Introduction To ebXML

In various presentations and reports, analysts such as the Gartner group have described the evolution in e-business as a movement from tactical applications with limited scope towards increasingly strategic e-business initiatives. Up to around 1995, information technology in most organizations was only used to support internal business applications, often at departmental level. Network connectivity in general was limited to being used within the enterprise. From the mid-1990s until the end of that decade, with the arrival of widespread Internet use, the focus broadened to include direct web-based business-to-consumer (B2C) applications.

At the same time, companies increasingly started supporting external electronic commerce transactions with traditional business partners in their value chains. They invested heavily in **Enterprise Resource Planning** (**ERP**) software solutions (from vendors like BAAN, PeopleSoft and SAP) to replace legacy departmental systems. These systems now provide the backbone for most (if not all) of their enterprise applications in use in larger companies, and increasingly include extended facilities for electronic commerce, customer relationship management, and supply chain management.

The challenge for this next century is to open up these enterprise systems, and the business applications used in small and medium-size enterprises, and to integrate them into a multi-enterprise **collaborative commerce** framework, based on interactions between businesses operating within "a single global electronic marketplace where enterprises of any size and in any geographical location can meet and conduct business with each other" (in the words of the ebXML executive white paper, found at http://www.ebxml.org/white_papers/whitepaper.htm).

The **ebXML framework** fits very well in this vision of collaborative commerce and can be viewed as a set of specifications that (again quoting the ebXML white paper) "together enable a modular, yet complete electronic business framework" for collaborative commerce. As ebXML is further developed, adopted in the market, and supported by compliant e-business middleware software, off-the-shelf software for vertical (industry-specific) or horizontal (cross-industry) applications, systems integration services, as well as complementary specifications, it will enable such a global marketplace to be built.

In this chapter we'll provide an introduction to the ebXML framework, both by example and in relation to the more general topic of business-to-business (B2B) integration. This chapter includes the following sections:

- We'll start with a brief description of the project in which the ebXML specifications were developed: its structure, the organizations that were involved, and the status of ebXML-related activities now that the project has been completed.
- Then we'll discuss a typical complete e-business scenario that ebXML supports, and use this scenario to provide a brief introduction to the main components that make up ebXML.
- Next, we'll position ebXML in the context of the state-of-the-art in e-business application architectures, showing how it supports high-level e-business application integration applications and modern e-business architecture "patterns". We'll provide plenty of background to application integration and middleware technology for the benefit of readers who are (relatively) new to the topic.
- The next section will show that ebXML is more than just a state-of-the-art framework of e-business interoperability specifications, but that it provides support for developing lower-cost, more flexible e-business systems that may increase the applicability of ebXML to small and medium-sized organizations.
- The final section provides a summary, some additional scenarios, and a forward reference to some of the other chapters in this book.

The ebXML Project

The ebXML framework, as delivered in the summer of 2001, is the result of a specific project that has been completed; there is no ebXML organization, not even a virtual organization. Instead, ebXML represents a collaborative effort of two organizations, their members and supporters:

- The **United Nations Center for Trade Facilitation and Electronic Business** (**UN/CEFACT**, http://www.uncefact.org/) – a global organization responsible for worldwide policy and technical development in the area of trade facilitation and electronic business for trade facilitation. It is well known for delivering the UN/EDIFACT framework for **Electronic Data Interchange** (**EDI**).
- **OASIS** (http://www.oasis-open.org/) – a not-for-profit, member-based consortium that identifies, builds, and maintains industry-standard specifications for interoperability. OASIS has a strong background in providing a forum for developers and vendors to identify and resolve interoperability issues regarding SGML/XML software products. OASIS is also home to the XML.org portal (http://www.xml.org/), a registry for XML Schemas and XML news source geared toward industry, and to Robin Cover's *Cover Pages*, the authoritative web-based bibliography for SGML/XML related matters (http://xml.coverpages.org/).

Background, purpose, and scope of the ebXML project are described in the invitation letter sent out in 1999 to potential participants in the ebXML project, (which can be viewed by checking out http://www.ebxml.org/documents/199909/ebXML_invitation.htm), and in the ebXML terms of reference document (http://www.ebxml.org/documents/199909/terms_of_reference.htm). These documents identify two main issues that ebXML set out to address as it was initiated:

- At the time, XML was emerging as the technology of choice for the exchange of structured information over the Web, including business transaction related information. As a Web-based technology, XML was felt to offer opportunities for small and medium-size enterprises (SMEs), developing countries, and economies in transition, which had been unable to benefit from the traditional EDI frameworks.
- The downside of the successful adoption of XML was a proliferation of XML-based specifications, many of which are overlapping, thus causing confusion and unnecessary duplication of efforts among users.

In this context, ebXML was intended to offer (to quote from the terms of reference document) an "open technical framework to enable XML to be utilized in a consistent and uniform manner for the exchange of electronic business data in application-to-application, application-to-person and person-to-application environments". In addition to developing a framework for XML-based e-business, the project was chartered to analyze current e-business and data interchange processes and issues, and to provide an assessment of the advantages and disadvantages of utilizing XML.

The ebXML project was organized as an executive committee, a steering committee, and a number of project teams working on specific topics:

- Requirements
- Registry/repository
- Business process
- Core components
- Trading partner agreements
- Quality review
- Technical architecture
- Proof of concept
- Transport, routing and packaging
- Marketing, awareness and education
- Security

Leaving aside the more general project-related topics (such as quality and marketing), these working groups cover specific functional parts of the ebXML framework, or aspects (such as architecture and security) that relate to the framework as a whole. We'll touch upon the results of these groups in this chapter, with further chapters dedicated to them later in the book.

The ebXML project involved over 300 active participants working in the various teams, and over 4000 subscribers to mailing lists. The project had its first meeting in November 1999; this was followed by a series of quarterly meetings. The final meeting took place in May 2001 in Vienna, Austria. At this meeting, the project delivered some 25 documents, all of which are available for download from the project's web site, http://www.ebxml.org/.

At the project completion stage it was clear that, although an impressive amount of work had been delivered, additional work would be needed in all areas. The two founding organizations agreed on a Memorandum of Understanding regarding future work on ebXML. Organizationally, OASIS and UN/CEFACT decided to maintain joint committees for coordination, architecture, and marketing activities, and to divide the technical work areas between them. OASIS Technical Committees have since focused on infrastructure-related activities. These include messaging, protocol collaboration, registry and repository, interoperability, implementation, and conformance. Business content-related work has continued within UN/CEFACT, with business collaborations and core components as ongoing work items.

The ebXML project has managed to obtain endorsements from some of the major e-business initiatives. These endorsements indicate that ebXML has indeed become a credible core e-business framework on top of which other (horizontal or vertical) initiatives can build:

- **RosettaNet** is an e-business standardization initiative in the high tech industry. As an early adopter of XML and in the absence of horizontal frameworks, RosettaNet had to develop its own messaging framework as part of its **RosettaNet Implementation Framework** (**RNIF**). RosettaNet has decided to incorporate the ebXML messaging specification into future versions of RNIF and acknowledges that ebXML offers significant complementary functionality in the areas of registries and trading partner agreements. (You can read the full story by following a link to the press release news page at http://www.rosettanet.org/).
- The **Open Applications Group (OAG)** is the largest XML-based horizontal e-business framework. The OAG has decided to integrate the ebXML specifications into the 182 business transaction standards currently published by the organization (http://www.openapplications.org/news/010730.htm).
- The **Open Travel Alliance (OTA)** is an initiative to develop standards for the travel industry. Applications standardized by the OTA include those in airline and car rental industries, for example for requesting availability or booking reservations, as well as for booking holiday tours. The OTA has endorsed ebXML in the most recent versions of its specifications (http://www.ebxml.org/news/pr_20010801.htm).
- **Covisint** is one of the best-known and successful digital marketplaces in the world. Founded by some of the world's major automotive companies, Covisint supports very large-scale procurement processes in the automotive industry (http://www.covisint.com/). Covisint is currently expanding its services to become an application service provider for supply chain management services. Covisint has announced public support for ebXML (http://www.covisint.com/about/pressroom/pr/ebxml.shtml).
- The **Global Commerce Initiative (GCI)** is an international standardization consortium for companies in the consumer goods industry, supported by the major players in this industry (go to http://www.globalcommerceinitiative.org/ for more information). The GCI announced plans to use ebXML as the backbone for its new data exchange standard for B2B trade in the consumer goods industry (http://www.ebxml.org/news/pr_20000911.htm).

Announcements like these are evidence that ebXML has achieved its objective of helping vertical or horizontal e-business initiatives by providing functionality that is common to all such initiatives, such as messaging, partner agreements, registries and repositories, and business processes. With ebXML this is standard functionality that more specialized initiatives can build upon, and for which there is no longer any need to develop proprietary mechanisms.

Various vendors, many of whom are OASIS members, are currently working on products that support parts of ebXML, and some have released (beta) versions in late 2001. We'll refrain from mentioning any of these, as this is an area where developments are particularly subject to change. The ebXML project site, http://www.ebxml.org/, is dedicated to maintaining an up-to-date listing of ebXML-related software.

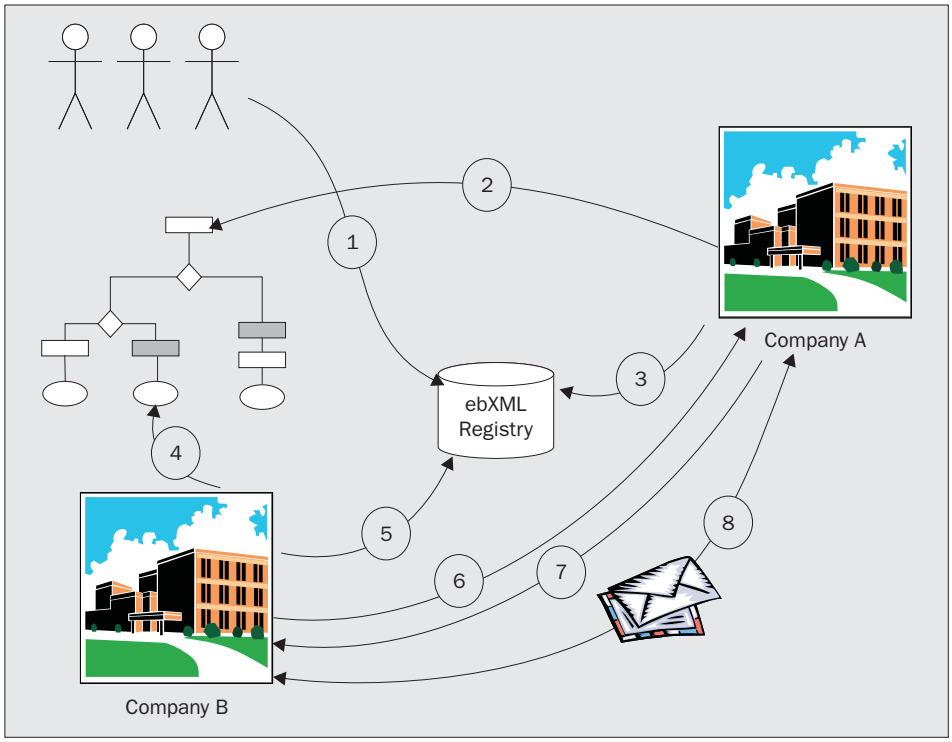
Overview of the ebXML Framework

Now that we've discussed the background to the project that resulted in the ebXML framework, we'll look at the components that make up the framework. We'll achieve this by working through a typical situation that will become one of multiple ways of using ebXML-compliant solutions as the ebXML framework gets more widely adopted. This situation has the virtue of including all components in a single scenario, and has been used in various presentations by the ebXML project team.

After we've presented the scenario we'll summarize the main components of ebXML.

A Sample Scenario for ebXML

Consider the case depicted in the following figure. We will discuss each numbered step in turn:



1. As a first step, a project team will analyze a particular B2B process in order to enable organizations to engage in e-business interactions that implement this process. This project team may be a vertical industry standardization initiative, or a project initiated by one or several business partners. The project team may adopt an e-business methodology, such as the **UN/CEFACT Modeling Methodology** (**UMM**; see Chapter 2), to structure this analysis, but this is not essential.

   In any case, the results of their work will be a formal description of the process encoded according to the **ebXML Business Process Specification Schema** (**BPSS**; see Chapter 5 for more on this). This description will reference certain **roles** that partners may fulfill and will consist of a number of choreographed business transaction **activities**. For instance, in the business of international shipping, there will be a company that acts as an 'Exporter', a 'Freight Forwarder', and 'Customs', and they will perform activities such as 'Make Shipment Reservation', 'Obtain Export License', etc. See Chapter 2 for a more detailed example.

   As part of this work, the project team will look at the types of business information that need to be exchanged in these activities. The **ebXML Core Components** framework (see Chapter 10) may be used for this purpose, or other types of "payload" content can be adopted (see Chapters 11 and 12).

   Once completed, the project team can store this BPSS representation, its documentation, and any core components needed to implement it in an **ebXML-compliant repository**, and make it available through an **ebXML registry** (both of which are covered in Chapter 7).

2. A particular company that wants to expand its electronic business can use the ebXML registry's search functionality to find business processes in which it can perform a particular role. For instance, an insurance company may find the international shipping business process (modeled at Step 1 of this scenario) and determine that it wants to sell its shipping insurance services here. To do so, it should see what kind of ebXML message interactions it needs to support to perform the 'Insurer' role in this process.

   Once this is done, it also needs to make sure it can support this process, which will probably require development of interfaces to its backend systems, or modification of the backend systems themselves. The company can provide such support in various ways, for example using its in-house staff, contracting an external systems integrator, or investigating whether the backend systems vendor or other software company has an off-the-shelf e-business "adaptor" for this process.

3. The insurance company can formally state its capability to fulfill its role in the international shipping business in a document known as a **collaboration protocol profile** (**CPP**; see Chapter 8). In addition to referencing the business process document, this document encodes much more information – such as message payload packing information, and digital signatures.

4. Other companies can browse the registry to find relevant business processes. For example, the diagram might illustrate a company that wants to export and to have its goods insured during transport. This is similar to Step 2.

5. Once this company has found the same business process, it can start a search for business partners that offer complementary services (in this case, insurance). It will then find the CPP registered in Step 3 and find the insurance company that registered it.

6. The exporting company now needs to make sure its B2B integration product supports the ebXML framework and that its back-end system can be integrated with it to fulfill its role. This process will yield information similar to the one encoded by the insurance company in its CPP. The exporter can create a proposal for a trading partner agreement, called a **collaboration protocol agreement** (**CPA**, also discussed in Chapter 8) and propose this to the insurer directly.

   It can also create and register a CPP of its own, but this is not strictly necessary. If it did, you could imagine third party "match-making" agents that would assist in the business partner discovery process.

   A CPA does not cover all aspects the companies may want to agree on, and there will probably be some additional paperwork to be settled. The CPA may reference such a mutually agreed contract.

7. The insurance company can agree with this proposal or reply to it with a counter-proposal. There may be several iterations of Steps 6 and 7.

   Once there is an agreement, companies can configure their systems and make other internal preparations needed to support the business interactions controlled by the CPA.

8. Instances of the business collaboration agreed upon in the CPA can now be started as long as the CPA remains valid. The messages exchanged here and in the other interactions can use the facilities of the **ebXML messaging services** (discussed in Chapter 13.)

This completes the sample scenario. To further whet your appetite, here are some additional scenarios and enhancements:

- Software engineering tools or business modeling tools could support ebXML and offer an integrated development environment, including graphical process modeling tools that support the creation of BPSS, CPP, and CPA documents.
- Other development tools, perhaps integrated with the modeling tools just mentioned or with an application server, could offer code generation based on CPA and BPSS documents.
- Some software products that interact with information servers on the Internet may build on top of the ebXML infrastructure. For example, consider a home banking product distributed by a bank to its customers. This product could use an implicit CPA, which is created when the user fills out a set of forms as part of the installation process and finally presses the "OK" button that both signals agreement with the license terms and conditions of use, and completes the configuration information. This would then cause the software to send an initial message to the bank that results in the service being activated.

Hopefully this book will provide you with enough information and inspiration to start designing and implementing e-business scenarios that are at least as exciting as the ones we could think of at the time we wrote this book, when ebXML was still a very new technology and when the developer community had just started to explore its possibilities.

Overview of ebXML Framework Components

We've already discussed the main components of the ebXML framework, in the scenario section above. They are summarized here for convenience:

- **Business Process Specification Schema**: BPSS is an XML-based specification language that formally defines "public" business processes. It focuses on the collaboration of trading partners, the binary collaborations that these trading partners are engaged in bilaterally (that is, in pairs), and the business transaction activities they perform in the context of those collaborations (see Chapter 5 for discussion). The BPSS is strongly influenced by UMM, a modeling methodology developed by UN/CEFACT (discussed in Chapter 2), but does not require it.
- **Core Components**: these provide the business information that is encoded in business documents that are exchanged between business partners. As components, you should be able to assemble these core components from public or private registries into structures, thus re-using common business structures and eliminating overlap. Core components are tagged with universal identifiers and facilitate multilingual environments. The work on core components did not obtain the formal label of "ebXML specification", and is currently being worked on as part of the follow-on activities (see Chapter 10 for discussion).
- **Registry/Repository**: the ebXML registry/repository deliverables specify a general-purpose repository that is useful for more than merely conducting business searches. Some scenarios, including our example scenario, depend heavily on registries to support setting up business relationships (see Chapter 7 for discussion).
- **Collaboration Protocol Profiles and Agreement**: these are XML documents that encode a party's e-business capabilities or two parties' e-business agreements, respectively. They are closely related to BPSS. With the messaging service, they provide configuration information to generic, high-level ebXML-compliant B2B integration products. With the registry, they support business discovery and the process of setting up new e-business relations (see Chapter 8 for discussion).
- **Transport, Routing, and Packaging**: the ebXML messaging services provide an elegant general-purpose messaging mechanism. It is quite a mature specification that is required implicitly by many other components, for instance to access the registry (see Chapter 13 for discussion). The ebXML messaging service is layered over SOAP with Attachments (see Chapter 4), and can transport arbitrary types of business content.
- **Security**: this is a topic that is pervasive to all the components and is critical for a production e-business system. In this book we've included a dedicated chapter (Chapter 15) on security, which discusses ebXML security issues and related initiatives in W3C and OASIS.
- **Architecture**: the ebXML architecture is discussed in the next section.

e-Business Architecture

As stated in the ebXML white paper, the ebXML framework is evolutionary rather than revolutionary. It represents the state-of-the-art in e-business architecture, addresses the issue of integration at a high level, namely the level of public process interfaces, and supports the public process management application pattern – concepts which you'll find explained in more detail later in this chapter.

In this section we'll provide an overview of e-business integration and some common approaches and technological solutions that will allow you to appreciate the kind of architectural options that ebXML supports. Keep in mind that, in isolation, the ebXML specifications are not a functional description of an e-business integration product. They are *specifications* that enable interoperability of software products (especially at the messaging level) and provide high-level systems configuration information (especially the protocol agreement and business process layers). This means that developers can use a variety of middleware products to implement an ebXML-compliant system.

We'll address the following topics in this section:

- Types of integration: enterprise integration and B2B integration.
- Integration and a multi-tier application model.
- Integration middleware technology.
- E-business integration patterns.

While the first three of these subsections contain preparatory material that provides no original or innovative content, they introduce concepts that you'll find useful, when attempting to understand ebXML and e-business integration at a high level. The fourth subsection is also important, as we'll use it to position ebXML in relation to other common approaches to B2B interaction, including the Web Services integration model.

Types of Integration

Application integration is concerned with the issue of connecting two or more applications that were not initially designed to be connected, in order to have them interoperate and/or share data. Application integration in itself is not a new problem. In fact, it predates the Internet and even client/server architecture; systems integration accounts for a significant percentage of the IT budget in many companies.

When discussing application integration, it is useful to distinguish two major classes of application integration:

- **Enterprise Application Integration (EAI)** is concerned with integrating applications within an enterprise. It is concerned with "internal" processes that are typically not visible to the outside world and the software applications used to implement them.
- **Business-to-business Integration (B2Bi)** is concerned with the application integration to automate interactions of "external" business processes across enterprises. B2Bi is sometimes referred to as the "extended enterprise" model.

In practice, the differences between these two types of integration are not always as clear-cut as they are presented here, but the distinction is nevertheless important and useful to make. Some technologies or approaches that work well in one area are less appropriate for, or perhaps just less commonly used in, the other area. Typically, B2Bi is more high-level than EAI, although EAI products are increasingly incorporating B2Bi functionality. You are most likely to encounter ebXML in B2Bi projects, or in large-scale EAI projects in large companies.

You will often need to address both B2Bi and EAI in a single project, because the external processes that B2Bi is concerned with, viewed in isolation, do not constitute an end-to-end business system but need to be associated with back-end systems and processes. Another way of thinking about this is to view external interactions between two parties as process synchronization signals for their internal processes, where the overall business state is maintained by the backend applications at the two sides. This is an idea that is developed further in Chapter 6.

You can further classify and compare the various approaches to application integration (whether EAI or B2Bi), as well as the software that implements those approaches, along a number of dimensions. Two common dimensions are:

- The application **tiers** at which the integration is achieved.
- The type of **middleware** technology that is used.

A third dimension is to consider common e-business application architectural **patterns** for solving business-to-business integration issues. These B2Bi e-business patterns are more easily understood in reference to the first two classifications, which is why we'll look at them first. The discussion in this chapter is partly based on overviews of application integration – examples include *Building B2B Applications with XML*, by M. Fitzgerald (Wiley, 2001, ISBN 0-471404-01-2), and *B2B Application Integration. e-Business-Enable your Enterprise*, by D. Linthicum (Addison-Wesley, 2001, ISBN 0-201709-36-8).

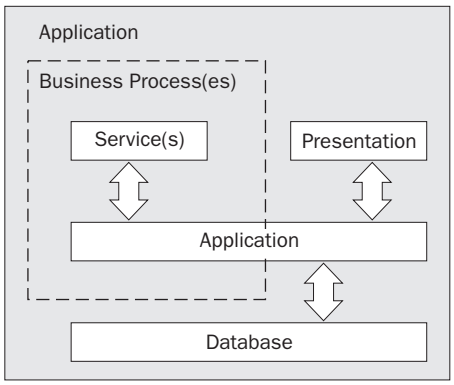
Application Tiers

Applications can be integrated at multiple levels, and the choice depends on the situation. The classic three-tier client/server model is a common reference model for application layers. For our purposes, we've extended it with a fourth, business service/process level, following recent proposals (for example, see http://eai.ebizq.net/bpm/white_1.html), as it is particularly useful in the B2Bi context. These tiers are:

- The **database** tier deals with data integrity and consistency, maintained using mechanisms such as SQL triggers, stored procedures, keys, and data types.
- The **application** tier is where the core program business logic is encoded.
- The **presentation** tier is used to separate the (graphical) user interface from the application code.
- The **service** or **business process** tier is concerned with higher-level, business process-oriented interfaces. Business processes may be composite entities, involving multiple service invocations, organized in orchestration flow models.

The fourth layer is the non-standard tier in this system, and it is an alternative or replacement for the presentation tier, rather than a higher-level tier. You can think of it as a control layer for the application layer, as a kind of meta application layer, or (traditionally) as design documentation only. We'll talk about business process management systems that can be configured using executable process models later in this chapter, as well as in Chapters 5 and 6.

The following diagram visualizes these layers and their relations. The diagram shows how access to the database layer from both the presentation layer and the business service/process layer passes through the application layer. We've displayed presentation and services as alternative layers that access application functionality. The presentation layer provides this access (synchronously) to humans. The service and process layers provide this access (quite often asynchronously) to external applications. We draw the business process interface as a box to show that it can encapsulate multiple services and application interfaces.



This diagram opens up many issues and questions (such as the relation between business service interfaces and business process interfaces) that we will get into later in this chapter, when we discuss e-business patterns. One thing to note here is that in this diagram we've assumed (for expository reasons) that there is only one application and that (business) service and business process interactions are part of this application. We'll later refine this, and view business services and business process interfaces as a layer that can actually provide access to multiple applications and may have its own mechanisms to maintain state information.

In an e-business context, distinguishing a business service and process layer makes it easier to define processes that involve multiple business partners, as in a B2B supply chain. It is often a first step for organizations that are planning to outsource part of their business; the IT systems that manage that part of the business would then move out of the outsourcer's control but could still be interfaced with their IT systems.

When you look at two arbitrary applications, you can in principle use any of these four application tiers as starting points for integration. In fact, the number of possible combinations is sixteen rather than four, because the integration can attach to the two applications at different levels. For instance, if your only access to an application is by running a Perl script to extract information from the HTML generated by its web interface, you are accessing that application's *presentation* layer from your system's *application* layer. Similarly, in EAI applications you sometimes have to bypass an application layer and access the data layer directly. In practice, only a few of these combinations make sense, and we'll concentrate on integration at a single level.

At the *database* layer, one application could directly manipulate information managed by another application's database layer or obtain information via calls to an intermediate data access layer. The application layer and higher levels of the receiving application don't even need to be aware of the fact that part of its information is obtained from an external source. In general, this approach is limited to EAI scenarios where you cannot (easily) modify the application layer, but are able to bypass it to (safely, hopefully) access the database layer directly. You'll rarely encounter this scenario in a B2B context.

Access at the *application* layer is the preferred type of application integration within the enterprise. You can integrate with an application in a variety of ways, for instance by calling its API or by using a batch file format interface. It's easier to integrate with larger applications if you break them down into components that each perform more specialized functionality, and use a component framework to build new, integrated enterprise applications using these components. System integrators often use the term "legacy wrapping" to refer to the process of encapsulating the functionality of existing enterprise systems (some of which may represent millions of lines of COBOL code and hundreds of programmer-years of development) within a component to open up its functionality to develop the new applications.

Larger enterprises typically not only have many large custom-developed applications, they also use package-based applications, developed using ERP and database software packages. While offering similar functionality (for instance, enterprise resource planning), the products in a particular category (such as SAP, Baan, or Peoplesoft) are often based on radically different approaches. The need to integrate package-based applications has created an entire category of integration software called EAI **adaptors** (or **connectors**) that offer standard access components for the various products, and for the various releases or versions of those products.

Enterprise Information Portals are sometimes mentioned as instances of *presentation*-level integration, although strictly speaking they don't provide information to external applications but rather to externally-based users. Portals occasionally use presentation-level access. This happens when they need to obtain their information by retrieving and transforming applications via their web interface, or when "screen scraping" is needed to get information out of terminal-based legacy mainframe applications.

In a B2B context, the number of enterprise applications involved is multiplied by the number of partners involved, and rapidly becomes unmanageable. There are also other complicating issues, both technical and business-related. Companies may not allow direct access to enterprise applications for security reasons and may be committed to different, incompatible middleware (perhaps even to different EAI middleware!), which none of them can enforce upon their business partners. There may also be business-level (procedural) mismatches between the internal business processes of various companies.

In the discussion of patterns, we'll see that it is often useful to make a distinction between exposing the "raw" *application* interface and exposing a *service* interface, where a service is a lightweight business-to-business interface component that can provide a higher-level interface. It may even interface intelligently with multiple applications. The service interfaces are still defined by the exposing organization, and will typically be limited to synchronous interactions.

The very reason that companies become interested in collaborative commerce also makes any tight coupling of applications undesirable. Companies outsource activities to reduce their dependence on a single (formerly internal) supplier of those services, so the IT systems integration needs to be flexible to support other partner systems quickly. In this case, the approach to take is to integrate at the *business process* level. This means that companies need to define high-level, generic interfaces that expose their e-business capabilities. These business process interfaces will have a many-to-many relationship with back-end enterprise systems. Preferably, you will organize those external interfaces according to industry-standard business interface models to reduce the cost of development and to increase the number of potential partners that this allows you to do e-business with.

Integration Middleware

We've seen two ways of looking at integration solutions so far: one based on whether the integration is within the enterprise or between connecting businesses, and the other based on the application level at which integration is addressed. A third way of classifying approaches to integration is to look at the specific integration middleware technology used in a solution. In this section we'll provide a high-level summary of the main features of, and requirements for, integration middleware, and briefly review some of the more common types of middleware. We'll adopt a broad definition of "technology" here, which includes product categories, programming interfaces, and communication protocols. Clearly, a detailed discussion of middleware frameworks that can be used with ebXML would constitute a book in itself. The purpose of this overview is to introduce some of the technologies you're likely to encounter in a practical ebXML project, and to provide a reference for the discussion of e-business integration patterns.

Features and Requirements

The middleware we'll discuss in this section serves as the run-time framework that allows applications to exchange information with each other, and to jointly implement an (intra-enterprise) internal or (inter-enterprise) external business process. Minimally, it serves as the transport mechanism that moves the data to be processed between the applications or components. In practice, middleware systems also provide some or all of the following features:

- Supporting distribution of processing over multiple computer nodes, hiding the complexity of various network protocols and operating systems from the developer.
- Providing availability; preventing loss of data and allowing systems to function in spite of network, hardware, or software failures ("failover").
- System administration and management.
- Tracking and tracing.
- Data representation, translation, and transformation.
- Intelligent routing.
- High-level, graphical development tools.
- Adaptors (or connectors) for commonly used applications packages.
- Security features such as encryption and digital signatures.

Component Frameworks

As we mentioned, component frameworks are important enablers for application integration. You can design (or re-design) an application as a collection of collaborating components to facilitate the construction of new applications and integration with other applications. You can "wrap" a legacy system in a component to hide its complexity from other developers who need to access its services. A component should be modular, hide its internal structure and implementation, and present itself to the outside world using a well-defined interface. In many frameworks, that interface is programming language independent.

A component middleware framework allows components to access each other's functionality by offering a communication bus through which the components can communicate. A distributed framework makes accessing a component that runs on a completely different platform as easy as connecting to a component running on the same computer, and hides this complexity from the developer. Three widely-supported component middleware frameworks are:

- Microsoft **COM+** and the (upcoming) **.NET Framework**.
- **Java 2 Platform, Enterprise Edition** (**J2EE**).
- The Object Management Group's **Common Object Request Broker Architecture** (OMG **CORBA**). The **Internet Inter-ORB Protocol** (**IIOP**) interface is the CORBA standard interface to connect component buses over the Internet. CORBA is widely used in some industries, particularly in the telecom industry.

Component frameworks have been adopted in many enterprises as the basis for new applications and to open up functionality in legacy applications. In the mid-1990s, component frameworks also received much attention as a potential future implementation platform for business-to-business integration. Various initiatives were exploring the possibilities they offered in this space, including the following:

- The Open Applications Group (OAG, http://www.openapplications.org/) was formed in 1995 to address the interoperability of business software, with some of the major ERP vendors as founders.
- The United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT), aside from being responsible for maintaining and extending the UN/EDIFACT framework, was carrying out work on an object-oriented EDI successor to the EDIFACT framework, to operate over component frameworks like IIOP.

The OAG officially adopted XML as a business message notation in 1997, finding it a better fit for the loosely coupled, asynchronous business interaction model it felt to be appropriate for business-to-business e-business. Part of UN/CEFACT's work on object-oriented models for e-business is still visible in the UN/CEFACT modeling methodology (UMM) that we'll discuss in Chapter 2. Like OAG and other organizations, UN/CEFACT gradually adopted XML as the business message format of choice and, in late 1999, became one of the founding organizations of the ebXML project.

It is commonly felt that the component frameworks we've discussed are best positioned within the enterprise, rather than as a business-to-business integration technology. This is partly due to the complexity of these frameworks, the fact that you normally can't deploy them across firewalls, and the fact that there isn't a single component framework, as each of the three main frameworks is widely supported. Recently, this problem has been addressed by defining an interoperability layer that effectively hides these frameworks under an XML interface. This is the field of Web Services, with **SOAP** (**Simple Object Access Interface**) as the most important standard. SOAP is discussed in detail in Chapter 4 of this book.

Message Queuing

Message queuing has been used to integrate applications in production systems for over a decade. It is based on an idea that is both simple and powerful. Instead of invoking a service by performing a synchronous (remote) procedure call, the sending application asynchronously puts a request in a message queue. The processing application reads requests from that queue and processes them. This may occur a few milliseconds after the request is put on the queue, or hours later, for example if the receiving application is a legacy mainframe application that is scheduled to do a batch run overnight. Once processing has completed, the processing application puts a reply in the receive queue of the sending application, again using asynchronous communication.

Message queuing products support distributed processing by using a store-and-forward mechanism to transport messages from one machine to another. This process is very robust; if there are temporary network or software failures, the queuing software can just wait until the connection is re-established. It is also scalable, as it doesn't overload applications at peak load time. The messaging framework can also take care of multiple protocol support, perform encryption and authentication, and may offer functionality such as load balancing or priority-based handling. The downside of using message queuing is that existing applications need to be modified if they assume a synchronous (remote) procedure call style of interaction.

The **Java Messaging Service** (**JMS**) API of the Java 2 platform is a product-independent API that allows developers to use message queuing in their applications without being tied to a particular implementation. Many vendors of message queuing products support JMS. This subject is covered further in Chapter 14.

Message queuing is the prevailing middleware used to implement application level EAI. This is because of the need to access batch applications distributed across multiple computers, its support for reliable, asynchronous messaging, and because you may have to support a variety of hardware platforms, network protocols, and operating systems to interact with enterprise systems. If the message queuing product supports raw TCP/IP or Internet protocols like HTTP and especially SMTP, you can also use it for business-to-business integration. However, this often requires software from the same vendor both at the sending and receiving ends.

EAI Platforms

Several vendors have developed specialized solutions for EAI. These products are typically built on top of some of the other middleware solutions we've mentioned so far, message queuing software in particular, and provide numerous adapters to common applications and database systems. They have built-in management tools and development tools.

You can use such an EAI platform as a bridge between an application server or B2Bi server, and packaged or legacy applications.

Application Servers

Application servers have emerged as the integration and development tools of choice for web-based transactional systems. They provide a standard middle-tier environment for application developers and offer efficient support for scalability and high availability. Most application servers support the Java 2 platform for enterprise computing, which means support for various APIs, including:

- **Enterprise JavaBeans (EJB)**: a generic component model that provides a standard framework for transaction management, security, and other issues such as persistence and resource pooling, thus allowing the programmer to focus purely on implementing business logic.
- **Java Interface Definition Language (IDL)**, a standard interface between Java programs and distributed services and objects built using the CORBA component model.
- **RMI (Remote Method Invocation)**, an approach to distributed computing based on remote invocation of Java methods.
- **Java Database Connectivity (JDBC)**, a vendor-independent API for accessing relational database systems.
- **Java Servlets**, a mechanism for web client interaction control and page construction.
- **Java Naming and Directory Interface (JNDI)**, an API for accessing naming and directory services.
- **Java Messaging Service (JMS)**, the generic API for invoking message services, including queue-based messaging.
- **Java Connector Architecture (JCA)**, a work-in-progress standard API for developing connectors to legacy enterprise applications.

An application server can access enterprise systems using custom code components, components developed using JCA, or via a specialized EAI infrastructure product.

The most common way of integrating two component-based applications using an application server results in a tightly coupled solution, requiring synchronous application invocation. As a result, application servers are more suited for integration within a company, rather than for business-to-business integration. In a business-to-business context, you can use the application server's facilities to interface with web servers to manage the message traffic with external business partners, and to provide scalability for the application components that implement the e-business interaction.

Electronic Data Interchange (EDI)

Electronic Data Interchange (**EDI**) is a technology that has supported a large-scale business-to-business e-business infrastructure for many years. EDI predates XML and even the Web. EDI is successfully used in production systems that support large-scale e-business, both in terms of transaction volumes and in economic value. Nevertheless, EDI has not managed to extend its reach outside a limited number of application areas; in particular, it hasn't extended its reach to **small and medium-size enterprises** (**SMEs**). This is commonly attributed to a number of factors:

- The EDI message syntax is cryptic and hard to understand.
- EDI relies on expensive private **value added networks** (**VANs**).
- Software and services for EDI are very expensive.
- EDI is tied to batch applications.
- The EDI world is fragmented in multiple communities.

EDI uses its own syntax to encode messages. For illustration, the following example is a piece of EDIFACT (copied from M. Bryan's article *Using XML for Electronic Data Interchange: ISIS European XML/EDI Pilot Project Deliverable D9*, found at http://palvelut.tieke.fi/edi/isis-xmledi/deliver/d9.doc) that encodes an `ORDERS` message:

```
UNH+1857+ORDERS:D:99A:UN:FI0084'BGM+220+1999B2734:9'DTM+137:19991105:102'
RFF+CT:652744'NAD+BY+5012345678900::9'NAD+SU+6012345678900::9'
NAD+CA+7012345678900::9'NAD+CZ+7012345678950::9'
NAD+CN+++THE VILLAGE STORE+2 THE REDDINGS:CHELTENHAM+GLOS++GL51 2UP'
LIN++1+37534656:EN'IMD+F+8+:::SUPER PARTY POPPERS'QTY+21:100'
DTM+2:1999121:102'UNT+13+1857''
```

The EDI message format uses short alphanumeric codes and has various ways to omit unused message sections to reduce the overall message size. You need external code tables and knowledge of the message structure to interpret the message. Message size was certainly an important issue at the time EDI was designed, as bandwidth was expensive. This issue is less relevant nowadays because Internet bandwidth is much less expensive, and because compression technology can reduce message size at transport (rather than application) level.

The EDI syntax is undeniably terse, and already SGML, the ancestor of XML, offered advantages over the EDI format because of support for validation using standard parsers, availability of more and better conversion tools to translate EDI messages in SGML format to other formats, and SGML's superior support for character sets – see *Practical SGML*, by E. van Herwijnen (Kluwer, 1994, ISBN 0-792394-34-8), Chapter 21. This argument has only gained in strength now that XML has become the ubiquitous Internet content encoding language, and now that software facilities for XML, such as DOM or XSLT, are widespread.

XML documents, in a sense, include their own meta data, in the form of element and attribute names that label information. This allows you to find information in a document using the meta data, rather than using positional information, which offers increased flexibility to extend the XML document with additional information without breaking existing applications. Various people in the EDI community have acknowledged these advantages and agree that, with XML, the EDI syntax has become obsolete.

Perhaps most significantly of all, EDI suffers from being unfashionable, from being associated with an expensive pre-Internet infrastructure, and from being viewed as fax or paper document replacement rather than as a business application integration technology. The main (and often underestimated) value of EDI for the broader e-business developer community is represented by the standard directories of message definitions, which incorporate a tremendous amount of business knowledge and are based on detailed analysis of inter-enterprise business processes. We'll elaborate on this in Chapter 12.

EDI syntax obsolescence is probably more important than its reliance on VANs, the cost of EDI software, or its traditional batch orientation. This is because some EDI software vendors actually support the Internet as an alternative transport channel; sometimes VANs are preferred, for example because of security or because changing the interface to use the Internet would cost more than it would save. The EDI format, while less easy to process than XML, is not exceedingly complex, and there is even a free `XML::Edifact` Perl module that facilitates EDI message processing (see http://www.xml-edifact.org/). EDI is not by nature restricted to batch applications, and there are "interactive" EDIFACT message sets designed for use in, for instance, travel reservation systems. Finally, the XML vocabulary space is much more fragmented than the EDI world, however, this hasn't stopped it from becoming the open Internet data format of choice.

As argued by the ISO **Open-edi** project (http://www.iso.ch/cate/d25154.html), which we'll discuss later in this chapter, the main reason why EDI has proved to be too expensive may well be a different one from those we've discussed so far, namely the absence of support for "common business scenarios" that causes the process of setting up partner agreements to be too expensive. This issue is really independent of interchange format syntax and needs other solutions, which we'll come back to later in this chapter.

XML Vocabularies

There is every reason to consider interoperability specifications as an e-business integration technology, like the other technologies we've grouped under this heading. XML-based initiatives and standards, often referred to as "vocabularies", have grown considerably, both in importance and in diversity, and are relevant at various architectural levels. Often, messages combine elements from multiple vocabularies and use XML Namespaces to manage this. There are some common ways to classify XML vocabularies as relevant to e-business, and we'll mention a few of these here.

Some XML vocabularies are not used (directly) to encode e-business messages, but serve to configure an e-business system or otherwise specify its behavior. In ebXML, the XML-based specification languages for business collaborations (BPSS) and trading protocols (CPP/CPA) fall into this category.

When looking at e-business as an exchange of business messages, it is common to distinguish messaging *framework*-related content from *payload*-related content, typically transported in a message body or attachment. (See Chapter 11 for more discussion on the relationship between these two).

Framework-related content determines the message "envelope" and handles lower-level information typically included in headers – such as identification of sender and intended recipient, what request the message is a response to (if the message is a response message), references to external protocol agreements, timing, routing or priority information, etc. Various e-business frameworks (including ebXML, RosettaNet, and BizTalk) actually use the MIME general-purpose format to provide an ability to transport both XML and non-XML content. Frameworks are application-neutral and can be used both across and within a particular industry.

Security-related standards for issues like digital signing, encryption, and authorization are an important class of supporting vocabularies. Security can sometimes be addressed at multiple protocol levels, and may affect either the framework-dependent message structure or the payload. The payload-related content reflects the diversity of the e-business application space. A coarse distinction is used to separate functions and verticals, as seen in *Even More Extensible; An Updated Survey of XML Business Vocabularies* by A. Kotok (http://www.xml.com/pub/a/2000/08/02/ebiz/extensible.html). These are defined as follows:

- **Functions**: guidelines for specific business operations that cut across industry boundaries.
- **Verticals**: messages for exchanges within a specific industry.

The RosettaNet consortium (http://www.rosettanet.org) makes a more refined distinction between standards concerned with universal business processes, standards that are concerned with specific business models, and standards concerned with specific supply chains. They also identify the need to standardize dictionary structure and content, at both the "universal" and supply chain levels.

In this book we'll discuss the XML vocabularies that are defined by the ebXML framework in the relevant chapters. With respect to payload, ebXML is designed to be independent of a particular payload format, and is even compatible with non-XML formats, including EDI and multimedia payloads. We'll discuss EDI and XML-based payload standards in Chapter 11.

XML has traditionally been more important for B2Bi than for EAI, as the need for open standards is more pressing in communication *between* businesses than *within* an enterprise. However, XML is increasingly being adopted in EAI projects too, and many adopters are offering XML interfaces to enterprise systems. Enterprise software products are often adopting direct XML interfaces, allowing them to be plugged into B2Bi products without the need for intermediate EAI adapter technology. These developments further narrow the gap between EAI and B2Bi.

B2B Integration Products

A specialized category of products has emerged that is designed to support development of B2B e-commerce applications. A B2Bi product is responsible for managing the e-business interactions of an enterprise and its external partners. The precise capabilities of a B2Bi product vary depending on the specific integration pattern the product implements, but will include some of the following features:

- Partner management, including management of bilateral (technical or business) agreements and support for the process of setting up such agreements.
- Management of external business processes/collaborations.
- Encryption, authentication, authorization, and non-repudiation.
- Interfaces with external and internal registries and repositories.
- Support for Internet communication protocols.
- Support for XML and XML-related standards, such as DOM, XSLT, and SOAP.
- Support for non-Internet communication channels like VANs and fax.
- Support for common business-to-business protocols such as EDI and XML-based frameworks such as RosettaNet.

- Data translation from external (possibly partner-specific) formats to internal formats, XML-based or other.
- Integration with back-end enterprise systems.

A J2EE-compliant B2B integration product can use the facilities of an application server for scalability and failover. A B2Bi product can use a specialized EAI product to integrate with enterprise systems, or use other components or interfaces to integrate with internal processes. It can access a workflow management system to interface with non-automated internal processes that require human intervention.

As we've outlined in this chapter (and will elaborate in the remainder of this book), ebXML is a framework of interoperability specifications that standardizes some of these functional requirements. This means that different implementations of those specifications in ebXML-compliant B2Bi products will interoperate. Currently, some B2Bi products require that all partners engaged in an e-business collaboration managed by the product use the run-time version of that product. As a result of ebXML and related standards, companies will be less tied to the proprietary features of a product and will be able to migrate more easily from one implementation to a competing product that supports the same ebXML interfaces. Furthermore, members in a trading community do not need to standardize on a particular product to engage in collaborative e-business.

e-Business Integration Patterns

So far, we've looked at three ways of classifying application integration scenarios, being focused on: enterprise level integration or integration of multiple enterprises; the level at which integration is applied in a three-plus-one tier model of application logic; and the specific middleware technology used to perform the integration. Theoretically, this yields a three-dimensional classification matrix with an enormous variety of potential approaches to integration. If you were to fill this matrix with references to actually-deployed solutions, you would find this to be a rather sparse matrix. This is because some combinations are much more common than others, typically because they represent common solutions to common problems that have proven themselves repeatedly in practical projects.

Such common approaches to common problems, and the application architecture and technology selection that go with them, are often called **patterns**. Patterns also reflect progress in the e-business industry, either because new middleware technology becomes available that makes more advanced approaches possible or affordable, or because specifications are developed that improve their interoperability, or simply because of "best practices": lessons learned, often the hard way, by software developers and systems integrators developing e-business solutions for their users or customers.

The ebXML project is a framework of interoperability specifications for e-business, and it is very instructive to position the ebXML specifications in a reference framework for e-business patterns. In this section we'll review a leading overview of e-business patterns, developed by IBM Corporation. IBM's patterns for e-business are documented at the IBM DeveloperWorks web site (http://www.ibm.com/developerworks/patterns/). We'll focus on the subset of business-to-business integration patterns, referred to as "extended enterprise patterns" in their classification. After this, we'll position the ebXML specifications in relation to these patterns.

In their overview, the authors of the extended enterprise site distinguish five application patterns:

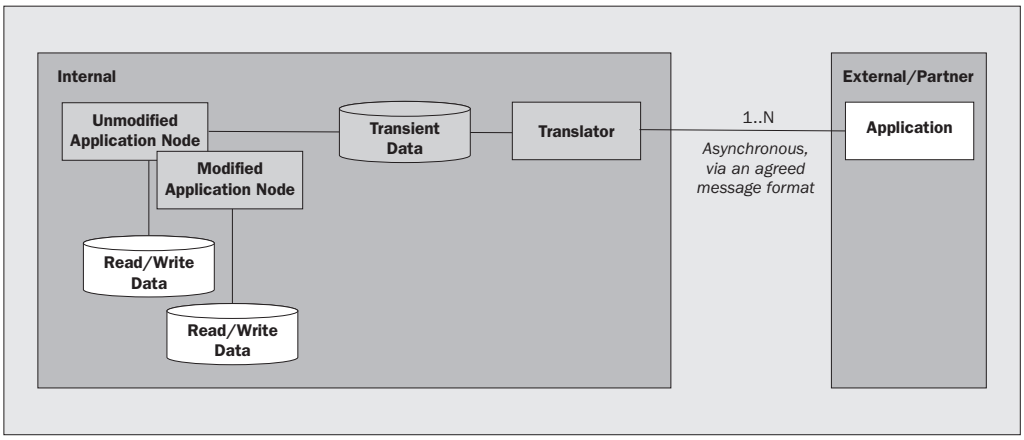
- The **document exchange** application pattern.
- The **exposed applications** application pattern.

- The **exposed business services** application pattern.
- The **managed public processes** application.
- The **managed public and private processes** application.

We'll look at each of these five application patterns. We've simplified the pattern diagrams here; for more detail please refer to http://www-106.ibm.com/developerworks/patterns/b2bi/select-application-topology.html.

The Document Exchange Pattern

The document exchange pattern is suited for business partners who replace paper document-based communication by electronic batched data interchange. The overall architecture is displayed in the following diagram:

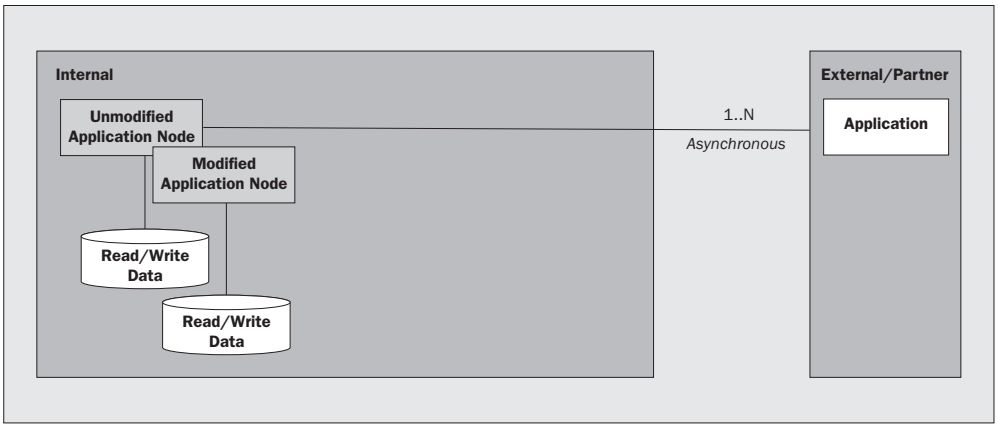


This diagram essentially describes classic EDI implementation scenarios. The parties that exchange information need to have detailed agreements on document format and the communication channel used to transport those messages, typically a particular VAN. In practice, such agreements are often biased towards the requirements of the dominant partner, such as a large manufacturer or retailer who can impose these agreements on its smaller suppliers. In such a situation, the internal, private business applications serve to support specific, fixed public processes. The integration of the various layers in this model reflects this. The EDI translator is directly integrated in the enterprise application and maps the internal data structures of that application to EDI message formats.

This setup has several drawbacks that limit its suitability for more general e-business interactions. The direct association of external and internal processes reduces the overall flexibility of the architecture, as internal applications may need to be changed if external business processes change, and vice versa. The document exchange paradigm is also typically limited to batch applications.

The Exposed Applications Application Pattern

The exposed application pattern is one where an application's application tier is exposed directly to the outside world, rather than via an intermediate presentation layer. The integration can use either message queuing or a component framework to communicate with partners, depending on the need for such communication to be asynchronous or synchronous. Typically, asynchronous communication is preferred to manage application load and to make the business-to-business connection less vulnerable to network or system failures. Usually, this also means that all partners need to standardize on a common middleware platform. For network connectivity you can use the public Internet instead of private networks, with security addressed by installing a **virtual private network** (**VPN**).



Due to the direct association of external processes and internal applications, this architecture is quite inflexible. If you directly access a partner application, you will need to accommodate any changes to that application in your own application, depending on how well the partner API hides its internal complexity from you. More complex interactions, which may involve multiple applications, essentially require you to familiarize yourself in detail with the application infrastructure of all partners you integrate with. This quickly becomes unmanageable and very expensive. If this is the case, direct application integration is actually far worse than using EDI, as EDI message sets are at least defined to express interactions at business process level, a process that emphasizes the commonality of business-level interactions over the uncontrollable variation of arbitrary applications.

The Exposed Business Services Application Pattern

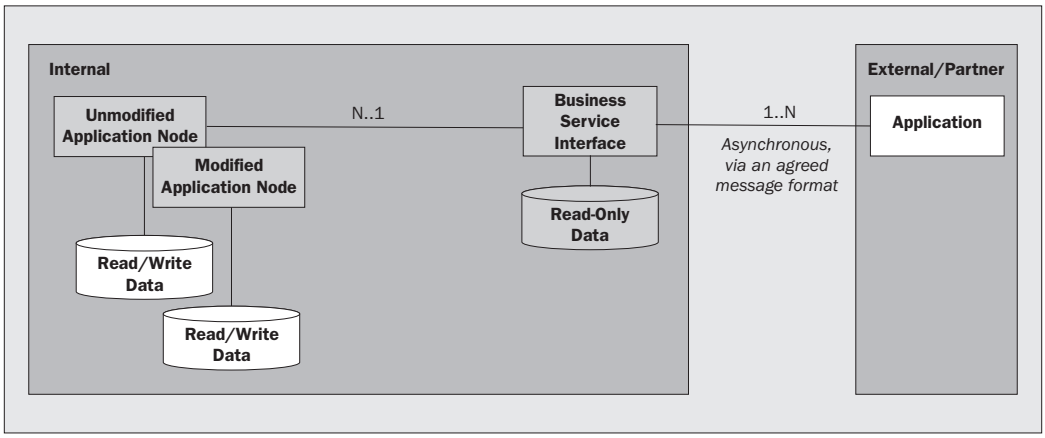
You can mitigate the complexity and high cost that the exposed application pattern potentially brings with it, by putting a layer between the backend enterprise applications and the partner tier. This layer exposes an interface that is much more oriented towards e-business interactions and manages the interaction with the backend applications. Such an interface is called a **business service** interface.

The complexity of a business service interface can vary from being a straightforward delegation component to a more intelligent component that performs data translation and invokes multiple applications. These applications may be invoked in a fixed or variable order, and the order and structure of the application invocations can perhaps be guided by the content of the request and/or the return values of invoked components. The core business logic and state information would still remain within the enterprise applications.

The main advantage of having a separate business service interface is that an enterprise remains in control of its internal systems and is not tied to partner agreements. As long as the service interface remains constant, they can improve their internal systems independently (perhaps replacing multiple legacy systems by a single modern ERP system). Multiple companies can offer the same business interface but implement it in radically different ways.

The interface exposed by a business layer can be determined (or dictated) by a single organization, or it can be agreed bilaterally between the organization that exposes the service and the organization that uses it. In this sense, there is no logical distinction between internal (private) and external (public) business processes. A single business service interface also cannot be parameterized for different business partners. This means that you cannot easily separate the "generic" aspects of a business service interface (common to all parties that invoke it) from aspects of a business service interface that are specific to (the requirements of) a particular party.

Business services can be implemented both in closed trading communities, which mutually agree which protocols and infrastructure products they use, and in open environments that use open Internet standards. In closed environments, organizations would typically use a message queuing product to benefit from advanced features such as support for asynchronous and reliable communication.



Much work has been done recently to make it easier to develop such service interfaces using the public Internet and web infrastructure. Collectively, this technology is referenced as **Web Services**. The Web Services area is organized around a number of interoperability specifications, including the Simple Object Access Protocol (SOAP), the Web Services Description Language (WSDL), and the Universal Description, Discovery and Integration (UDDI) initiative. We won't discuss any of these here, as we will look at them in more detail later in this book where they intersect with or complement parts of ebXML (see Chapter 9). A thorough coverage of Web Services is provided by *Professional XML Web Services* by Patrick Cauldwell *et al* (Wrox, ISBN 1-861005-09-1).

The Managed Public Processes Application Pattern

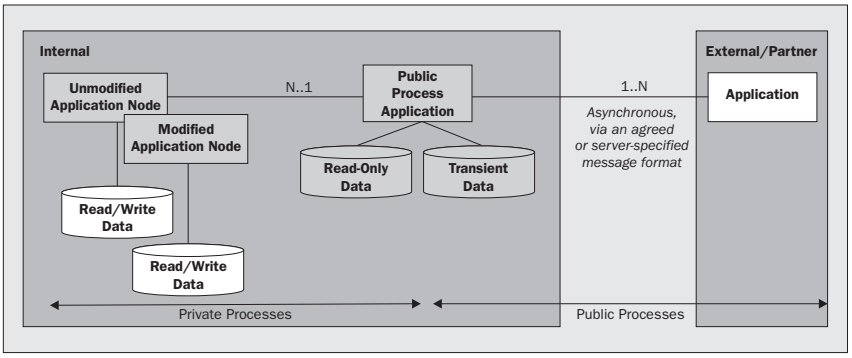
When we move to consider larger e-business communities, the sheer increase in the number of organizations, backend enterprise systems, and internal and external processes involved makes a straightforward application of the exposed services pattern very difficult to manage. Each new organization will introduce its own, possibly unique set of services, which it may need to expose in different ways to accommodate interactions with different (kinds of) business partners. With this model, inter-enterprise e-business becomes unmanageable and unaffordable for all but a limited set of very high-volume or high-value business transactions.

The managed public processes application pattern addresses integration at a higher level than the business service level, namely at the public process level. It focuses on open standards, for transport protocols (Internet protocols such as HTTP, SMTP), for business message encoding (Internet standards like MIME and XML), and for external business process descriptions.

To organize such larger e-business interactions, a cleaner organization of the business services interfaces is needed. In particular, a stricter separation of internal (private) processes and external public processes, or collaboration interactions, is needed. If a trading community identifies, analyzes, and formally describes its public processes, its members can align their business interfaces with standardized descriptions of these processes. This idea already underlies the efforts in the EDI standardization bodies to develop standard EDI message sets, which reflect (interactions in) common business processes.

The RosettaNet consortium is perhaps one of the best-known, successful initiatives to develop an XML-based e-business framework focused on business process analysis (see http://www.rosettanet.org/). The RosettaNet consortium is made up of companies in the electronic components, information technology, and semiconductor manufacturing industries, dedicated to developing e-business interoperability specifications for supply chain management applications in its member industry base.

Among other results, the RosettaNet consortium delivered and maintains a set of process specifications, called **partner interface processes** (**PIPs**), for application in its target industry supply chains. Individual PIPs (for instance, PIP 3A4, 'Manage Purchase Order') are organized in **segments** (such as Segment A, 'Quote and Order Entry'), which in turn are grouped in **clusters** (for example, Cluster 3, 'Order Management'). A PIP includes role-based references to the partner business roles, activities performed by each partner, the sequence in which those activities occur, structure and content of documents exchanged, as well as requirements for timing, security, and performance constraints.

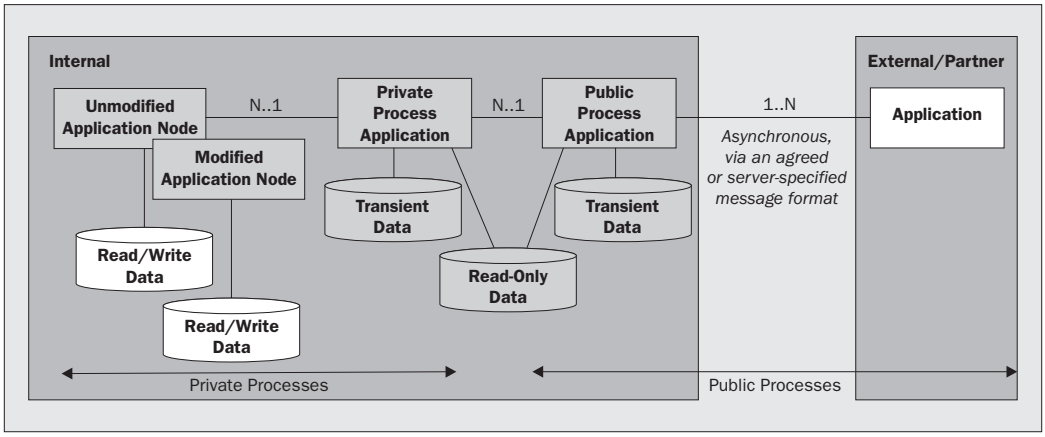


To manage a public business process, you need a software middleware layer that interfaces between enterprise applications and external partner systems. Note that a single PIP, or (more generally) business process description, may involve multiple interrelated message exchanges. Therefore, the managed public process pattern is a higher-level concept than an exposed service. The middleware layer extends the functionality of a business service interface layer in its ability to orchestrate and interrelate the various service invocations that jointly constitute the business process interaction. You could use a Web Services infrastructure as an implementation framework for a business process integration framework.

Aligning external business processes with industry standards, and grouping services that together implement a composite business process, are two ways of organizing business processes and thus of managing the complexity of the services that implement them. A third way is to separate partner-specific agreements from partner-independent aspects of an e-business interaction. To interact, two partners must make agreements on a number of issues, some of which are about business collaboration aspects (roles, message choreography, formats) and others about technical infrastructure issues (security, transport protocols) or service level agreements (such as agreed response times). Together, agreements on these are referred to as business of **trading partner agreements** (**TPA**). A modular design of the process management layer would separate those aspects, thus allowing you to sign up additional business partners, and support additional business processes or additional transport protocols, all independently of one another.

The Managed Public and Private Processes

In the previous pattern, the process management interface only covers public processes. A major part of the actual application logic is still handled by backend enterprise systems, which are also responsible for managing state information. For instance, an enterprise ERP system may have numerous status fields to encode the internal states a particular business interaction may be in. The B2Bi layer can interface with those systems in a variety ways, including use of a specialized EAI product or by invoking wrapper components for those legacy systems. Some of the business processes may require more or less manual intervention, which could be managed by a traditional **workflow management** (**WfM**) application. Taken together, internal and external business processes are long running processes, which may take anything from hours to months to complete.



The managed public and private processes pattern is an extension of the previous pattern that intends to provide a unified management environment for internal and external processes. In this scenario, the B2Bi product does not interface with internal applications directly, but rather provides an interface to an enterprise business process management system. Such a business process management system is configured by unified specifications of business processes, and can be thought of as "next generation EAI", extending the external process management layer into the enterprise.

This pattern is much more ambitious than the previous one, as in practice it will mean that you have to redesign your applications to externalize the business process state and the process flow logic that specifies the conditions that govern navigation among business states. Such a redesign is likely to be very expensive and complex, especially for complex, non-modular legacy systems. The upside of this is that it does result in a very flexible IT infrastructure, where you can easily redesign business processes (say, in response to business challenges like increased competition, changing regulation, etc.) at a very high level, and rely on the process management middleware to percolate those changes down to the information systems that support those processes.

e-Business Integration and ebXML

So far we've provided an overview of:

- The application *tier* at which you can implement e-business integration.
- The *middleware* commonly used to integrate systems.
- The five application *patterns* for business-to-business integration.

We are now ready to position ebXML in relation to this categorization system. The most important dimensions are the first and the third, and these determine which middleware is appropriate for implementing ebXML. In principle, an ebXML platform developer can build the ebXML messaging service on top of various middleware technologies, including a Web Services framework and an existing message queuing product. An ebXML project or derived product can use an ebXML infrastructure platform or implement (parts of) ebXML itself.

The ebXML framework is an advanced framework for e-business, rather than for application integration. It is clearly focused on providing a flexible, business process tier oriented approach to B2B interactions. As we've remarked earlier, business processes are composite and long running, and therefore constitute an even higher-level concept than pure service-based integration architectures, that focus on relatively simple interactions and in practice don't go beyond simple, synchronous, request/response-based interactions.

The ebXML framework is designed to support development of the managed public processes pattern. It does so, not by specifying the precise functionality of an ebXML middleware product or an application programming interface that such products should make available, but by providing a number of declarative, XML-based specifications that can be used as ways to configure or integrate an ebXML product with other (ebXML-based) products.

It offers a declarative, XML-based language for the specification of public processes, or business collaborations – the ebXML BPSS. Similarly, the ebXML CPA is an XML-based specification language that allows partners to formally define partner protocol agreements at a bilateral level. BPSS, CPA, and the ebXML messaging service together specify precisely the business services that implement the business collaborations agreed in the CPA. Together, these three specifications can be used as input to configure an ebXML-compliant software platform.

Standardization of such specification languages greatly simplifies development of B2Bi projects, as you can develop such a business collaboration using generic ebXML middleware and XML-based configuration documents, instead of having to program *ad hoc* business process interfaces.

The ebXML framework does not offer direct support for implementing the managed public and private processes pattern (apart, of course, from functionality shared with it). This is because BPSS is actually more a specification of business collaborations (public business process interfaces), and leaves the internal processes unspecified. Some other specifications, not in the scope of ebXML, actually go beyond BPSS in this respect and offer a more complete framework; in particular the **Business Process Management Initiative** (**BPMI**, see http://www.bpmi.org/) is addressing this field. BPMI and its unified process management markup language are discussed in Chapter 15.

Flexible and Dynamic e-Business

The previous discussion has focused on positioning ebXML as a set of e-business interoperability specifications for state-of-the-art XML-based business-to-business integration solutions based on the "managed public process pattern". As such, it obviates the needs for industry initiatives to develop specifications for the more generic platform interoperability issues. Announcements such as the support expressed by RosettaNet, OAG, OTA, and GCI for ebXML in future versions of their specifications confirm that ebXML has managed to fulfill this part of its charter: unifying the disparity in XML-based initiatives, and providing a standard framework on top of which additional specifications, horizontal or vertical, or ad hoc solutions can be developed. However, this is only part of the motivation for ebXML.

The other main concern the organizations that founded ebXML wanted to address was the problem that e-business, prior to ebXML, had not managed to extend its reach beyond a limited set of scenarios, involving large companies and fixed, high business transaction volumes. The challenge for an e-business framework is to enable solutions that are affordable for SMEs, or in economically less developed parts of the world.

A significant number of production e-business systems are based on EDI. Earlier in this chapter we looked at EDI and have pointed out some of its problems, including its proprietary syntax, use of expensive private networks, and the non-modular application designs often associated with EDI implementations. However, many of these problems could have been addressed and, to some extent, have indeed been addressed, in the period from 1995 onwards, by re-deploying the EDI infrastructure on the public Internet.

The main problem with re-deploying EDI over the Internet is that this only results in lowering the operational costs of a deployed e-business system, but not in lowering its initial and maintenance costs, as discussed in the introduction to Open-edi, one of the more recent and interesting EDI-related ISO standards (ISO/IEC 14662: *Open-edi reference model)*. As this document points out, two organizations need to establish and implement detailed bilateral business and technical agreements before they can actually exchange EDI messages. The initial costs of setting up such trading agreements are very high, as are the costs to modify them if business arrangements need to be modified. Second, these costs increase non-linearly with an increasing number of business partners. As a result, successful EDI implementations have been restricted to long-term partnerships between a limited number of partners, which automate high-volume (and/or high revenue) transaction volumes.

Standard Business Scenarios

The solution envisaged in the Open-edi document is to identify "standard business scenarios", which essentially are pre-defined specifications of very common business processes and of the business information that needs to be exchanged to support these processes. These scenarios are to be defined sufficiently rigorously to enable conformance testing of compliant applications. If two business partners recognize that there is a standard scenario that describes their intended interaction, they would be able to go out and purchase an off-the-shelf compliant application, or a relevant B2B "adaptor" that allows their enterprise system to support this scenario. In doing this, they would avoid the initial cost of setting up the business relation and e-business would become affordable for shorter-term relationships or lower volume (or lower margin) applications.

The concept of standard business scenarios is supported by ebXML in the following way:

- The Business Process Specification Schema (see the *ebXML Business Process Specification Schema* at http://www.ebxml.org/specs/ebBPSS.pdf, and Chapter 5) provides a way to define business collaborations in terms of the roles that business partners fulfill in them. Industry standards bodies can use it to express common processes in their industry, in the same way the RosettaNet consortium has done for its industry.
- Standardized descriptions of those collaborations can be published using the ebXML registry/repository mechanism (see the *ebXML Registry Services Specification* and the *ebXML Registry Information Model* at http://www.ebxml.org/specs/ebRS.pdf and http://www.ebxml.org/specs/ebRIM.pdf, respectively, and Chapter 7).
- Organizations can reference their capability to perform a role in such collaborations by referring to it from a CPP (collaboration protocol profile) document that describes its e-business capabilities. They can publish this document on the web site or using the same registry mechanism. Two organizations can formally specify protocol agreements in a similar CPA (collaboration protocol **agreement**) document (see the *Collaboration Protocol Profile and Agreement Specification* at http://www.ebxml.org/specs/ebCCP.pdf (sic), and Chapter 8).
- Independent software vendors can develop software components that automate the e-business interaction for a particular role in a business scenario. Vendors of software products targeted at SMEs (such as finance or administration) could incorporate those components in their products to open these up for e-business interactions, and could offer support for CPA formation from this application as well.

Note that the preceding paragraphs include a lot of "coulds" and "ifs". At this point, we're assuming a lot of development to have been completed that, at the time of writing, has just started. At present, all ebXML has to offer is an infrastructure framework that supports the concept of standard scenarios. The task of defining the actual scenarios is to be taken up by standards bodies or industry associations. The development of ebXML compliant commercial-off-the-shelf software is also beyond the scope of the specifications and is to be taken up by developers of such products.

Modular and Flexible e-Business Systems

As mentioned in the discussion of EDI, many EDI system implementations provide a limited separation of internal business processes and external business processes. As a result, these systems are difficult to adapt when the organization needs to support new or modified business interactions, or when the internal IT systems change. This inflexibility is another reason that many potential applications found EDI to be too expensive a solution.

The Open-edi specification proposes an architectural distinction in two levels in an e-business interaction that increases the flexibility of e-business systems. This model is adopted in both the RosettaNet implementation framework and in the ebXML technical architecture. These levels are referred to as the **Business Operational View** (**BOV**) and the **Functional Service View** (**FSV**), respectively. They address different aspects of e-business transactions, and different people are involved with them:

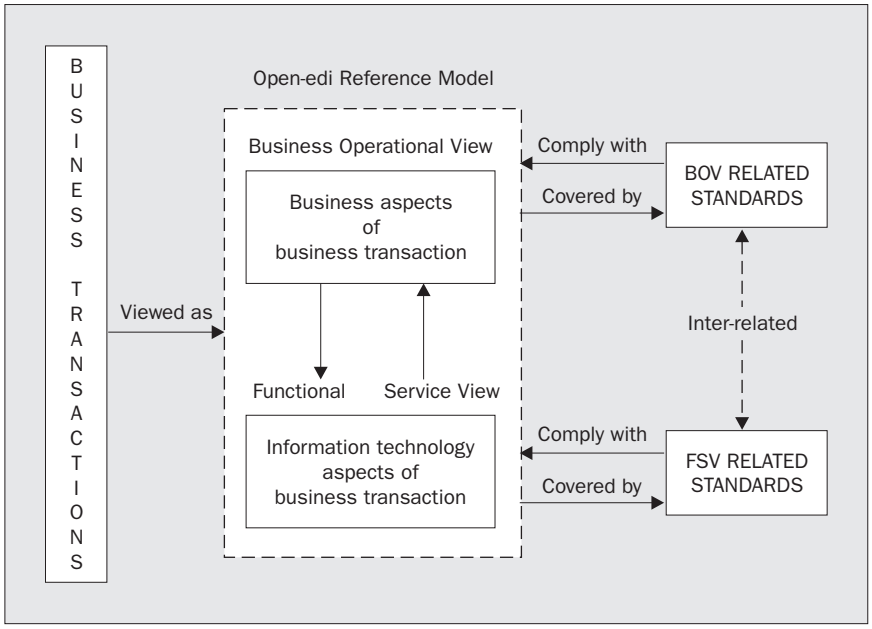
- The BOV is concerned with the semantics of business data in business transactions and associated message exchange. It is also concerned with business rules for business transactions, including operational conventions, agreements and mutual obligations.
- The FSV addresses the supporting services of the information systems that implement the information exchange. It focuses on functional capabilities, service interfaces and communication protocols. These capabilities include initiating, operating and tracking the progress of transactions, security, protocols and data or message translation.

The people involved in creating and maintaining the business layers of an e-business system are likely to be different from the people involved in the information systems implementation (with some people operating at both levels to provide the interface). Developments at FSV level should be able to have their own life cycle independent of the BOV lifecycle.

When working at the BOV level and interacting with business people, you should be able to use methods and tools that are appropriate for and commonly used at that level, such as graphical process modeling tools. Fortunately, the modeling tools used at business level are increasingly based on the same modeling language as is used for modeling software: the **Unified Modeling Language** (**UML**). We'll discuss this issue in more detail in Chapter 2. Ideally, at BOV level the entire issue of what notation is used to transport the data over the network (XML, EDIFACT, or perhaps even something like CORBA IIOP) should be irrelevant.

By contrast, at FSV level, document format and message encoding will be important considerations, as will be issues like security, message compression, or application interfaces and data translation layers (for example, from XML to a legacy EDIFACT interface).

The following figure is from the ISO/IEC 14662: *Open-edi reference model*, and illustrates these two layers and their interrelationship. It also indicates that each layer will be governed by its own set of (interrelated) standards:



The UN/CEFACT Modeling Methodology (UMM) distinguishes various abstraction levels, workflows, and project phases and supports an e-business process complementary to the Open-edi specification. We'll discuss UMM and illustrate its application to a use case in Chapter 2. UMM is formally defined in the UN/CEFACT Technology and Methodology Working Group *UN/CEFACT Modelling Methodology* document (CEFACT/TMWG/N090R9.1). You can find this at http://www.unece.org/cefact/docum/download/91-1.zip.

Summary

In this chapter we provided an introduction to the ebXML project, the main constituent parts of the ebXML framework, and its architecture in relation to the more general topic of e-business integration. We've seen that e-business integration can be addressed at several application **tiers** in a multi-tier architecture model for software application, and can build on a variety of existing **middleware** technology. We've also seen that practical business-to-business integration projects tend to cluster themselves around a limited number of **patterns**, or common solutions to common problems. These patterns show a progress towards increasingly flexible architectures, facilitated by increasingly capable middleware technology and configured by declarative configuration languages rather than procedural programming interfaces.

The ebXML framework is a set of specifications that is focused on supporting solutions that adopt the **managed public processes** pattern for e-business integration at the **business process tier**. This pattern is a state-of-the-art integration pattern that addresses e-business integration at the level of public processes (also referred to as collaborations). It reduces the operational complexity of having to manage a multitude of business services interfaces to multiple business partners by:

- Aligning business processes with industry standards
- Grouping **services** that together implement a composite business **process**
- Separating generic aspects of collaboration from **trading partner agreements**

The ebXML framework includes declarative, executable languages to express e-business collaborations (BPSS) and protocol profiles and agreements (CPP/CPA) in a non-proprietary, XML-based format. These specification documents can be shared and ported between compliant implementations. The ebXML messaging services complements these by offering a very capable standards-based e-business messaging system. Jointly, these can be used to configure and manage ebXML-based e-business message exchange.

As a collection of e-business infrastructure interoperability specifications, the ebXML framework allows horizontal or vertical standards bodies to focus on their specific needs, and obviates the need for them to develop proprietary solutions for generic e-business infrastructure issues. Initial announcements by leading organizations developing such horizontal or vertical standards seems to indicate that ebXML is indeed successful, and provides a generic framework to support them.

In addition to offering a state-of-the-art framework for e-business, ebXML also incorporates some mechanisms that lower the cost of setting up and maintaining e-business relations, an issue that has restricted the use of e-business to large companies and stable, (semi-)permanent business relations. These mechanisms are:

- Facilitation of registering standard business scenarios and collaboration protocols of organizations
- Facilitation of partially automating the process of negotiating and setting up trading partner agreements dynamically
- Separation of business aspects from implementation aspects of e-business, thus making it easier for business people to concentrate on defining (and refining) the business aspects of e-business collaboration

While it is widely acknowledged that we still have a long way to go here, many people feel that these are important initial attempts to increase ebXML's utility for small and medium-size enterprises or enterprises in parts of the world other than the highly developed countries.