Chapter 2
Enabling Interactions on the Web: A Taxonomic Perspective

The growth of the Web is revolutionizing the way organizations interact with their partners and customers. Businesses and government agencies are moving or have already moved their main operations to the Web to take advantage of the potential of more automation, efficient business processes, and global visibility [63, 64]. This has elicited the formation of alliances in which different partners join their applications and systems to share costs, skills and resources in offering value-added services. The ultimate goal is to have inter and intra-organization applications evolve independently, yet allow them to effectively and conveniently *interact* with each other. *Interaction* is defined as consisting of *interoperation* and *integration* with both internal and external enterprise applications.

Interactions among loosely coupled and tightly coupled systems has been, over the past twenty years, an active research topic in areas such as databases, knowledge-based systems, and digital libraries [24, 173]. However, the emerging Semantic Web has opened new research avenues because of issues such as semantics, heterogeneity, scalability, and automation. The Semantic Web requires the integration and interoperation of both applications and data. Disparate data representations between partner's systems must be dealt with. Interaction is also required at a higher level for connecting (i) front-end with back-end systems, (ii) proprietary/legacy data sources, applications, processes, and workflows to the Web, and (iii) partners' systems.

In this chapter, we survey the main issues and concepts to interactions on the Semantic Web [140, 233]. We propose a framework for comparing Semantic Web interaction technologies. The framework identifies the interaction layers, i.e., communication, content, and business process. It also proposes a set of dimensions to study interaction solutions. We present an in-depth study of interaction technologies on the pre Semantic Web and Semantic Web eras, and compare these technologies using the proposed framework. Previous work dealing with interoperation in loosely coupled systems mostly focused on databases and digital libraries [193, 175]. Recent surveys addressing in-

B. Medjahed and A. Bouguettaya, *Service Composition for the Semantic Web*,
DOI 10.1007/978-1-4419-8465-4_2, © Springer Science+Business Media, LLC 2011

teractions on the Semantic Web (e.g., [3, 31, 62, 65, 117, 194]) were mostly fragmented and lacked a holistic view of the problem.

The Chapter's organization reflects the historical evolution of interaction technologies in the pre-Semantic Web and Semantic Web eras. In Section 2.1, we present a typical architecture for a Semantic Web Interaction framework. In Section 2.2, we define the different interaction layers. We then identify a set of dimensions for comparing interaction solutions across these layers. In Section 2.3, we study several popular interaction solutions for the pre-Semantic Web, namely, EDI, components, and workflows. These solutions are evaluated against a pre-defined set of dimensions. In Section 2.4, we survey and evaluate the trends in supporting interactions in the Semantic Web. These include ontologies, Web services, semantic Web services, software agents, and XML-based standards. In Section 2.5, we describe current commercial platforms for supporting interactions on the Web. In Section 2.6, we overview Web service composition research prototypes. Finally, Section 2.7 provides a tabular comparison summary of the existing solutions for Semantic Web interactions.

2.1 Architecture of a Web-based Interaction Framework

Web-based applications generally involve several partners that interact via computerized systems (e.g., Web servers, networking services, databases) for conducting their daily business (e.g., exchanging documents, selling products, filing taxes) [29]. The building blocks for enabling such applications are provided through an *interaction framework* (Figure 2.1). These include modules for (1) defining and managing internal and external *business processes*, and (2) integrating those processes, and (3) supporting interactions with back-end application systems such as ERPs (*Enterprise Resource Planning*) [31]. A *business process* is defined as a multi-step activity that supports an organization's mission such as manufacturing a product and processing insurance claims [31].

We depict in Figure 2.1 the main components of an interaction framework. Translation facilities (e.g., application adapters) may be used to interconnect back-end systems (e.g., databases, ERPs) and internal business processes (e.g., workflows, applications). An external business process implements the business logic of an organization with regard to its external partners such as processing messages sent by trading partners' systems. Interactions between partners' external business processes may be carried out based on a specific *standard* (e.g., EDI [163, 49], *RossettaNet* [48]) or bilateral agreements. *Interaction standards* define the format and semantics of messages (e.g., request for quote), bindings to communication protocols (e.g., HTTP, FTP), business process conversations (e.g., joint business process), security mechanisms

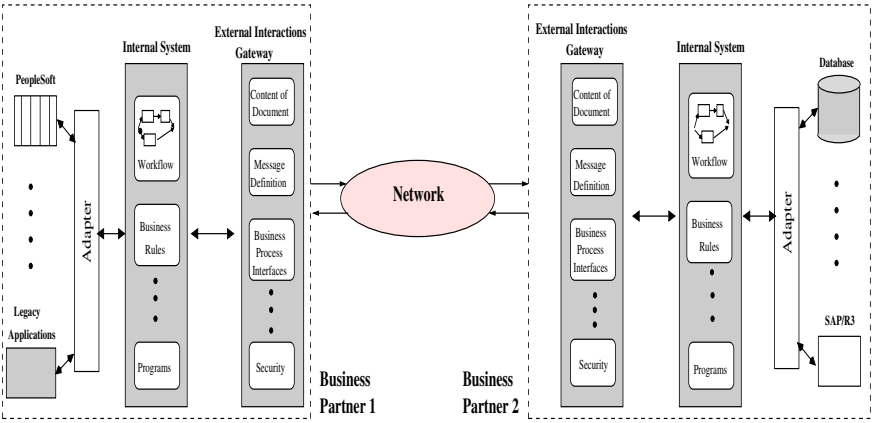


Fig. 2.1 Architecture of a Web-based Interaction Framework

(e.g., encryption, non-repudiation), etc. Interaction frameworks may have to support several standards and proprietary interaction protocols.

Figure 2.2 depicts an example of interaction between the `AreaAgencyAging` and `HealthDepartment`. Assume that *John*, `AreaAgencyAging`'s case worker, wants to organize a walk-in immunization for a group of disabled senior citizens. Such immunization is fee-based and provided by a separate agency, namely the `HealthDepartment`. *John* first issues a *request for purchase*. Upon approval of this request, a *purchase order* is issued and sent to `HealthDepartment` along with the list of citizens interested by the immunization campaign. The purchase order is transformed into an *immunization order* at `HealthDepartment`'s order processing system. After satisfactory *credit check*, an *order fulfillment* is issued by `HealthDepartment`. An *invoice* and *immunization schedule* is finally sent to `AreaAgencyAging`.

2.2 A Taxonomy for Semantic Web Interactions

In the first part of this section, we identify the different layers that make up an interaction framework on the Semantic Web. We then define the dimensions for assessing interactions across these layers. These dimensions are used as a benchmark for evaluating Semantic Web interaction solutions.

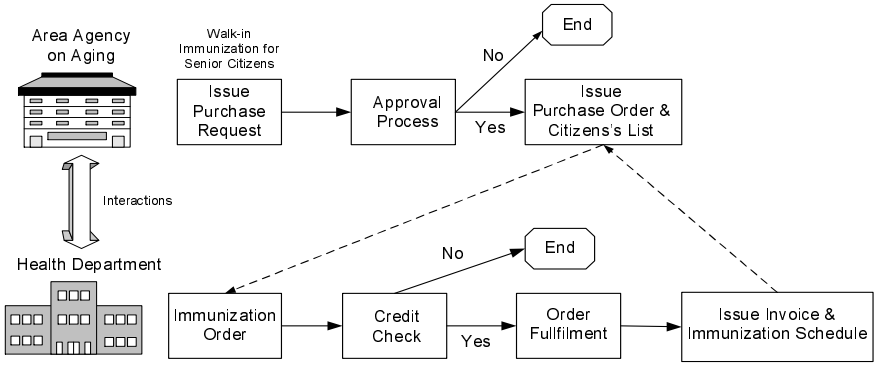


Fig. 2.2 Semantic Web Interactions: A Running Example

2.2.1 Interaction Layers

Interactions on the Semantic Web occur in three layers: *communication*, *content*, and *business process* layers (Table 2.1). For example, `AreaAgencyAging` and `HealthDepartment` need to agree on their joint business process: *John* expects to receive an invoice and immunization schedule from the `HealthDepartment` after sending a purchase order. `HealthDepartment` needs also to "understand" the content of the purchase order sent by `AreaAgencyAging`. Finally, there must be an agreed upon communication protocol to exchange messages between `AreaAgencyAging` and `HealthDepartment`.

Layer	*Definition*	*Examples of Enabling Tech.*
Communication	Protocols for exchanging messages among remotely located partners	HTTP, SOAP, REST, CORBA ORB, etc.
Content	Languages and models to describe and organize information in such a way that it can be understood and used	Ontologies, XML-based standards (e.g., eCO, cXML), etc.
Business Process	Enable autonomous and heterogeneous partners to engage in peer-to-peer interactions with each other	Web services (e.g., BPEL4WS), XML-based standards (e.g., RosettaNet), Inter-enterprise Workflows

Table 2.1 Interaction Layers

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. For example, `AreaAgencyAging` and `HealthDepartment` may use *Java RMI*

(*Remote Method Invocation*) [156] and IBM's *MQSeries* [104] respectively for internal communications. The objective of integration at this layer is to achieve a seamless integration of the communication protocols.

The content layer provides languages and models to describe and organize information in such a way that it can be understood and used. Content interactions require that the involved systems understand the semantics of content and types of business documents. For instance, if `HealthDepartment` receives a message that contains a document, it must determine whether the document represents a purchase order or request for quotation. Information translation, transformation, and integration capabilities are needed to provide for reconciliation among disparate representations, vocabularies, and semantics. The objective of interactions at this layer is to achieve a seamless integration of data formats, data models, and languages. For example, if `AreaAgencyAging` uses xCBL (*XML Common Business Library*) [47] to represent business documents and `HealthDepartment` expects documents in cXML (*Commerce XML*) [54], there is a need for a conversion between these two formats.

The business process layer is concerned with the conversational interactions (i.e, joint business process) among services. Before engaging in a transaction, `AreaAgencyAging` and `HealthDepartment` need to agree on the procedures of their joint business process. The semantics of interactions among `AreaAgencyAging` and `HealthDepartment` 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 interactions 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. Interoperability at this higher level is a challenging issue because it requires the understanding of the semantics of partner business processes.

2.2.2 Dimensions for Semantic Web Interactions

The Semantic Web covers a wide spectrum of interactions among different partners. The type of interactions depend on the usage scenarios, involved parties, and business requirements. Each framework makes specific tradeoffs with regard to the requirements of Semantic Web interactions. It is therefore important to determine the relevant requirements and understand the related tradeoffs when evaluating models of interactions. In this section, we identify the following set of dimensions to study interaction issues the Semantic Web (Table 2.2):

- *Coupling among partners:* this dimension refers to the degree of *tightness* and *duration* of coupling among business partners. Two partners are *tightly*

Dimension	*Definition*
Coupling among Partners	Degree of *tightness* and *duration* of coupling among business partners
Heterogeneity	Degree of dissimilarity among business partners
Autonomy	Degree of compliance of a partner to the global control rules
External Manageability	Degree of external visibility and manageability of partners' applications.
Adaptability	Degree to which an application is able to quickly adapt to changes
Security	Measures must be in place to boost partners' confidence that their transactions are safely handled
Scalability	Ability of a system to grow in one or more dimensions such the number of relationships that can be supported

Table 2.2 Interaction Dimensions

coupled if they are strongly dependent on each other. For example, one partner may control the other, or they may control one another. *Loosely coupled* partners exchange business information on demand. The *duration* of a relationship may be *transient* (also called *dynamic*) or *long term*. In *transient* relationships, businesses may need to form a fast and short term partnership (e.g., for one transaction), and then disband when it is no longer profitable to stay together. Partners need to dynamically discover partners to team up with to deliver the required service. In *long term* relationships, businesses assume an *a priori* defined partnership.

- *Heterogeneity:* heterogeneity refers to the degree of dissimilarity among business partners. The need to access data across multiple types of systems has arisen due to the increased level of connectivity and increased complexity of the data types. Applications use different data structures (e.g., XML, relational databases), standard or propriety semantics (e.g., standardized ontologies). There may also be structural heterogeneity at the business process layer (e.g., use of APIs, document exchange protocols, inter-enterprise workflows). In addition, organizations may, from a semantic point of view, use different strategies for conducting business that depend on business laws and practices [34].
- *Autonomy:* autonomy refers to the degree of compliance of a partner to the global control rules. Partner systems may be autonomous in their design, communication, and execution. This means that individual partners select the process and content description models, programming models, interaction models with the outside world, etc. In a fully autonomous collaboration, each partner is viewed as a black box that is able to exchange information (i.e., send and receive messages). Partners interact via well-defined interfaces allowing them to have more local control over implementation and operation of services, and flexibility to change their processes without affecting each other. Usually, a completely autonomous collaboration may be difficult to achieve because it may require sophisticated translation facilities.

- *External Manageability:* this dimension refers to the degree of external visibility and manageability of partners' applications. In order to be effectively monitored by external partners, an application must be defined in a way that facilitates the supervision and control of its execution, measurement of its performance, and prediction of its status and availability. For example, `AreaAgencyAging` may need to get the status (e.g., pending, approved) of the purchase order sent to `HealthDepartment`. This requires that `HealthDepartment` exposes sufficient information pertaining to measurements and control points to be used by `AreaAgencyAging`. While desirable in principle, high visibility may require complex descriptions of partners' applications. However, the overhead to provide such descriptions may be well justified if it provides other advantages such as *Quality of Service* (QoS).
- *Adaptability:* adaptability refers to the degree to which an application is able to quickly adapt to changes. Semantic Web applications operate in a highly dynamic environment where new services could come on-line, existing services might be removed, and the content and capabilities of services may be updated. Businesses must be able to respond rapidly to changes whereby both operational (e.g., server load) and market (e.g., changes of availability status, changes of user's requirements) environment are not predictable. For example, if `HealthDepartment` decides to stop its walk-in immunization activities, `AreaAgencyAging` would then need to adapt to such change. Changes may be initiated to adapt applications to actual business climate (e.g., economic, policy, or organizational changes). They may also be initiated to take advantage of new business opportunities. Since applications interact with both local back-end systems and partner applications, it is important to consider the impact of changes in both local and external applications to ensure local and global consistency. In general, the impact of changes depends on the degree of tightness among applications.
- *Security:* security is a major concern for inter-enterprise applications. Before Semantic Web applications reach their real potential, sophisticated security measures must be in place to boost partners' confidence that their transactions are safely handled [227]. For instance, `HealthDepartment` may need to check the authenticity of the purchase order before processing it. Semantic Web applications must support mutual authentication, fine grain authentication, communication integrity, confidentiality, non-repudiation, and authorization. Interactions may be based on limited mutual trust, little or no prior knowledge of partners, and transient collaborative agreements. Shared information may include limited capabilities of services.
- *Scalability:* scalability refers to the ability of a system to grow in one or more dimensions such as the volume of accessible data, the number of transactions that can be supported in a given unit of time, and the number of relationships that can be supported. More importantly, changes in

business climate are forcing organizations to merge to be effective in the global market. Thus, the cost and effort to support new relationships is an important criterion to consider when evaluating interaction solutions in the Semantic Web. Clearly, a low cost establishment of new relationships is desirable. However, in case of long-term relationships, the cost of establishing a new relationship is not of great significance.

2.3 Interactions in the Pre Semantic Web Era

Technologies for interactions on the pre-Web era have been around for almost three decades providing businesses, such as the banking industry, with a secure framework for sharing and exchanging data electronically. The most widely used and earliest framework is the *Electronic Data Interchange* (EDI) standard that runs on dedicated computer networks. Later, advances in software technology gave rise to a new breed of affordable software for distributed messaging and computing that can securely run on public computer networks: *component-based frameworks*. With corporate takeovers and consolidations coupled with the need of agile, just-in-time inter-enterprise cooperation on the Web, pressure mounted to provide solutions for enabling *inter-enterprise workflows*. Tomorrow's silver bullet applications such as *Virtual Enterprises* [17, 81, 82], will heavily draw on these solutions. In this section, we describe major interaction frameworks developed in the pre Semantic Web era: EDI, components, and workflows.

2.3.1 Electronic Data Interchange (EDI)

EDI [163, 49] is commonly defined as the inter-organizational application-to-application transfer of business documents (e.g., purchase orders, invoices, shipping notices) between computers in a compact form. Its primary aim is to minimize the cost, effort, and time incurred by the paper-based transfer of business documents [2]. EDI documents are structured according to a standard (e.g., ANSI X12 [49] and UN/EDIFACT [163]) and machine-processable format.

Figure 2.3 depicts two trading partners `AreaAgencyAging` and `HealthDepartment` exchanging business documents via a *Value-Added Network* (VAN). The document (e.g., purchase order) must be created in the business application of the sender (i.e., `AreaAgencyAging`). The *mapper* software is used to describe the relationship between the information elements in the application and the EDI standard. The EDI *translator* software converts the document into an EDI message according to the standard used.

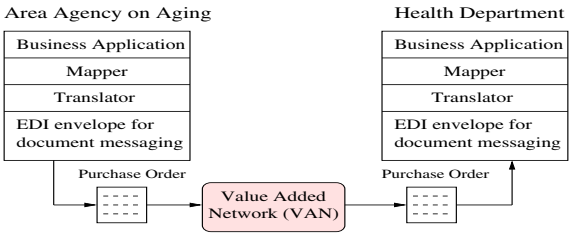


Fig. 2.3 EDI-based Interactions

The translator wraps the EDI message in an electronic envelope that has an identifier for the receiver (i.e., `HealthDepartment`). The actual transmission of the electronic envelope is performed by the *communication* software. This software maintains the trading partners' phone numbers to dial-up and exchange operations. The communication software can be a separate application or part of the translator. The VAN reads the identifier on the envelope and places it in the mailbox of `HealthDepartment`. At the `HealthDepartment` side, the reverse process takes place.

2.3.1.1 Interactions in EDI-based Solutions

EDI focuses mostly on interoperability at the communication and content layers. VANs are used to handle message delivery and routing among business partners. EDI standards provide a single homogeneous solution for content interoperability. They define a set of types for describing business documents. However, there is a limited (albeit large) number of predetermined documents supported by EDI standards. While these documents represent a large number of business transactions (e.g., shipping invoices, health care claim status reports), companies are limited to that set of EDI documents for which standards already exist [2]. It would be difficult for trading partners to conduct transactions whose parameters are not included in an EDI document. In that regard, EDI is hardly flexible in its ability to expand the set of supported document types. The introduction of a new type or changing an existing type of business transaction is complex and time consuming [2]. This kind of changes requires modification to the configuration of the translation software and must be validated in the related standard or EDI guideline committee which usually takes a long time [2]. For example the *EDI Guideline Consistency Subcommittee* (EGCS) is responsible for the content and maintenance of all TCIF (*Telecommunications Industry Forum*) EDI-maintained code lists [9]. Any modification to these code lists has to be reviewed by the EGCS. The EGCS is also responsible for notifying the *TCIF*

Secretariat of any changes in the electronic documentation. Interoperability at the business process layers is supported through pre-defined business processes. For example, if `AreaAgencyAging`'s purchase order is accepted then the `AreaAgencyAging` expects a purchase order acknowledgment, an invoice, and the immunization schedule.

The EDI approach is particularly strong along the criteria of security and heterogeneity. EDI is based on document exchange over private or value-added networks. Business partners do not concern themselves with those security issues encountered in public networks. Moreover, business partners do not need to directly reference each other systems. Therefore, critical security issues are bypassed. All partners are required to comply with the EDI standard. As a result, heterogeneity is not a problem. However, understanding all information in an EDI document is not a simple task. For example, there are data elements (UNH and UNT) in EDI document whose sole purpose is to indicate the start and end of a message. The impact of local changes is limited as partners do not directly reference each others' systems.

Although several EDI implementations have shown impressive results as set in the example of SEWP [162], the cost of establishing a new relationship usually requires a significant overhead. Because EDI is based on proprietary and expensive networks, organizations, predominantly small and medium, could not afford EDI. They were, *de facto*, excluded from being partners with larger organizations that mandate the use of EDI [2, 114]. Typically, VAN services entail three types of costs: account start-up costs, usage of variable costs, and VAN-to-VAN interconnect costs for the number of characters in each document [114]. The final cost of an EDI solution depends on several factors such as the expected volume of documents, economics of the EDI translation software, and implementation time. Maintenance fees and VAN charges can vary considerably and as such affect the cost of EDI systems. Some VAN providers do their billing on a per document basis. Others charge based on the number of characters in each documents [114]. It has been reported that 90% of the Fortune 500 companies in the United States uses EDI; only 6% of the other 10 million companies can make that claim [2]. Efforts to reduce the cost of using VAN networks include Internet-based EDI solutions such as EDIINT [109] and OBI [168].

Each EDI deployment involves negotiation and agreement on a set of implementation conventions describing the extensions to the standard documents and actual formats that would be exchanged. This negotiation and agreement process represents a significant cost in EDI deployment. To address this issue, EDIFACT and ANSI X.12 have undertaken an effort to standardize sets of documents for various industries. For example, ANSI X.12 has recently released a set of standard EDI document definitions for the health care industry. Using these industry standard document definitions, the customizations required per relationship can be reduced, although per-relationship work is generally still required. Additionally, once implementation conventions are decided upon, custom integration work must be performed at both partner

organizations for the existing enterprise systems to process the EDI documents. This typically involves purchasing a commercial EDI system, integrating it with the enterprise systems, and writing custom code to translate the EDI system document definitions to the corresponding enterprise system records.

2.3.1.2 Internet-based EDI Initiatives

EDI has been extended in many directions. For instance, business documents in EDI standards have been mapped to XML documents (e.g., XML/EDI [50]). More specifically, the combination of EDI and Internet technologies seems to overcome several shortcomings of the traditional EDI (e.g., VAN charges). Indeed, several organizations are already using EDI for transacting over the Internet. For example, EDI purchase orders and invoices are now routinely exchanged via the Internet by NASA, Sun Microsystems, and Cisco systems. Major Internet-based EDI initiatives include EDIINT (*EDI over the Internet*) [109] and OBI (*Open Buying on the Internet*) [168].

EDIINT [109] – EDIINT is essentially the same as traditional EDI, but uses the Internet as a communication medium instead of VANs. The aim is mainly to reduce EDI communication charges due to the use of VANs. EDIINT was initiated by the *Uniform Code Council* (UCC) to standardize the method to exchange EDI documents over the Internet. EDIINT is similar to EDI in terms of interoperability at the content and business process layers. At the communication layer, the first EDIINT standard (emerged in 2000) was EDIINT AS1 (*Applicability Statement 1*). EDIINT AS1 set the rules to exchange EDI documents using SMTP protocol. The second standard (completed in 2001) was EDIINT AS2 standard. It supported communication of EDI documents using the HTTP protocol.

Initially, there was reluctance to use the Internet for exchanging critical business information due to concerns about security. To deal with this problem, EDIINT AS2 specifies standard mechanisms for securing documents using PGP (*Pretty Good Privacy*) encryption and digital signatures [110]. The standards referenced by EDIINT AS2 include RFC1847 and MIME Security with PGP [110]. EDIINT offers lower entry cost than EDI since it is Internet-based. However, the quality of service (e.g., automatic error detection and correction) associated with VANs is lost. EDIINT offers similar characteristics as EDI with respect to the other dimensions (i.e, coupling, heterogeneity, autonomy, external manageability, and adaptability).

OBI [168] – OBI is a standard that leverages EDI to define an Internet-based procurement framework. It is clearly stated that OBI aims to complement EDI standards, not replace them. OBI is intended for high-volume, low-dollar amount transactions, which account for 80% of the purchasing activities in

most organizations. At the communication level, OBI uses HTTP protocol for exchanging messages. OBI relies on the ANSI X12 EDI standard to describe the content of order documents. Order documents are encapsulated in OBI objects. OBI objects also encapsulate other non-EDI messages such as buyers' and sellers' digital signatures. OBI does not introduce a specific model for describing locally maintained information (e.g., product and price information). This information may be described in the partner's database. At the business process level, OBI defines a simple and pre-defined operational protocol for Internet-based purchasing. This protocol consists of a number of commonly agreed upon activities (e.g., select a supplier, create order) for purchasing non-strategic material (e.g., office supplies, laboratory supplies). In fact, this protocol only specifies the way partner OBI systems interact. It is the responsibility of each partner to integrate its internal applications (catalogs, inventory and order management systems, etc) with OBI servers.

OBI makes a strong attempt to provide a robust security infrastructure. It uses the SSL (*Secure Sockets Layer*) [164] over HTTP for securing communications. It also uses digital signatures and digital certificates for ensuring messages authenticity and integrity. OBI rates higher than EDI with regard to the scalability and adaptability dimensions. First, the extensibility of order documents is not an important requirement. OBI targets simple and predefined purchasing transactions. Second, it offers lower entry cost as it is an Internet-based framework. OBI offers similar properties as EDI and EDIINT with regard to the other dimensions (i.e., coupling, heterogeneity, autonomy, and external manageability).

2.3.2 Software Components

Software components (simply, *components*) are program modules that can be independently developed and delivered [22, 203]. They may be newly developed or wrap existing functionalities provided by databases, legacy systems or packages. Although most of the fundamental ideas that define object technology are applicable to components, components are not necessarily created using object-oriented tools and languages [153, 96]. For example, components may be realized using a functional language, an assembly language, or any other programming language [203].

The development of component-based applications generally requires the interconnection of geographically distributed components. The availability of a *middleware* that provides more effective ways of programming is important to the development of distributed component-based applications. A *component middleware* is an infrastructure that supports the creation, deployment, and interactions among components [210]. Figure 2.4 depicts `AreaAgencyAging`'s and `HealthDepartment`'s applications assembled from components. Each component represents an independent unit of a business

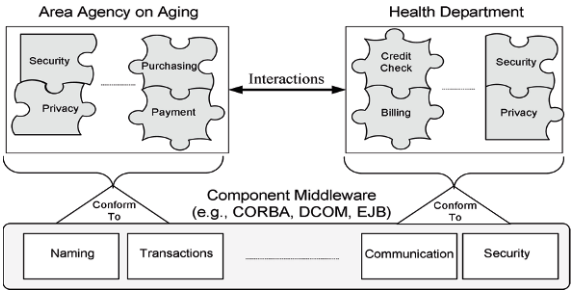


Fig. 2.4 Component-based Interactions

functionality such as *payment*, *purchasing*, *privacy*, and *security*. The different components work together to serve the needs of `AreaAgencyAging`'s and `HealthDepartment`'s business processes. They are built on top of a set of basic services. Functions provided by these services include distributed communication, security, transactions, and naming schema.

Three major component middleware frameworks have been developed during the past decade:

- CORBA (*Common Object Request Broker Architecture*) [170]: CORBA is the standard promoted by the OMG (*Object Management Group*), an international industry consortium. It is part of a general architecture called the *Object Management Architecture* (OMA). The backbone of CORBA is the *Object Request Broker* (ORB) which allows communication between client and server components.
- DCOM (*Distributed Component Object Model*) [154]: DCOM is Microsoft's technology for distributed components. It is an extension of COM, the Microsoft's component software architecture. COM and its DCOM extension are merged into a single runtime which provides both local and remote access.
- EJB (*Enterprise Java Beans*) [184]: EJB is one of several technologies which make up Sun's J2EE (*Java 2 Platform, Enterprise Edition*) specification. It provides a component model for the Java programming language. In EJB, pieces of business logic or functions can be written in Java and encapsulated to become components known as *beans*. The *container* is the core of EJB component model. It provides a runtime environment that hosts and controls the beans.

The component-based approach is more appropriate for a small number of partners within an enterprise [45]. However, with companies being merged and acquired at the current rate, there is a need to address interactions within an enterprise. Components mainly cover interactions at the communication

layer. They exhibit limited capabilities dealing with interactions at the content layer. They focus on the syntactic integration to wrap heterogeneous applications. At the business process layer, applications (e.g., scheduling an immunization campaign for `AreaAgencyAging`) may be assembled from independently developed components (e.g., payment, purchasing, privacy, security) [198, 79]. However, businesses generally would need to develop *ad hoc* solutions for defining intra and inter-enterprise business processes.

2.3.2.1 CORBA-based Interactions

At the communication layer, the use of ORBs in CORBA hides the underlying complexity of network communications from application developers. When a client issues a method invocation on a server component, the ORB intercepts the invocation and routes it across the network to the appropriate server. It is also possible that components distributed on different ORBs communicate over the Internet through the *Internet Inter-ORB Protocol* (IIOP).

CORBA provides a *trader service* through which businesses can find each other by assigning a set of properties to each component. However, these properties are simply defined as (name,value) pairs. They do not provide support for semantic description of components. Recent efforts have been made to add semantic features to CORBA through the ECDTF (*Electronic Commerce Domain Task Force*) reference model which includes a *semantic data* facility [170]. However, the model is still at its very early stage. Additionally, very little work has been done so far to define a specification for the semantic data facility.

CORBA enable tightly coupled and long term business relationships between components. Once interfaces are expressed in IDL (*Interface Definition Language*), they are compiled by an IDL compiler into *stubs* and *skeletons*. The *stub*, used on the client side, invokes remote operations via the ORB to the corresponding skeleton on the server side. The *skeleton* gets the call parameters, invokes the actual operation implementation, collects results, and returns values back to the client through the ORB. Efforts are being made to add messaging capabilities to CORBA [45]. The new messaging specification defines a number of asynchronous and time-independent invocation modes for CORBA. It allows both static and dynamic invocations to use all modes. The use of message driven interactions among components allows the support of loosely coupled relationships. CORBA components are mostly based on static operation invocation. Although the *Dynamic Invocation Interface* (DII) in CORBA allows components to learn about other components' operations at run time, the utility of DII is yet to be exploited due to its complexity.

Components shield application developers from implementation details. Interfaces are the only considerations businesses must make when interacting with each other. Business partners have the latitude to implement their

interfaces in ways that best fit their internal needs and requirements. Each CORBA component has an IDL that includes the name of the operations to be called by clients together with the name and types of all parameters and return values. However, all participants in a certain market need to agree on a predefined interface. This means that businesses are bound to interfaces published by their trading partners. In terms of heterogeneity, CORBA was designed to be independent of implementation languages, operating systems, and other factors that normally affect interactions. Components can be implemented using diverse programming language such Java, C++, and Smalltalk.

External manageability is partially addressed in CORBA through the *event service*. The CORBA *event service* allows components to inform each other of the occurrence of specific events. It divides components into suppliers and consumers. Suppliers generate notifications of events while consumers register to be notified about the occurrence of events so that they can perform specific actions in response of those events.

CORBA provides little or no support for adaptability. As mentioned before, businesses are tightly bound to interfaces published by their trading partners. Hence, any change to a partner's interface may need the corresponding interface to be re-compiled. To date, CORBA does not provide mechanism to respond rapidly to changes in component interfaces.

Security is addressed in CORBA through the CORBA *security service*. This service provides a number of mechanisms such as authentication, authorization, and encryption of messages to build secure Web-based applications. Major CORBA vendors provide implementations of the security service.

The complexity of CORBA development increases the cost of entry in CORBA-based solutions. For example, developers in CORBA must generate binary code packages and deploy them on client sides when building new applications or when modifying the interfaces of existing applications. Although the dynamic invocation interface in CORBA alleviates this problem, programming calls with such interface is fairly complicated.

2.3.2.2 DCOM-based Interactions

Similarly to CORBA, DCOM-based solutions mainly deal with interactions at the communication layer. They present little or no support for interactions at the content and business process layers. For a DCOM client to access an operation of another component at the communication layer, it must use virtual lookup tables to obtain a pointer to that operation. The DCOM runtime environment ensures that the pointer is local to the invoking process by using proxies [121].

DCOM components enable tightly coupled and long term business relationships. Proxies need to be created at the client side to communicate with stubs on the serving end [121]. The operation invocation process is static

in DCOM which prevents establishing dynamic relationships among components. In terms of heterogeneity, current DCOM implementations are mostly based on Windows platforms although some experimentation have been done to port DCOM to other platforms (e.g., UNIX). Also, the languages that are mostly used to write DCOM components are Microsoft J++ (Microsoft's implementation of Java), C, C++, and Visual Basic. Additionally, DCOM's IDL is neither CORBA nor DCE (*Distributed Computing Environment*) compliant [121]. Security in DCOM relies on the Windows NT security model. Although this allows developers to build secure applications on Windows platforms, it is not clear how security will be provided when DCOM is used on other platforms. DCOM has similar characteristics as CORBA with respect to autonomy, external manageability, adaptability, and scalability.

2.3.2.3 EJB-based Interactions

At the communication layer, EJB uses the *Java RMI* [156] to enable interactions among beans. The use of RMI makes the location of the server transparent to the client. Similarly to CORBA and DCOM, EJB is fairly limited in terms of interactions at the content and business process layers.

Similarly to CORBA and DCOM, EJB caters for tightly coupled and long term business relationships. Developers must define an RMI remote interface for each bean. The RMI compiler generates a stub for each remote interface. The stub is installed on the client system and provides a local proxy for the client. The stub implements all the remote interfaces and transparently delegates all method calls across the network to the remote bean. A new specification of EJB (EJB Version 2) has recently been made available. It uses JMS (*Java Messaging Service*) to add support for message driven beans, extending the EJB component model to support both tightly and loosely coupled applications [45]. Static operation invocation is found in most EJB implementations. However, some implementations such as *JBoss* integrate more dynamic features.

In EJB, each bean has a *remote* interface which defines the methods that carry out the business logic of the bean. The EJB remote interface provides functions that are similar to those provided by CORBA and DCOM IDL. Hence, EJB is similar to CORBA and DCOM in terms of autonomy. EJB does not support heterogeneous platforms although it is fully based on Java. Indeed, most of the current EJB implementations do not offer direct interoperability with non-Java platforms. In addition, communicating between components deployed on heterogeneous application servers, such as invoking a *BEA WebLogic* component from an *IBM WebSphere* server, requires operations in degraded mode.

Several implementations of an event service have also been provided for EJB to support external manageability. An example of EJB's event service is the *Drala Event Broker* [66]. EJB provides some support for adaptability by

associating a *deployment descriptor* to each bean. The descriptor describes the way in which a bean interacts with its environment. Application developers declaratively define contracts in their descriptors. This contract describes the type of services (such as the form of transaction management to be used) required by the bean. It can be changed independently of the business logic.

The EJB container provides security features to EJB components. Each deployment descriptor contains declarations about the access control for the corresponding enterprise bean. When a client calls an operation of that bean, the container is responsible for checking that the requester has the right to invoke that operation by accessing an access control list. Finally, EJB offers similar properties as CORBA and DCOM with respect to scalability.

2.3.3 Workflow

Workflow management is concerned with the declarative definition, enactment, administration and monitoring of *business processes*. A *business process* (or workflow process) consists of a collection of activities related by data and control flow relationships (Figure 2.5). An activity is typically performed by executing a program, enacting a human/machine action, or invoking another process (called sub-process). Programs, persons, machines, and data used to perform workflow processes are called *workflow resources*. For example, `AreaAgencyAging`'s business process includes several activities such as issuing a purchase request, approving it, and issuing a purchase order. The information sent from `AreaAgencyAging`'s *Issue_Purchase_Request* activity to the *Approval_Process* activity includes the number of citizens eligible for immunization service. The scripting of activities and resource policies through *business process analysis, modeling, and definition tools* defines a *business process definition* (workflow schema) [57]. The *workflow enactment service* enables different parts of the business process to be enacted by providing interfaces to users, applications, and databases distributed across the workflow domain (Figure 2.5).

Workflow is a key technology for automating business processes that involve access to several applications. However, traditional workflow systems are ineffective when we consider the needs of Web-based applications, with their complex partnerships, possibly among a large number of highly evolving processes. Current efforts (e.g., the Business Process Initiative - BPMI.org) promise to deliver a next generation workflow systems (*Inter-Enterprise Workflow Systems* - IEWSs) that have the ability to thread together cross-organizational business processes, supporting the integration of diverse users, applications, and systems [227]. IEWSs focus mainly on interactions at the business process layer. Their purpose is to automate business processes that interconnect and manage communication among disparate systems.

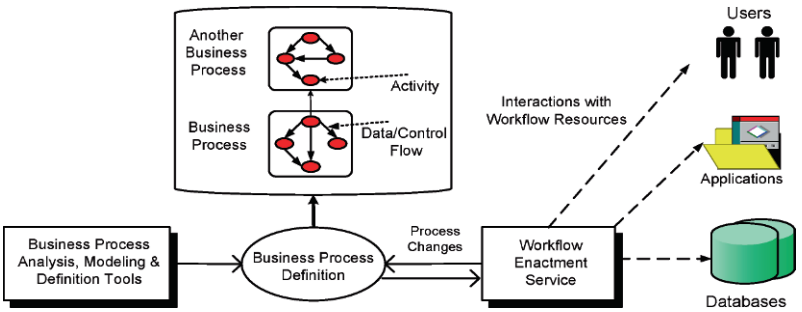


Fig. 2.5 Workflow System Characteristics

2.3.3.1 Distributed Workflow Systems

The emphasis in *Distributed Workflow Systems* (DWSs) is on partitioning the overall workflow specification into several sub-workflows, each encompassing all the activities that are to be executed by a given entity within an organization [160]. DWSs impose that each organization participating in a distributed workflow deploy a full-fledged execution engine, capable of interpreting the workflow definition. The same workflow model must be adopted by each participant in the global workflow. This approach assumes that global and sub-business processes use the same process definition and data exchange model. This is a quite restrictive assumption in the context of Web-based applications where: (i) partners may use disparate data and process representation models (e.g., `AreaAgencyAging` uses EDI and `HealthDepartment` uses RosettaNet), and (ii) private business processes may require access to proprietary/legacy data sources and applications (e.g., Oracle database for `AreaAgencyAging` and SAP application for `HealthDepartment`). In addition, DWSs assume a tight coupling model among the distributed sub-workflows. Thus, modifications to back-end applications, sub-workflows, and global workflow need to be coordinated. The cost of establishing a new relationship may be significant as business processes must be modeled and deployed in concert across all participants. DWSs are appropriate for the development of a business process of a single organization that needs to integrate multiple distributed sub-workflows.

2.3.3.2 Collaborative Process Management

Inter-enterprise business processes management features the separation between *public* and *private processes* [31, 57]. A *public process* defines an external message exchange of an organization with its partners according to a message

exchange protocol such as EDI and RosettaNet. A *private process* describes internal executable activities that support the activities of public processes. Public and private processes interact through *process wrappers*. Process wrappers consist of pre-defined activities that can be used in a private business process to send/receive messages to/from public business processes. For example, if a public process uses *xCBL* [47] to represent business documents, and the private business process expects documents in *cXML* [54], the conversion between these two formats is handled by a wrapper. Private processes may also interact with back-end applications through *application adapters*. In this approach there is no requirement that local process management engines (e.g., engines which are responsible for managing private business processes) be identical. It is possible for example, that one engine is based on IBM's *MQSeries* [104] and another based on HP's *Process Manager* [100].

The separation between back-end applications, public, and private processes has the advantage that local changes (i.e, those that concern only private processes) have no impact on public processes and back-end applications. However, changes related to interactions (e.g., changing the formats of incoming and outgoing messages) between a public process (or a back-end application) and a private business process may require the modification of some wrappers. The separation between components of a Web-based application (public processes, private processes, business rules, and back-end systems) contributes to the scalability of this approach. The support of a new interaction protocol (e.g., EDI) requires only the creation of a new public process and process wrappers. The support of new a back-end application requires the creation of new application adapters. The creation of a new relationship with a new partner may require a few adjustments. If the new partner does not comply to an already supported interaction protocol, a new public process must be created to support the protocol used by the new partner. The support of a new back-end application requires only the creation of a new application adapter. The above discussion shows that the addition of interaction protocols, back-end applications, or partners does not require the modification of private business processes.

The separation between public and private business process provides for a greater degree of autonomy and bridging of heterogeneity. With regard to security, IEWs may leverage techniques used in other frameworks (i.e, document-based or component-based frameworks). External manageability can be provided by adding specific activities in public processes.

2.4 Trends in Supporting Semantic Web Interactions

The traditional approaches for Web-based interactions were not devised for Semantic Web. They provide little or no support for capturing the semantics of messages exchanged among partners or partners' business processes.

Therefore, recent research has focused on developing concepts and techniques for enabling interactions on the Semantic Web. In this section, we overview major Semantic Web technologies, namely, ontologies, Web services, agents, and XML-based interaction standards.

2.4.1 Ontology

The Web is evolving from a set of single isolated application systems into a World wide network of disparate systems interacting with each other via information exchange. This requires means to represent the semantics of the exchanged information so that it could be automatically understood. This is where the concept of*ontology* would play a crucial role, providing a machine processable semantics of the information communicated between heterogeneous systems.

Ontologies were initially developed in Artificial Intelligence to facilitate knowledge sharing and reuse. They have since then been recognized as a popular research topic in various research communities such as knowledge engineering, e-commerce, natural language processing, cooperative information systems, and information integration. In this section, we define the concept of ontology and give the motivations behind it. We then overview major languages and standards for describing ontologies on the Web. Finally, we evaluate ontology-based systems using interaction layers and dimensions.

2.4.1.1 Modeling Semantics Through Ontology

An *ontology* is a *formal* and *explicit* specification of a *shared conceptualization* [60, 171]. "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, 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 described using a well-defined model or language called *ontology language*.

An ontology typically consists of a hierarchical description of important concepts in a domain, along with descriptions of the properties of each concepts. Formally, an ontology Ω contains a set of *concepts* (also called *classes*) $\{c_1,...,c_n\}$ which constitutes the core of the ontology. The notion of concept in ontologies is similar to the notion of class in object-oriented programming.

Each concept c_i has a set of *properties* $P_i = \{p_{i1},...p_{im}\}$ associated to it. This set describes the different features of the class. Each property p_{ik} has a *range* R_{ik} (also called *type*) indicating a restriction on the values p_{ik} can take. An ontology relates classes to each other through *ontology relationships*. Examples of relationships include "subclassof", "superclassof". Properties are also related through similar relationships such as "subpropertyof" and "supropertyof".

We identify three different types of ontologies depending on their generality level: *vertical*, *horizontal*, and *metadata* ontologies. Other types of ontologies such as *representational*, *method* and *task* ontologies also exist but are out of the scope of this book [71]. *Vertical* ontologies capture the knowledge valid for a particular domain such as medical, mechanic, chemistry, and electronic. *Horizontal* ontologies describe general knowledge that is valid across several domains. They define basic notions and concepts (e.g., time, space) applicable in many technical domains. *Metadata* ontologies provide concepts that allow the description of other concepts. For example, *Dublin Core* is an ontology for describing the content of on-line information source.

2.4.1.2 Ontology Models and Languages

Several ontology-based approaches have been developed for Web applications during the past few years. A common claim for all these approaches is the need for models and languages to specify ontologies. In this section, we overview a representative set of these languages and models: RDF/RDF Schema, Description Logic, DAML+OIL, and OWL. An exhaustive list of ontology languages for the Semantic Web is presented in [84].

RDF and RDF Schema [71]: RDF (*Resource Description Framework*) provides a domain-neutral framework to model metadata about *resources* on the Web. It is recommended by *World Wide Web Consortium* (W3C) working groups. A *resource* is any "thing" on the Web that has a *Uniform Resource Identifiers* (URI) such as Web pages and files. Besides resources, RDF's basic model includes *properties* and *statements*. A *property* is a characteristic, attribute, or relation that describes a resource. A *statement* consists of a specific resource with a named property plus that property's value for that resource. This value can be another resource or a literal value (free text). An RDF description is a list of triples: *(resource, property, value)*. RDF does not provide mechanisms for defining relationships between properties and resources. This is the role of *RDF Schema*. RDF Schema is a type system for RDF. It provides a mechanism to define domain-specific properties and classes of resources to which we can apply those properties. Basic primitives of RDF include *class* and *subclass* which together allow the definition of class hierarchy, *property* and *subproperty* to build property hierarchies, and *do-*

main and *range* to restrict the possible combinations of properties and classes.

Description Logic (DL) [10]: DLs are a family of logic based formalisms for describing knowledge. A DL considers unary relations (called concepts) and binary relations (called roles). A DL terminology includes a set of *primitive* concepts and roles and definitions of *complex* concepts and roles. The definitions of complex concepts and roles are given via *descriptions* that are built using a set of *constructors*. Consider an example in which the primitive concepts are *person*, *smart* and *footballPlayer*, and *child* is a primitive role. Various constructors can be used to define new concepts by imposing some restrictions on the number of fillers of a role, or on the classes to which fillers of a certain role have to belong. For instance, ($\leq$ 3 child) is a complex concept built by applying a number restriction constructor on the role child: it defines the set of objects that have at most 3 fillers for the role child. ($\forall$.child.smart) is a description which is built from the *universal quantification* constructor $\forall$, and defines the class of objects for which all fillers of the role child belong to the class smart. $\exists$child.footballPlayer uses the *existential quantification* constructor and defines the set of objects for which there exists at least one filler of the role child which belongs to the class footballPlayer. The following describes the complex concept c1, which is defined to include set of persons such that all of their children are smart, have at most 3 children, and have a child who is a football player: $c1 := person \cap (\forall child.smart) \cap (\leq 3child) \cap (\exists child.footballPlayer)$. A particular description logic language may include several constructors other that the Boolean ones such as the Number restrictions or roles also known as cardinality constraints (e.g., $\geq$ 3 hasChild), Qualified number restriction (e.g., $\geq$ 3 hasChild.Female), Nominal (e.g., c := Italy), Inverse roles (e.g., $hasChild^{-}$(hasParent)), Transitive role, and Role composition (e.g., $Uncl := hasParent \circ hasBrother$). Particular DL languages are characterized by the set of constructors allowed for building complex concepts and roles from simpler ones and by the set of axioms for asserting facts about concepts, roles and individuals.

DAML+OIL [99]: DAML+OIL is the result of a merger between DAML-ONT, a language developed as part of the US DARPA Agent Markup Language (DAML) and OIL (the Ontology Inference Layer), developed by a group of (mostly) European researchers. It builds on earlier Web ontology standards such as RDF and RDF Schema and extends those languages with richer modeling primitives (e.g., cardinality). DAML+OIL adopts an object oriented approach, describing ontologies in terms of classes, properties, and axioms (e.g., subsumption relationships between classes or properties) [97]. When a resource R is an instance of a class C, we say that R has type C. From a formal point of view, DAML+OIL can be seen to be equivalent to description logic (DL), with a DAML+OIL ontology corresponding to a DL terminology (Tbox). As in a DL, DAML+OIL classes can be names (URIs)

or expressions, and a variety of constructors are provided for building class expressions. Example of DAML+OIL constructors include *intersectionOf*, *unionOf*, *complementOf*, *oneOf*, *minCardinalityQ*, *maxCardinalityQ*. The meaning of the first three constructors (intersectionOf, unionOf and complementOf) is relatively self-explanatory: they are the standard boolean operators that allow classes to be formed from the intersection, union and negation of other classes. The oneOf constructor allows classes to be defined existentially, i.e., by enumerating their members. The minCardinalityQ, maxCardinalityQ, cardinalityQ constructors are known in DLs as qualified number restrictions. The minCardinalityQ (maxCardinalityQ, cardinalityQ) constructor gives the class whose instances are related via a given property to at least (at most, exactly) n different resources of type C. The number n and class C are specified in the constructor.

OWL [98]: 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. The SH family of description logic is the basis of OWL. OWL has three (increasingly expressive) species: OWL lite, OWL DL, and OWL full. OWL lite: equivalent to SHIF(D_n). Inference in SHIF(D) is of worst-case deterministic exponential time (ExpTime) complexity. OWL lite includes all the vocabularies of RDFS schema. From DL boolean operators, OWL lite allows only the usage of the intersection operator via its axiom *intersectionOf*. The cardinality that can be placed on properties is restricted to "0" or "1". With OWL lite one can declare a property to be *transitive*, *symmetric* or *inverse* to another one. OWL DL: equivalent to SHQIN(D), where the inference is of worst-case non deterministic exponential time. OWL DL allows the different boolean operators and it allows to specify different values for the cardinality on properties. OWL full is union of OWL syntax and RDFS. OWL full is undecidable.

2.4.1.3 Interactions in Ontology-based Systems

Ontologies are mostly developed to support interactions at the content layer. At the communication layer, they may rely on existing communication protocol (e.g., HTTP). Ontologies may be used, at the business process layer, to provide a set of pre-defined business documents to be exchanged among partners. Each document would contain the actions that should take place during message exchange.

Participants in an ontology-based system are bound to the concepts and properties defined within the ontology. This may impact their autonomy and heterogeneity. Ontologies are often not stable definitions. Indeed, a shared conceptualization of a domain has to be reached in a process of social communication. Changes in the domain generally trigger the modification of the ontology. The evolution of ontologies causes operability problems, which hamper

their effective reuse. Solutions are still required to allow changes to ontologies without making current use invalid. The cost of entry in an ontology-based system may be significant since participants must describe their applications according to a given ontology. Finally, the issues of external manageability and security are not addressed in ontology-based systems.

2.4.2 Web Services

The precise definition of Web services is still evolving as witnessed by the various definitions in the literature. One such definition is that a Web service is a "business function made available via the Internet by a service provider, and accessible by clients that could be human users or software applications" [40]. It is also defined as "loosely coupled applications using open, cross-platform standards and which interoperate across organizational and trust boundaries" [208]. The W3C (*World Wide Web Consortium*) defines a Web service as a "software application identified by a URI (*Uniform Resource Identifier*), whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols". The aforementioned definitions can be seen as complementary. Each definition emphasizes some part of the Web service characteristics (discovery, invocation, etc). In this section, we define Web services as business functionalities that are:

- *Programmatically accessible*: Web services are mainly designed to be invoked by other Web services and applications. They are distributed over the Web and accessible via widely deployed protocols such as HTTP and SMTP. Web services must describe their capabilities to other services including their operations, input and output messages, and the way they can be invoked.
- *Loosely coupled*: Communication among Web services is *document-based*. Web services generally communicate with each other by exchanging XML documents. The use of a document-based communication model caters for loosely coupled relationships among Web services.

2.4.2.1 The Web Service Reference Model

Interactions among Web services involve three types of participants: *service providers*, *service registry*, and *service consumers* (Figure 2.6). *Service providers* are the parties that offer services. They define descriptions of their services and publish them in the *service registry*, a searchable repository of service descriptions. Each description contains details about the correspond-

ing service such as its data types, operations, and network location. *Service consumers* use a *find* operation to locate services of interest. The registry returns the description of each relevant service. The consumer uses this description (e.g., network location) to invoke the corresponding Web service. For example, `HealthDepartment` may advertise a Web service that includes a `scheduleImmunization` operation. `AreaAgencyAging`'s provider would then access the registry, discover `HealthDepartment`'s service, and invoke its operations.

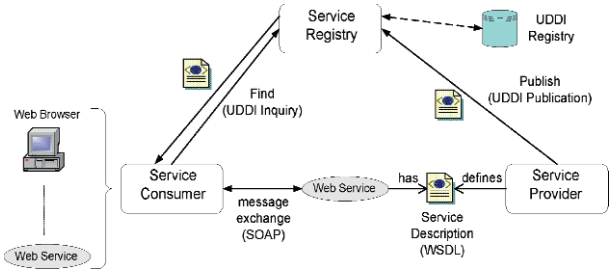


Fig. 2.6 The Web Service Reference Model

Three major standardization initiatives have been submitted to the W3C consortium to support interactions among Web services (Figure 2.6):

- WSDL (*Web Services Description Language*) [217]: WSDL is an XML-based language for describing operational features of Web services. WSDL descriptions are composed of *interface* and *implementation* definitions. The *interface* is an abstract and reusable service definition that can be referenced by multiple *implementations*. The *implementation* describes how the interface is implemented by a given service provider.
- *UDDI* (*Universal Description, Discovery and Integration*) [216]: UDDI defines a programmatic interface for publishing (*publication API*) and discovering (*inquiry API*) Web services [216]. The core component of UDDI is the *business registry*, an XML repository where businesses advertise services so that other businesses can find them. Conceptually, the information provided in a UDDI business registration consists of *white pages* (contact information), *yellow pages* (industrial categorization), and *green pages* (technical information about services).
- SOAP (*Simple Object Access Protocol*) [215]: SOAP is a lightweight messaging framework for exchanging XML formatted data among Web services. SOAP can be used with a variety of transport protocols such as HTTP, SMTP, and FTP. A SOAP message has a very simple structure: an XML element (called *envelope*) with two child elements. The first ele-

ment, the *header* includes features such as security and transactions. The second element, the *body* includes the actual exchanged data.

2.4.2.2 Web Service Stack

The Web service architecture is organized into a layered stack. The four key layers, including communication, messages, description, and processes, are shown along the perpendicular direction in Figure 2.7. The layered stack is an extension of the W3C service stack [218]. Service operations including publication, lookup, and binding could get support from the service stack. The upper layers rely on the lower layers to realize their functionalities:

- *Communications:* The underpinning of the Web services stack is the network, where the underlying communications take place. A set of network protocols help 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 Internet protocols could also be supported, such as SMTP.
- *Messaging:* The massaging layer provides a document-based messaging model for the interaction with Web services. The messaging model works with a wide variety of transport 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.
- *Representation:* The 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 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. 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.
- *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.

The variety of 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

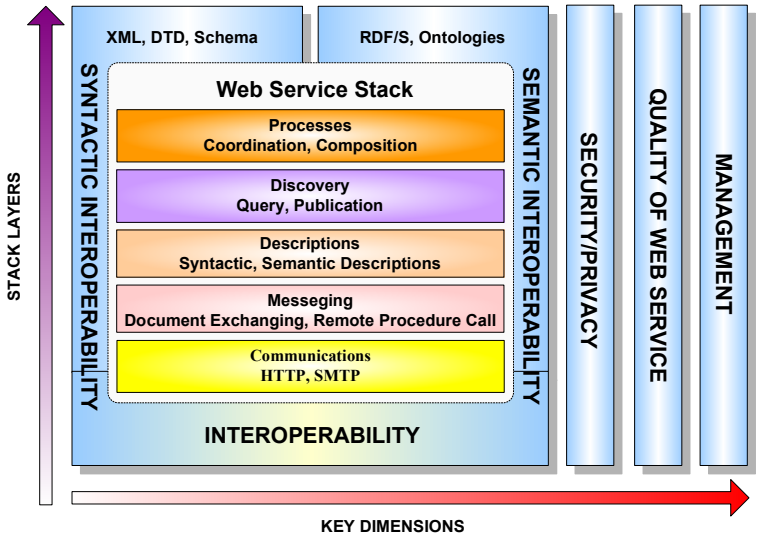


Fig. 2.7 Web Service Stack and Key Dimensions

requirements for deploying Web service. We identify a set of dimensions to evaluate Web service technologies. These dimensions are in line with the key requirements for deploying Web services. The dimensions are defined in line with the vertical layers across the Web service stack (see Figure 2.7), which include *interoperability*, *security & privacy*, *Quality of Web Services (QoWS)*, and *management*.

Interoperability: Web services are designed to bring together applications from geographically distributed and heterogeneous environments and provide interoperability between them. Therefore, interoperability is a core requirement that Web services aim at achieving. Interoperation occurs at two levels: *syntactic* and *semantic levels*. *Syntactic interoperability* is concerned with the compatibility of the input/output of the operations, the binding types, the data format, and so on. XML would help achieve this interoperability by encoding this syntactic information into XML messages. *Semantic interoperability* is the most challenging issue for achieving the truly seamless interoperation. It would rely on some machine interpretable languages to address the semantic issues. The envisioned Semantic Web could be a good resource for providing Web service description languages with rich semantics.

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 [80]. 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 encrypting the message. Lastly, *integrity* is to protect the information from being tampered by putting digital signatures on the messages. Privacy is another major concern of Web service deployment [182, 132]. During the service interactions, personal data or business secrets (e.g., billing information, shipping address, or product preference) might be unintentionally or even inevitably released [1]. Conventional privacy protection mainly relies on the law enforcement and restriction of social values. Lack of sufficient technological support, privacy is still left in a vulnerable state. Emerging technologies for preserving privacy in Web services include digital privacy credentials, data filters, and mobile privacy preserving agents [183].

QoWS: The proliferation of Web services will introduce the competition between 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 [231, 234, 185, 232, 172]. *QoWS* encompasses different quality parameters that characterize the behavior of a Web service in delivering its functionalities. Examples of parameters include availability, latency, and fees.

Management: Web service management refers to the management of Web services through a set of management capabilities to *control* and *monitor* service qualities and service usage. Web service management mechanisms are highly coupled with QoWS of a Web service. *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 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* aims to rate 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 QoWS parameter values or assessing a Web service claim in terms of promised QoWS.

2.4.2.3 RESTful Services

REST (Representational State Transfer) or RESTful Web services may be considered a lightweight version of SOAP Web services in that they do not

require XML messages or WSDLs [73]. RESTful Web services use HTTP to transfer information between other RESTful Web services. Critics of SOAP Web services state that they are too complex. Another concern is the brittleness of the WSDL itself-minor changes to the WSDL residing on the server or an upgrade to the Web service stack can cause each client using the Web service to have to make changes to its code. Further, there are concerns over the performance of SOAP Web services. Once the requestor has received the SOAP message response, an XML parser must be used to parse the response, and retrieve the results. Searching the UDDI repository to look up a service is also costly. REST has been chosen by many developers for it requires less overhead. REST is used by Google, Amazon, and Flickr, among other mainstream Web sites. RESTful services tend to use custom developed tokens that are passed in the URL QueryStrings. REST uses HTTP variables such as GET, POST, UPDATE, DELETE, and OPTIONS. The REST architecture is based on four principles [73]:

- Resource identification through URI - Resources are identified by URIs, which provides a global addressing space for resource and service discovery.
- Uniform interface - GET retrieves the current state of a resource in some representation. POST transfers a new state of a resource.
- Self-descriptive messages - Resources are decoupled from their representation so that their content can be accessed in variety of formats (e.g., HTML, XML, plain text, PDF, JPEG).
- Stateful interactions through hyperlinks - Several techniques exist to change (e.g., URI rewriting, cookies, and hidden form fields). State can be embedded in response messages to point to valid future states of the interaction.

2.4.2.4 Semantic Web Services

Semantic Web services integrate ontologies into Web services to pave the way for automated selection, composition, and monitoring of Web services [134, 147, 136]. The ontology-based Web service representation enriches the Web service description with machine interpretable semantics. It uses ontology language constructs to describe the functionalities of services. Web service 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. Web service processes can rely on the behavior information to decide the order of different Web services that participate in the process. Major semantic Web frameworks include *OWL-S* [56], *WSMF* [72], and *WSMO* [89].

OWL-S [56] – OWL-S is an OWL-based ontology for describing Web services. It is an extension of DAML-S, a DAML+OIL upper ontology for Web service. The `Service` class sits on top of the hierarchical structure. It consists of three components: a `ServiceProfile`, a `ServiceModel`, and a `ServiceGrounding`.

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 the 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 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* [72, 89] – Web Service Modeling Ontology (WSMO) is an ontology for describing several aspects of semantic Web services. It takes Web Service Modeling Framework (WSMF) [72] as a starting point. The 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 the 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.

WSMO refines the 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 a 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. *Web Service Modeling Language* (WSML) [88] 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.

2.4.2.5 Web Service Composition

Web service composition is a process that combines outsourced Web services to offer *value-added* services [67, 7, 140]. 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. Major standardization efforts for service composition include BPEL and ebXML's *business process specification*.

BPEL [15]: BPEL 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.

ebXML's business process specification [167]: ebXML's business process specification is another process-based composition approach. The business process specification schema 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 choreography. 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).

2.4.2.6 Automatic Web Service Composition

One of the major challenges for a wide adoption of the service-oriented approach is the automated composition of Web service, i.e. the development of concepts, techniques, technologies, and tools that support an easy-to-use, low-cost, dynamic, and time-efficient composition of Web services [177]. However, the development of automated composition techniques is still an open problem. In this section, we overview major research techniques for supporting automatic service composition.

Interactive Service 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 [197]. 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.

AI Planning Techniques for Service Composition – Another trend for dealing with the automatic service composition is the use of Artificial Intelligence (AI) planning techniques. The composition engine treats service composition as a planning problem. *SHOP2* adopts the concept of HTN (Hierarchical Task Network) as a planning methodology [226]. It decomposes tasks into smaller and smaller subtasks, until primitive tasks are found that can be performed directly. SHOP2 uses a knowledge base which contains *operators* and *methods*. Each operator is a description of what needs to be done to accomplish a primitive task. Each method describes the process of decomposing a compound task into partially ordered subtasks. *Estimated-regression* is another planning technique for service composition [133]. The situation-space search is guided by a heuristic estimator obtained by backward chaining in a relaxed problem space. The resulting space is so much smaller than situation space that a planner can build complete representation of it, called a regression graph. The regression graph reveals, for each conjunct of a goal, the minimal sequence of actions that could achieve it.

Techniques Dealing with Service Behavior – A number of techniques dealing with service behavior have been proposed (e.g., [103, 33, 230]). These techniques deal with services whose life-cycle consists of several related calls. Models and formalisms for describing service behaviors have been discussed, including *finite state machine*, *Mealy machine*, and *pi-calculus* [103]. This book focuses on "isolated" operations, that is services whose life-cycle consists of independent operation invocations. The techniques described in [103] can be adopted to define dynamic semantic rules dealing with the composability of service behaviors. A pi-calculus model for extending CORBA IDL with protocol descriptions is proposed in [33]. The model is based on the concept of role which allows the specification of the observable behavior of CORBA objects. Techniques for testing the compatibility of behaviors are also presented. *Protocol Specifications* is another approach for describing object service protocols using finite state machines [230]. This approach describes both the services offered and required by objects. It also defines techniques that allow participants to be checked for protocol compatibility. However, the aforementioned techniques deal with objects and components *not* Web services.

Matching Techniques for Web Services – Several techniques have been proposed to deal with Web service matching. LARKS defines five techniques for service matchmaking: context matching, profile comparison, similarity matching, signature matching, and constraint matching [202]. Matching services to requests is performed by using any combination of the above techniques. The ATLAS matchmaker defines two methods for comparing service capabilities described in DAML-S [179, 176]. The first method

compares functional attributes to check whether advertisements support the required type of service or deliver sufficient quality of service. The second compares the functional capabilities of Web services in terms of inputs and outputs. [42] presents a service matching technique for pervasive computing environments. Service descriptions are provided in DAML-S. They also include platform specific information such as processor type, speed, and memory availability. The composition manager uses a semantic service discovery mechanism to select participant services. This mechanism is based on DReggie, a Jini-based semantic discovery framework [41]. [124] adopts techniques from knowledge representation to match DAML-S service capabilities. In particular, it defines a description logic (DL) reasoner; advertisements and requests are represented in DL notations. Another DAML-S based matchmaker implementation is KarmaSIM [161] where DAML-S descriptions are described in terms of a first-order logic language (predicates) and then converted to Petri-nets where the composition can be simulated, evaluated and performed. Other service matching techniques are also presented in [12, 135, 95]. However, they techniques mostly focus on comparing syntactic attributes of Web services.

2.4.2.7 Interactions in Web Services

Web services allow interactions at the communication layer by using SOAP or REST as messaging protocols. RESTful services do not require XML messages or WSDLs, but as in the case of SOAP uses HTTP to transfer information between other RESTful Web services. The adoption of a messaging over well-established protocols such as HTTP enables communication among heterogeneous systems. At the content layer, Web services use WSDL language. WSDL recommends the use of XML Schema as a canonical type system (to associate data types to message parameters). However, the current version of WSDL does not model semantic features of Web services. For example, no constructs are defined to describe document types (e.g., whether an operation is a request for quotation or a purchase order). Semantic Web service frameworks such as OWL-S, WSMF, and WSMO deal with semantic interoperability.

Web services are still at a maturing stage. Hence, they still lack the support for interactions at the business process layer. To date, enabling interactions among Web services has largely been an *ad hoc* process involving repetitive low level programming. Standardization efforts such as BPEL4WS (*Business Process Execution Language for Web Services*) [15] and ebXML's business process specification are underway for enabling the definition of business processes through Web service composition.

The use of a document-based messaging model in Web services caters for loosely coupled relationships. Additionally, Web services are not statically

bound to each other. New partners with relevant features can be discovered and invoked. However to date, dynamic discovery of Web services takes place mostly at development time. Heterogeneous applications (e.g., Java, CORBA objects) may be wrapped and exposed as Web services. For example, the *Axis's Java2WSDL* utility in *IBM's Web Services Toolkit* enables the generation of WSDL descriptions from Java class files. *IONA*'s *Orbix E2A Web Services Integration Platform* may be used to create Web services from existing EJBs or CORBA objects. In terms of autonomy, Web services are accessible through published interfaces. Partners interact with Web services without having to be aware of what is happening behind the scene. They are not required to know how the operations provided by the service are internally implemented. Some operations can even be transparently outsourced from third parties.

WSDL does not currently include operations for monitoring Web services such as checking the availability of an operation or the status of a submitted request. Additionally, neither UDDI nor WSDL currently define quality of service parameters such as cost and time. In terms of adaptability, changes may occur in operation signatures (e.g., name), messages (e.g., number of parameters, data types), service access (e.g., port address), and service and operation availability. The process of dealing with changes is currently *ad hoc* and manually performed. More efforts need to be done to cater for automatic detection and handling of changes.

Security in Web services needs to be addressed at different levels including communication, description, and firewall. At the communication level, enabling security in XML and HTTP is an important factor towards securing Web services. Current standardization efforts include securing XML-based messages through the creation of an XML digital signature standard and S-HTTP, a protocol for transmitting data securely over the Web. Other work is also being done to extend SOAP to include a security specification at the receiving endpoints (e.g., specify which users are authorized to receive and process messages). At the service description level, WSDL does not include security-oriented information such as role-based access control and other authorization information. Finally, since SOAP messages carried over HTTP traverse firewalls, network administrators would need to configure their firewalls so that malicious requests (e.g., code embedded in SOAP messages) are not tunneled through SOAP messages. For example, *application firewalls*, sitting behind networks firewalls, have been developed (e.g., *iSecureWeb*). Application firewalls check all HTTP traffic to validate and authorize messages based on security policies.

Security with REST is limited compared to SOAP Web services. Access Control Lists (ACLs) may be used when the POST method is used. Filtering of the QueryString on the server-side can be used, which is a similar approach used for HTTP Web applications. An audit trail should be kept when RESTful services are used to trace security violations. When the POST command is used, ACLs may be used which protect against unauthorized use of a ser-

vice. Also, since there are no standards for RESTful services security, each API may use a different security model.

The emergence of tools to describe, advertise, and invoke Web services facilitates the development of Web service-based solutions. However, the use of a tagged language such as XML increases the volume of information to be exchanged among Web services. This might overload the network in presence of a large number of services, hence penalizing the scalability of the Web service approach. Additionally, SOAP defines only simple data types. Using complex data types may require the XML parser to get the corresponding XML Schema definitions from remote locations. This might add an overhead for processing SOAP messages. The registry presents another scalability issue. A centralized registry might result in a single point of failure and bottleneck for accessing and publishing Web services. A distributed registry would cater for a more reliable and scalable solution. However, this incurs an additional overhead of managing distributed repositories. An intermediary solution is adopted in UDDI where the registry is physically replicated over multiple nodes. This solution solves the problem of centralized registry. However, it still requires the nodes to exchange data with each other to maintain registry consistency.

2.4.3 Software Agents

The concept of *software agent* (or simply *agent*) emerged as an important paradigm for organizing many classes of distributed applications such as e-commerce, digital libraries, information retrieval, and data mining. It is defined as a piece of software capable of acting exactingly to accomplish tasks on behalf of its consumers. Consumers may be end-users, other agents, or applications using the agent. A more comprehensive definition would be to view an agent as an umbrella term which covers a range of properties. Agents typically possess several (or all) of the properties summarized in Table 2.3. Ideally, an agent should exhibit all the properties equally well, but this is an aspiration rather than the reality [165]. Other properties such as *emotion* and *character* (knowledge, belief, intention, etc.) have also been defined [224]. They are mainly used in artificial intelligence applications and so, are out of the scope of this book.

2.4.3.1 Agent Communication Language

A multi-agent system is a collaborating set of agents in which the membership may change over time. Individual agents in the community view each other as peers and are responsible for providing specific services according

Property	*Synonyms*	*Definition*
Continuous	Not Available	An agent is a continuously running process
Autonomous	Not Available	Agents operate without the direct intervention of humans or others and exercise control over their own actions
Co-operative	Communicative Socially able	Agents interact with each other via some kind of agent-communication language
Reactive	Sensing and acting	Agents perceive their environment and respond in a timely fashion to changes that occur in it
Adaptive	Learning	Agents change their behavior based on their previous experience
Pro-active	Goal-oriented Purposeful	Agents do not simply act in response to their environment, they are able to take initiatives to achieve a certain goal
Mobile	Not Available	Agents are able to transport themselves from one site (in a network) to another

Table 2.3 Agent Properties

to their role. The processing of a request in such an environment is distributed among the different agents. The success of a multi-agent system relies on how the agents interact and interoperate. This involves the ability of agents to communicate with each other via an *Agent communication language* (ACL). An ACL consists of three major parts: a *vocabulary*, an inner language called *KIF* (*Knowledge Interchange Format*) and an outer language called *KQML* (*Knowledge Query and Manipulation Language*). An ACL message is a KQML expression in which the arguments are terms in KIF formed from words in the ACL vocabulary.

Agents should share a common understanding of what is being communicated. Ontologies facilitate agent communication by providing a common vocabulary. The vocabulary is an open-ended dictionary or list of words appropriate to common application areas. Each word has an English description for use by humans and formal annotations (written in KIF) for use by programs. The dictionary allows for the addition of new words within both existing and new application areas. KIF is a prefix version of first order predicate calculus, with various extensions to enhance its expressiveness. It provides for the encoding of simple data, constraints, negations, and quantified expressions. A KQML message is a piece of dialogue between a sender and a receiver. Each message is a list of participants enclosed in matching parentheses. The first word in the list indicates the type of dialogue (ask, tell, recommend, bid, etc.). The subsequent entries are the arguments, that is KIF expressions appropriate to that dialogue.

Below is an example of KQML message where a sender (Mary) is asking the receiver (Bookstore) to send information regarding all books related to "Agents" which costs below $30:

```
(ask
  :sender Mary
  :receiver Bookstore
```

:content "select * from book where topic="Agents" and price<30"
:language "Oracle SQL"
:ontology "online bookstore")

2.4.3.2 Interactions in Multi-Agent Systems

Multi-agent systems cater for interactions at the communication layer by having a standard communication language, namely ACL. This assumes that the underlying communication and transport protocols are already established. At the content layer, ontologies are used to facilitate communication between agents and add semantics to their messages. Multi-agent systems lack support for specifying complex business processes. A process is generally spread through the implementation code of every agent involved in the process. This *ad hoc* definition of business processes does not scale because of the size of partners that may be involved in business processes. One solution would be to have a separate *control* agent that handles the business process. However, the control agent's design is time consuming. Additionally, the business logic is embedded in the control agent, making it hard to reuse for different business processes [180].

Multi-agent systems assume that relationships between different partners have to be pre-established. Additionally, data to be interchanged among agents generally needs to be manually or semi-automatically mapped to a common ontology. This makes agents suitable for long-term and tightly coupled relationships. Agents may be added to existing applications to add new functions or customize the execution of existing functions with little impact on the autonomy of those applications. However, they must adhere to the concepts and properties of a common ontology which may limit their autonomy.

Agents may wrap proprietary and heterogeneous applications. The adoption of an ontology enables the abstraction from heterogeneous information representations. Event subscriptions and monitoring agents may be used to address the issue of external manageability. However, adaptability is not adequately addressed in multi-agent systems. Agents are typically preprogrammed with certain functionalities and can hardly be configured to switch roles or change interaction patterns. Agents interact to carry out sophisticated tasks on behalf of users. In the course of this interaction, they may exchange sensitive information about these users. More research is needed to ensure that multi-agent systems support the security features required in Semantic Web interactions. The cost of entry in an agent-based system is significant. Agents must be able to exchange information via Agent Communication Language. They also need to formulate queries or requests based on the common ontology. They finally must have a way of matching queries/requests with their known ontologies (i.e. reason over its repository of ontologies).

2.4.4 XML-based Interaction Standards

A large number of contemporary interaction standards are based on XML. The vision behind this approach is to allow the use of services on the Web without dedicated transformation and mediation facilities or custom integration of partners' systems. Business partners would form a *trading community* based on their capabilities to produce and consume those XML documents. The business process of the trading community is specified by the shared document definitions. The partners are interconnected in terms of largely agreed upon documents. The business logic implementation at a partner side is invisible to other trading partners. In general, a complete XML-based integration requires standardized domain-specific ontologies (such as an agreed upon DTD or XML Schema), mappings between different ontology descriptions, and means for processing XML documents and invoking appropriate services (e.g., workflows and legacy systems) to handle requests.

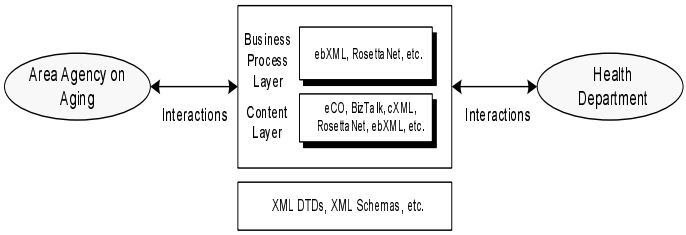


Fig. 2.8 XML-based Interaction Standards

Most of the existing XML-based interaction standards were developed for business-to-business (B2B) interactions. However, they may be extended to deal with other types of applications such as e-government and digital library. In what follows, we describe a representative set of those standards (Figure 2.8). The proposed standards generally deal with enabling interactions at the content (e.g., *eCO*, *cXML*) and business process (e.g., *RosettaNet*) layers. However they sometimes overlap or even compete with each other [31]. The issue of interoperability has thus shifted from the level of applications to the level of standards. A trading partner has to deal with several standards at the same time. In case one trading partner exchanges messages across industries, the variety of standards is likely to increase even more [31].

2.4.4.1 eCO

eCO [47] aims at providing means to businesses to discover and access services regardless of the standards and protocols each potential partner adopts. At the content level, eCO introduces xCBL (*XML Common Business Library*) to define business documents. xCBL consists of a set of XML *core documents* that are used to represent common interactions in business transactions. It does not target vertical industry domains. It attempts to provide a generic framework for describing the content of core business documents. 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. For example, `HealthDepartment` can use xCBL to create an XML document of its service catalog by customizing the generic *xCBL catalog DTD* with specific information about the search method (e.g., immunization services by type). 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.

At the business process level, eCO focuses more on providing a common building blocks for interactions among businesses. Businesses can advertise their online services as *Business Interface Definitions (BIDs)*. BIDs are XML descriptions that specify business services in terms of documents they accept and produce. It does not mandate a global business process definition. eCO uses xCBL as a basis to define both the interfaces of processes and content of data elements. Since every partner is forced to use the same tag to define the same type of information, the structural heterogeneity is not a problem. As is any standard, there is however a non-trivial issue: the meaning and types of services and their interfaces can vary among businesses although a group of partners in a specific marketplace may select to adopt common conventions. In generic frameworks such as eCO, it is difficult to address semantic heterogeneity because of the wide range of E-commerce applications. One solution is to use several schemas (or *ontologies*) including horizontal (i.e., across domains such as computer manufacturing and healthcare) and vertical (i.e., within a specific domain) domains. This solution requires the support of data normalization, mapping and conversion between schemas or ontologies.

Although, eCO requires that services be described using XML schemas, it does address, albeit in a limited way, the issue of autonomy. eCO separates the description of services and their implementations. Note that a marketplace may adopt some common conventions for describing services. This may as a result, negatively impact on the partners' autonomy. For example, a change in the name of a tag, requires all partners to make that specific change at the same time. In eCO, the use of security mechanisms is optional.

However, business partners do not need to directly gain access to each other systems. The establishment of a new relationship with an existing partner does not require any additional work. The creation of a new service requires the provision of its description (types, interfaces, etc). It also requires the integration of the interfaces of the service with internal applications. The integration cost in an XML-based approach tends to be less significant than other approaches because of widely available XML processing tools. The eCO framework rates high in adaptability. The impact of local changes is limited as partner systems are loosely coupled. In addition, eCO offers extensibility to accommodate changes.

2.4.4.2 cXML

cXML (*Commerce XML*) [54] consists of an XML-based schema language and a protocol for online purchasing transactions. It targets business transactions that involve non-production Maintenance, Repair, and Operating (MRO) goods and services. In a nutshell, cXML can be considered as a simplified XML and Internet-based version of EDI. cXML assumes the existence of intermediaries (*E-commerce hubs*) that act as trusted third parties between procurement systems and supplier systems. The functions provided by an E-commerce hub (e.g., Ariba Network, Extricity Software) are similar to those provided by the BizTalk repository. However, cXML does not prescribe a specific intermediary architecture.

cXML supports two communication models: *request-response* and *one-way*. The request-response provides for synchronous communication through HTTP. The one-way provides for asynchronous communication through HTTP or other protocols. Currently, the one-way model supports HTTP and URL Form Encoding.

At the content level, 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). Catalogs can be static or dynamic. In the cXML terminology a dynamic catalog is called a *punchout*.

At the business process level, the cXML approach is similar to OBI's (see Section 2.3.1.2). cXML defines a generic procurement protocol. This protocol consists of a number of commonly agreed upon online procurement activities (e.g., product selection, order request, order approval, order transmission, order routing). E-commerce hubs provide means for catalog and purchase order

management (e.g, catalog publishing and subscription, automated purchase order routing and tracking).

cXML offers similar properties to those in OBI, namely, heterogeneity, autonomy, and adaptability. cXML appears to rate higher than OBI with regard to scalability because the integration cost in an XML-based approach tends to be less significant than other approaches. cXML addresses security by including authentication information message headers. One advantage of cXML approach is economy of scale and ease of managing business relationships. Both suppliers and buyers only need to manage relationships with the trusted intermediary rather than with all their business partners.

2.4.4.3 RosettaNet

RosettaNet [48] 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). RosettaNet focuses on three key areas of standardization to automate Web-based interactions. First, the vocabulary needs to be aligned. 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). Second, the way in which business messages are wrapped and transported must be specified. The *RosettaNet Implementation Framework* specifies content of messages, transport protocols (HTTP, CGI, email, SSL) for communication and common security mechanism (digital certificates, digital signatures). Third, the business process governing the interchange of the business messages themselves must be harmonized and specified. *RosettaNet*'s PIPs (*Partner Interface Processes*) are pre-defined XML-based *conversations*. A *conversation* consists of a set of business documents (e.g., purchase order, purchase order acknowledgment) and message exchange logic (e.g., the sequencing of the actions that take place during a product quote request). A PIP is defined using a combination of textual and graphical (UML-based state machine) representations.

At the communication layer, common Internet transport protocols are supported. At the content layer, RosettaNet uses an XML-based schema as document content model. The use of a vertical ontology (i.e, common vocabulary with information technology supply chain domain) contributes to solving the problem of semantic heterogeneity. At the business process layer, RosettaNet focuses on providing a common basis for public interactions via PIPs. The integration of PIPs with internal business processes is performed by partners. RosettaNet does not provide means to define arbitrary global business processes. RosettaNet offers similar properties as OBI with regards to security.

It offers similar properties as eCO with regard to autonomy, adaptability, scalability, coupling, and external manageability.

2.5 Deployment Platforms for Web-based Interactions

Major software vendors (*IBM*, *Microsoft*, *Sun Microsystems*, *HP*, *Oracle*, *BEA systems*, etc) are currently working on implementing Web service platforms. The purpose of this section is not to compare commercial products but to overview their main features. Because there are a large number of products, this section does not attempt to cover all of them. Instead, we focus on the major players in this arena. Our coverage is based on user manuals and white papers since there are few or no published technical papers detailing commercial products. Additionally, existing products are at various development stages and operate at different levels of disclosure.

Microsoft .NET [155] – *.NET* embraces the concept of Web services to enable B2B interaction. It consists of three key elements: *.NET Framework and tools*, *.NET Enterprise Servers*, and *.NET Service Building Blocks*. *.NET Framework and tools* provides the standard-based tools for SOAP, WSDL, and UDDI. *.NET Enterprise Servers* provides the core components for building Web services. These include database like *SQL Server 2000*, messaging software like *Exchange 2000 Server*, business process technology like *BizTalk Server 2000*, and *Internet Security and Acceleration Server*. *.NET Service Building Blocks* contains pre-defined Web services created using the *.NET* infrastructure (e.g., *Passport* and *HailStorm*).

SOAP is used as the main transport protocol in the communication layer. Interoperability at the communication layer is also supported by *Microsoft Message Queue* (MSMQ) supplemented with gateways for sending and receiving documents in various formats from trading partners. *Microsoft Host Integration Server* is used to support connection to proprietary systems like IBM mainframes. Heterogeneity at the content layer is addressed by adhering to open standards (XML and WSDL) and the wrapping of applications as *.NET Managed Components*. Building business processes (called *Orchestration*) is done through *BizTalk Server*. Developers use the *Biztalk Orchestration Designer* to create *Biztalk processes*. These are compiled into *XLANG schedules* which are executed by the *Biztalk Scheduler Engine*.

WebSphere [104] – *WebSphere* is a family of IBM products for B2B interactions. The *application server* is the cornerstone of *WebSphere*. It aims at providing database and backend integration as well as security and performance capability (e.g. workload management). The *WebSphere application server Advanced Edition* adds support for J2EE specification. It also extends J2EE with direct access to advanced CORBA services for

greater flexibility and improved interoperability. The advanced edition integrates support for key Web service standards such as SOAP, UDDI, and WSDL. Additionally, it provides distributed transaction support for major database systems including *IBM*'s *DB2*, *Oracle*, *Sybase*, and *Informix*. Other products make up the *WebSphere* platform. These include *WebSphere Business Components*, *WebSphere Commerce*, and *WebSphere MQ Family*. The *WebSphere Business Components* provides pre–built, tested, and "plug and play" components for building new applications or extending existing ones. *WebSphere Commerce* provides mechanisms for building B2B sites including catalog creation and payment processing. *WebSphere MQ Family*, formerly known as as *MQSeries*, is a family of message-oriented middleware products that enable communication between applications running on different hardware platforms.

Sun ONE (*Sun Open Net Environment*) [157] – *Sun ONE* is a platform for Web services developed by *Sun*. Two main product lines make up the *Sun ONE* platform: *Forte tools* and *iPlanet*. *Forte tools* offer *Integrated Development Environment* (IDE) for the Java, C, C++, and Fortran languages. It enables developers to access the plug-ins they need and hence speed the development of Web services. *iPlanet* is the core of *Sun ONE* platform. It includes a stack of products that allow the creation, deployment, and execution of Web services. Examples of such products are the *iPlanet Portal Server*, *iPlanet Application Server*, and *iPlanet Integration Server*. The *iPlanet Portal Server* is the representation layer of *iPlanet*. It delivers services to end-users by aggregating content and providing security, personalization, and knowledge management. The *iPlanet Application Server* enables access to legacy applications and databases. It also provides a J2EE execution environment for Web services. The *iPlanet Integration Server* is a workflow-based engine that enables businesses to define workflows across legacy applications and create services.

Sun ONE uses workflows to ensure interoperability at the business process layer. However, it is not clear how services are composed using the *iPlanet Application Server*. *Sun ONE* supports the emerging Web service standards such as SOAP, WSDL, and UDDI. The *iPlanet Portal Server* enables the integration of any HTML or XML encoded content and heterogeneous applications that run on major operating systems such as *Microsoft Windows*, and *Unix*. Complementary packages provide additional functionality including secure communications. *iPlanet* addresses scalability by offering built-in services such as load balancing.

Vitria BusinessWare [214] – *Vitria BusinessWare* emphasizes on business process management and automation. It adopts *UML* and *WfMC reference model* for modeling business processes. The exchange of information between trading partners is done using XML. However, *BusinessWare* assumes that those partners will agree upon a common standard XML DTD to describe the

documents to be exchanged. *BusinessWare* also requires businesses to agree on the semantics of business processes' activities. *BusinessWare*'s processes are divided into two types: *public* and *private*. A placement of purchase order described in RosettaNet PIPs is an example of public process. The way that different companies deal with an incoming order from a customer is an example of private process. The separation between private and public process allows trading partners to change their private process without affecting the cross-organization public business process.

BusinessWare is composed of four modules: *Business process management*, *Business-to-Business communications*, *Enterprise application integration*, and *Real-time analysis*. The *Business process management* controls and coordinates the flow of information between internal and external process systems. Both private and public processes can be defined using the graphical modeling tool. The *Business-to-Business communications* is responsible for interactions with trading partners using multiple protocols (HTTP-S, FTP, IIOP, SOAP, EDI, fax and email) and data formats (XML, IDL, EDI, RosettaNet). The *Enterprise application integration* provides connectors for major databases, messaging systems, and packaged applications. The *Real-time analysis* enables the gathering and analysis of process information. It allows businesses to identify processing bottlenecks and react to fast-changing business conditions.

Oracle Integration Server [169] – *Oracle Integration Server* is one of the products of *Oracle Application Server* which is based on J2EE and emerging Web service standards. It supports transport protocols such as SOAP, HTTP-S, SMTP, FTP/S, IIOP, and various messaging systems (JMS, IBM MQSeries, TIBCO/Rendezvous). The *Integrator* has two main components. The first component provides an EJB container for executing the designed business process. The second component consists of the design and management tools which include *Integration Modeler*, *Business Process Monitor*, and *Business Intelligence*. The *Integration Modeler* offers a set of Web-based tools to model business process, map data sources from one form to another, and set up relationships with trading partners. The *Business Process Monitor* provides means for users to monitor, analyze, and drill down on the state of the business process (such as start, stop, resume). The *Business Process Intelligence* uses *Oracle* data warehousing facilities to analyze and gather information about the overall flow of business processes (i.e., the frequency of messages being sent/received).

HP NetAction [100] – The *HP NetAction* software suite includes the *HP NetAction Internet Operating Environment* (IOE), a platform for building B2B applications. The IOE includes the *HP Process Manager* and *HP Web Services Platform*. *HP Process Manager* (formerly called *ChangeEngine*) allows the graphical definition of business processes and provides an environment that automates the execution of those processes. It has a

component-based architecture based on J2EE. *HP Process Manager* also provides an audit logger that can be used to read information in XML format from a JMS (*Java Message Service*) queue. It allows the definition of audit nodes within a business process to indicate the points in the process at which audit information should be collected. *HP Web Services Platform* is a standards-based architecture for developing Web services. Key components of the *HP Web Services Platform* include *HP-SOAP 2.0*, *HP Service Composer* (a graphical tool for creating and mapping WSDL interfaces), *HP Registry Composer* (a graphical tool for registering and discovering Web services in UDDI registries). HP announced in July 2002 it was discontinuing its development and support of *NetAction*.

BEA WebLogic Integrator [14] – *BEA WebLogic Integrator* is the cornerstone of *BEA WebLogic E-Business* Platform. It is built on top of a J2EE compliant application server and J2EE connector architecture. It supports current Web service standards such as SOAP, UDDI, and WSDL. The *Integrator* is composed of four major modules: *Application server*, *Application integration*, *Business process management*, and *B2B integration*. The *Application server* provides the infrastructure and functionalities for developing and deploying multi-tiers distributed applications as EJB components. The *Application integration* leverages the J2EE connector architecture to simplify integration with existing enterprise applications such as SAP R/3 and PeopleSoft. The *Business process management* provides a design tool and execution engine for business processes. The *B2B integration* manages interactions with external business processes. A separate module called *B2B integration/collaboration* is used to manage different B2B protocols (such as RosettaNet PIPs, BEA's eXtensible Open Collaboration Protocol) and *Quality of Service* (QoS) of the trading partners.

WebMethods [220] – *WebMethods* is composed of three modules: *WebMethods Enterprise Server*, *WebMethods Enterprise Adaptor* and *WebMethods Enterprise Rule Agent*. The *Rule Agent* is used to set up specific business rules that are required for integrating business processes across different enterprises. The *adaptors* connect information sources to *WebMethods Enterprise Server* and provide bi-directional mapping of information between the native format and the server's. Several adaptors are provided to allow the mapping of XML messages to industry-adopted message types (RossettaNet, cXML, OBI, EDI). The hub of the system is *WebMethods Enterprise Server* which acts as the central control and storage point. It uses XML for exchanging messages between trading partners. The server supports multiple transport protocols such as SOAP, HTTP, HTTP-S, RMI-IIOP, SMTP and FTP. It also defines a process-oriented language called *Flow* to visually compose services.

TIBCO ActiveEnterprise [206] – *ActiveEnterprise* uses a set of products to enable B2B interactions. *TIBCO InConcert* is a tool for defining and managing dynamic workflows. *TIBCO IntegrationManager* defines and manages automated business processes that span multiple applications and transactions. *TIBCO MessageBroker* performs rule-based transformation and mapping of messages between different messaging softwares. *TIBCO Hawk* is a sophisticated tool for administrating and monitoring of system behaviors within *ActiveEnterprise*. *TIBCO Rendezvous* is an advanced messaging system that supports publish/subscribe, request/reply, synchronous/asynchronous, certified and transactional messaging paradigms. *ActiveEnterprise* supports other messaging protocols such as JMS, HTTP/S, COM, CORBA and MQSeries. At the content layer, *ActiveEnterprise* supports various vertical and horizontal industry standards such as cXML, RosettaNet, EDI, and HealthCare standards.

In Table 2.4, commercial Web service platforms are summarized using the following set of parameters: major modules, communication standards, content and business process standards, and key technologies. For example, BEA Weblogic Integrator includes an application server, application integration, business process management, and B2B Integration. The communication standards supported in BEA Weblogic Integrator are SOAP, JMS and IIOP. BEA Weblogic Integrator supports WSDL, XML, RosettaNet-PIP, and BEA-XOCP as content and business process standards. The key technologies that are supported include components (J2EE), XML, workflows, and Web services. Note that all deployment platforms support HTTP as a communication protocol. Additionally, the list of supported standards (communication, content, and B2B protocol) is non-exhaustive as new standards are constantly being added.

2.6 Research Prototypes for Web Service Composition

In this section, we overview research prototypes that support Web service composition. Note that the list of prototypes described in this section is not exhaustive. We focus on a representative set of such prototypes.

CMI (*Collaboration Management Infrastructure*) [190, 82] – CMI provides an architecture for inter-enterprise workflows. The main components of CMI engine includes the *CORE*, *coordination* and the *awareness engines*. The *CORE engine* provides basic primitives used by the coordination and awareness engines. These primitives include constructs for defining resources, roles, and generic state machines. CMI's *coordination model* extends the traditional workflow coordination primitives with advanced primitives such as *placeholder*. The concept of placeholder enables the dynamic establish-

	Major Modules	*Communication Standards*	*Content and Business Process Standards*	*Key Technologies*
IBM WebSphere	Application Server, MQSeries, Business Components, WebSphere Commerce	MQSeries, JMS, IIOP, SOAP, HTTP	WSDL, XML, RosettaNet-PIP, cXML, EDI	Components (J2EE), XML, Web Services
Sun ONE	Forte tools and iPlanet	JMS, SOAP, LDAP, WAP, IIOP, HTTP	EDI, XML, WSDL	Components (J2EE), XML, Web services and Workflow
Oracle Integration Server	Integration Modeler, System Monitoring and Administration, Business Process Monitor, Business Intelligence	Oracle Queue, JMS, SOAP, IIOP, MQSeries, TIBCO/rendezvous, HTTP	XML, WSDL, EDI, RosettaNet-PIP, ebXML	Components (J2EE), workflow, XML, data mining, Web services
HP NetAction	HP Opencall, HP Chat, HP NetAction Internet Operating Environment	SOAP, JMS, IIOP, HTTP	XML, WSDL	Components (J2EE), XML, workflow (ChangeEngine), Web services
Microsoft .NET	.NET Framework and Tools, .NET Enterprise Servers, .NET Service Building Blocks	MSMQ, SOAP, Microsoft Host Integration Server, HTTP	XML, WSDL, RosettaNet-PIP, XLANG from BizTalk Server	DCOM, MSMQ, Web services, XML, BizTalk Orchestration Engine
BEA WebLogic Integrator	Application Server, Application Integration, Business Process Management, B2B Integration	SOAP, JMS, IIOP, HTTP	WSDL, XML, RosettaNet-PIP, BEA-XOCP	Components (J2EE), XML, workflow, Web services
WebMethods	Enterprise Server, Enterprise Adaptor, and Enterprise Rule Agent	SOAP, IIOP, JMS, HTTP	WSDL, XML, EDI, RosettaNet-PIP, ebXML, cXML, OBI	Components, workflow, Web services and Agents
Vitria Business Ware	Business Process Management, B2B Communications, Enterprise Application Integration and Real-Time Analysis	SOAP, IIOP, JMS, HTTP	XML, EDI, RosettaNet-PIP, ebXML, xCBL, cXML	Components, XML, workflow, process model, process analysis
TIBCO Active Enterprise	InConcert, IntegrationManager, MessageBroker, Hawk, and Rendezvous	SOAP, JMS, IIOP, MQSeries, HTTP	WSDL, XML, HL7, EDI, RosettaNet-PIP, BizTalk, ebXML, cXML, xCBL	Messaging software, XML, workflow, Web services

Table 2.4 Deployment Platforms

ment of trading relationships. A placeholder activity is replaced at runtime with a concrete activity having the same input and output as those defined as part of the placeholder. A selection policy is specified to indicate the activity that should be executed. If multiple providers offer implementations for an activity interface, the selection policy may use a broker to choose the implementation that offers the "best" quality of service. CMI's *awareness model* captures information that is closely related to a specific role and situation of a process participant. Awareness information is specified by process/awareness designers using *awareness specifications*.

eFlow [37] – *eFlow* is a platform that supports the specification, enactment, and management of composite services. A composite service is described as a process schema that combines basic or composite services. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. It may include *service*, *decision*, and *event* nodes. Service nodes represent the invocation of a basic or composite service. The definition of a service node contains a *search recipe* represented in a query language. When a service node is invoked, a search recipe is executed to select a reference to a specific service. Decision nodes specify the alternatives and rules controlling the execution flow. Event nodes enable service processes to send and receive several types of events. A *service process instance* is an enactment of a process schema. To support heterogeneity of services, eFlow provides adapters for services that support various B2B interaction protocols such as OBI and RosettaNet.

WISE (*Workflow based Internet SErvices*) [189, 118] – WISE aims at providing an infrastructure for the support of cross-organizational business processes in *virtual enterprises*. WISE architecture is organized into four components: process *definition*, *enactment*, *monitoring* and *coordination*. The *process definition* component allows *Virtual Business Process* (VBPs) to be defined using as building blocks the entries of a catalog where companies within a trading community (TC) can post their services. The *process enactment* component compiles the description of the VBP into a representation suitable for enactment and controls the execution of the process by invoking the corresponding services of the TC. The *process monitoring* component keeps track of the progress made in the execution of the VBP. The information produced by this tool is used to create an awareness model used for load balancing, routing, quality of service, and analysis purposes. The *process coordination* component supports multimedia conferencing and cooperative browsing of relevant information between all participants in the TC.

CrossFlow [127] – The main contribution of CrossFlow is in using the concept of *contracts* as a basic tool for cooperation. Businesses specify their interactions through *contracts* (e.g., purchase and employment contracts).

When a provider wants to advertise a service, it uses its contract manager to send a contract template to a *trader* or *matchmaking engine*. When a consumer wants to outsource a service, it uses a contract template to search for relevant providers via the trader. If a matching is found between consumer's requirements and provider's offer, an electronic contract is made by filling in the template. Based on the specifications in the contract, a dynamic contract and service enactment infrastructure are set up. The symmetrical infrastructure in provider's and consumer's sides contains proxy gateways that control their interactions. The dynamically created modules can be removed after contract completion.

Mentor-Lite [222] – Mentor-Lite addresses the problem of distributing the execution of workflows. The idea is to partition the overall workflow specification into several sub-workflows, each encompassing all the activities that are to be executed by a given entity within an organization. The basic building block of Mentor-Lite is an interpreter for workflow based on state charts. Two other modules are integrated with the workflow interpreter defining the workflow engine: *communication manager* and *log manager*. The *communication manager* is responsible for sending and receiving synchronization messages between the engines. It uses the *Transaction Processing* (TP) monitor *Tuxedo* for delivering synchronization messages within queued transactions. The *log manager* provides logging and recovery facilities. A separate workflow log is used at each site where a workflow engine is running.

XL (*XML Language*) [74, 75] – XL defines an XML language for the specification of Web services. An XL service specification contains *local declarations*, *declarative clauses*, and *operation specifications*. Two kinds of local variables can be declared in XL. The first kind of variable represents the internal state of the service. The second kind of variable represents the internal state of a particular conversation in which the service is involved (e.g., session ID). Declarative clauses include variables that control the Web service global state. In particular, the *history* and *on change* clauses address the issue of external manageability and adaptability. If the *history* clause is specified, all operation invocations are automatically logged. The *on change* clause uses triggers to detect changes in variables declared in the Web services' local declarations.

SELF-SERV (*compoSing wEb accessibLe inFormation and buSiness sERvices*) [16, 192] – SELF-SERV proposes a process-based language for composing Web services based on *state charts*. It also defines a *peer-to-peer* Web service execution model in which the responsibility of coordinating the execution of a composite service is distributed across several peer components called *coordinators*. The coordinator is a lightweight scheduler which determines when a state within a state chart should be entered and what should be done when the state is entered. It also determines when

should a state be exited and what should be done after the state is exited. The knowledge needed by a coordinator to answer these questions at runtime is statically extracted from the state chart describing the composite service operations and represented in the form of routing tables.

The aforementioned prototypes are compared in Table 2.5 using key interaction layers. For example, eFlow uses RMI at the communication layer. At the content layer, eFlow provides adapters to support different interaction protocols such as OBI and RosettaNet. Interoperability at the business layer is enabled through a process description model based on state machines.

	Communication	*Content*	*Business Process*
CMI	Transport protocols (e.g., HTTP, CORBA) must *a priori* be agreed upon	Message format (e.g., XML, EDI) must *a priori* be agreed upon	State machine based model for process description
WISE	Coordination and communication module	Not Addressed	Virtual business processes
Cross Flow	Java RMI	Contract in XML	Contracts
Mentor-Lite	Transaction Processing (TP) monitor (Tuxedo)	Not Addressed	Business processes expressed as state and activity charts
eFlow	Java RMI	Provides adapters to support different protocols such as OBI and RosettaNet	State machine based model for process description
XL	SOAP	XML Schema	Little or no statements for inter-service business processes
SELF-SERV	SOAP	Not Addressed	State charts

Table 2.5 Prototypes vs. Interaction Layers

The same prototypes are compared in Table 2.6 using key interaction dimensions. For example, eFlow allows loose coupling among participants. In terms of autonomy, trading partners do not need to reveal how their services are implemented. Heterogeneous interaction protocols are supported through adapters. External manageability and adaptability are possible via event tracking and process templates respectively. Security, however is not addressed. Scalability is accommodated using distributed service enactment engines.

	Coupling	*Autonomy*	*Het.*	*Ext. Manag.*	*Adapt.*	*Security*	*Scal.*
CMI	Tight and long term	External systems only need to reveal the state they are in after they accomplish a task, not how they accomplish the task	Use of object-oriented proxies	State dependent control flow and use of awareness events	Primitives such as *optional* and *inhibitor* can be used for coping with some unforeseen events	Role	Distributed and parallel engines for execution
WISE	Tight and long term	Partners must advertise services in encapsulated objects	Object-based middleware	Process monitoring and analysis module	Execution guarantee	Not Addressed	Distributed architecture
Cross Flow	Loose and transient	Partners must agree on service contract definition	Partners must install service contract run time environment	Quality of Service (QoS) module provides monitoring facilities	Primitives for flexible execution are restricted to those provided by traditional workflows	Not Addressed	Cost of entry: participants must locally install contract run time environment
Mentor-Lite	Tight and long term	Participants do not need to reveal how services are implemented	Application programs are connected to the workflow engine by specific wrappers	Not Addressed	Not Addressed	Not Addressed	Workflows are partitioned into several subworkflows and distributed
eFlow	Loose and long term	External systems need to describe their services not their implementation	Provides adapters for different protocols and platforms such as OBI, RosettaNet, and e-speak	Event tracking	Provides process templates, service nodes, and service data repositories for reuse	Not Addressed	Distributed service enactment engines
XL	Loose and transient	Participants do not need to reveal how services are implemented	Web services can be written in XL, Java, or other languages	History clauses	Change clauses	Security features of J2EE	Not Addressed
SELF-SERV	Loose and transient	Participants do not need to reveal how services are implemented	Service wrappers	Not Addressed	Not Addressed	Not Addressed	Peer-to-peer execution model

Table 2.6 Prototypes vs. Interaction Dimensions

2.7 Summary and Discussion

In this section, we compare the different approaches to interactions on the pre-Semantic Web and Semantic Web eras. We discuss the commonalities and differences between Web services and related technologies such as components, agents, and workflows. Finally, we illustrate the role of Web services on the Semantic Web.

2.7.1 Comparison of Semantic Web Interaction Technologies

We identify two sets of parameters that together exhaustively define how applications interact on the Web. The first set (applicable to enabling technologies and prototypes) consists of the following parameters: communication layer, content layer, and business process layer. The second set (applicable to enabling technologies and prototypes) consists of the following parameters: coupling, autonomy, heterogeneity, external manageability, adaptability, security, and scalability.

In Table 2.7, enabling technologies technologies for the pre Semantic Web era are compared using the most important architectural layers. For example, communications in EDI are conducted through VANs. The content layer is supported through ANSI X12 and EDIFACT formatted documents. Interactions at the business process layer are limited to pre-defined business processes. In Table 2.8, key Semantic Web technologies are compared using the most important architectural layers. For example, *Web Services*' communication layer is typically provided by SOAP. The content layer is supported by using WSDL language. However, WSDL currently provides little support for semantic description of business documents. One of the current trends to support semantic interoperability is the use of ontologies (e.g., DAML-S). WSFL, XLANG, and BPEL4WS languages provide support for interactions at the business process layer. However, these languages are still at their early stage.

In Table 2.9, enabling technologies For the pre Semantic Web era are now compared using salient interaction parameters (dimensions). For example, *EJB-based* B2B's coupling is tight and long term. Autonomy is provided by separating the interface and the implementation using the EJB remote interface. Heterogeneity is supported at the platform levels (e.g., Unix and Windows) but only Java is supported. The use of event services provides support for external manageability. Adaptability is partially addressed through the separation between contracts and business logic. Security is provided through the EJB container. Scalability is not much of a concern for intra-enterprise applications. In Table 2.10, Semantic Web interaction technologies

	Communication	*Content*	*Business Process*
EDI	VANs	ANSI X12 and EDIFACT formatted documents	Pre-defined business processes
EDIINT	SMTP (for EDIINT AS1) and HTTP (for EDIINT AS2)	ANSI X12 and EDIFACT formatted documents	Pre-defined business processes
OBI	HTTP	ANSI X12 formatted documents	Pre-defined protocol for Internet procurement
CORBA	ORBs and IIOP	Not Addressed	Ad hoc: hand-coded programming of the integration logic
DCOM	DCOM runtime environment	Not Addressed	Ad hoc: hand-coded programming of the integration logic
EJB	RMI/JMS	Not Addressed	Ad hoc: hand-coded programming of the integration logic
Workflows	Not Addressed	Not Addressed	Inter-enterprise business processes (public and private)

Table 2.7 Pre Semantic Web Technologies vs. Interaction Layers

	Communication	*Content*	*Business Process*
Ontologies	Not Addressed	Shared Conceptualization	Not Addressed
Web Services	SOAP and REST	WSDL focuses on syntactic interoperability. Semantic Web service frameworks provide support for semantic interoperability	BPEL4WS and ebXML's business process specification
Agents	Agent Communication Language	Common Ontology	Not Addressed
eCO	HTTP	xCBL	Not Addressed
cXML	HTTP and URL form encoding	XML DTDs	Pre-defined protocol for Internet procurement
RosettaNet	HTTP, E-mail, etc.	RosettaNet Business Dictionary and RosettaNet Technical Dictionary	Partner Interface Processes (PIPs). Pre-defined protocol for Internet procurement

Table 2.8 Semantic Technologies vs. Interaction Layers

are compared using the same parameters. For example, Web services support both transient and long-term as well as loose relationships. The separation between WSDL interface and implementation definition caters for the autonomy of participants. WSDL descriptions allow the wrapping of heterogeneous applications. The issues of external manageability and adaptability are currently not addressed. Research efforts are still on-going to provide security mechanisms for Web services.

2.7.2 Web Services and Related Technologies

Web services share some features with the component, agent, and workflow-based approaches. However, several characteristics differentiate Web services from components, agents, and workflows.
Web Services and Components – Web services use document-based communication. This in in contrast with component-based frameworks which use object-based communication, thereby yielding systems where the coupling between components is tight [213]. Additionally, by using HTTP as a communication protocol, Web services enables much more firewall-friendly computing than component-based systems. For example there is no standard port for IIOP, so it normally does not traverse firewalls easily. Although a specification has been submitted to OMG to dealing with IIOP firewall traversal, ORB implementations are still using their own proprietary solutions such as *VisiBroker's GateKeeper* and *IONA's WonderWall*.

Web Services and Agents – As in Web services, agent-based systems provide "directories" where an agent advertise their distinct functionalities and other agents search to locate it [101]. However, agents are inherently communicative, whereas Web services are passive until invoked. Web services use XML formatted message over ubiquitous protocols such as HTTP to interact with each other. However, agents use ACL language to format their messages. As discussed in Section 2.4, agents are loosely coupled while agents are tightly coupled. All these reasons make Web service suitable for enabling cross-organizational interactions. Agents generally provide alerts and updates when new events become available.

Web Services and Workflows – Service composition shares some features with workflows. For example, to define a composite service, the provider needs to specify the flow of service invocations (i.e., services to be invoked, their input and output data, and their execution dependencies). Similarly, in a workflow, the designer must specify the flow of work (i.e., work items to be invoked, their input and output data, and their execution dependencies). However, several features distinguish between composite services and workflows. We list below the main differences:

- Distributed workflow systems require that each participating enterprise deploys a full-fledged execution engine, capable of interpreting the workflow definition. The same workflow model must be adopted by all the participants in the global workflow. They assume a tight coupling among the distributed sub-workflows. Modifications to sub-workflows need to be coordinated. Hence, distributed workflow systems are appropriate for the development of business processes within a single enterprise.
- In workflows, input and output data are typically specified by a set of variable names [38]. The values of the input variables are passed to a

selected resource at the time an activity is started. Activity execution results are inserted into the output variables. Communications between the workflow system and resources is done through *adapters* which deal with data mapping issues. However, the development of adaptors is a difficult and tedious task [38]. In contrast, Web services interact by exchanging messages, thereby avoiding the need for adapters.

- A number of XML-based interaction standards (e.g., RosettaNet, cXML) are being defined to support B2B interactions. Many applications that support such standards are being or have been developed. It is likely that composite services will interact with services that follow one of these standards [38]. However, many workflow systems do not support such standards, although some vendors are moving in this direction [38].

2.7.3 The Role of Web Services in the Semantic Web Landscape

The current technologies for Web-based interactions may be viewed as complementary. In Figure 2.9, we summarize these technologies and the way they co–exist in a Semantic Web interactions framework. Component middleware, workflows, Web services, and agents define the building blocks for developing Semantic Web applications. Ontologies, EDI, OBI, and XML-based standards (eCO, RosettaNet, etc.) focus on defining the semantics of interactions among businesses. Component middleware (CORBA, DCOM, and EJB) are suitable for building robust and secure applications *within* an enterprise. Intra-enterprise business processes are managed using systems such as ERPs (e.g., *SAP/R3*) and workflows (e.g., IBM's *MQSeries*). Inter-enterprise workflows mostly focus on interactions at the business process layer. Their aim is to automate business processes that span the boundaries of disparate enterprises.

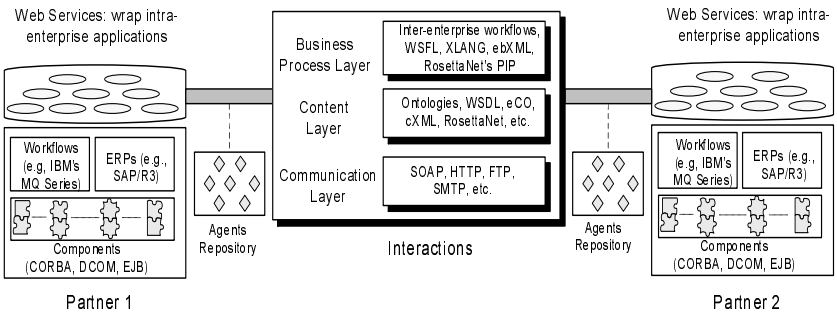


Fig. 2.9 Technologies in a Semantic Web Interaction Framework

Web services take components a step further by enabling inter-enterprise interactions. They define the entry points to enterprises' internal systems. Web services may wrap intra-enterprise components to provide connectivity between autonomous and heterogeneous inter-enterprise applications. They may also wrap other applications developed, for example, in Java or Visual Basic. Agents may be used add new functions to Web services (e.g., change monitoring) or customize existing functions based on operating conditions or observations on users' behavior. Web services may use a workflow-based approach or service composition techniques and standards (e.g., BPEL4WS) to support inter-enterprise business processes. Web services adopt ontologies and XML-based standards to capture the semantics of documents and business processes. For example, Web services may use *RosettaNet* or *cXML* to carry out interactions among businesses according to these frameworks. *EDI* and *OBI* standards may also be used to enable interactions at the content layer. However, they provide little support in terms of expanding the set of supported document types.

	Coupling	*Auto.*	*Het.*	*Ext. Manag.*	*Adapt.*	*Security*	*Scal.*
EDI	Loose and long term	New documents must be approved by EDI guideline committees	Support of heterogeneous applications thanks to translator software	Not Addressed	Impact of local changes limited. New documents must be approved	Private VANs	Expensive networks. Need to agree on implementation conventions
EDIINT	Loose and long term	New documents must be approved by EDI guideline committees	Support of heterogeneous applications thanks to translator software	Not Addressed	Impact of local changes limited. New documents must be approved	RFC 1847, MIME security with PGP, etc.	Lower entry-cost than EDI (Internet-based)
OBI	Loose and long term	Higher than EDI (document extensibility is not an important requirement)	Support of heterogeneous applications thanks to translator software	Not Addressed	Higher than EDI (document extensibility is not an important requirement)	SSL/HTTP, digital signatures, and digital credentials	Lower entry-cost than EDI (Internet-based)
CORBA	Tight and long term	Separation between interface and implementation (IDL)	Different languages (e.g., Java, C++) and platforms (Unix, Windows)	Event service	Not Addressed	Supported by CORBA security service	Suitable for intra-enterprise applications. Participants need to have a stub for each component server
DCOM	Tight and long term	Separation between interface and implementation (IDL)	Different languages (e.g., Microsoft J++, C++) but Windows platform	Event service	Not Addressed	Based on Windows NT security model	Suitable for intra-enterprise applications. Proxies needed at client side
EJB	Tight and long term	Separation between interface and implementation (EJB remote interface)	Java language and different platforms (Unix, windows)	Event service	Contracts can be changed independently of the business logic	Security features provided in EJB container	Suitable for intra-enterprise applications. Participants must define a remote interface for each bean
Workf.	System-specific	Separation between public and private business processes	Different workflow engines	Adding appl-specific states	Process wrappers and adapters help localize changes	Not Addressed	May require creation of new process wrappers and adapters

Table 2.9 Pre Semantic Web Technologies vs. Interaction Dimensions

	Coupling	*Auto.*	*Het.*	*Ext. Manag.*	*Adapt.*	*Security*	*Scal.*
Ontology	Loose and long-term	Participants bound to concepts defined within the ontology	Participants bound to concepts defined within the ontology	Not Addressed	Not Addressed	Not Addressed	Participants need to describe their applications according to a given ontology
Web Services	Loose, transient, and long-term	Separation between WSDL interface and implementation definitions	WSDL descriptions to wrap underlying applications	Not Addressed	Not Addressed	On-going efforts (e.g., XML digital signatures, S-HTTP)	Availability of development tools. XML tags increase the volume of information. Replicated registries must exchange data to maintain coherence
Agents	Tight and long-term	Participants must adhere to concepts defined within a common ontology	Agents may be used to wrap heterogeneous applications	Events and monitoring agents	Not Addressed	Not Addressed	Agents need to formulate requests based on the common ontology. They must have a way of matching requests with their known ontologies
eCO	Loose, transient, and long term	Separation between description and implementation. Marketplaces may hinder autonomy	XML Schemas. Marketplaces may help addressing semantic heterogeneity.	Not Addressed	Impact of local changes limited. Support of extensibility documents	Optional	Establishment of a relationship with a partner does not require additional work from this partner
cXML	Loose, transient, and long term	Higher than EDI because document extensibility is not an important requirement	XML DTDs	Not Addressed	Impact of local changes limited. Extensibility of documents is not required	Authentication information. Trusted intermediaries	Cost of entry is lower than in OBI thanks to the use of XML
Rosetta Net	Loose and long term	Separation between description and implementation. Marketplaces may hinder autonomy	RosettaNet business and technical dictionaries. Use of vertical technology	Not Addressed	Impact of local changes limited. Support of extensibility documents	SSL/HTTP Digital certificates and signatures	Establishment of a relationship with a partner does not require additional work from this partner

Table 2.10 Semantic Web Technologies vs. Interaction Dimensions



http://www.springer.com/978-1-4419-8464-7

Service Composition for the Semantic Web
Medjahed, B.; Bouguettaya, A.
2011, XV, 191 p., Hardcover
ISBN: 978-1-4419-8464-7