The VLDB Journal (2003) 12: 59–85 / Digital Object Identifier (DOI) 10.1007/s00778-003-0087-z

Business-to-business interactions: issues and enabling technologies

Brahim Medjahed[1], **Boualem Benatallah**[2,⋆], **Athman Bouguettaya**[1,⋆⋆], **Anne H. H. Ngu**[3], **Ahmed K. Elmagarmid**[4,⋆⋆⋆]

[1] Department of Computer Science, Virginia Tech, 7054 Haycock Road, Falls Church, VA 22043 USA; e-mail: {brahim,athman}@vt.edu
[2] School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052 Australia; e-mail: boualem@cse.unsw.edu.au
[3] Department of Computer Science, Southwest Texas State University, San Marcos, TX 78666 USA; e-mail: angu@swt.edu
[4] Department of Computer Sciences, Purdue University, 250 N. University Street, West Lafayette, IN 47907 USA; e-mail: ake@cs.purdue.edu

Edited by A. Dogac. Received: July 23, 2002 / Accepted: January 6, 2003
Published online: April 3, 2003 –

Abstract. Business-to-Business (B2B) technologies pre-date the Web. They have existed for at least as long as the Internet. B2B applications were among the first to take advantage of advances in computer networking. The Electronic Data Interchange (EDI) business standard is an illustration of such an early adoption of the advances in computer networking. The ubiquity and the affordability of the Web has made it possible for the masses of businesses to automate their B2B interactions. However, several issues related to scale, content exchange, autonomy, heterogeneity, and other issues still need to be addressed. In this paper, we survey the main techniques, systems, products, and standards for B2B interactions. We propose a set of criteria for assessing the different B2B interaction techniques, standards, and products.

Keywords: E-commerce – B2B Interactions – EDI – XML – Components – Workflows – Web services

1 Introduction

The growth of the Web is revolutionizing the way businesses interact with their partners and customers. Millions of organizations 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 [27,28]. For instance, *Dell*'s computer online sales exceeded $18 million per day in the year 2000 [1]. The current scale of E-commerce has been phenomenal in several domains, including healthcare, travel, auto supply chain, e-procurement, shipping, and warehousing. All predictions agree that E-commerce will be worth billions of dollars in new investments [1]. According to a recent study by *Gartner* (a leading research and advisory firm), the E-commerce market is on track, despite the slowdown in the economy, to total US$8.5 trillion by the year 2005 [38].

The Web offers a unique opportunity for E-commerce to take a central stage in the fast growing online economy [8,16,28]. With the advent of the Web, the first generation of Web-based E-commerce was born: *Business-to-Customer* (B2C) Applications. Examples of B2C applications include virtual malls, customized news delivery, traffic monitoring, and route planning. Another quieter E-commerce revolution with far more dramatic economic implications has been taking place away from the spotlights: *Business-to-Business* (B2B) E-commerce. Examples of B2B applications include procurement, Customer Relationship Management (CRM), billing, accounting, human resources, supply chain, and manufacturing. B2B E-commerce far exceeds B2C E-commerce both in the volume of transactions and rate of growth [34]. Despite the dot-com debacle that shook the US economy, B2B E-commerce is still strong and predictions agree that B2B E-commerce future looks even brighter [34]. While B2B E-commerce has been around for at least as long as the Internet, it reached its full potential with the emergence of the Web as a conduit for efficient B2B transacting. Numerous organizations started using the Web as a means to automate relationships with their business partners. This has elicited the formation of alliances in which businesses joined their applications, databases, and systems to share costs, skills and resources in offering value-added services. The ultimate goal of B2B E-commerce is therefore to have inter- and intra-enterprise applications evolve independently, yet allow them to effectively and conveniently use each other's functionality.

An important challenge in B2B E-commerce is *interaction*. Interaction is defined as consisting of *interoperation* and *integration* with both internal and external enterprise applications [35]. This has been a central concern because B2B applications are composed of autonomous, heterogeneous, and distributed components. Interactions among loosely coupled and tightly coupled systems has been, over the past 20 years, an active research topic in areas such as databases, knowledge-

⋆ This author's work is partially supported by the Australian Research Council's Discovery Grant DP0211207.
⋆⋆ This author's work is supported by the National Science Foundation's Digital Government Program under grant 9983249-EIA and by a grant from the Commonwealth Information Security Center (CISC).
⋆⋆⋆ This author's work is supported by the National Science Foundation's Digital Government Program under grant 9983249-EIA.

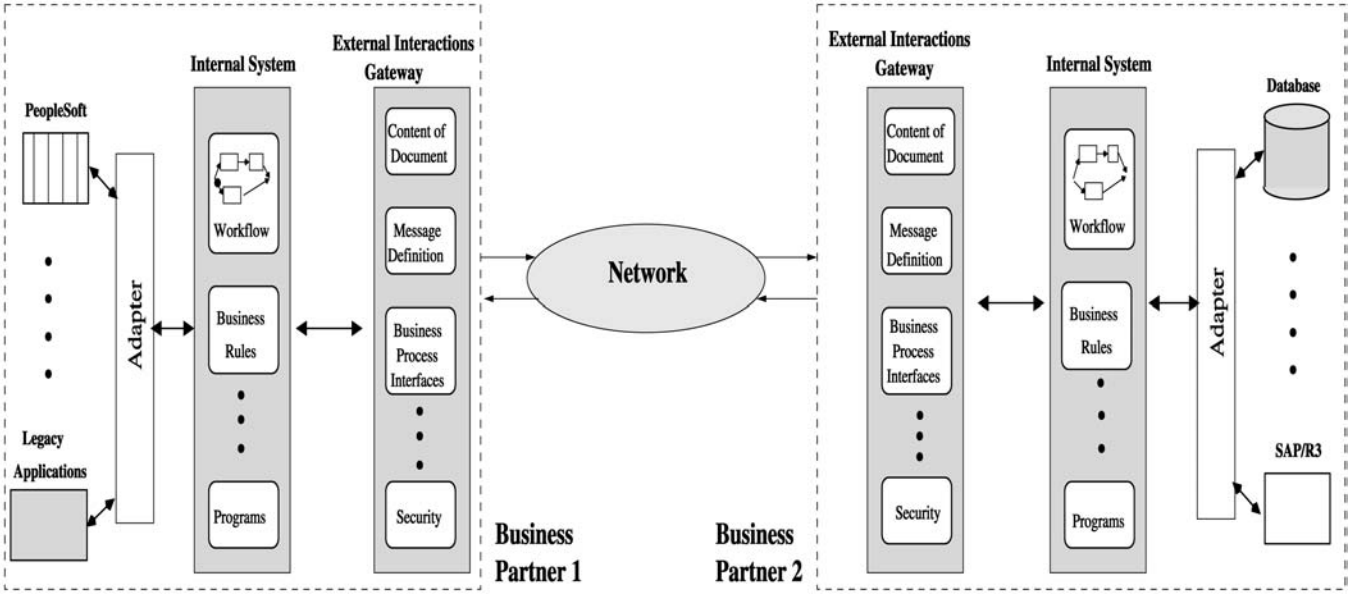

Fig. 1. Architecture of a B2B interaction framework

based systems, and digital libraries [15]. Interactions in B2B E-commerce offer unique challenges because of issues such as scalability, volatility (dynamism), autonomy, heterogeneity, and legacy systems. B2B E-commerce requires the integration and interoperation of both applications and data. Disparate data representations between partners' 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) trading partners' systems.

The aim of this work is to survey the main issues and solutions to B2B E-commerce interactions. Previous work dealing with interoperation in loosely coupled systems mostly focused on databases and digital libraries [80,73]. Recent surveys addressing B2B E-commerce (e.g., [3,17,26,29,54,81]) were mostly fragmented and lacked a holistic view of the problem. In this paper, we propose a framework for comparing B2B interaction technologies. The framework identifies the interaction layers, i.e., communication, content, and business process. It also proposes a set of dimensions to study B2B interaction solutions. Additionally, the work covered in those surveys did not specifically focus on B2B interactions. In this paper, we take a broad approach to study B2B interactions.

The survey's organization reflects the historical evolution of B2B solutions and supporting technologies. In Sect. 2, we define the different interaction layers in B2B E-commerce. We then identify a set of dimensions for comparing B2B solutions across these layers. In Sect. 3, we study several popular B2B E-commerce approaches, namely, EDI, components, and workflows. These approaches are evaluated against a predefined set of dimensions. In Sect. 4, we survey and evaluate the trends in B2B enabling techniques that include XML-based frameworks and Web services. In Sect. 5, we discuss a few representative research projects that enable B2B interactions. In Sect. 6, we overview some of the most popular B2B deployment platforms. Finally, Sect. 7 provides a tabular comparison summary of B2B enabling technologies, prototypes, and commercial platforms.

2 Overview of B2B interaction frameworks

In the first part of this section, we present a typical architecture of a B2B interaction framework. We then identify the different layers that make up such framework. Finally, we define the dimensions for assessing B2B architectures across these layers. These dimensions are used as a benchmark for evaluating B2B E-commerce interaction solutions.

2.1 Architecture of a B2B interaction framework

B2B applications refer to the use of computerized systems (e.g., Web servers, networking services, databases) for conducting business (e.g., exchanging documents, selling products) among different partners [16]. The building blocks for B2B applications are provided through a *B2B interaction framework* (Fig. 1). These include modules for: (1) defining and managing internal and external *business processes*, (2) integrating those processes, and (3) supporting interactions with back-end application systems such as ERPs (*Enterprise Resource Planning*) [17]. 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 [17].

We depict in Fig. 1 the main components of a B2B 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 B2B standard (e.g., EDI [67,101], *RossettaNet* [76]) or bilateral agreements. B2B 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 (e.g., encryption,

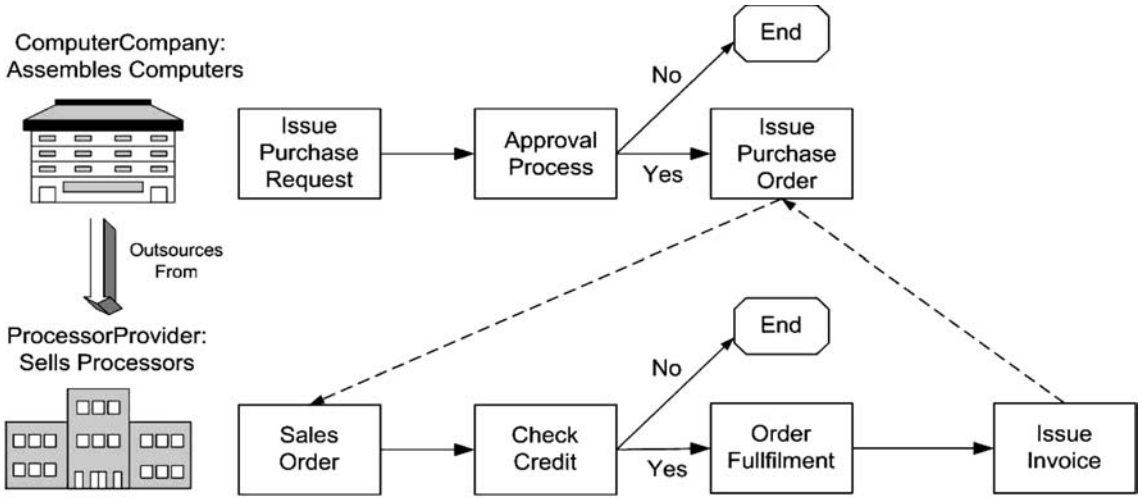

Fig. 2. B2B interactions: a running example

non-repudiation), etc. A B2B framework may have to support several B2B standards and proprietary interaction protocols.

Example. As a running example, we use an application from the computer manufacturing domain. We consider a company, called `ComputerCompany`, that offers complete computer configurations. Assume this company does not own all the hardware parts (e.g., monitors, processors) needed for assembling computers. It would interact with other business partners (e.g., `ProcessorProvider`) to acquire those parts. Interactions between `ComputerCompany` and `ProcessorProvider` are depicted in Fig. 2. `ComputerCompany` wants to purchase processors in bulk from `ProcessorProvider`. An employee in `ComputerCompany` first issues a *request for purchase*. Upon approval of this request, a *purchase order* is issued and sent to `ProcessorProvider`. The purchase order is transformed into a *sale order* at `ProcessorProvider`'s order processing system. After satisfactory *credit checking*, an *order fulfillment* is issued by `ProcessorProvider` and an *invoice* is finally sent to `ComputerCompany`.

2.2 Layers of B2B interaction frameworks

Interactions in B2B applications occur in three layers: *communication*, *content*, and *business process* layers. For example, `ComputerCompany` and `ProcessorProvider` need to agree on their joint business process (e.g., delivery mode, contracts). `ProcessorProvider` needs also to "understand" the content of the purchase order sent by `ComputerCompany`. Finally, there must be an agreed upon communication protocol to exchange messages between `ComputerCompany` and `ProcessorProvider`.

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, `ComputerCompany` and `ProcessorProvider` may use *Java RMI* (*Remote Method Invocation*) [82] and IBM's *MQSeries* [48], 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 `ProcessorProvider` 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 `ComputerCompany` uses xCBL (*XML Common Business Library*) [32] to represent business documents and `ProcessorProvider` expects documents in cXML (*Commerce XML*) [23], 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, `ComputerCompany` and `ProcessorProvider` need to agree on the procedures of their joint business process. The semantics of interactions among `ComputerCompany` and `ProcessorProvider` 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 [58].

2.3 Dimensions for evaluating B2B interaction frameworks

B2B E-commerce covers a wide spectrum of interactions among business partners. The types of interactions depend on the usage scenarios, parties involved, and business requirements. Each framework makes specific tradeoffs with regard to the requirements of B2B 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 a set of dimensions to study interaction issues in B2B E-commerce. We consider the following dimensions: *coupling among partners*, *heterogeneity*, *autonomy*, *external manageability*, *adaptability*, *security*, and *scalability*.

- *Coupling among partners:* this dimension refers to the degree of *tightness* and *duration* of coupling among business partners. Two partners are *tightly 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 B2B 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. Businesses 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 [18].
- *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, `ComputerCompany` may need to get the status (e.g., pending, approved) of the purchase order sent to `ProcessorProvider`. This requires that `ProcessorProvider` exposes sufficient information pertaining to measurements and control points to be used by `ComputerCompany`. 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. B2B 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. For example, `ComputerCompany` may decide to partner with a new processor provider for QoS purposes (e.g., cost, time). 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 `ProcessorProvider` decides to stop its supply activities (e.g., for local maintenance), `ComputerCompany` 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 B2B E-commerce reaches its real potential, sophisticated security measures must be in place to boost E-commerce partners confidence that their transactions are safely handled [103]. For instance, `ProcessorProvider` may need to check the authenticity of the purchase order before processing it. B2B applications must support mutual authentication, fine grain authentication, communication integrity, confidentiality, non-repudiation, and authorization. B2B 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 in order 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 B2B E-commerce. 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.

3 State-of-the-art technologies for B2B interactions

Technologies for B2B E-commerce 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* [10,39,40], will heavily draw on these solutions.

In this section, we overview these technologies in detail. Note that for the sake of space, other relevant technologies (e.g., agents [52]) and standards (e.g., SWIFT [84]) are not covered. An exhaustive list of standards can be found in [17].

3.1 Electronic data interchange (EDI)

EDI [67,101] 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 [101] and UN/EDIFACT [67]) and machine-processable format.

Figure 3 depicts two trading partners `ComputerCompany` and `ProcessorProvider` 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., `ComputerCompany`). 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. The translator wraps the EDI message in an electronic envelope that has an identifier for the receiver (i.e., `ProcessorProvider`). 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 `ProcessorProvider`. At the `ProcessorProvider` side, the reverse process takes place.

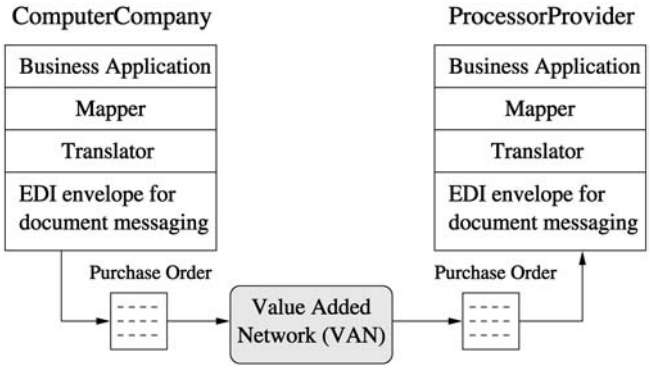

Fig. 3. B2B interactions in EDI

3.1.1 B2B 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 may be complex and time consuming [2]. This kind of change 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 [5]. 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 `ComputerCompany`'s purchase order is accepted then the `ComputerCompany` expects a purchase order acknowledgment, an invoice, and the delivery of the ordered processors in the time frame specified in the purchase order.

The EDI approach to B2B E-commerce 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's 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 documents 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 other's systems.

Although several EDI implementations have shown impressive results as set in the example of SEWP [66], 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 [2,53]. 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 [53]. 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 document [53]. It has been reported that 90% of the Fortune 500 companies in the United States use 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 [50] and OBI [70].

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.

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 [102]). 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*) [50] and OBI (*Open Buying on the Internet*) [70].

EDIINT [50] – 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 [49]. The standards referenced by EDIINT AS2 include RFC1847 and MIME Security with PGP [49]. 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 [70] – 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*) [68] over HTTP for securing communications. It also uses digital signatures and digital certificates for ensuring message 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 pre-defined 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).

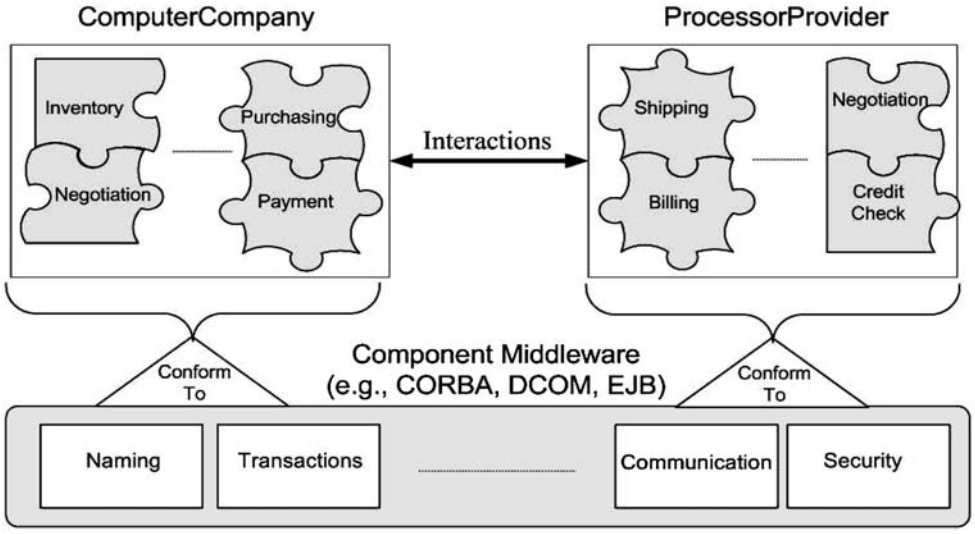

Fig. 4. Component-based B2B E-commerce

3.2 Components

Components are program modules that can be independently developed and delivered [12,85]. 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 [61,43]. For example, components may be realized using a functional language, an assembly language, or any other programming language [85].

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 [88]. Figure 4 depicts `ComputerCompany`'s and `ProcessorProvider`'s applications assembled from components. Each component represents an independent unit of a business functionality such as *payment*, *purchasing*, *billing*, and *shipping*. The different components work together to serve the needs of `ComputerCompany`'s and `ProcessorProvider`'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*) [72]: 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*) [62]: DCOM is Microsoft's technology for distributed components. It is an extension of COM, 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*) [75]: 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 the EJB component model. It provides a runtime environment that hosts and controls the beans.

The component-based approach for B2B E-commerce is more appropriate for a small number of partners within an enterprise [22]. However, with companies being merged and acquired at the current rate, there is a need to address B2B 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., ordering a processor for `ComputerCompany`) may be assembled from independently developed components (e.g., inventory, payment, purchasing). However, businesses generally would need to develop *ad hoc* solutions for defining intra and inter-enterprise business processes.

3.2.1 CORBA-based B2B E-commerce

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 [72]. 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 enables 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 [22]. 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 B2B 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 B2B E-commerce. 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.

3.2.2 DCOM-based B2B E-commerce

Similarly to CORBA, DCOM-based solutions for B2B E-commerce 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 [56].

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 [56]. 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). In addition, 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 [56]. 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.

3.2.3 EJB-based B2B E-commerce

At the communication layer, EJB uses *Java RMI* [82] 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 [22]. 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* [30]. 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) 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.

3.3 Workflows

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 (Fig. 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, `ComputerCompany`'s business process includes several activities such as issuing a purchase request, approving it, and issuing a purchase order. The information sent from `ComputerCompany`'s *Issue_Purchase_Request* activity to the *Approval_Process* activity includes the characteristics of the processor to be ordered (e.g., CPU speed). The scripting of activities and resource policies through *business process analysis, modeling, and definition tools* defines a *business process definition* (workflow schema) [25]. 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 (Fig. 5).

Workflow is a key technology for automating business processes that involve access to several applications. This makes workflow technology one of the most important candidates for integrating, automating and monitoring processes [20]. However, traditional workflow systems are based on the premise that the success of an enterprise requires the management of business processes in their entirety. Indeed, an increasing number of organizations have already automated their internal process management using workflows and enjoyed substantial benefits in doing so. Current business processes within an organization are integrated and managed either using ERP systems (e.g., *SAP/R3*, *Baan*, *PeopleSoft*) or various workflow systems such as IBM's *MQSeries* or integrated manually in on-demand basis. However, B2B E-commerce requires the flexible support of cross-enterprise relationships. Traditional workflow systems are ineffective when we consider the needs of B2B E-commerce, with its complex partnerships, possibly among a large number of highly evolving processes.

Current efforts (e.g., the Business Process Initiative - BPMI.org) promise to deliver 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 [103]. 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. Early projects in this direction focus mostly on the integration of a known and small number of tightly coupled business processes [39–41,24]. Recent workflow projects focus on loosely coupled processes (e.g., *eFlow* [19] and *WISE* [55]). They consider some critical requirements of B2B E-commerce such as adaptability and external manageability.

3.3.1 The workflow reference model

There are numerous workflow specification languages that are based on different paradigms. Usually, each commercial *Workflow Management System* (WfMS) implements its own specification language, with little attention paid to offering uniformity among products. To address this issue, the *Workflow Management Coalition* (WfMC) has defined the *Workflow Reference Model* [42]. The model includes a standardized set of interfaces and data interchange formats between workflow systems' components. The *WfMC*'s model puts more emphasis on the syntactic integration of workflow processes. It provides little support for inter-enterprise business processes.

Standardization efforts based on the *WfMC*'s model have been proposed. The *OMG Workflow Management Facility* has developed the *jointFlow* [51] standard. This standard translates the *WfMC* standards (except process definition) into an object-oriented framework and embeds this framework into the existing CORBA infrastructure. The object model provided by *jointFlow* has been used for the development of two standards: the *Simple Workflow Access Protocol* (*SWAP*) [14] and *Wf-XML* message set [100]. *SWAP* introduces an Internet-based protocol to instantiate, control, and monitor workflow processes. It defines a binding of *WfMC* standards using an HTTP-based interaction protocol. *Wf-XML* defines the XML data content required to communicate between workflow engines. However, the method of transport of these messages (HTTP, SMTP, CORBA IIOP, etc.) is left to the solution provider.

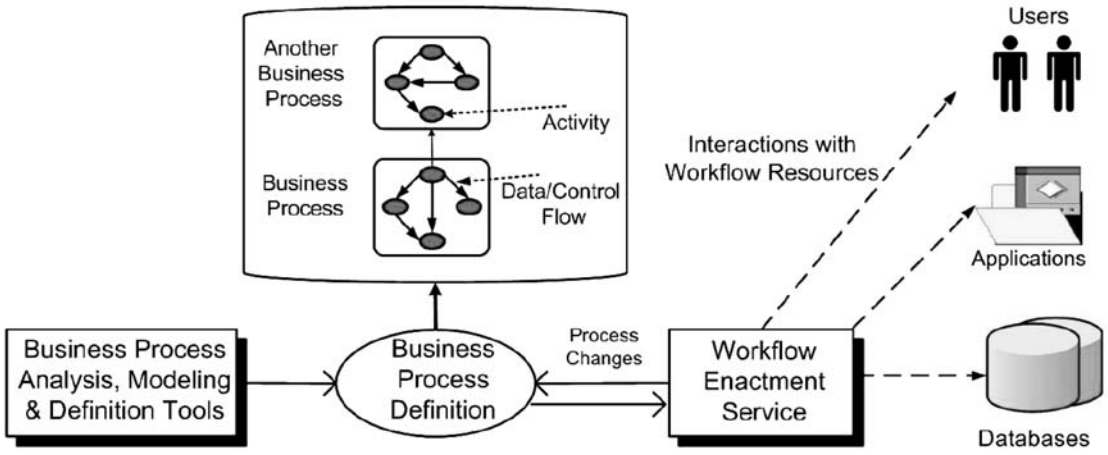

Fig. 5. Workflow system characteristics

3.3.2 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 [65]. 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 B2B E-commerce where: (i) partners may use disparate data and process representation models (e.g., `ComputerCompany` uses EDI and `ProcessorProvider` uses RosettaNet); and (ii) private business processes may require access to proprietary/legacy data sources and applications (e.g., Oracle database for `ComputerCompany` and SAP application for `ProcessorProvider`). 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.

3.3.3 Collaborative process management

Inter-enterprise business processes management features the separation between *public* and *private processes* [17,25]. 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. For example, *Order Processor* and *Check Credit* are, respectively, public and private processes of `ProcessorProvider`. Public and private processes interact through *process wrappers*. Process wrappers consist of predefined 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* [32] to represent business documents, and the private business process expects documents in *cXML* [23], 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* [48] and another based on HP's *Process Manager* [44].

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 B2B 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 B2B frameworks). External manageability

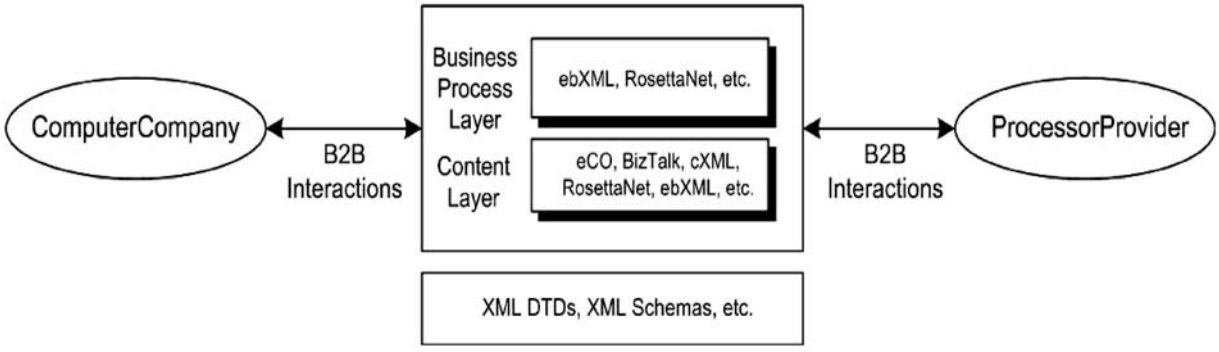

Fig. 6. XML-based frameworks for B2B interactions

can be provided by adding specific activities in public processes.

4 Trends in supporting B2B interactions

The exponential growth of the Web opened opportunities for businesses to transact across all types of boundaries (geographical, national, business category, etc.). It is noteworthy that the traditional approaches for B2B interactions were not devised for the Web. Therefore, early research had focused on providing a lingua franca for B2B E-commerce that went beyond HTML to reflect the richness of the data being advertised/published. Such an effort resulted in the development of XML (*eXtensible Markup Language*) [92]. However, XML was not developed to define semantics, description of message exchange sequences, or definition of correct interpretations of exchanged messages [17]. To address this issue, standardization committees defined XML-based B2B interaction frameworks (or standards). A parallel effort is the work on the *Semantic Web* [11]. The idea behind the Semantic Web is to readily make more semantics available over the Web and enable machines to automatically process applications. *Web services* are slated to play a major role in enabling the Semantic Web. They would provide value added services to users and businesses. In this section, we overview major XML-based B2B interaction frameworks and Web services technologies.

4.1 XML-based B2B interaction frameworks

A large number of contemporary B2B interaction standards are based on XML, an emerging standard for data representation and exchange on the Internet. They aim at overcoming some of the limitations of traditional EDI standards (e.g., high costs in terms of expensive VAN infrastructure and EDI software). For example, several commercial and public XML processing and integration tools (e.g., DOM parser [91]) are available.

An XML document consists of nested data items called elements which can have sub-elements and attributes. It is associated with a type generally defined as a *DTD* (*Document Type Definition*) [92] or *XML Schema* [97]. The type describes the structure of the document and the relationships between the various elements that form the document. Encoding business information (e.g., service requests and responses) in an XML document with a common XML Schema eliminates the need for one-to-one information translation. An organization would create and publish XML documents that describe its offers, requirements, assumptions, and terms for doing business. Partners would then interact with each other after inspecting and understanding each other's descriptions. For example, `ProcessorProvider` would provide `ComputerCompany` with the possibility of ordering processors by submitting a *purchase order*. As a result, `ProcessorProvider` would commit to send back an *invoice* and a *shipping notice*. 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.

There is a large number of XML-based frameworks for B2B interactions (Fig. 6). In what follows, we describe a representative set of XML-based interaction frameworks. An exhaustive list of XML-based B2B standardization efforts can be found in [69]. Existing frameworks mostly deal with enabling B2B interactions at the content (e.g., *eCO*, *cXML*) and business process (e.g., *RosettaNet*, *ebXML*) layers. However these frameworks sometimes overlap or even compete with each other [17]. 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 [17]. One solution to deal with such problem has been described in [17] through the use *B2B protocol and integration engines*. These execute actual message exchanges according to various standards.

4.1.1 eCO

eCO [32] aims at providing means to businesses to discover and access services regardless of the E-commerce 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, `ProcessorProvider` can use xCBL to create an XML document of its search catalog by customizing the generic *xCBL catalog DTD* with specific information about the search method (e.g., by CPU speed). 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 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 in 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 domain-specific 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, as with any document-based approach (e.g., EDI), business partners do not need to directly access each other's 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.

4.1.2 BizTalk

The *BizTalk*[1] approach [13] for enabling B2B interactions is based on leveraging several standards and technologies including the *Simple Object Access Protocol (SOAP)*, *XML*, and *Multipurpose Internet Mail Extensions (MIME)*. It relies on a centralized schema repository and layered logical architecture. The schema repository provides means to publish and validate XML-based schemas (e.g., verify their compliance with BizTalk), and manage their evolution (e.g., schema versioning) and relationships (e.g., specialization). The architecture consists of three layers: *application*, *BizTalk Framework Compliant (BFC) server*, and *transport*. Applications communicate with each other by sending business documents through BFC servers (one per end). BFC servers send BizTalk messages to each other via multiple communication protocols.

At the communication level, BizTalk leverages existing communication protocols such as HTTP, SMTP, and Microsoft Message Queue (MSMQ). Currently, BizTalk provides transport bindings for HTTP and SMTP. At the content level, BizTalk does not promote any specific language or standard. Instead, it refers to external XML-based schemas for describing the content and structure of business documents. BizTalk differentiates between documents and messages. A *BizTalk document* contains message-handling instructions (e.g., routing, identification, delivery, tracking, remote procedure call) and attached business documents (e.g., purchase order, invoice). Message-handling instructions are described using a standardized set of XML elements and attributes called *BizTags*. In essence, a BizTalk document is a variation of a SOAP message. Business documents are well-formed XML documents containing business data. A BizTalk message is the unit of communication between BFC servers. It contains a primary BizTalk document and one or more attachments (e.g., other BizTalk documents, XML documents). It also contains transport-specific headers (e.g., HTTP binding headers). At the business process level, BizTalk offers a special BizTag that may be used to include information about the business process that provides the processing context of a BizTalk document. *BizTalk Orchestration* is proposed as a technology to define and execute inter-enterprise processes. However, this effort is still in its infancy.

BizTalk's centralized repository provides interesting features such as schema validation and control. However, it falls short on support for scalability. The BizTalk framework supports S/MIME (version 3) for securing BizTalk messages. Finally, BizTalk is unique in defining specific BizTags (e.g., delivery and commitment receipts) to ensure reliable delivery of BizTalk documents. This feature provides a starting point for supporting external manageability.

[1] This discussion is based on the BizTalk Framework 2.0.

4.1.3 cXML

cXML (*Commerce XML*) [23] 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 Sect. 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 in 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.

4.1.4 RosettaNet

RosettaNet [76] aims at standardizing product descriptions and business processes in information technology supply chain applications. RosettaNet's supply chain includes 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 B2B 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 B2B 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.

4.1.5 ebXML

ebXML (*Electronic Business XML*) [31] aims at defining a set of specifications for enabling B2B interactions among companies of any size. The basic part of the ebXML infrastructure is the *repository*. It stores important information about businesses along with the products and services they offer. At the communication layer, businesses exchange messages through the *messaging service*. One important feature of the ebXML messaging service is that it does not rely on a specific transport protocol. It allows for the use of any common protocol such as SMTP, HTTP, and FTP.

At the content layer, companies interact through *business documents*. A *business document* is a set of information components that are interchanged as part of a business process. Business documents are composed of three types of components: *core components*, *domain components*, and *business information objects*. *Core components*, stored in the *core library*, are information components that are re-usable across industries. *Domain components* and *business information objects* are larger components stored in the *domain library* and *business library*, respectively. Core components are provided by the ebXML library while domain component and business information objects are provided by specific industries or businesses.

At the business process layer, ebXML defines a *business process specification schema* available in UML and XML versions. The UML version only defines a UML class diagram. It is not intended for the direct creation of a business process specification but provides a representation of all the elements and relationships required for its creation. The XML version allows the creation of XML documents representing ebXML-compliant *business process specifications*. ebXML provides a set of common business process specifications that are shared by multiple industries. These specifications, stored in the *business library*, can be used by companies to build customized business processes. Interactions between business processes are represented through *choreographies*. A *choreography* specifies the ordering and transitions between business transactions. 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 terms of payment).

The ebXML infrastructure enables secure and reliable communications by using emerging security standards (e.g., SSL and S-HTTP). In addition, digital signatures can be applied to individual messages or a group of related messages to guarantee authenticity. With regard to autonomy and adaptability, ebXML appears to offer the same kind of properties as eCO. External manageability can be provided by adding specific activities in shared business processes. The initial goal of the ebXML initiative was to support a fully distributed set of repositories which is an interesting feature for improving scalability. However, to date, only a single repository is specified.

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" [21]. It is also defined as "loosely coupled applications using open, cross-platform standards and which interoperate across organizational and trust boundaries" [87]. 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.

Web services share some features with the component-based approach [89]. For example, CORBA objects can be advertised in a *trader service* so that clients can find and invoke them. However, several characteristics differentiate Web services from components. First, Web services use document-based communication. This is in contrast with component-based frameworks which use object-based communication, thereby yielding systems where the coupling between components is tight. Additionally, by using HTTP as a communication protocol, Web services enable 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*.

4.2.1 The Web service reference model

Interactions among Web services involve three types of participants: *service provider*, *service registry*, and *service consumer* (Fig. 7). *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 corresponding 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, `ProcessorProvider` may advertise a Web service that includes `searchProcessor` and `orderProcessor` operations. `ComputerCompany`'s provider would then access the registry, discover `ProcessorProvider`'s service, and invoke its operations.

Three major standardization initiatives have been submitted to the W3C consortium to support interactions among Web services (Fig. 7):

- WSDL (*Web Services Description Language*) [96]: 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*) [95]: UDDI defines a programmatic interface for publishing (*publication API*) and discovering (*inquiry API*) Web services [95]. The core component of UDDI is the *business registry*, an XML repository where businesses advertise services so that other businesses can find them.

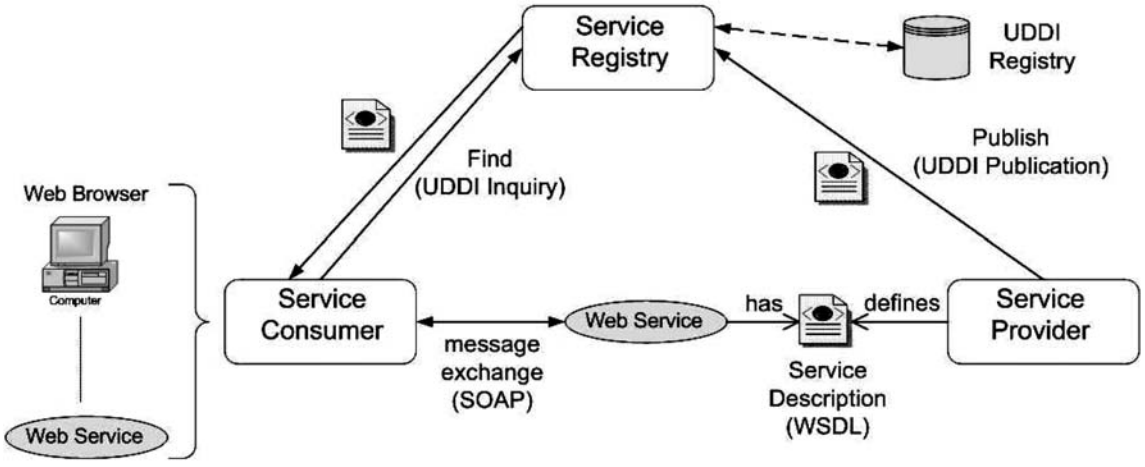

Fig. 7. The Web service reference model

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*) [94]: 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 element, the *header* includes features such as security and transactions. The second element, the *body* includes the actual exchanged data.

4.2.2 Web service composition

One of the most important issues elicited by Web services is the use of the Web as a facilitator for service *outsourcing* [87]. Existing enterprises would combine their core skills and resources to create *composite* services. Simply put, a *composite* service is a conglomeration of outsourced services working in tandem to offer a *value-added* service [60]. For example, `ComputerCompany` may define a composite service that would outsource from pre-existing Web services such as `ProcessorProvider` and `MonitorProvider`.

Efforts are underway to define standards for composing Web services [4]. These include WSFL (*Web Services Flow Language*) [46], XLANG [64], and BPEL4WS (*Business Process Execution Language for Web Services*) [7].

WSFL [46] – WSFL introduces the notions of *flow* and *global model* for defining composite services. The *flow model* specifies the execution sequence between component services. It is represented by a directed graph. Each node of the graph, called *activity*, models a single step of the overall business goal to be achieved through composition. Activities are bound to services through a *locator* element. This binding can be either *static* or *dynamic*. In a *static* binding, the service is directly specified in the locator. In a *dynamic* binding, the locator may, for example, contain a UDDI query that returns a list of candidate services; a service is then selected through a given *selection policy* (e.g., the first service in the list). Two types of edges are used to connect activities: *control links* and *data links*. *Control links* prescribe the order in which activities have to be performed. *Data links* represent the flow of information between activities. The *global model* specifies how component services interact. It includes a set of *plug link* elements. A *plug link* connects an operation of the composite service (called *exported* operation) to an operation of a component service. This indicates that the corresponding interaction must take place to completely implement an activity.

XLANG [64] – XLANG provides language constructs for describing behavioral aspects of Web services and combining those services to build multi-party business processes. At the intra-service level, XLANG extends WSDL language by adding a *behavior* element. A *behavior* defines the list of actions that belong to the service and the order in which these actions must be performed. XLANG defines two types of actions: regular WSDL operations and XLANG-specific actions (e.g., timeout operations). At the inter-service level, XLANG defines a *contract* element which provides means for interconnecting several XLANG service descriptions. The execution order of XLANG actions is defined through *control processes* (e.g., *sequence*, *while*). A particular control process named *context* enables the support of *transactions*. The concept of transaction, as used in databases, guarantees that in case of failure,the partial updates of a service execution are rolled back. XLANG adopts a looser notion of transaction based on compensations. The execution of actions in a *context* may fail or be cancelled for a variety of business and technical reasons (e.g., communication failure). In this case a compensation code, explicitly specified by the provider, is executed.

BPEL4WS [7] – BPEL4WS combines the features of both WSFL (support for graph oriented processes) and XLANG (structural constructs for processes) for defining business processes. A business process is composed of several steps called *activities*. BPEL4WS defines a collection of primitive activities such as *invoke* to invoke a Web service operation. These primitive activities can be combined into more complex primitives using any of the structure activities provided in BPEL4WS. These include the ability to: (1) define an ordered sequence of steps (*sequence*); (2) have branching using the now common "case-statement" approach (*switch*); (3) de-

fine a loop (*while*); (4) execute one of several alternative paths (*pick*); and (5) indicate that a collection of steps should be executed in parallel (*flow*). BPEL4WS provides mechanisms to handle and recover from errors in business processes (*throw* and *catch* constructs). It also adopts the notion of *compensating* actions defined in XLANG. Fault handling and compensating are supported by introducing the notion of a *scope*. A *scope* is the unit of fault handling and/or compensation.

4.2.3 B2B interactions in Web services

Web services allow interactions at the communication layer by using SOAP as a messaging protocol. The adoption of an XML-based messaging over well-established protocols (e.g., HTTP, SMTP, and FTP) enables communication among heterogeneous systems. For example, major existing environments are able to communicate via HTTP and parse XML documents. However, SOAP protocol is still at its early stage; current implementation do not yet meet the reliability and workload constraints needed in enterprises.

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). Recent efforts towards dealing with semantic interoperability include the development of content markup languages such as OIL (*Ontology Inference Layer*) and DAML+OIL (DAML stands for *DARPA Agent Markup Language*) [59]. However, such efforts are still in their infancy.

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. As mentioned in Sect. 4.2.2, standardization efforts 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.

The emergence of tools to describe, advertise, and invoke Web services facilitate 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 (e.g., String and Int). 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.

5 Research prototypes

In this section, we overview a set of representative research prototypes. These prototypes represent the different technologies supporting B2B interactions including components, workflows, and Web services.

5.1 CMI

CMI (*Collaboration Management Infrastructure*) [78,40] 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 establishment 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*.

CMI's trading partners are tightly-coupled. For example, the message format and the communication protocol to be used between partners must be agreed upon before service activity definition. Heterogeneity is addressed through object-oriented proxies which enable access to different information sources such as relational databases, EJBs, and CORBA objects. CMI provides application-specific state machines and operations for modeling services. This allows for the selective monitoring of state changes in external services. In term of adaptability, primitives such as *optional* and *inhibitor*, may be used for coping with unforeseen events. CMI addresses security only at the process model level through a role-based process and activity execution.

5.2 eFlow

eFlow [19] 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.

Composite services are specified through the *service process composer*. Services are enacted by the *service process engine*. This is composed of the *scheduler*, the *event manager*, and the *transaction manager*. The scheduler processes completion notification messages from service nodes and then contacts the *service process broker* to discover the actual services that can fulfill the actions specified in the service node definitions. eFlow provides a default broker. However, users have the option to plug-in the service broker that best fits their needs. The *event manager* monitors event occurrences by detecting temporal data, workflow events, and notifications of application-specific events from external applications. The *transaction manager* enables the execution of portions of a process graph, called *transactional regions*, in an atomic fashion using compensating actions.

5.3 WebBIS

WebBIS (*WebBase of Internet-accessible Services*) [10] proposes a declarative language for composing Web services. Three types of service are introduced in WebBIS: *wrapper services*, *pull-communities*, and *push-communities*. A *wrapper service* is an object that encapsulates the content and capabilities of the underlying application. It uses *translators* to access the operations of proprietary applications. A wrapper service includes a set of *notifications* which describe events that can be sent by a service to its requesters. These events can be used to inform the requester about the service state and situations reached during operation execution. A *pull-community* provides a means to compose a new service from a collection of existing ones. It includes a *components* clause which introduces the list of services that compose the pull-community. Methods of a pull-community are performed by invoking internal operations or methods of component services. A pull-community has also the possibility of subscribing to notifications of the component services. This addresses the issue of external manageability. *Push-communities* cater for the establishment of dynamic relationships among services. They describe the capabilities of a desired service without referring to any actual provider. To be accessible through a push-community, a service needs to register with this community. WebBIS uses ECA (*Event-Condition-Action*) rules as a basis for the declarative specification of the business logic of services. Wrapper services' ECA rules specify constraints on the service properties and methods. They also specify the reaction to requests and responses from translators. Pull (resp. push) communities use ECA rules to invoke operations provided by their components (resp. members) and react to notifications from the underlying services.

WebBIS addresses the issue of adaptability by defining a mechanism to propagate changes . All changes performed to a service are propagated to other services that rely on it to ensure global consistency. Hence, if a component service is deleted, operations or events of pull-communities depending on it are made unavailable. For this purpose, WebBIS defines meta-services called *change notifiers*. Each service has a notifier attached to. The notifier maintains information about the availability of the related service. During its lifespan, a service can be available, temporarily unavailable, or permanently unavailable. A modification of the availability status results in generating an instance of an event that may trigger change operations.

5.4 Other prototypes

WISE (*Workflow-based Internet SErvices*) [77,55] – 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 [57] – 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 [99] – 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.

SELF-SERV (*compoSing wEb accessibLe inFormation and buSiness sERvices*) [9,79] – 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.

XL (*XML Language*) [36,37] – 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.

6 Deployment platforms

Major software vendors (*IBM*, *Microsoft*, *Sun Microsystems*, *HP*, *Oracle*, *BEA systems*, etc.) are currently working on implementing B2B interaction 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.

6.1 Microsoft .NET

.NET [63] 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 of 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*.

6.2 IBM WebSphere

WebSphere [48] 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.

To support interoperability at the business process layer, IBM has just completed acquiring *CrossWorld*, a business process integration product. *WebSphere application server* supports different platforms such as *Windows NT*, *Sun Solaris*, and *IBM mainframes*. It also provides security controls and application access protection. Performance tuning wizards along with log analyzers are added to *WebSphere application server Advanced Edition* for monitoring purpose.

6.3 Sun ONE

Sun ONE (*Sun Open Net Environment*) [83] 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.

6.4 Vitria BusinessWare

Vitria BusinessWare [90] 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.

6.5 Other platforms

Oracle Integration Server [71] – *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 [44] – 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 [6] – *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 [98] – *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 [86] – *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-base 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.

7 Summary discussion

In this section, we discuss and compare the different B2B approaches to interactions on the Web. We identified three sets of parameters that together exhaustively define how B2B E-commerce 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. The third set (applicable to commercial B2B platforms) consists of the following parameters: major modules, communication standards, content and business process standards, and key technologies.

7.1 Evaluation of B2B interaction technologies

In Table 1, key B2B enabling 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+OIL). WSFL, XLANG, and BPEL4WS languages provide support for interactions at the business process layer. However, these languages are still at their early stage.

Table 1. Enabling technologies vs interaction layers

	Communication layer	Content layer	Business process layer
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)
eCO	HTTP	xCBL	Not Addressed
BizTalk	HTTP, SMTP, etc.	External schemas	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
ebXML	Messaging service (HTTP, SMTP, etc.)	Business documents (core components, domain components, and business information objects)	Business process specification schema. Collaboration protocol agreements
Web Services	SOAP	WSDL but little support for semantic description. Use of ontologies for semantic interoperability	WSFL, XLANG, and BPEL4WS

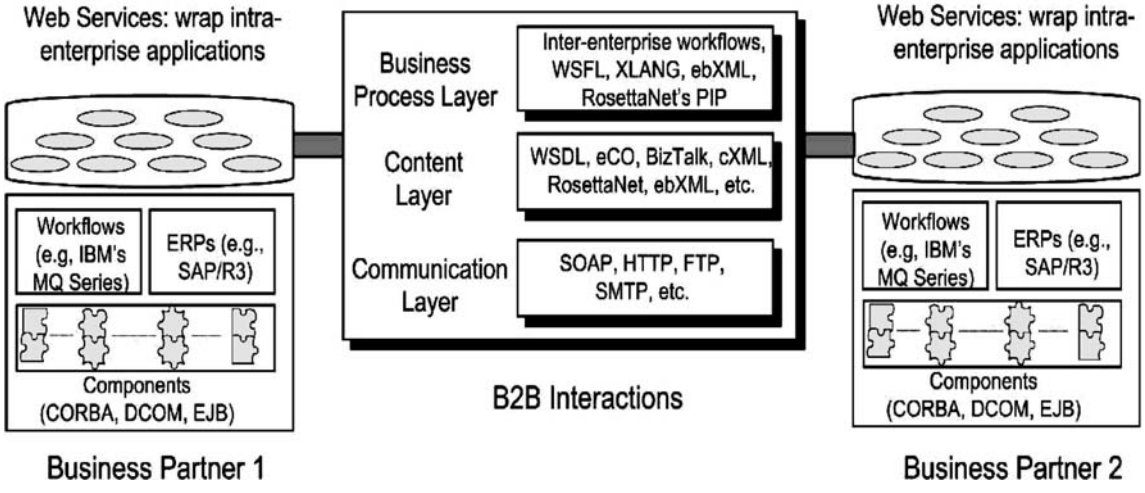

Fig. 8. Technologies in a B2B interactions framework

In Table 2, key B2B enabling technologies 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.

The current technologies for B2B interactions may be viewed as complementary. In Fig. 8, we summarize these technologies and the way they co-exist in a B2B interactions framework. Component middleware, workflows, and Web services define the building blocks for developing B2B applications. EDI, OBI, and XML-based B2B frameworks (eCO, BizTalk, 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. 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. 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. Web services may use the workflow approach (e.g., WSFL and XLANG languages) to support inter-enterprise business processes. Component middleware, workflows, and Web services adopt XML-based B2B frameworks to capture the semantics of documents and business processes. For ex-

Table 2. Enabling technologies vs interaction dimensions

	Type of coupling	Autonomy	Heterogeneity	External manageability	Adaptability	Security	Scalability
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 need 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
Workflows	System-specific	Separation between public and private business processes	Different workflow engines	Adding application-specific states	Process wrappers and adapters help localize changes	Not Addressed	May require creation of new process wrappers and adapters
eCO	Loose, transient, and long term	Separation between service 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
BizTalk	Loose, transient, and long term	Separation between description of services and implementation	External XML Schemas	Specific BizTags (e.g., delivery and commitment receipts)	Impact of local changes limited. Support of extensibility documents	Leverages existing standards	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
RosettaNet	Loose and long term	Separation between service 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
ebXML	Loose and long term	Separation between service description and implementation.	Business service interface to wrap legacy applications	Adding specific activities in shared business processes	Not Addressed	Security protocols (e.g., SSL). Digital signatures.	Distributed set of repositories but to date, only a single repository is specified
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.

ample, 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.

In Table 3, major B2B prototypes are compared using the first set of parameters, i.e., B2B key architectural 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.

Table 3. Prototypes vs interaction layers

	Communication Layer	Content Layer	Business process Layer	Key Features
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	Placeholders for dynamic selection. Application-specific status and awareness events for process monitoring
WISE	Coordination and communication module	Not Addressed	Virtual business processes	Use of exception handling model. Process monitoring and analysis module
Cross Flow	Java RMI	Contract in XML	Contracts	Contract management. Quality of service monitoring.
Mentor-Lite	Transaction Processing (TP) monitor (Tuxedo)	Not Addressed	Business processes expressed as state and activity charts	Partition the overall workflow specification into several sub-workflows.
eFlow	Java RMI	Provides adapters to support different protocols such as OBI and RosettaNet	State machine-based model for process description	Search recipe for dynamic selection. Event nodes: processes can send and receive several types of events
WebBIS	Java RMI	Not Addressed	ECA rules for describing interactions among services	Service communities for dynamic selection. Change notifiers for monitoring and controlling changes.
XL	SOAP	XML Schema	Little or no statements for inter-service business processes	Specification Language based on workflows, dataflow languages, and XQuery expressions
SELF-SERV	SOAP	Not Addressed	State charts	Peer-to-peer model for coordinating the execution of composite services

The same prototypes are compared in Table 4 using the second set of parameters, i.e., B2B key interaction dimensions. For example, eFlow allows loose coupling among B2B 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.

In Table 5, commercial B2B platforms are summarized using the third set of parameters, i.e., 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.

7.2 Open issues

For B2B E-commerce to scale to the Internet, there is a need for efficient integration with all relevant partners, established *a priori* or on demand. The need for interoperability in B2B applications is more pronounced than usual partly because of the way businesses operate, the systems they have, and the difficulties created by systems' autonomy and heterogeneity. Although the current technologies provide the foundation for building B2B integration frameworks, several research issues still need to be addressed. These include *process-based integration of services*, *dependable integration of services*, *support of standardized interactions*, *security*, and *privacy*.

Process-based integration of services – In spite of the potential opportunities, B2B integration solutions are mainly used by large organizations [2,53]. One of the main reasons is that the development of integrated services is still largely hand-coded and requires a considerable effort of low-level programming. Since the components of an integrated service may be heterogeneous, distributed, and autonomous, the integration process may be time consuming. It typically requires the intimate knowledge of the underlying communication protocols, data formats and access interfaces. Additionally, B2B services integration requires flexibility to dynamically adapt to changes that may occur in partners' applications. Businesses must be able to respond rapidly to changes where both operational (e.g., server load) and market (e.g., changes in regulations) environments are not easily predictable. The extension of traditional business process modeling techniques to streamline B2B services integration is a natural step in this direction. Indeed, several standardization efforts for process-based integration of Web services are emerging such as BPEL4WS and ebXML's business process specification schema.

Dynamic and scalable orchestration of services – The number of services to be integrated may be large and continuously changing. Consequently, approaches where the development of an integrated service requires identifying, understanding, and establishing interactions among component services at service-definition time, are inappropriate. Instead, divide-and-conquer approaches should be adopted, whereby services providing similar capabilities (also called *alternative services*) are grouped together. These groups take over some of the responsibilities of service integration . Examples of such responsibilities include dynamic discovery of services based on their availability, characteristics, organizational policies, and

Table 4. Prototypes vs interaction dimensions

	Type of coupling	Autonomy	Heterogeneity	External manageability	Adaptability	Security	Scalability
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
CrossFlow	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 sub-workflows 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
WebBIS	Loose, transient, and long term	Participants do not need to reveal how services are implemented	Service wrappers hide the heterogeneity of underlying applications	Event notifications (ECA rules)	Change notifiers	Not addressed	Participants need to warp their applications. Peer-to-peer approach for managing changes
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

resources that are needed to accomplish the integrated service. Given the highly distributed nature of services, and large number of network nodes that are capable of service execution, we believe that novel mechanisms involving scalable and completely decentralized execution of services will become increasingly important.

Dependable Integration of Services – Transaction support is required to provide reliable and dependable execution of composite services. Traditional transaction management techniques [33] are not appropriate in the context of composite services. The components of a composite service may be heterogeneous and autonomous. They may not be transactional and if they are, their transactional features may not be compatible with each other. In addition, component services, for different reasons (e.g., quality of services), may not be willing to comply with constraints such as resource locking, until the termination of the composite service execution. New transaction techniques are required in the context of Web services. For instance, it is important to extend the description of services by explicitly describing transactional semantics of Web service operations. An example is to specify that an operation can be aborted without effect from a requester's perspective. It is also imperative to extend service composition models to specify transactional semantics of an operation or a group of operations. An example is to specify how to handle the unavailability of a component service. The effective handling of transactional aspects at the composite service level, should be facilitated by exploiting the transactional capabilities of component services. A few industry standards such as *WS-Coordination* [45], *WS-Transaction* [47], and *Business Transaction Protocol (BTP)* [69] are already emerging for transaction support of integrated services.

Security – Security is a critical issue that must be dealt with in B2B E-commerce. Security must be enforced to give businesses the confidence that their transactions are safely handled. A few *de facto* standards are available for transport-level security (e.g., SSL) and message-level security (e.g., SMIME). However, issues such as authentication and authorization still need to be addressed. Businesses generally perform controls over the internal use of their business processes. In B2B E-commerce, there is a need to extend this controlled access to outside companies' boundaries. The concept of access control, traditionally studied in the context of databases, must be thoroughly investigated in the context of B2B applications.

Table 5. Deployment platforms

	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

Research on specifying, validating, and enforcing access control policies for B2B applications is one where intensive work is needed. In particular, access control should be performed at both the database and application levels. In addition, businesses generally have to deal with various and even contradictory access control policies while transacting with their partners.

Privacy – Privacy refers to the restriction of knowledge about various pieces of business transactions to parties involved in the transactions. It is generally (mis)perceived as an issue whose *natural* solution consists of good security mechanisms. Although security and privacy are two tightly interrelated issues, secure B2B frameworks do not necessarily ensure privacy [60,74]. The importance of the privacy problem does not seem to have triggered the right level research efforts. In fact, few techniques and standards have addressed the issue of preserving privacy in Web-based applications. One such standard is W3C's *Platform for Privacy Preferences Project* [93] (P3P). However, P3P enables the specification of the privacy of Web sites but *not* B2B applications. Worse, P3P provides no mechanisms that guarantee that Web sites actually implement their stated privacy policy. A major issue in B2B E-commerce is the ability for businesses to understand each others' privacy policy. There is also a need to provide mechanisms to address how the privacy policy of integrated services is derived from the individual policies of the trading partners.

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

References

1. Aaron R, Decina M, Skillen R (1999) Electronic commerce: enablers and implications. IEEE Comm Mag 37(9):47–52
2. Adam N, Dogramaci O, Gangopadhyay A, Yesha Y (1998) Electronic commerce: technical, business, and legal issues. Prentice-Hall, Englewood cliffs, N.J., USA
3. Adam NR, Yesha Y (1996) Strategic directions in electronic commerce and digital libraries: towards a digital Agora. ACM Comput Surv 28(4):818–835
4. Aissi S, Malu P, Srinivasan K (2002) E-Business process modeling: the next big step. IEEE Comput 35(5):55–62
5. ATIS (2003) EDI Guideline Consistency Subcommittee (EGCS). http://www.atis.org/atis/tcif
6. BEA (2003) WebLogic Integrator. http://www.bea.com/products/weblogic/integrator
7. BEA, IBM, and Microsoft (2003) Business Process Execution Language for Web Services (BPEL4WS). http://xml.coverpages.org/bpel4ws.html
8. Benatallah B, Casati F (eds) (2002) Special Issue on Web Services. Distrib Parallel Databases 12(2)
9. Benatallah B, Dumas M, Sheng M, Ngu AHH (2002) Declarative composition and peer-to-peer provisioning of dynamic Web services. In: ICDE Conference, pp 297–308, San Jose, California, USA
10. Benatallah , Medjahed B, Bouguettaya A, Elmagarmid A, Beard J (2000) Composing and maintaining Web-based virtual enterprises. In: 1st VLDB TES Workshop, pp 71–90, Cairo, Egypt
11. Berners-Lee T, Hendler J, Lassila O (2001) The Semantic Web. Sci Am May:7–15
12. Bichler M, Segev A, Zhao JL (1998) Component-based E-commerce: assessment of current practices and future directions . ACM SIGMOD Rec 27(4):7–14
13. BizTalk (2003) http://www.BizTalk.org

14. Bolcer GA, Kaiser G (1999) SWAP: Leveraging the Web to manage workflow. IEEE Internet Comput 3(1):55–88
15. Bouguettaya A, Benatallah B, Elmagarmid AK (1998) Interconnecting heterogeneous information systems. Kluwer, Boston, Mass., USA
16. Brodie M (2000) The B2B E-commerce revolution: convergence, chaos, and holistic computing. In: Brinkkemper S, Lindencrona E, Solvberg A (eds) Information system engineering: state of the art and research themes. Springer, London, UK
17. Bussler C (2001) B2B Protocol standards and their role in semantic B2B integration engines. Bull Tech Comm Data Eng 24(1):3–11
18. Casati F, Dayal U, Shan MC (2001) E-Business applications for supply chain automation: challenges and solutions. In: ICDE Conference, pp 71–78, Heidelberg, Germany
19. Casati F, Ilnicki S, Jin LJ, Krishnamoorthy V, Shan MC (2000) eFlow: a platform for developing and managing composite e-services. Technical Report HPL-2000-36, HP Laboratoris, Palo Alto, Calif., USA
20. Casati F, Shan MC (2000) Process automation as the foundation for e-business. In: VLDB Conference, pp 688–691, Cairo, Egypt
21. Casati F, Shan MC (2001) Models and languages for describing and discovering e-services (tutorial). In: SIGMOD Conference, p. 626, Santa Barbara, Calif., USA
22. Cobb E (2001) The evolution of distributed component architectures. In: CoopIS Conference, pp 7–21, Trento, Italy
23. cXML (2003) http://www.cxml.org
24. Dadam P, Reichert M (eds) (1999) Proc. Informatik'99 Workshop on Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, October
25. Dayal U, Hsu M, Ladin R (2001) Business process coordination: state of the art, trends, and open issues. In: VLDB Conference, pp 3–11, Rome, Italy
26. Dogac A (1998) A survey of the current state-of-the-art in electronic commerce and research issues in enabling technologies. In: Euro-Med Net 98 Conference, Electronic Commerce Track, Nicosia, Cyprus
27. Dogac A (ed) (1998) Special Issue on Electronic Commerce. ACM SIGMOD Rec 27(4)
28. Dogac A (ed) (1999) Special Issue on Electronic Commerce. Distrib Parallel Databases 7(2)
29. Dogac A, Cingil I (2001) A survey and comparison of business-to-business e-commerce frameworks. ACM SIGecom Exch 2(2):16–27
30. Drala (2003) Drala Event Broker. http://www.dralasoft.com
31. ebXML (2003) http://www.ebxml.org
32. eCO (2003) http://eco.commerce.net
33. Elmagarmid AK (ed) (1992) Database transaction models for advanced applications. Morgan Kaufmann, San Francisco, Calif., USA
34. eMarketer (2002) E-Commerce trade and B2B Exchanges. http://www.emarketer.com
35. Fastwater (2003) http://www.fastwater.com
36. Florescu D, Grünhagen A, Kossmann D (2002) XL: An XML programming language for web service specification and composition. In: WWW Conference, pp 65–76, Honolulu, Hawaii, USA
37. Florescu D, Grünhagen A, Kossmann D, Rost S (2002) XL: Platform for web services. In: SIGMOD Conference, p. 625, Madison, Wis., USA
38. Gartner (2001) The economic downturn is not an excuse to retrench B2B efforts. http://www.gartner.com
39. Georgakopoulos D (ed) (1999) Information technology for virtual enterprises. 9th International Workshop on Research Issues on Data Engineering
40. Georgakopoulos D, Schuster H, Cichocki A, Baker D (1999) Managing process and service fusion in virtual enterprises. Inf Syst 24(6):429–456
41. Geppert A, Tombros D (1998) Event-based distributed workflow execution with EVE. In: Middleware'98 Workshop, pp 427–442, Cumbria, UK
42. Hollinsworth D (1994) The workflow reference model. Brussels, Belgium. TC00-1003. http://www.aiai.ed.ac.uk/WfMC/DOCS/refmodel/rmv1-16.html
43. Hopkins J (2003) Component primer. Comm ACM 43(10):27–30
44. HP (2003) NetAction. http://www.hp.com
45. IBM (2003) Web services coordination. http://www-106.ibm.com/developerworks/library/ws-coor
46. IBM (2003) Web Services Flow Language (WSFL). http://xml.coverpages.org/wsfl.html
47. IBM (2003) Web Services transaction. http://www-106.ibm.com/developer-works/webservices/library/ws-transpec
48. IBM (2003) WebSphere. http://www.ibm.com
49. IETF (2003) http://www.ietf.org
50. IETF (2003) Electronic Data Interchange - Internet Integration (EDIINT). http://www.ietf.org
51. JoinFlow (1998) Workflow management facility, revised submission. OMG Document Number: bom/98-06-07
52. Joshi A, Singh MP (1999) Multiagent systems on the net. Comm ACM 42(3):38–40
53. Kalakota R, Whinston AB (2000) Frontiers of electronic commerce. Addison-Wesley, Reading, Mass., USA
54. Larsen G (2000) Component-based enterprise frameworks. Comm ACM 43(10):24–26
55. Lazcano A, Alonso G, Schuldt H, Schuler C (2000) The WISE approach to electronic commerce. Int J Comput Syst Sci Eng 15(5):343–355
56. Lewandowski SM (1998) Frameworks for component-based client/server computing. ACM Comput Surv 30(1):3–27
57. Ludwig H, Hoffner Y (1999) Contract-based cross-organisational workflows - the Crossflow Project. In: Georgakopoulos D, Prinz W, Wolf AL (eds) Proc. International Joint Conference on Work Activities Coordination and Collaboration (WACC'99)
58. Mackie S (1998) ERP Meets web e-commerce. DBMS Mag 11(8):38–45
59. McIlraith SA, Son TC, Zeng H (2001) Semantic web services. IEEE Intell Syst 16(2):46–53
60. Medjahed B, Rezgui A, Bouguettaya A, Ouzzani M (2003) Infrastructure for e-government web services. IEEE Internet Comput 7(1):58–65
61. Meyer B (1999) On to components. IEEE Comput 32(1):139–140
62. Microsoft (2003) Distributed Component Object Model (DCOM). http://www.microsoft.com
63. Microsoft (2003) .NET. http://www.microsoft.com/net/
64. Microsoft (2003) Web Services for Business Process Design (XLANG). http://xml.coverpages.org/xlang.html
65. Muth P, Wodtke D, Weissenfels J, Dittrich AK, Weikum G (1998) From centralized workflow specification to distributed workflow execution. J Intell Inf Syst 10(2):159–184
66. Nasa (2003) Scientific and Engineering Workstation Procurement (SEWP). http://www.sewp.nasa.gov

67. United Nations (2003) United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport (UN/EDIFACT). http://www.unece.org/trade/untdid/welcome.htm
68. Netscape (2003) Secure Socket Layer (SSL) 3.0 Specification. http://wp.netscape.com/eng/ssl3/
69. OASIS (2003) http://www.oasis-open.org/cover
70. OBI (2003) OpenBuy. http://www.openbuy.org
71. Oracle (2003) Integration server starter pack. http://otn.oracle.com/software
72. Orfali R, Harkey D (1998) Client/server programming With Java and CORBA. Wiley, New York
73. Paepcke A, Chang CK, Garcia-Molina H, Winograd T (1998) Interoperability for digital libraries worldwide. Comm ACM 41(4):33–43
74. Rezgui A, Ouzzani M, Bouguettaya A, Medjahed B (2002) Preserving privacy in web services. In: 4th International ACM Workshop on Web Information and Data Management
75. Roman E, Ambler SW, Jewell T (2001) Mastering Enterprise JavaBeans. Wiley, New York
76. RosettaNet (2003) http://www.rosettanet.org
77. Schuler C, Schuldt H, Alonso G, Schek HJ (1999) Workflows over workflows: practical experiences with the integration of SAP R/3 business workflows in WISE. In: Proc. Informatik'99 Workshop: Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications
78. Schuster H, Baker D, Cichocki A, Georgakopoulos D, Rusinkiewicz M (2000) The collaboration management infrastructure. In: ICDE Conference, pp 677–678, San Diego, Calif., USA
79. Shen M, Benatallah B, Dumas M, Mak EOY (2002) SELF-SERV: A platform for rapid composition of web services in a peer-to-peer environment. In: VLDB Conference, pp 1051–1054, Hong Kong, China
80. Sheth AP, Larson JA (1990) Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput Surv 22(3):183–236
81. Shim SSY, Pendyala VS, Sundaram M, Gao JZ (2000) Business-to-Business E-Commerce frameworks. IEEE Comput 33(10):40–47
82. Sun (2003) Java RMI (Remote Method Invocation). http://java.sun.com/products/jdk/rmi
83. Sun (2003) Sun ONE. http://www.sun.com
84. SWIFT (2003) http://www.swift.com
85. Szyperski C (2002) Component software - beyond object-oriented programming. Addison-Wesley, Reading, Mass., USA
86. TIBCO (2003) ActiveEnterprise. http://www.tibco.com
87. Tsur S, Abiteboul S, Agrawal R, Dayal U, Klein J, Weikum G (2001) Are web services the next revolution in e-commerce? (Panel). In: VLDB Conference, pp 614–617, Rome, Italy
88. Urban SD, Dietrich SW, Saxena A, Sundermier A (2001) Interconnection of distributed components: an overview of current middleware solutions. J Comput Inf Sci Eng 1(1):23–31
89. Vinoski S (2002) Web services interaction models, part 1: current practice. IEEE Internet Comput 6(3):89–91
90. Vitria (2003) BusinessWare. http://www.vitria.com
91. W3C (2003) Document Object Model (DOM). http://www.w3.org/DOM
92. W3C (2003) Extensible Markup Language (XML). http://www.w3.org/XML
93. W3C (2003) The Platform for Privacy Preferences Specification (P3P). http://www.w3.org/TR/P3P
94. W3C (2003) Simple Object Access Protocol (SOAP). http://www.w3.org/TR/soap
95. W3C (2003) Universal Description, Discovery, and Integration (UDDI). http://www.uddi.org
96. W3C (2003) Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl
97. W3C (2003) XML Schema. http://www.w3.org/XML/Schema
98. WebMethods (2003) http://www.webmethods.com
99. Weissenfels J, Gillmann M, Roth O, Shegalov G, Wonner W (2000) The mentor-lite prototype: a light-weight workflow management system. In: ICDE Conference, pp 658–686, San Diego, Calif., USA
100. WfXML (2003) WfXML. http://www.wfmc.org
101. X12 (2003) EDI (Electronic Data Interchange) ANSI X12. http://www.x12.org
102. XML/EDI (2003) http://www.xmledi-group.org
103. Yang J, Papazoglou MP (2000) Interoperation support for electronic business. Comm ACM 43(6):39–47