Creating A Single Global Electronic Market

ebXML Technical Architecture Specification

ebXML Technical Architecture Team

17 October 2000

1.0 Status of this Document

This document represents a work in progress upon which no reliance should be made. Distribution of this document is unlimited. The document formatting is based on the Internet Society’s Standard RFC format.

This version:

ebXML_TA_v0.9.doc

***Latest version*:**

N/A

Previous version:

EbXML_TA_v0.8.72i

2.0 Scope

This document describes the underlying *Architecture* for ebXML. It provides a high level overview of ebXML and describes the relationships, interactions, and basic functionality of ebXML *Components*. It should be used as a roadmap to learn: (1) what ebXML is, (2) what problems ebXML solves, and (3) core ebXML functionality. This document does not go into the level of detail required to build an ebXML application. Please refer to each of the ebXML component specifications for the exact information needed to build ebXML applications and related *Components*.

3.0 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this specification. At the time of publication, the editions indicated below were valid. All standards are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below.

W3C XML v1.0 specification

ISO/IEC 14662: Open-edi Reference Model
ISO 11179 Metadata Repository
ISO 10646: Character Encoding
ISO 8601:2000 Date/Time/Number Datatyping

4.0 ebXML Technical Architecture Participants

We would like to recognize the following for their significant participation in the development of this document.

Editors: Duane Nickull, XML Global Technologies
Brian Eisenberg, DataChannel

Participants: Colin Barham, TIE
Al Boseman
Dick Brooks, Group 8760
Cory Casanave, DataAccess Technologies
Robert Cunningham, Military Traffic Management Command, US Army
Christopher Ferris, Sun Microsystems
Anders Grangard, EDIFrance
Kris Ketels, SWIFT
Piming Kuo, Worldspan
Kyu-Chul Lee, Chungnam National University
Henry Lowe, OMG
Melanie McCarthy, General Motors
Klaus-Dieter Naujok, NextEra Interactive
Bruce Peat, eProcessSolutions
John Petit, KPMG Consulting
Mark Heller, MITRE
Scott Hinkelman, IBM
Karsten Riemer, Sun Microsystems
Lynne Rosenthal, NIST
Nikola Stojanovic, Columbine JDS Systems
Jeff Sutor, Sun Microsystems
David RR Webber, XML Global Technologies

5.0 Table of Contents

6.0 Introduction

Over 25 years ago the idea was born to eliminate the use of paper documents for exchanging business data by linking computer systems together so that the data, normally on paper, could be sent from one system to the other. This concept became known as *Electronic Data Interchange (EDI)*. The advantages are still valid today: single point of information capture, electronic delivery, low storage and retrieval costs, to mention just a few. However, looking at the statistics of who is currently utilizing *EDI* only the top 10,000 companies on a global scale (Fortune 1000 in the top 10 countries) are using *EDI*. For the rest of the business world only 5% are using *EDI* and therefore today common *Business Processes* are dominated by paper transactions.

Today, *Extensible Markup Language (XML)* is at the forefront of efforts to replace paper-based business transactions. In order for *Small to Medium Enterprises (SMEs)* to benefit from the next generation of *eBusiness* standards, these standards must contain all the information to allow software developers to create programs that can be purchased off-the-self (shrink-wrapped-solutions) or developed in-house. The success of any new way to exchange data among businesses depends not only on the adoption by the Fortune 1000 companies of standard agreements, but on their adoption by the other estimated 25,000,000 *SMEs* in the world. Without an economic incentive for the *SMEs*, any new method of accomplishing *eBusiness* is just re-inventing the status quo instead of delivering a pervasive solution.

The answer is to document and capture in an unambiguous way the *Business Processes* and associated information requirements for a particular business goal, which can then be processed by a computer program. The use of *XML* technologies combined with *Business Process and Information Modeling* and object-oriented technology can achieve this objective. Instead of looking at the data requirements based on internal legacy database records, *Business Experts* identify the collaborations with other parties in order to achieve a certain business goal. Those collaborations are documented in a model developed in the *Unified Modeling Language (UML)*. Each activity requires the exchange of business information. Instead of taking the data element (*EDI*) approach, objects are used to describe and model *Business Processes*.

With the advent of *XML*, it is easier to identify and define objects with attributes (data) along with functions that can be performed on those attributes. There are many objects that are common to many *Business Processes* (goals), such as address, party, and location.. By allowing these objects to be reused, ebXML can provide the means to unify cross-industry exchanges with a single consistent *Lexicon*. However the role of ebXML is not to replicate the reliance on electronic versions of common paper documents such as purchase orders, invoices and tender requests and to offer up and develop such implementation examples. Instead the ebXML specifications provide a framework where *SMEs*, software engineers, and other organizations can create consistent, robust, and interoperable *eBusiness* services and *Components*, ultimately leading to the realization of global *eBusiness*.

7.0 ebXML Abstract Overview

Although *XML* is a recent newcomer in the *eBusiness* landscape, *Supply Chains* in many industries, as well as industry consortiums and standards organizations are using *XML* to define their own vocabularies for business relationships and transactions. The vocabularies, business templates, and *Business Processes* used by these groups to transact business must be accessible by all partners at any time.

Furthermore, newcomers to the *Supply Chain* or business partnerships must be able to discover and implement *eBusiness* interfaces to interoperate in a secure, reliable and consistent manner. In order to facilitate these needs, mechanisms must be in place that can provide information about each participant (*Trading Partner*), including what they support for *Business Processes* and their implemented service interfaces. This includes information about what business information is required for each instance of a business message, and a mechanism to allow dynamic discovery of the semantic meaning of that business information. The entire mechanism must be able to recognize semantic meanings at the business element level and be implemented using *XML* based representations and systems. The complete set of ebXML Specifications explains this functionality in detail.

8.0 ebXML Conceptual Overview

Figure 1 shows a conceptual model for two *Trading Partners*, first configuring and then engaging in a simple business transaction interchange. This model is provided as an illustration of the process and steps that may typically be required using ebXML applications and related *Components*. The ebXML specifications are not limited to this simple model, provided here as quick introduction to the concepts. Further examples of ebXML implementation models are provided at the end of this section. Specific implementation examples are described in Appendix A.

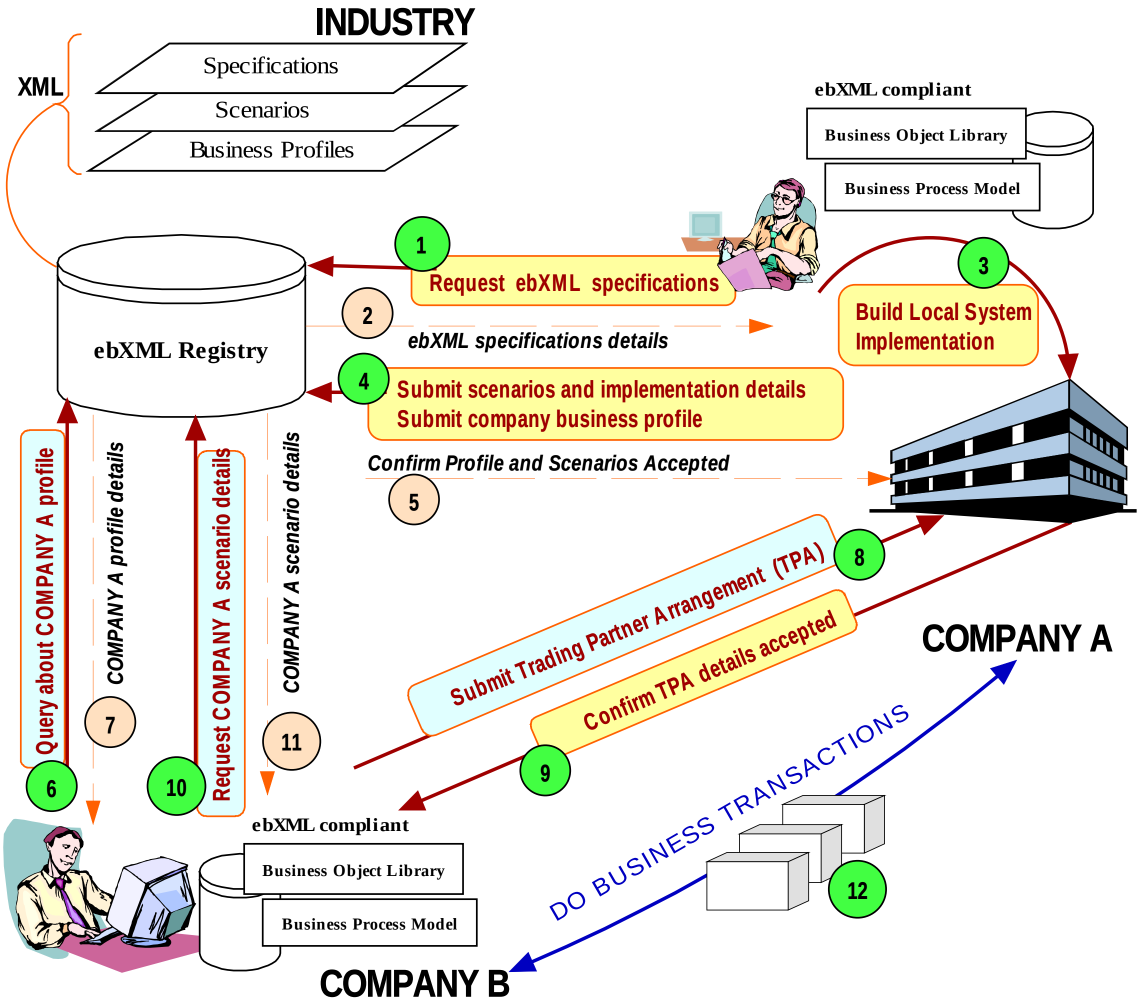

INDUSTRY
XML
Specifications
Scenarios
Business Profiles
ebXML compliant
Business Object Library
Business Process Model
1
Request ebXML specifications
3
Build Local System Implementation
ebXML Registry
2
ebXML specifications details
4
Submit scenarios and implementation details
Submit company business profile
Confirm Profile and Scenarios Accepted
5
Query about COMPANY A profile
COMPANY A profile details
Request COMPANY A scenario details
COMPANY A scenario details
6
7
10
11
Submit Trading Partner Arrangement (TPA)
8
Confirm TPA details accepted
9
COMPANY A
DO BUSINESS TRANSACTIONS
12
ebXML compliant
Business Object Library
Business Process Model
COMPANY B

Figure 1: *a high level overview of ebXML functionality*

In Figure 1, Company A has become aware of an ebXML Registry that contains a set of ebXML specifications. Company A requests an ebXML specification in order to determine if it wants to become an ebXML compliant participant (Figure 1, step 1). The request results in the ebXML process specification being sent to Company A (Figure 1, step 2). Company A, after reviewing the specification, decides to build and deploy its own ebXML compliant application (Figure 1, step 3). [Note: custom software development is not a necessary prerequisite for ebXML participation, user applications will be also commercially available as turn-key solutions.]

Company A then submits its own implementation details, reference links, and *Trading Partner Profile* (*TPP*) as a request to the ebXML registry (Figure 1, step 4). The *TPP* submitted describes the company's ebXML capabilities and constraints, as well as its supported business scenarios. These scenarios are *XML* versions of the *Business Processes* and associated information parcels (based on business objects: for example a sales tax calculation) that the company is able to engage in. After receiving verification that the format and usage of a business object is correct, an acknowledgment is sent to Company A by the ebXML Registry (Figure 1, step 5).

Company B (an SME) is then informed by Company A that they would like to engage in a business transaction using ebXML. Company B acquires a shrink-wrapped application that is ebXML compliant and able to interface with its existing (legacy) applications. The ebXML program already contains the base ebXML information bundles such as a library of *Business Object*s and Models for the specific industry they are part of. Company A knows that its *Business Processes* and *TPP* are compliant with the ebXML infrastructure from the information available in the ebXML specification package. However, since Company A just registered its scenarios, they are not yet part of the package. Therefore the ebXML application queries the ebXML Registry about Company A (Figure 1, step 6). Company A's profile is retrieved (Figure 1, step 7). Based on the *TPP*, the application determines that it is able to execute a specific scenario that Company A supports.

Before engaging in that the scenario Company B submits a proposed *Trading Partner* Agreement (*TPA*) directly to Company A's ebXML compliant software interface. The *TPA* outlines the *eBusiness* scenario and specific arrangement(s) it wants to use with Company A, as well as certain messaging, contingency and security-related requirements (Figure 1, step 8). Company A accepts the *TPA* and acknowledgement is sent directly to Company B's shrink-wrapped ebXML software application (Figure 1, step 9). Since the scenario from Company A was not available in the software package that Company B is using, the application requests it from the ebXML Registry (Figure 1, step 10). The scenario is then provided to Company B's application (Figure 1, step 11).

Based on the processes (contained in the process models) and information parcels (presented in class diagrams) Company A and B are now engaging in *eBusiness* utilizing ebXML specifications via their respective software applications (Figure 1, step 12).

The conceptual overview described in the scenario above introduced the following concepts and architectural *Components*:

1. A standard mechanism for describing a *Business Process* and its associated information model.
2. A mechanism for registering and storing a *Business Process* and information model so that it can be shared/reused.
3. Discovery of information about each participant including:
   - The *Business Processes* they support.
   - The business service interfaces they offer in support of the *Business Process*.
   - the business messages are to be exchanged between their respective service interfaces.
   - The technical configuration of the supported transport, security and encoding protocols.
4. A mechanism for registering the aforementioned information so that it may be discovered and retrieved.
5. A mechanism for describing a *Trading Partner* Agreement (*TPA*) which may be derived from the information about each participant from item 3 above.
6. A standardized messaging service which enables interoperable, secure and reliable exchange of messages between two parties.
7. A mechanism for configuration of the respective messaging services to engage in the agreed upon *Business Process* in accordance with the constraints defined in the *TPA*.

Using these *Components* ebXML compliant software can be used to implement popular, well-known *eBusiness* scenarios, examples include but are not limited to:

a) Two partners set-up an agreement and run the associated electronic exchange.
b) Three or more partners set-up a *Business Process* implementing a supply-chain and run the associated electronic exchanges
c) A company sets up a portal that defines a *Business Process* involving the use of external business services.
d) Three or more parties engage in multi-party *Business Process* and run the associated electronic exchanges.

The above examples are described in detail in Appendix A.

9.0 Relating the ebXML Architecture to Existing Standards

The ebXML approach utilizes public specifications and standards wherever applicable and consistent with the goals of the ebXML initiative. One such specification is the Open-edi work, an ISO/IEC 14662 (Open-edi Reference Model) vision of future *EDI*. The ebXML approach can benefit from the lessons learned by Open-edi work and utilize the related methodologies. Particularly, Open-edi takes a generic industry and technology neutral approach and by similarly utilizing this, ebXML will enable organizations to provide the opportunity to significantly lower the barriers to electronic data exchange by introducing standard business scenarios and the necessary services to support them. In principle, once a business scenario is agreed upon, and implementations conform to the standards, there is no need for prior agreement among *Trading Partners*, other than the decision to engage in the ebXML transaction in compliance with the business scenario. This will lead to the ability to establish short-term business relationships quickly and cost effectively.

The field of application of ebXML is the electronic processing of *XML*-based business transactions among autonomous multiple organizations within and across sectors (e.g., public, private, industrial, geographic). It includes business transactions that involve multiple data types such as numbers, characters, images and sound. The Open-edi Reference Model provides the standards required for the inter-working of organizations through interconnected information technology systems, and is independent of specific information technology (IT) implementations, business content or conventions, business activities, and organizations.

The Open-edi Reference Model places existing *EDI* standards in perspective using two views to describe the relevant aspects of business transactions: the Business Operational View (BOV) and the Functional Service View (FSV). The ebXML Architecture uses similar views of these definitions. The BOV expresses the users' requirements needed to achieve the common business goal. The FSV describes how the BOV is actually implemented using the selected technology.

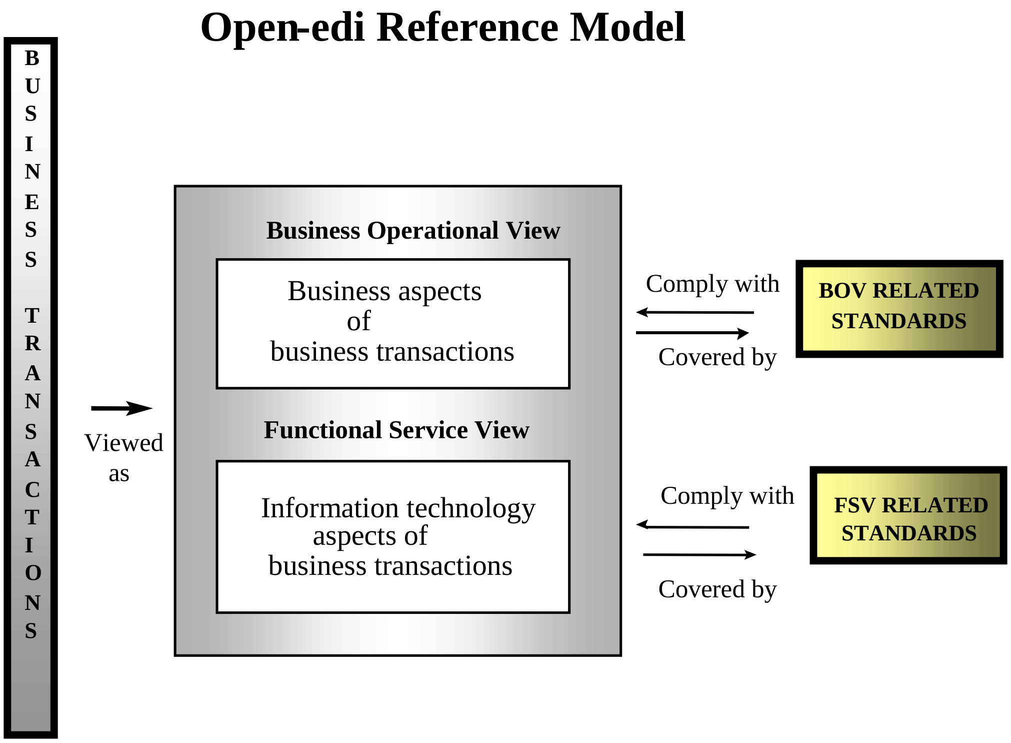


Figure 2: *Open-edi environment*

Figure 2 above sets out the relationship between the Open-edi Reference Model and these views.

The primary focus of ebXML resides with the FSV and the supporting BOV. The assumption for ebXML is that the FSV will be implemented by commercial software vendors and ensure backwards compatibility to traditional *EDI* systems (where applicable). As such, the resultant BOV-related standards provide the business and object class models needed to construct ebXML compliant *eBusiness* services and *Components*.

While business practices from one business organization to another are highly variable, most activities can be decomposed into *Business Processes* which are more generic to a specific type of business. This analysis through the modeling process will identify object classes and models that are likely candidates for standardization. The ebXML approach looks for standard reusable *Components* from which to construct information exchange software. While Open-edi is a theoretical syntax neutral approach, ebXML itself is focused on a physical implementation using specifically an XML-based syntax and related technologies.

10.0 ebXML Architecture

The ebXML Architecture Reference Model uses the following two views to describe the relevant aspects of business transactions:

- The Business Operational View (BOV)
- The Functional Service View (FSV)

The BOV addresses the semantics of:

a) The semantics of business data in transactions and associated data interchanges

b) The architecture for business transactions, including:

  - operational conventions;
  - agreements;
  - mutual obligations and requirements.

These specifically apply to the business needs of ebXML *Trading Partners*.

The FSV addresses the supporting services meeting the mechanistic needs of ebXML. It focuses on the Information Technology aspects of:

- functional capabilities;
- service interfaces;

- protocols.

Additionally, the functional capabilities, service interfaces and protocols include:

- capabilities for implementation, discovery, deployment and run time scenarios;
- user application interfaces;
- data transfer infrastructure interfaces;
- protocols for interworking of XML vocabulary deployments from different organizations.

The BOV and the FSV are discussed in detail in the following sections.

11.0 ebXML Business Operational View

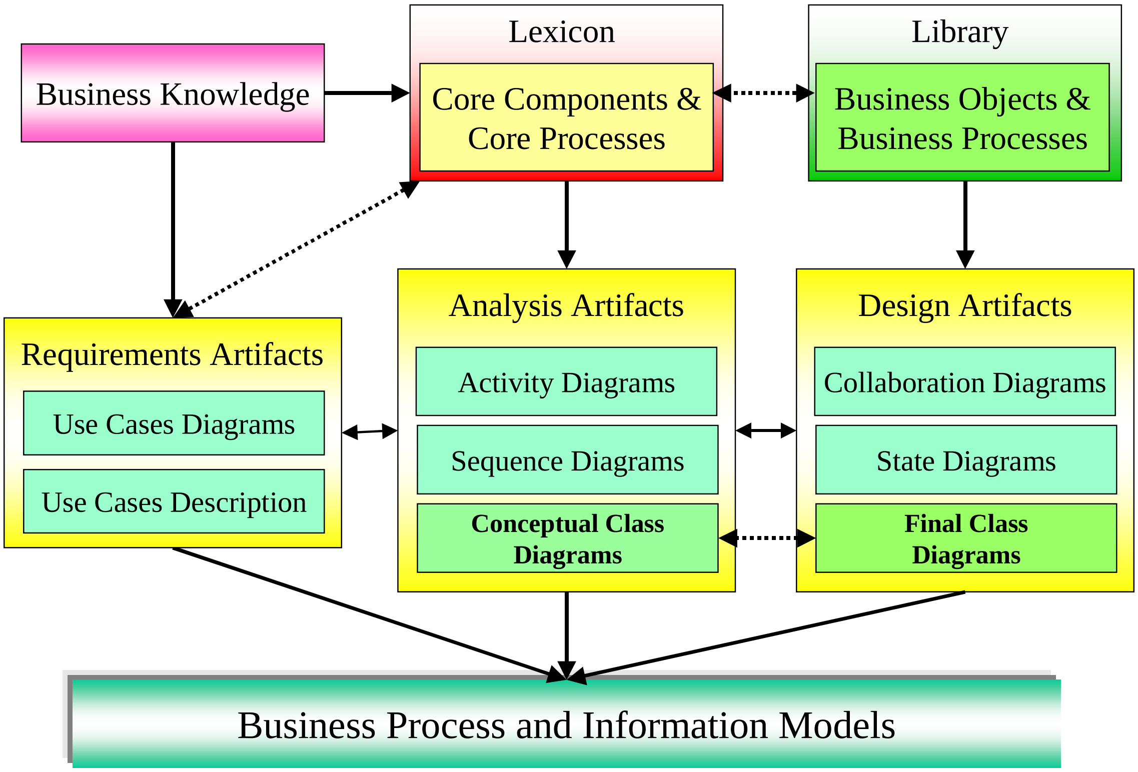


***Figure 3:** the Business Operational View*

ebXML Business and Information Models are created following the selected ebXML *Business Process and Information Modeling* (see section 17).

Business knowledge is captured in a *Lexicon*. The *Lexicon* contains data and process definitions including relationships and cross-references as expressed in business terminology and organized by industry domain. The *Lexicon* is the bridge between the specific business or industry language and the knowledge expressed by the models in a more generalized industry neutral language.

The first phase defines the requirements artifacts which describe the problem using Use Case Diagrams and Descriptions. If *Lexicon* entries are available they will be utilized, otherwise new *Lexicon* entries will be created.

The second phase (analysis) will create activity and sequence diagrams describing the *Business Processes*. Class diagrams will capture the associated information parcels (business messages). The analysis phase reflects the business knowledge contained in the *Lexicon*. No effort is made to force the application of object-oriented principles. The class diagram is a free structured data diagram.

The design phase is the last step of standardization, which may be accomplished by applying object-oriented principles. In addition to generating collaboration diagrams, a state diagram may also be created. The data diagram from the analysis phase will undergo harmonization to align it with other models in the same industry and across others.

Therefore in ebXML interoperability is achieved by applying *business objects* across all class models. The content of the *business object library* is created by analyzing existing *business objects* as used by many industries today in conjunction with the *Lexicon* content and ebXML selected modeling methodology.

Figure 4 shows how the user can see this correlation to the actual business roles:

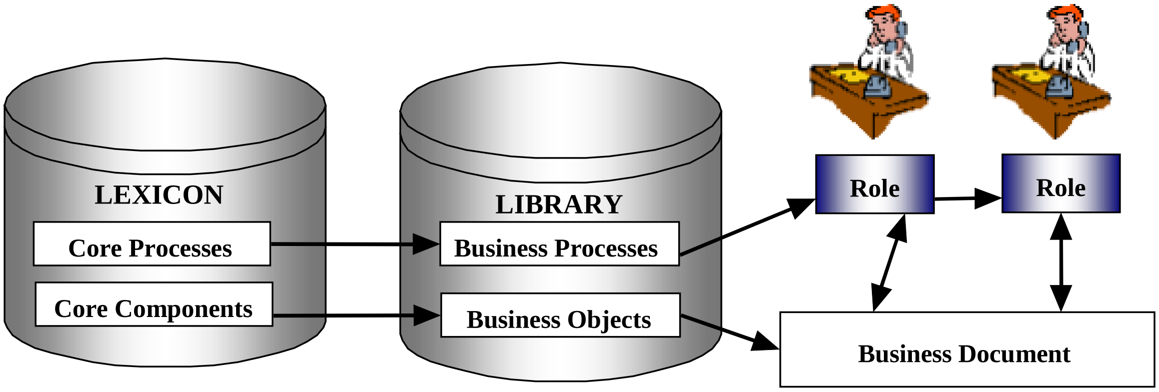


Figure 4: *Role Relation Model.*

12.0 ebXML Functional Service View

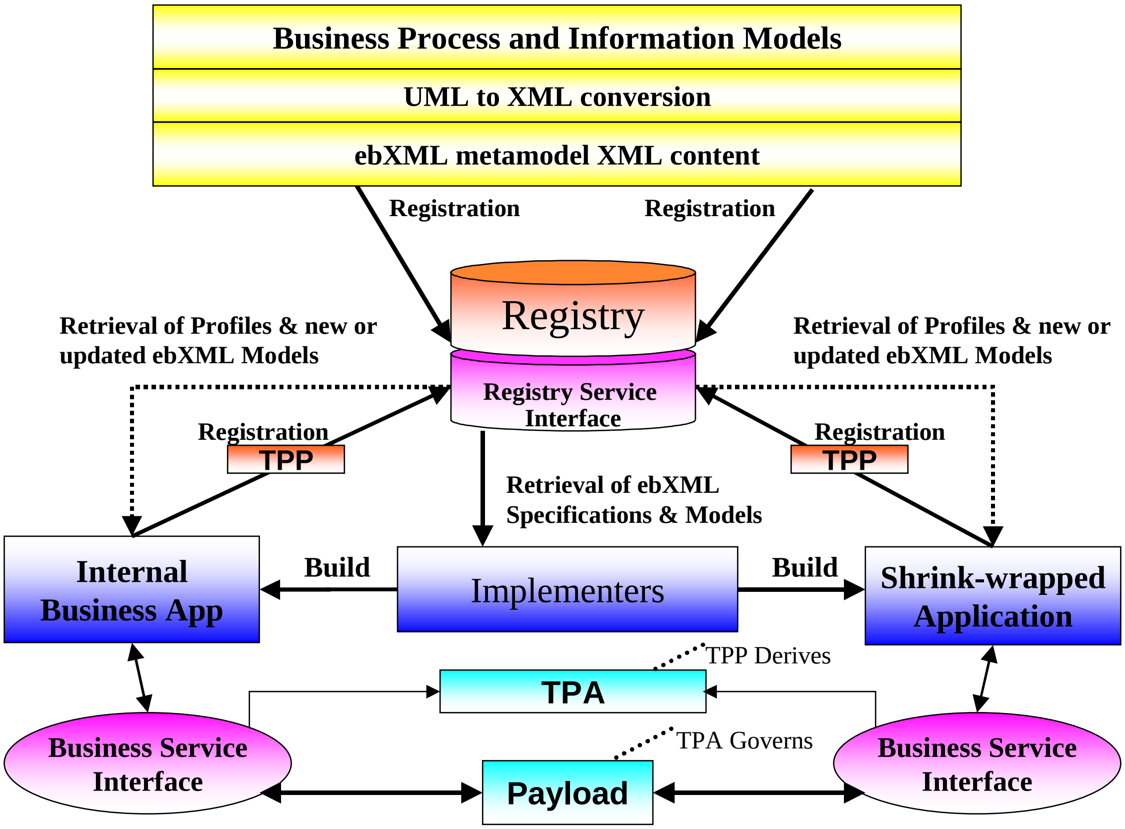

Business Process and Information Models
UML to XML conversion
ebXML metamodel XML content
Registration
Registration
Registry
Registry Service Interface
Retrieval of Profiles & new or updated ebXML Models
Retrieval of Profiles & new or updated ebXML Models
Registration
TPP
Registration
TPP
Retrieval of ebXML Specifications & Models
Internal Business App
Build
Implementers
Build
Shrink-wrapped Application
TPP Derives
TPA
TPA Governs
Business Service Interface
Payload
Business Service Interface

Figure 5: *ebXML Functional Service View*

The ebXML Registry system is an important part of ebXML. The Registries not only hold the ebXML base reference specifications, but also the *Business Process* and information models developed by industry groups, *SMEs*, and other organizations. These models are compliant with the ebXML Metamodel and related methodologies. In order to store the models they are converted from *UML* to *XML*. ebXML Registries store these models as instances of *XML* that are compliant to the ebXML metamodel.

This *XML*-based business information shall be expressed in a manner that will allow discovery down to the attribute level via a consistent methodology. In order to enable this functionality, the use of Unique Identifiers (UIDs) is required for all items within an ebXML Registry and Repository System. These UID references are implemented as *XML* attributes, expressed as fixed value attributes for each of the physical *XML* elements and structures. UID keys are required references for all ebXML content. The UID keys themselves do not contain explicit versioning control, but may be used with versioning control mechanisms, either as an extension to the UID key value itself, or within the ebXML Registry and Repository System. The latter is the preferred approach since it provides a single access and maintenance and control point.

Additionally the UID keys may be implemented in physical *XML* syntax in a variety of ways. The architectural needs require that several mechanisms be supported. These mechanisms include, but are not limited to:

- A pure explicit reference mechanism (*XML* URN:UID method),
- A referential method (*XML* URI:UID / namespace:UID),
- An object-based reference compatible with W3C Schema (*XML* URN:complextype name), and
- A datatype based reference (for ISO 8601:2000 Date/Time/Number datatyping and then legacy datatyping).

Examples of each of these in *XML* syntax in the order noted include:

- An URN:UID method,
- An URI:UID / namespace:UID method,
- An URN:complextype name method, and
- An explicit type encoding values as outlined in ISO 8601.

Additionally, all participating *Components* in ebXML must facilitate multilingual support. Again, a UID reference is particularly important here as it provides a language neutral reference mechanism. To enable multilingual support, the ebXML specification must be compliant with Unicode and ISO/IEC 10646 for character set and UTF-8 or UTF-16 for character encoding.

The underlying ebXML Architecture is distributed in such a manner to minimize the potential for a single point of failure within the ebXML infrastructure. This specifically refers to Registry and Repository Services (see Registry and Repository Functionality, Section 20 for details of this architecture).

The implementation of the FSV of ebXML, can be categorized as having the following three major phases:

a) The Implementation Phase

The implementation phase deals specifically with the procedures for creating an application of the ebXML infrastructure

b) The Discovery and Deployment Phase

The Discovery and Deployment Phase covers all aspects of actual discovery of ebXML related resources and self enabled into the ebXML infrastructure.

c) The Run Time Phase

The Run Time phase covers the execution of a ebXML scenario with the actual associated ebXML transactions.

These three phases are now discussed in greater detail.

13.0 Implementation Phase

A *Trading Partner* wishing to engage in an ebXML compliant transaction, must first request a copy of the ebXML specification. The Specification is then downloaded to the *Trading Partner*. The *Trading Partner* studies the ebXML specification. The *Trading Partner* subsequently requests to download the *Lexicon* and the *Business Object Library*. The *Trading Partner* may also request other *Trading Partners' Business Process* information (stored in its *TPP*) for analysis and review. The *Trading Partner* may also submit its own *Business Process* information to an ebXML compliant Registry.

Figure 6 below, illustrates a potential interaction between an ebXML Registry and a business service interface.

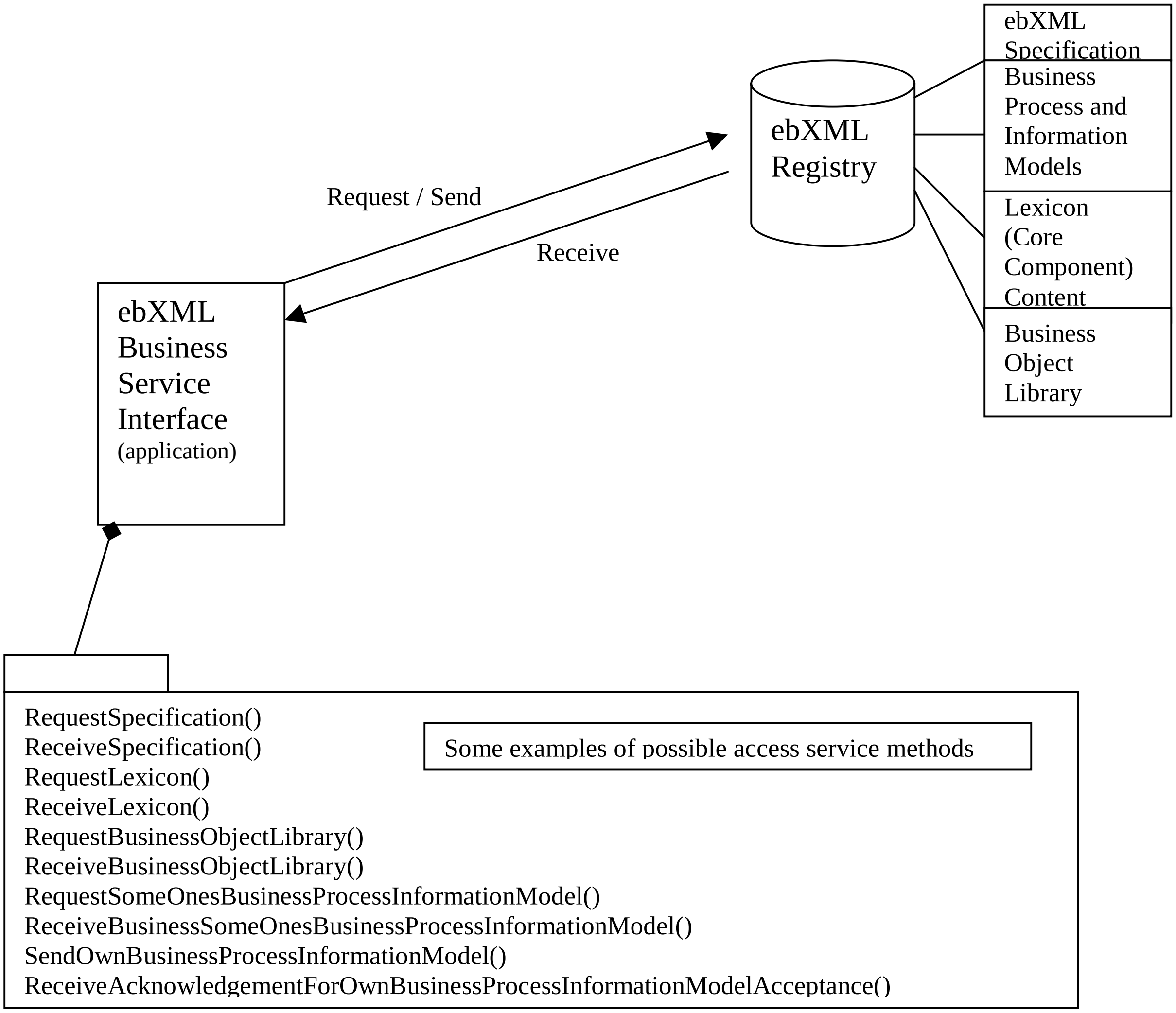
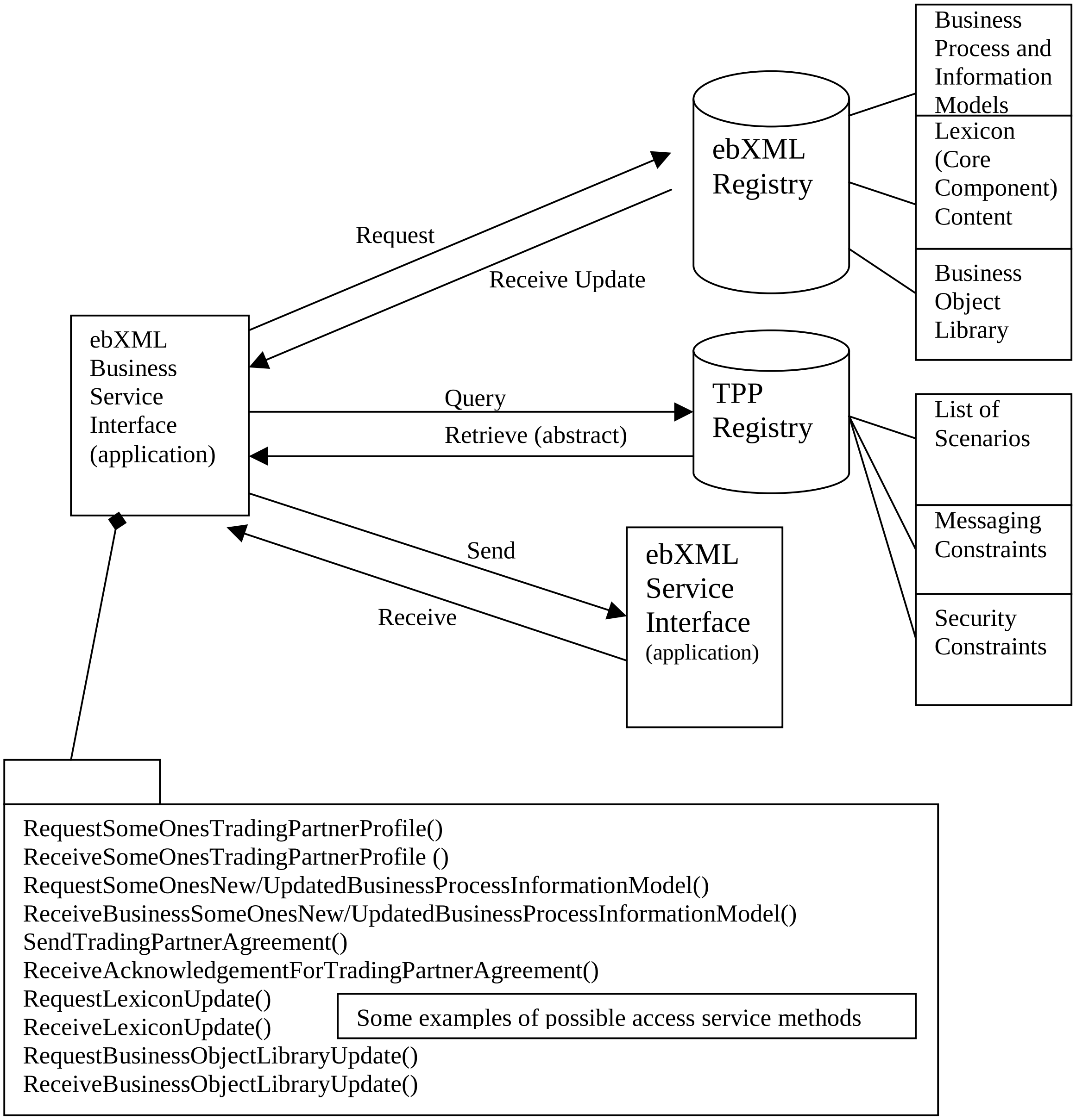

ebXML Specification
Business Process and Information Models
Lexicon (Core Component) Content
Business Object Library
ebXML Registry
Request / Send
Receive
ebXML Business Service Interface (application)
RequestSpecification()
ReceiveSpecification()
RequestLexicon()
ReceiveLexicon()
RequestBusinessObjectLibrary()
ReceiveBusinessObjectLibrary()
RequestSomeOnesBusinessProcessInformationModel()
ReceiveBusinessSomeOnesBusinessProcessInformationModel()
SendOwnBusinessProcessInformationModel()
ReceiveAcknowledgementForOwnBusinessProcessInformationModelAcceptance()
Some examples of possible access service methods

Figure 6: *Functional Service View: Implementation Phase*

14.0 Discovery and Deployment Phase

A *Trading Partner* who has implemented an ebXML Business Service Interface may now begin the process of discovery and deployment (Figure 7). One possible discovery method may be to request the *Trading Partner Profile* of another *Trading Partner*. Requests for updates to *Lexicons*, *Business Object Libraries* and updated or new *Business Process* and information models are also methods which shall be supported by an ebXML application. This is the phase where *Trading Partners* discover the semantic meaning of business information being requested by other *Trading Partners*.


Business Process and Information Models
Lexicon (Core Component) Content
Business Object Library
ebXML Registry
Request
Receive Update
ebXML Business Service Interface (application)
Query
Retrieve (abstract)
TPP Registry
List of Scenarios
Messaging Constraints
Security Constraints
Send
Receive
ebXML Service Interface (application)
Some examples of possible access service methods

RequestSomeOnesTradingPartnerProfile()
ReceiveSomeOnesTradingPartnerProfile ()
RequestSomeOnesNew/UpdatedBusinessProcessInformationModel()
ReceiveBusinessSomeOnesNew/UpdatedBusinessProcessInformationModel()
SendTradingPartnerAgreement()
ReceiveAcknowledgementForTradingPartnerAgreement()
RequestLexiconUpdate()
ReceiveLexiconUpdate()
RequestBusinessObjectLibraryUpdate()
ReceiveBusinessObjectLibraryUpdate()

***Figure 7:** Functional Service View: Discovery and Deployment Phase*

15.0 Run Time Phase

The Run Time phase is the least complex (Figure 8). Note that no Registry calls are required during the Run Time Phase. There are ebXML message instances being sent and received between *Trading Partners* utilizing the ebXML Messaging Service.

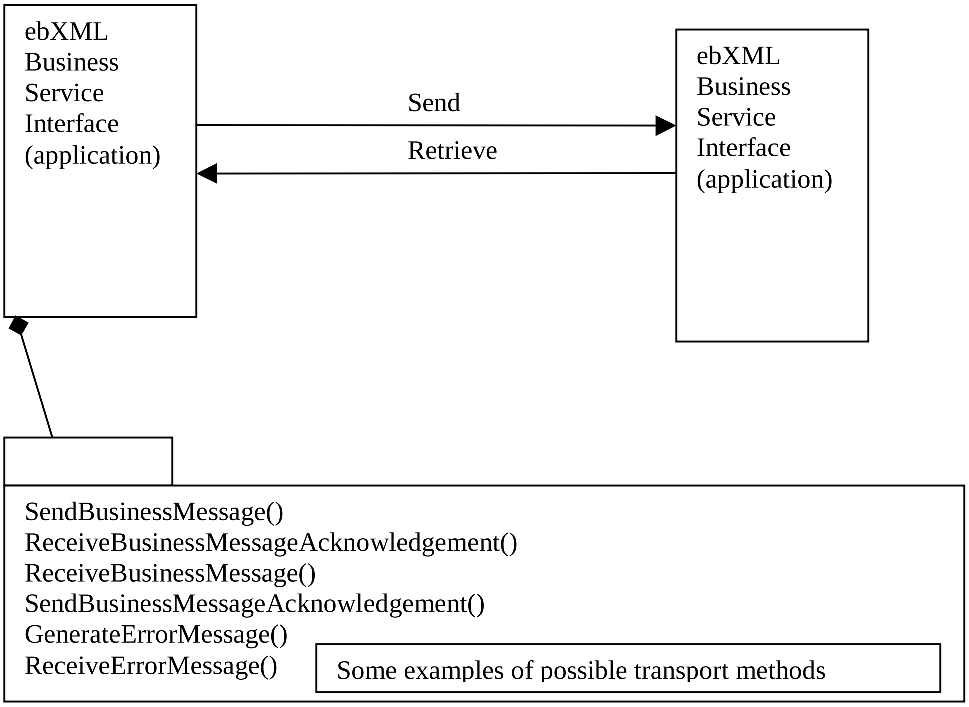

ebXML Business Service Interface (application)
Send
Retrieve
ebXML Business Service Interface (application)
SendBusinessMessage()
ReceiveBusinessMessageAcknowledgement()
ReceiveBusinessMessage()
SendBusinessMessageAcknowledgement()
GenerateErrorMessage()
ReceiveErrorMessage()
Some examples of possible transport methods

Figure 8: *Functional Service View: Run Time Phase*

16.0 Trading Partner Information

To facilitate the process of conducting *eBusiness*, *SMEs* and other organizations need a mechanism to publish information about the *Business Processes* they support along with specific technology implementation details about their capabilities for exchanging business information. This is accomplished by creating a *Trading Partner Profile (TPP).* The TPP is a document which allows a *Trading Partner* to express their minimum *Business Process* and Business Service Interface requirements in a manner where they can be universally understood by other ebXML compliant *Trading Partners*. The TPP describes the specific technology capabilities that a *Trading Partner* supports and the Service Interface requirements that need to be met in order to exchange business documents with that *Trading Partner*. The TPP of the a priori interchange information is stored in an ebXML Registry which provides a discovery mechanism for *Trading Partners* to find one another.

16.1 Support for Trading Partner Agreements

To facilitate the process of conducting electronic business, organizations need a mechanism to publish information about the *Business Processes* they support, along with specific technology details about their capabilities for sending and receiving business documents.

ebXML defines the ability for this to be realized under the broad notion of a *Trading Partner Agreement*.

A *Trading Partner Agreement* (*TPA*) is a document that describes: (1) the Messaging Service (technology), and (2) the Process (application) requirements that are agreed upon by two or more parties. A *TPA* is negotiated after the discovery process and is essentially a snapshot of the specific technology and process related information that two or more parties agree to use to exchange business information. If any of the parameters of an accepted *TPA* changes after the agreement has been executed, a new *TPA* shall be negotiated between all parties.

Conceptually, ebXML supports a three level view of narrowing subsets to arrive at agreements for transacting business. The outer-most scope relates to all of the possibilities that a Partner could do, with a subset of that of what a Partner is capable of doing, with a subset of what a Partner "will" do.

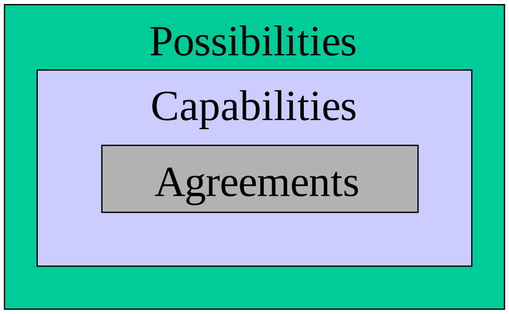


***Figure 9:** Three level view of TPA's*

ebXML acknowledges the global scope of a *Trading Partner Agreement*s to include such aspects as legal agreement elements and legal ramifications and other trade issues that are, from an over-arching business perspective, essential elements of "Agreements between Traders." ebXML limits its scope within this broad spectrum to addressing the needs of (electronic) *Business Collaborations*. This provides extensibility for ebXML to expand to encompass other aspects of *Trading Partner Agreement*s on its own or by embracing other work. Further, the entities engaged in Business Collaborations within ebXML are referred to as *Partners*. Business Collaborations are the first order of support that can be claimed by ebXML Partners. This "claiming of support" for specific Business Collaborations is facilitated by a distinct profile defined specifically for publishing, or advertising in a directory service, like the ebXML Registry/Repository or other available similar services.

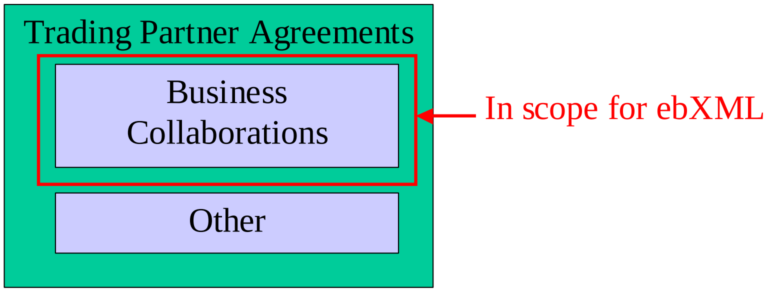


Figure 10: *Scope for TPA's*

17.0 Business Process and Information Modeling

17.1 Overview

The purpose of the *Business Process and Information Modeling* specification is to enable the modeling of the business relationships between partners in a shared *Business Process*, and their interaction and information exchange as they each perform roles within that process. In general terms, a *Business Process* is defined as a sequenced set of business transactions. A business transaction is a clearly defined exchange of business messages resulting in a new legal or commercial state between the two partners. The business semantics of each commercial transaction are defined in terms of the *Business Object*s affected, and the commitment(s) formed or agreed. The technical semantics of each commercial transaction are defined in terms of a 'handshake' protocol of required message (signal) exchanges.

17.2 Position within overall ebXML Architecture

The *Business Process and Information Modeling* specification has important semantic relationships to the *Core Component* specification and to the *Trading Partner* Specification. In addition, the business models produced are registered within an ebXML Registry/Repository.

17.3 Business Process and Information Modeling Functionality

The *Business Process and Information Modeling* specification supports *UML Business Process* and information modeling, along with conversion of *UML* models into *XML*, and direct access to the *XML* expression of the model. Within the *UML* modeling system the ebXML specification provides a *UML* profile, a set of recommended diagrams, and a selected methodology to follow in constructing those diagrams. For the conversion of *UML* to *XML* the specification provides a set of production rules. For further standardization, the specification provides a set of core processes, and a set of patterns from which to compose new process definitions.

The ebXML Metamodel specifications constitute a set of *XML* structures that can be populated and stored in an ebXML Registry and Repository System. The *XML* structures may utilize a classification system with UID reference linkages which are compatible with the Registry and Repository architecture requirements.

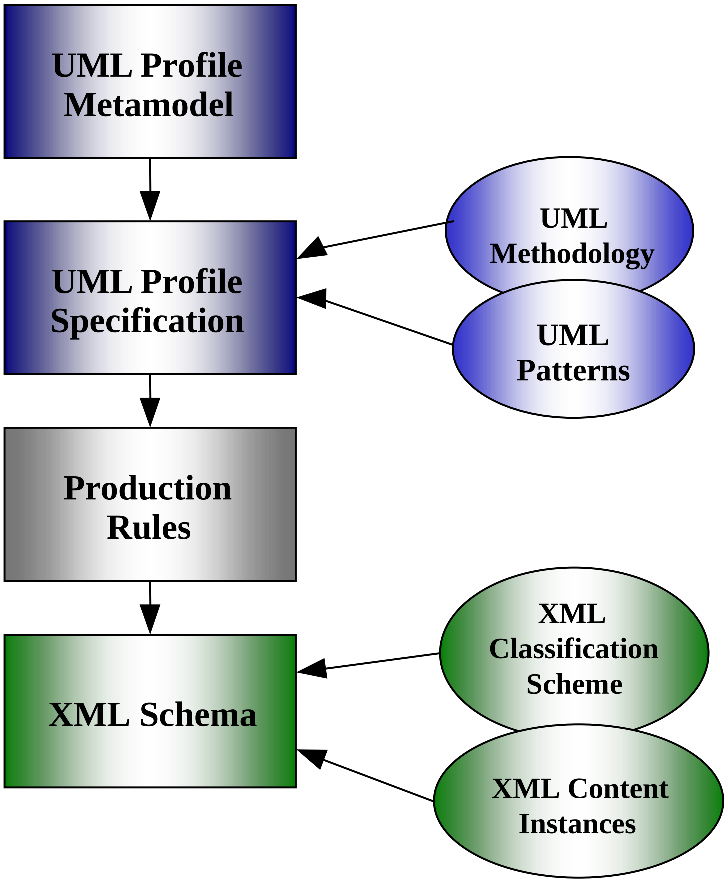


Figure 11: *Relationship of UML to ebXML metamodel content representation.*

17.4 The Business Process and Information Metamodel

The *Business Process* and Information Metamodel is composed of four layers (see Figure 12 below). The top layer consists of the Business Operational Map which supports the process of relating different *Business Processes* to each other into a map as well as the categorization of *Business Processes* by business or process area. The next layer, the Business Requirements View, supports definition of the partner type which partakes in a step within a *Business Process* along with the business agreements resulting from or governing that step and the economic resource commitment or exchange resulting from that step. The Business Transaction View supports the specification of Business Transactions in terms of exchanged business documents. The bottom layer, the Business Service View, captures the syntax and semantics of business messages and their exchange between business services.

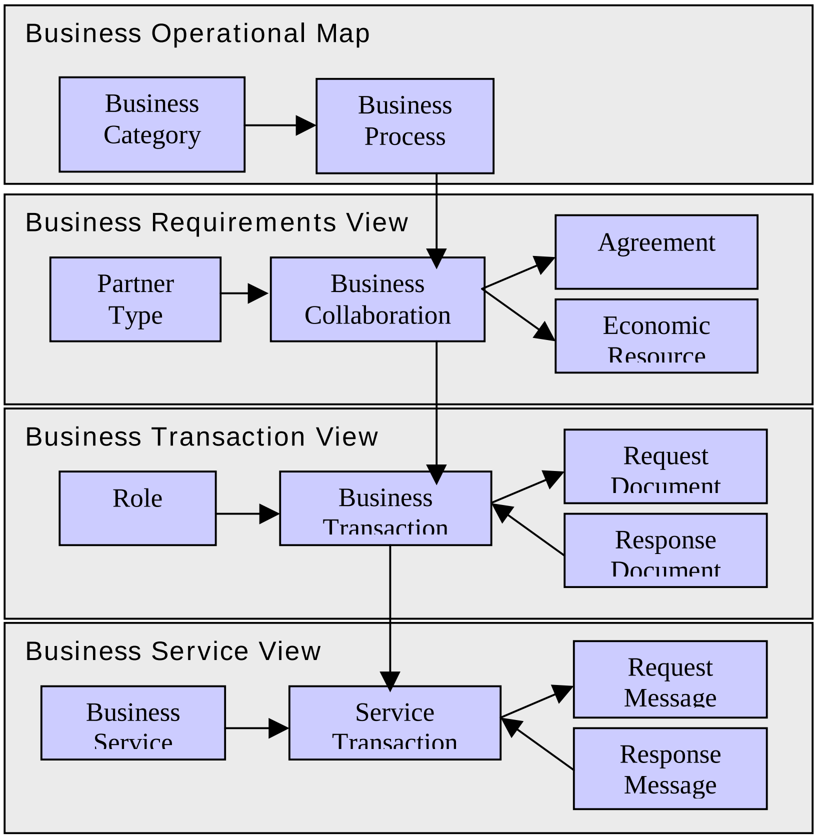


Figure 12: *Business Process and Information Metamodel Architecture*

17.5 Interfaces and Relationship

The interface from a *Business Process* and information model to other parts of the ebXML architecture, or to other tools and environments outside the scope of the ebXML specifications, is an *XML* document representing the *Business Process* and information model. Specifically, the interface between the *Business Process* and information model and the *Trading Partner* model is the part of such an *XML* document that represents the business transactional layer of the *Business Process* metamodel. The expression of the sequence of commercial transactions in *XML* is shared between the *Business Process* and *Trading Partner* models.

17.6 Relationship to Trading Partner Agreements

The interface between the *Business Process* and information model and the *Trading Partner* specification is the sequence of business transactions, the commercial business itself, and the message exchange in support of the business transaction. The profile of a *Trading Partner* defines that partner's functional and technical capability to support one or more roles in a *Business Process*. The agreement between two *Trading Partners* defines the actual conditions under which the two partners will conduct business transactions together.

17.7 Relationship to Core Components

A *Business Process* can be seen as a series of actions on entities within an enterprise, interleaved with a set of communications with parties outside the enterprise. The communication between the parties is the shared part of the *Business Process*. This is the focus of ebXML.

The entities within an enterprise are called business entities, and their data structure can be represented by *Business Object*s.

The communication with parties outside the enterprise takes place through an exchange of business documents.

Both *Business Object*s and business documents are composed from *Core Components*, re-useable low-level data structures.

The exact composition of a *Business Object* or a business document is guided by a set of contexts derived from (among other sources) the *Business Process*.

18.0 Core Component Functionality

A *Core Component* captures information about a real world (business) concept, and relationships between that concept and other business concepts.

A *Core Component* can be either an individual piece of business information, or a natural "go-together" family of business information pieces. It is 'Core' because it occurs in many different areas of industry/business information interaction.

A *Core Component* may contain:

- Another *Core Component* in combination with one or more individual business information pieces.
- Other *Core Components* in combination with zero or more individual business information pieces.

A *Core Component* needs to contain either attribute(s) or be part of another *Core Component*, thus specifying the precise context or combination of contexts in which it is used.

Context may be structural, identifying the placement of a *Core Component* within another *Core Component*. It may be a combination of structural contexts when the *Core Component* is re-used at different layers within another *Core Component*.

Context will also be defined by the *Business Process* model, which defines the instances in which the *Business Object* occurs.

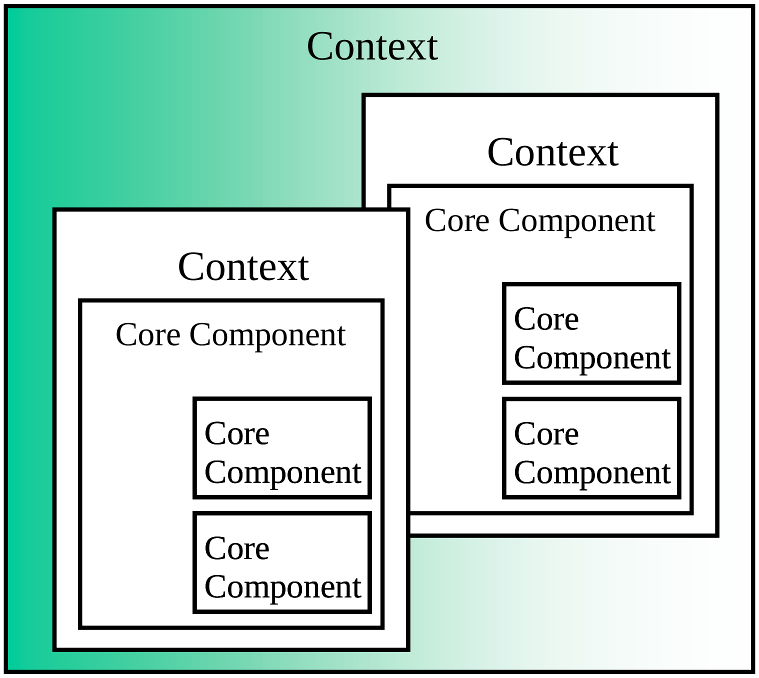


Figure 13: *Core components as contextual items.*

The pieces of information, or *Core Components*, within a generic *Core Component* may be either mandatory, or optional. A *Core Component* in a specific context or combination of contexts may alter the fundamental mandatory/optional cardinality.

Individual *Core Components* will in general match the "data list" part of *Business Objects*.

19.0 Business Object Functionality

19.1 Overview

The term *Business Object* is used in two distinct ways in ebXML, with different meanings for each usage:

- In a business model, *Business Objects* describe a business itself, and its business context. The *Business Objects* capture business concepts and expresses an abstract view of the business's "real world" functions.

- In a business software application or service, *Business Objects* reflects how business concepts are represented in software. The abstraction here reflects the transformation of business ideas (processes) into a software implementation.

Within the context of ebXML, only *Business Objects* represented in *Business Processes* and information models are of relevance.

19.2 Business Objects in Business Process and Information Models

A *Business Object* describes a thing, concept, process or event in operation, management, planning or accounting of a business or other organization. It is a conceptual object that has been specified for the purpose of directly describing and representing, and thus serving, a business concept or purpose. The focus/subject is the business subject/concept being modeled.

A *Business Object* in this usage is a specification for a kind of object which may exist in one or more business domains. The specification of a business object may include attributes, relationships, and actions/events that apply to these objects. These business object models may exist regardless of the existence of information systems, applications, software design or program code. They are independent of information systems because business object models directly reflect and abstract "real world" business concepts and scenarios. Thus business object models are defined independently of application systems.

The primary concern when creating business object models is capturing common business semantics and having a common idea or concept that is usable by different parts of a business or by different independent businesses.

19.3 Common Business Objects

A *Common Business Object* (CBO) is a business object that is specified in more than one Domain. For the purposes of defining CBOs, a domain is defined as an industry sector. As with all business objects in general, the most important issue with CBOs is a common concept and mutually agreed upon structure.

20.0 Registry and Repository Functionality

20.1 Overview

An ebXML Registry provides a set of distributed services that enable the sharing of information between interested parties for the purpose of enabling *Business Process* integration between such parties by utilizing the ebXML specifications. The shared information is maintained as objects in an ebXML Repository which is managed by ebXML Registry Services. Access to an ebXML Repository is provided by the interfaces (APIs) exposed by Registry Services.

Therefore, architecturally the Registry and Repository are tightly coupled *Components*. The Registry provides the access services interfacing, the information model and reference system implementation, while a Repository provides the physical backend information store. For example, an ebXML Registry may provide a *Trading Partner Profile* from the Repository in response to a query; or an ebXML Repository may contain reference DTD's or Schemas that are retrieved by the Registry as a result of searching a metadata classification of the DTD's or Schemas. Figure 14 provides an overview of this configuration.

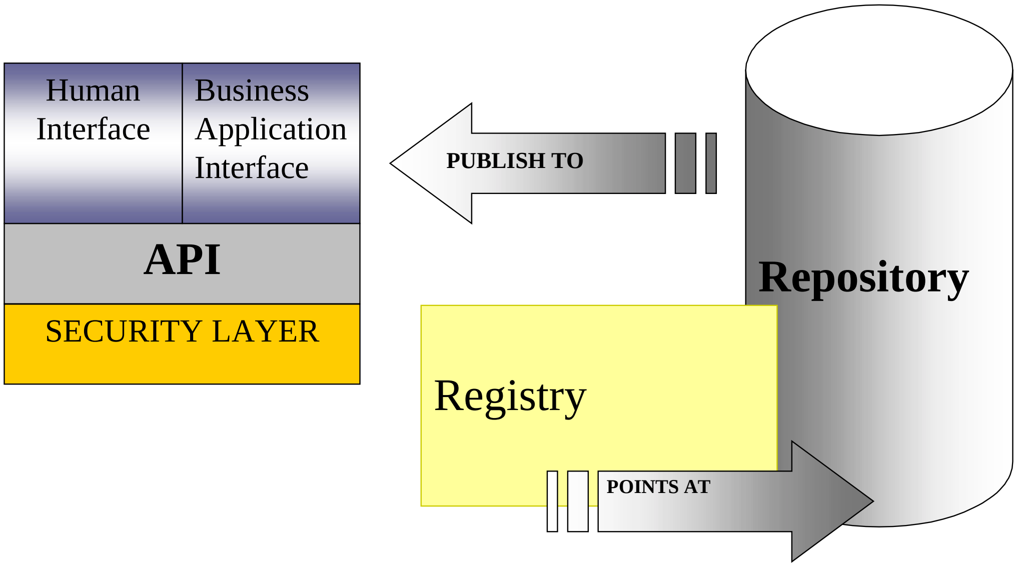


Figure 14: *Registry / Repository interaction overview.*

20.2 Information Model & Interface Constrains

In order to accurately and consistently store and retrieve information a registry requires a formal architecture that includes an information model. Traditional relational SQL databases have a very simple information model that includes tables, indexes and column definitions for content to be stored into them. However, when using *XML* structures to store and manage content we potentially have an infinite variety of ways to present information to a registry. We therefore to ensure there are particular aspects of those structures that allow us to manage them within the registry, and that also these aspects are linked to the access methods that will be used to interface to the registry. Providing the mechanisms to support the business functional capabilities expressed in these *XML* structures and content and ensuring it functions correctly is the role of the information model.

The information model for the ebXML Registry is an extension of the existing OASIS Registry information model, specifically tailored for the storage and retrieval of business information content, whereas the OASIS model is a superset designed for handling extended and generic information content. As such the ebXML Registry information model is designed to make it easier to implement and to provide explicit ebXML metamodel compliant instance structures to facilitate accessing and storing ebXML content.

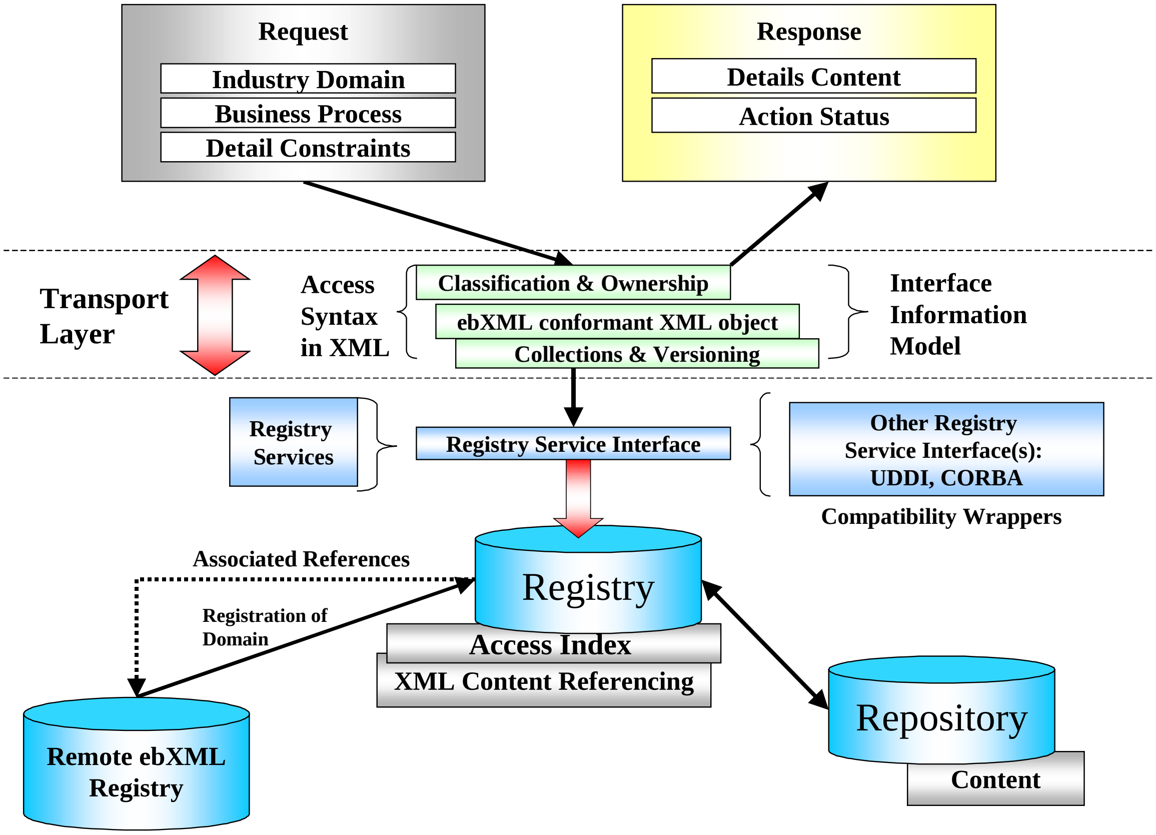


Figure 15: *Registry / Repository Architecture.*

A Registry maintains the metadata for a registered object, and a Repository maintains the file containing a registered object. The Registry and Repository are tied together in that the metadata for a registered object in the Registry includes a globally unique locator for a file, in some Repository, that contains the registered object.

A Registry Item contains information that identifies, names, and describes each registered object, gives its administrative and access status, defines its persistence and mutability, classifies it according to pre-defined classification categories, declares its file representation type, and identifies the submitting and responsible organizations.

Related to this the existing ISO11179/3 work on business semantic content registry implementations is used to provide a model for the ebXML Registry implementation. Again the approach is to take a tailored subset of the ISO11179 functionality that is applicable to the ebXML Registry requirements, and to make implementing ebXML Registry systems simpler for vendors to implement and librarians to manage.

Combined together these reference specifications are then exposed via the Registry Interface system itself. The Registry interface is the architectural component that provides both a machine-to-registry automated access system, and also a human-to-registry interactive visual access system. The Registry interface system is designed to be a primitive *XML*-based interface that is transport layer neutral. However, the reference implementation of the Registry interface is built using the ebXML Transport layer facilities only. Similarly the query syntax used by the Registry access mechanisms is designed to be a neutral syntax based solely in *XML* syntax, and independent of the physical product implementation of the backend Repository system.

20.3 Formal Functional Overview

A Registry/Repository system may have many deployment models that yield the same functionality. The initial specification and implementation will define the minimal functional requirements that a Registry/Repository System shall provide to facilitate its role in the ebXML infrastructure. It is expected that future specifications and implementations will evolve into more complex systems.

All interaction between a Registry clients and the Registry are treated as business transactions between parties. Thus the processes supported by the Registry are described in terms of:

- A special *TPA* between the Registry and Registry clients.
- A set of business functional processes involving the Registry and Registry clients.
- A set of business messages exchanged between a Registry client and the Registry as part of a specific business functional process.
- A set of primitive interface mechanisms to support the business messages and associated query and response mechanisms.
- A special *TPA* for between one Registry interoperating with another Registry.

- A set of functional processes involving Registry to Registry interactions.
- A set of error responses and conditions with remedial actions.

The Registry interactions supported here are intended to be a limited subset of the full requirements as defined by the ebXML Requirements documents. The architecture described here is based on supporting the conceptual ebXML architecture and business interactions as defined in Section 8 of this specification. Some of the extended functionality deferred to a subsequent phase includes transformation services, workflow services, quality assurance services and extended security mechanisms.

20.4 Sample Objects Residing in a Repository and Managed by a Registry

- **Schema**: These objects are documents that represent the schema (*XML* DTD, etc.) for *XML* documents.

- **Process**: These are objects that represent a *Business Process*. These could include a process description in an *XML* form such as XMI or could be actual software *Components* (e.g. Java Classes) that could represent an implementation of a *Business Process*.

- ***Trading Partner Profile***: These are *XML* documents that provide information about a party interested in participating in B2B interaction.

- **Reference Content:** there are two types of reference content, those that describe the reference information model and classification systems within the registry itself (schemas), and those that categorize industry business information (*XML* document instances). The later are often standard information sets that can be expected to reside in and be supported by the registry information model, such as ISO reference datatypes, ISO reference code tables and similar open public definitions.

- **Any object with metadata**: Elements provide standard metadata about the object being managed in the Repository. Note that the object metadata is separate from the object itself, thus allowing the ebXML Registry to catalog arbitrary objects.

20.5 Registry Management of Repository Objects and Metadata

Registry messages shall exist to create, modify and delete Repository objects and their metadata. Appropriate security protocols shall be deployed to offer authentication and protection for the Repository when accessed by the Registry.

Additionally all content stored into a Registry/Repository is implicitly public and open information. Therefore parties submitting information to an ebXML Registry should ensure that they have appropriate intellectual rights and permissions to submit this information. An ebXML Registry will provide administrative access rights to ensure only the submitting organization has formal access to change the content, however all other retrieval rights will be open. For this reason, *TPA*s, which are necessarily proprietary to

Trading Partners will not be stored within an ebXML Registry, only the public *TPP* details will be stored within an ebXML Registry.

20.6 Querying Registries and Returning Repository Objects and Metadata

A Registry query mechanism shall be employed to query for Repository objects and their metadata by either an Application automated interface or a Human software GUI interface.

Repository objects and their metadata shall be made available by ebXML messages sent to the Registry (typically an Application requestor service).

Repository objects and their metadata can also be addressable where applicable as an *XML* based URI reference using only HTTP for simple direct access.

Each Repository Object is identified by a Unique Identifier key (see Section 12 for an introduction on UID key mechanisms). A query on a Unique Identifier (UID) returns one and only one Repository object.

Metadata queries perform an object search based on the metadata defined for (but maintained outside) a managed object.

Browse and drill down queries are expected to be the primary use case for querying the Registry by Web based human interactions. In this scenario, a user browses the repository content using a Web browser via a HTTP protocol. The user may initially browse and traverse the content based on the built-in classification schemes.

20.7 Registry to Registry Interfacing Model

Since ebXML Registries are distributed each Registry may potentially interact with and cross-reference to another ebXML Registry. The following diagram provides an example of the architectural *Components* that facilitate these mechanisms.

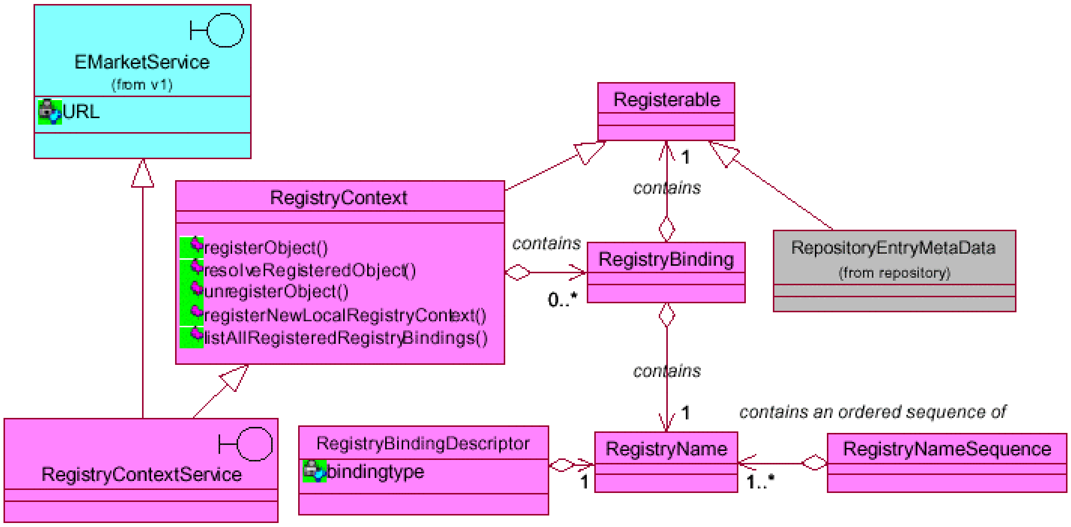


***Figure 16:** Registry to Registry Service Components.*

Referencing Figure 16 above, the following apply:

A *RegistryName* to *RegistryObject* association is called a *RegistryBindin*g. A RegistryBinding is always defined relative to a *RegistryContex*t. A *RegistryContext* is an object that contains a set of RegistryBindings in which each RegistryName is unique. Different RegistryNames can be bound to a RegisteredObject in the same or different RegistryContexts at the same time.

A RegistryObject specialization (a RegistryContext or a RepositoryEntryMetaData), which is bound into a RegistryContext is unaware of the fact that it has been associated to a RegistryName via a RegistryBinding, and that the RegistryBinding may be bound into a RegistryContext (not navigable).

A RegistryName is used to identify the binding within the RegistryContext for which it may be bound. A *RegistryNameSequence* is an ordered set of RegistryNames that can be used to resolve a RegisteredObject from a given target RegistryContext. RegistryContextService is RegistryContext boundary interface and is an EMarketService. For the extent of the model scope of this document, a URL is inherited and is used to facilitate distribution of RegistryContexts through URL addressing.

A RegistryBindingDescriptor describes a RegistryBinding by identifying the type of binding and the RegistryName. RegistryBindingDescriptors are returned on list messages on RegistryContexts.

The architecture of the ebXML metadata classification system within the ebXML Registry itself will be extended (see Figure 14 above). These extensions will support references to domains that are not directly managed by that Registry, and its associated Repository store.

20.8 Registry/Repository Business Scenario Example

In addition to the use of the ebXML Registry as a means to facilitate and enable the core architecture of ebXML compliant information exchanges, a Registry/Repository may also be used to facilitate business functional implementations. An example would be a network of *Trading Partners* similar to a telephone directory Yellow Pages system where businesses can be categorized by services that they provide.

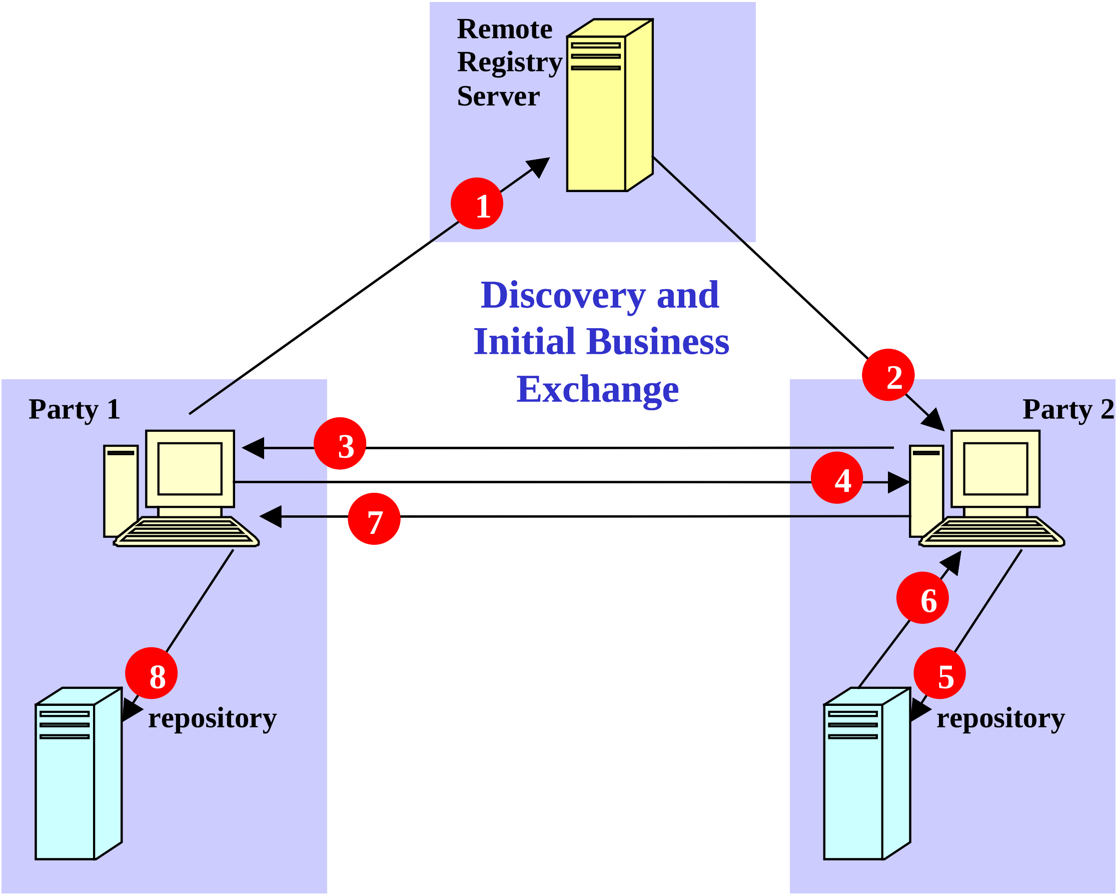


Figure 17: *Trading Partner Discovery.*

21.0 Messaging Service Functionality

21.1 Overview

The *ebXML Messaging Service* provides for the secure, reliable exchange of *ebXML Messages* between *Parties* over various transport protocols (SMTP, HTTP/S, FTP, etc.). The Messaging Service specification defines the MIME packaging and ebXML message *Header* information required by the *ebXML Messaging Service* to enable interoperable exchange of ebXML compliant messages.

The *ebXML Messaging Service* supports all messaging between distributed *Components* of the ebXML system including Registry/Repository and ebXML compliant applications.

It utilizes and enforces the "rules of engagement" defined in a *Trading Partner Agreement (TPA).* The *ebXML Messaging Service* supports simplex (one-way) and request/response (either synchronous or asynchronous) message exchange and can be mapped onto any transport service capable of transporting MIME (further discussed in section).

The *ebXML Messaging Service* is conceptually broken down into three parts: (1) an abstract service interface, (2) functions provided by the Messaging Service Layer, and (3) the mapping to underlying transport service(s). The relation of the abstract interface, Messaging Service Layer, and transport service(s) are shown in Figure 18 below:

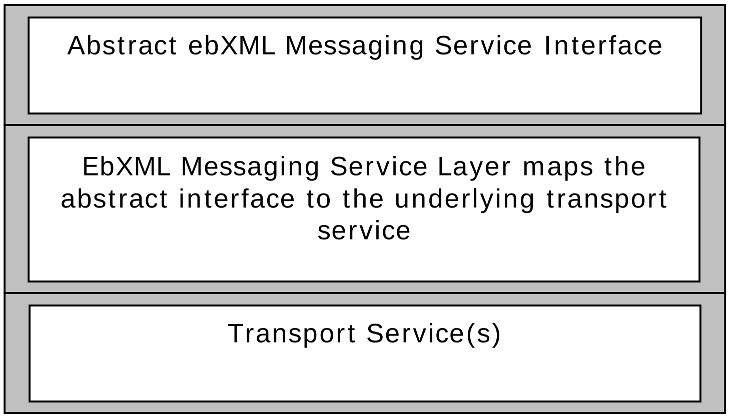

Abstract ebXML Messaging Service Interface
EbXML Messaging Service Layer maps the abstract interface to the underlying transport service
Transport Service(s)

Figure 18: *ebXML Messaging Service*

21.2 Abstract ebXML Messaging Service Interface

The ebXML Message Service provides ebXML with an abstract interface whose functions, at an abstract level, include:

- Send – send an *ebXML Message* – values for the parameters are derived from the *ebXML Message Headers*.
- Receive – indicates willingness to receive an *ebXML Message.*
- Notify – provides notification of expected and unexpected events.
- Inquire – provides a method of querying the status of the particular ebXML Message interchange.

21.3 ebXML Messaging Service Layer Functions

The ebXML Messaging Service Layer provides all of the services and functionality needed to manage the entire lifecycle of *ebXML Messages*. Functions provided by this layer include:

- The ability to construct and validate proper *ebXML Messages.*
- Enforcing the "rules of engagement" as defined by two parties in a *Trading Partner Agreement* (including security and *Business Process* functions related to message delivery). The *Trading Partner Agreement* defines the acceptable behavior by which each *Party* agrees to abide. The definition of these ground

rules can take many forms including formal *Trading Partner Agreements*, interactive agreements established at the time a business transaction occurs (e.g. buying a book online), or other forms of agreement. There are Messaging Service Layer functions that enforce these ground rules. Any violation of the ground rules result in an error condition, which is reported using the appropriate means.

- Support for the following reliability options:
    - "Best Effort" delivery
    - "Once and only once" delivery
    - Synchronous or Asynchronous messaging
    - Request/Response processing
    - Fire 'n forget processing
    - Allow for "multiparty" message delivery
- Perform all security related functions including:
    - Identification
    - Authentication (verification of identity)
    - Authorization (access controls)
    - Privacy (encryption)
    - Integrity (message signing)
    - Non-repudiation
    - Logging
- Interface with internal systems including:
    - Routing of received messages to internal systems
    - Error notification
- Administrative services including:
    - Notification, both system-to-system and system-to-human (via pagers or e-mail)
    - Track and report the status of message exchanges for auditing and diagnostic purposes
    - Logging of service related errors
    - Access to *Partner Agreement* information
    - Status inquiry
- Transport bindings:
    - Functions to enable the delivery of messages over various transport services (e.g. SMTP, FTP, HTTP, etc.)

The following diagram depicts a logical arrangement of the functional modules that exist within the ebXML Messaging Service architecture. These modules are arranged in a manner to indicate their inter-relationships and dependencies. This architecture diagram illustrates the flexibility of the ebXML Messaging Service, reflecting the broad spectrum of services and functionality that may be implemented in an ebXML system.

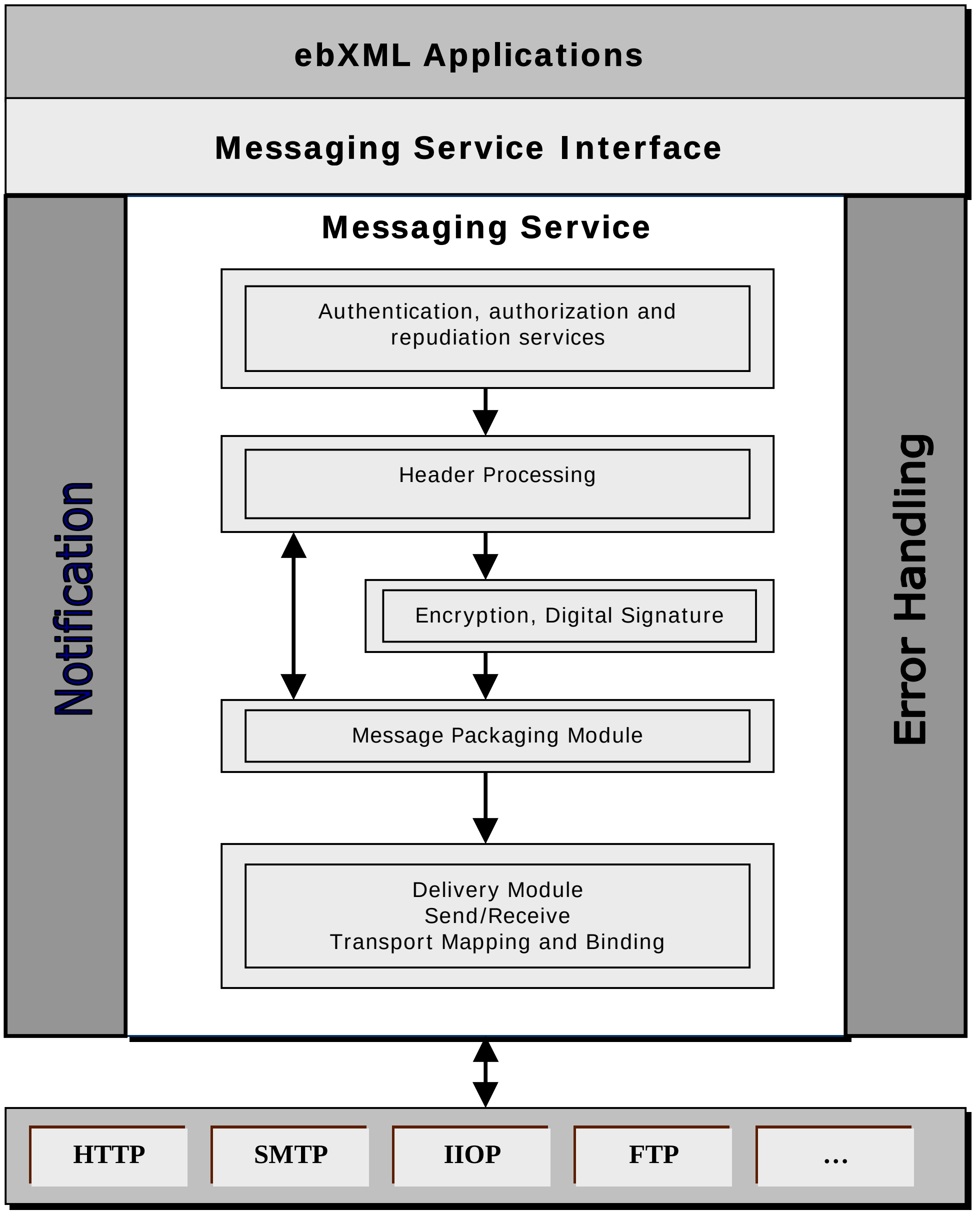


Figure 19: *The Messaging Service Architecture*

21.4 ebXML Message Structure and Packaging

Figure 20 below illustrates the logical structure of an ebXML compliant message.

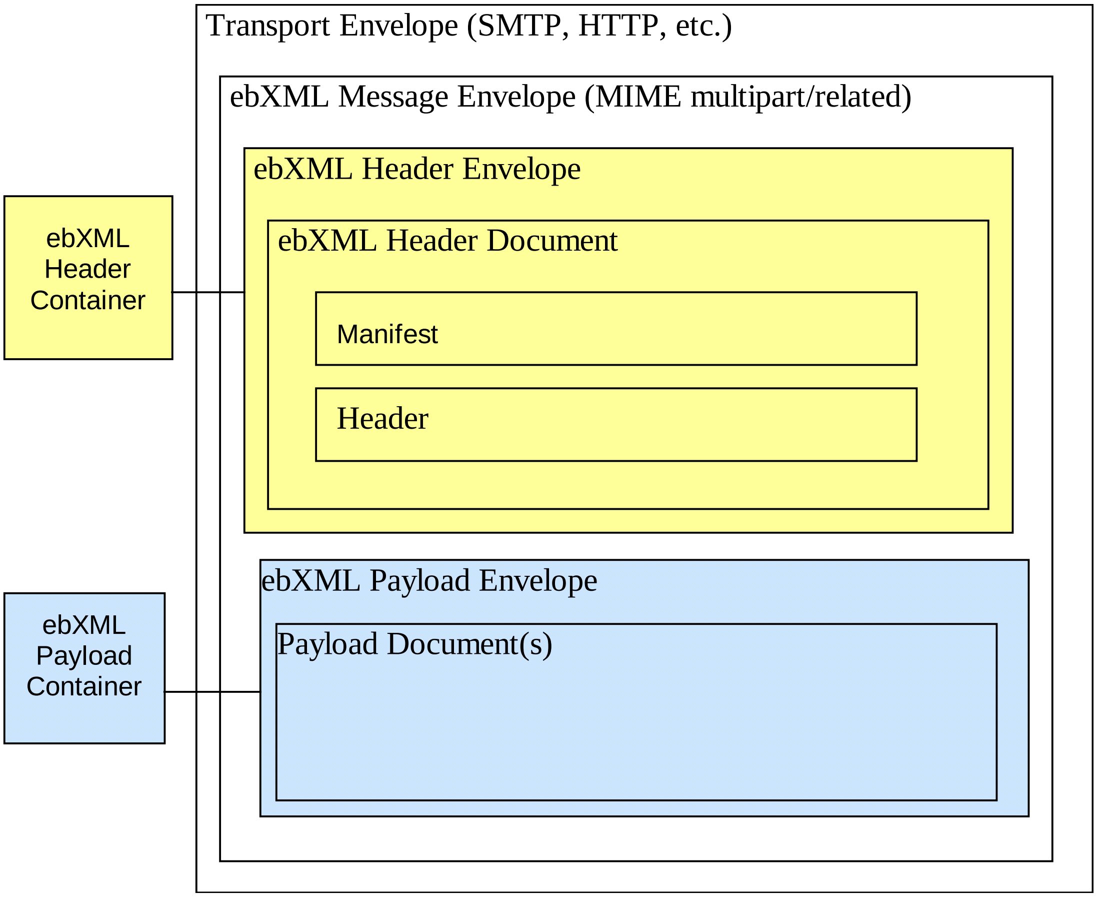


Figure 20: *ebXML Message Structure*

An *ebXML Message* consists of an optional transport protocol specific outer *Communication Protocol Envelope* and a protocol independent *ebXML Message Envelope*. The *ebXML Message Envelope* is packaged using the MIME multipart/related content type. MIME is used as a packaging solution because of the diverse nature of information exchanged between *Partners* in *eBusiness* environments. For example, a complex B2B business transaction between two or more *Trading Partners* might require a payload that contains an array of business documents (*XML* or other document formats), binary images, or other related business objects.

The *ebXML Message Envelope* is used to encapsulate the *Components* of an ebXML compliant message. This structure effectively separates ebXML header information from the payload content of the message. The separation of *Header* and *Payload* containers promotes system efficiency, as the ebXML Messaging Service only needs to access *Header* information to process the message. This provides a flexible mechanism for transparently passing diverse *Payloads* to appropriate business services without having to process them within the Messaging Service framework. It also allows encrypted and/or signed *Payloads* to be exchanged and forwarded with no processing overhead.

The *ebXML Payload Container* is an optional part of an *ebXML Message*. If a *Payload* is present in and *ebXML Message*, the *ebXML Payload Envelope* serves as the container for the actual content (*Payload*) of the *ebXML Message*. The *ebXML Payload Envelope* consists of a MIME header portion and a content portion (the *Payload* itself). The ebXML Messaging Service does not limit in any way the structure or content of payloads.

22.0 Conformance

22.1 Overview

The objectives of this section are to:

a) Ensure a common understanding of conformance and what is required to claim conformance;

b) Promote interoperability and open interchange of *Business Processes* and messages;

c) Promote uniformity in the development of conformance tests.

ebXML conformance is defined in terms of conformance to ebXML, conformance to each of the component specifications for ebXML, and conformance to this (Technical Architecture) specification.

All ebXML specifications shall contain a conformance clause. The conformance clause specifies explicitly all the requirements that have to be satisfied to claim conformance to that specification. These requirements may be applied and grouped at varying levels within each specification.

22.2 Conformance Requirements

Types of conformance requirements can be classified as:

a) Mandatory requirements: these are to be observed in all cases;

b) Conditional requirements: these are to be observed if certain conditions set out in the specification apply;

c) Optional requirements: these can be selected to suit the implementation, provided that any requirement applicable to the option is observed.

Furthermore, conformance requirements in a specification can be stated:

- Positively: they state what shall be done;
- Negatively (prohibitions): they state what shall not be done.

22.3 General Framework of Conformance Testing

The objective of conformance testing is to establish a set of criteria that enable vendors to implement compatible and interoperable systems built on the ebXML foundations. Since ebXML consists of many facets and *Components*, ebXML conformance shall take into account different layers and levels. These levels will be hierarchical and recursive, so conformance to a higher level will include conformance to a lower level.

Implementations and applications shall be tested to verify their conformance to ebXML Specifications.

Publicly available test suites from vendor neutral organizations such as OASIS and NIST should be used to verify the conformance of ebXML implementations, applications, and *Components* claiming conformance to ebXML. This will ensure that they are compliant with the base ebXML criteria. Live benchmark implementations may be available to allow vendors to test their products for interface compatibility and conformance.

Additional items of note include:

a) Extended implementations may include support for more than just the base ebXML protocols, including other popular or emerging formats, such as legacy *EDI* or messaging services such as Java Messaging Service (JMS) implementations.

b) Each ebXML working group will be responsible to coordinate with and determine what it means to conform to their specification and what should be included in the appropriate Conformance test suite(s).

Appendix A: Example ebXML Business Scenarios

Definition

This set of Scenarios defines how ebXML compliant software could be used to implement popular, well-known *eBusiness* models.

Scope

These Scenarios are oriented to properly position ebXML specifications as a convenient mean for Companies to properly run *eBusiness* over the Internet using open standards. They bridge the specifications to real life uses.

Audience

Companies planning to use ebXML compliant software will benefit from these Scenarios because they will show how these companies may be able to implement popular business scenarios onto the ebXML specifications.

List

e) Two Partners set-up an agreement and run the associated electronic exchange.

f) Three or more partners set-up a *Business Process* implementing a supply-chain and run the associated exchanges

g) A Company sets up a Portal which defines a *Business Process* involving the use of external business services.

h) Three or more parties engage in multi-Party *Business Process* and run the associated exchanges.

Scenario 1: Two Partners set-up an agreement and run the associated exchange

In this scenario:

- Each partner defines its own Party Profile.
  Each Party Profile references:
  - One or more existing *Business Process* found in the ebXML Repository
  - One of more Message Definitions. Each Message definition is built from reusable *Components* (*Core Components*) found in the ebXML Repository

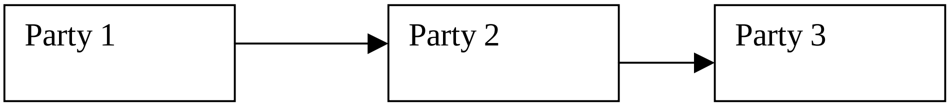
  Each Party Profile defines:
- The Commercial Transactions that the Party is able to engage into
- The Technical protocol (like HTPP, SMTP etc) and the technical properties (such as special encryption, validation, authentication) that the Party supports in the engagement
- The parties acknowledge each other's profile and create a Partner Agreement.
  The Partner Agreement references :
- The relevant Party Profiles
- The Legal terms and conditions related to the exchange
- The parties implement the respective part of the Profile.
  This is done:
- Either by creating/configuring a Business Service Interface.
- Or properly upgrading the legacy software running at their side
  In both cases, this step is about :

- Plugging the Legacy into the ebXML technical infrastructure as specified by the TR&P
- Granting that the software is able to properly engage the stated conversations
- Granting that the exchanges semantically conform to the agreed upon Message Definitions
- Granting that the exchanges technically conform with the underlying ebXML TR&P
- The parties start exchanging messages and performing the agreed upon commercial transactions.

Scenario 2: Three or more partners set-up a Business Process implementing a supply-chain and run the associated exchanges

The simple case of a supply-chain involving two parties can be reconstructed from the Scenario 1.

Here we are dealing with situations where more parties are involved. We consider a *Supply Chain* of the following type :



What fundamentally differs from Scenario 1 is that Party 2 is engaged at the same time with two different parties. The assumption is that the “state” of the entire *Business Process* is managed by each Party, i.e. that each Party is fully responsible of the Commercial Transaction involving it (Party 3 only knows about Party 2, Party 2 knows about Party 3 and Party 1, Party 1 knows about Party 2).

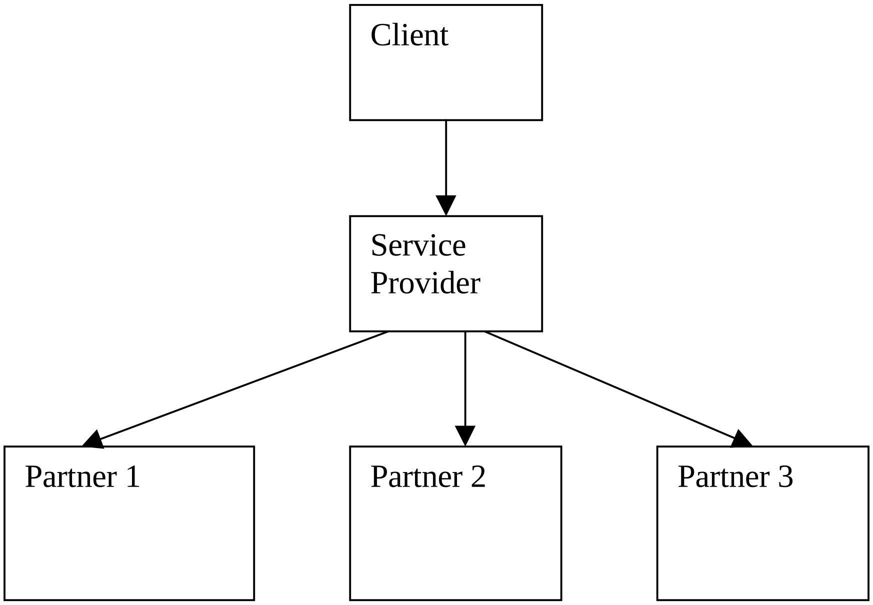
In this scenario:

- Each partner defines its own Party Profile.
  Each Party Profile references:
- One or more existing *Business Process* found in the ebXML Repository
- One of more Message Definitions. Each Message definition is built from reusable *Components* (*Core Components*) found in the ebXML Repository
  Each Party Profile defines:
- The Commercial Transactions that the Party is able to engage into
  Party 2 must be able to support at least 2 Commercial Transactions.
- The Technical protocol (like HTPP, SMTP etc) and the technical properties (such as special encryption, validation, authentication) that the Party supports in the engagement
  the technical requirements for the exchanges with Party 1 and Party 3 may be different. In such case, Party 2 must be able to support different protocols and/or properties.
- The parties acknowledge each other profile and create the relevant Partner Agreements (at least 2 in this Scenario).
  Each Partner Agreement references:
- The relevant Party Profiles
- The Legal terms and conditions related to the exchange
  Party 2 is engaged in 2 Party Agreements.
- The parties implement the respective part of the Profile.
  This is done:
- Either by creating/configuring a Business Service Interface.
- Or properly upgrading the legacy software running at their side
  In both cases, this step is about:

- Plugging the Legacy into the ebXML technical infrastructure as specified by the TR&P
- Granting that the software is able to properly engage the stated conversations
- Granting that the exchanges semantically conform to the agreed upon Message Definitions
- Granting that the exchanges technically conform with the underlying ebXML TR&P
- Party 2 may need to implement a complex Business Service Interface in order to be able to engage with different partners.
- The parties start exchanging messages and performing the agreed upon commercial transactions.
- Party 3 places an order at Party 2
- Party 2 (eventually) places an order with Party 1
- Party 1 fulfills the order
- Party 2 fulfill the order

Scenario 3 : A Company sets up a Portal which defines a Business Process involving the use of external business services

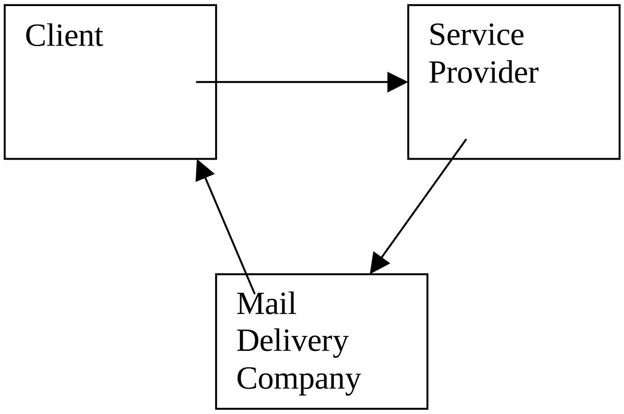
This is the Scenario describing a Service Provider. A "client" asks the Service Provider for a Service. The Service Provider fulfills the request by properly managing the exchanges with other partners, which provide information to build the final answer.
In the simplest case, this Scenario could be modeled as follows:



This is an evolution of Scenario 2. The Description of this scenario is omitted.

Scenario 4: Three or more parties engage in multi-party Business Process and run the associated exchanges

This Scenario is about 3 or more Parties having complex relationships. An example of this is the use of an external delivery service for delivering goods.



In this Scenario, each Party is involved with more than one other Party but the relationship is not linear. The good which is ordered by the Client with the Service Provider is delivered by a 3rd Party.

In this scenario:

- Each partner defines its own Party Profile.
  Each Party Profile references:
- One or more existing *Business Process* found in the ebXML Repository
- One of more Message Definitions. Each Message definition is built from reusable *Components* (*Core Components*) found in the ebXML Repository
  Each Party Profile defines:
- The Commercial Transactions that the Party is able to engage into
  In this case, each Party must be able to support at least 2 Commercial Transactions.
- The Technical protocol (like HTPP, SMTP etc) and the technical properties (such as special encryption, validation, authentication) that the Party supports in the engagement.
  In case the technical infrastructure underlying the different exchanges differs, each Party must be able to support different protocols and/or properties. (an example is that the order is done through a Web Site and the delivery is under the form of an email).
- The parties acknowledge each other profile and create a Partner Agreement.
  Each Party, in this Scenario, must be able to negotiate at least 2 Agreements.
  The Partner Agreement references :
- The relevant Party Profiles
- The Legal terms and conditions related to the exchange
  Each Party is engaged in 2 Party Agreements.
- The parties implement the respective part of the Profile.
  This is done:
- Either by creating/configuring a Business Service Interface.

- Or properly upgrading the legacy software running at their side

In both cases, this step is about:

- Plugging the Legacy into the ebXML technical infrastructure as specified by the TR&P
- Granting that the software is able to properly engage the stated conversations
- Granting that the exchanges semantically conform to the agreed upon Message Definitions
- Granting that the exchanges technically conform with the underlying ebXML TR&P
- All Parties may need to implement complex Business Service Interfaces to accommodate the differences in the Party Agreements with different Parties.
- The parties start exchanging messages and performing the agreed upon commercial transactions.
- The Client places an Order at the Service Provider
- The Service Provider Acknowledges the Order with The Client
- The Service Provider informs the Mail Delivery Service about a good to be delivered at the Client
- The Mail Delivery Service delivers the good at the Client
- The Clients notifies the Service Provider that the good is received.

Disclaimer

The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this design.

Copyright Statement