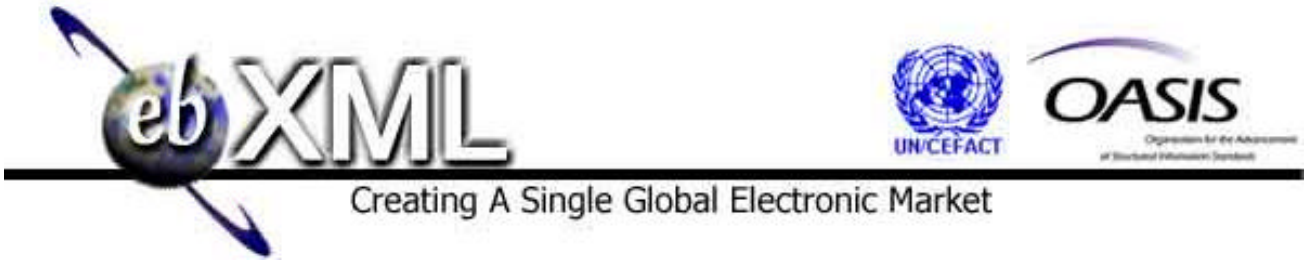


ebXML Technical Architecture Specification

ebXML Technical Architecture Project Team

22 December 2000

1 Status of this Document

This document is a working DRAFT for the *eBusiness* community. Distribution of this document is unlimited. This document will go through the formal *Quality Review* Process as defined by the *ebXML Requirements Document*. The formatting for this document is based on the Internet Society's Standard RFC format.

This version:

ebXML_TA_v0.95.doc

***Latest version*:**

ebXML_TA_v0.95.doc

Previous version:

ebXML_TA_v0.94.doc

2 ebXML Technical Architecture Participants

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

Team Lead: Anders Grangard, EDI France

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

Participants: Colin Barham, TIE
Al Boseman, ATPCO
Christian Barret, GIP-MDS
Dick Brooks, Group 8760
Cory Casanave, DataAccess Technologies
Robert Cunningham, Military Traffic Management Command, US Army
Christopher Ferris, Sun Microsystems
Kris Ketels, SWIFT
Piming Kuo, Worldspan

Kyu-Chul Lee, Chungnam National University
Henry Lowe, OMG
Melanie McCarthy, General Motors
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

4 Introduction

4.1 Summary of Contents of Document

4.2 Audience and Scope

This document is intended primarily for the *ebXML Project Teams* to help guide their work. Secondary audiences MAY include software implementers, international standards bodies, and other industry organizations.

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. It SHOULD be used as a roadmap to learn: (1) what ebXML is, (2) what problems ebXML solves, and (3) core ebXML functionality and architecture.

4.3 Related Documents

As mentioned above, other documents provide detailed definitions of some of the components of ebXML and of their inter-relationship. They include ebXML specifications on the following topics:

1. Requirements
2. Business Process and Information Meta Model
3. Core Components
4. Registry and Repository
5. Trading Partner Information
6. Messaging Services

These specifications are available for download at http://www.ebxml.org.

4.4 Normative References

The following standards contain provisions that, 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.

RFC 2119
W3C XML v1.0 Second Edition Specification
ISO/IEC 14662: Open-edi Reference Model
ISO 11179/3 Metadata Repository
ISO 10646: Character Encoding
ISO 8601:2000 Date/Time/Number Data typing
DC 128 GUID

4.5 Document Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

The following conventions are used throughout this document:

- *Capitalized Italics* words are defined in the ebXML Glossary.
- [NOTES: are used to further clarify the discussion or to offer additional suggestions and/or resources]

- [NOTES: in red represent outstanding issues that need further discussion and resolution within the Technical Architecture Project Team]

5 Design Objectives

5.1 Problem Description & Goals for ebXML

For over 25 years *Electronic Data Interchange (EDI)* has given companies the prospect of eliminating paper documents, reducing costs, and improving efficiency by exchanging business information in electronic form. Ideally, companies of all sizes could conduct *eBusiness* in a completely ad hoc fashion, without prior agreement of any kind. But this vision has not been realized with *EDI*; only large companies are able to afford to implement it, and much *EDI*-enabled *eBusiness* is centered around a dominant enterprise that imposes proprietary integration approaches on its trading partners.

In the last few years, the *Extensible Markup Language (XML)* has rapidly become the first choice for defining data interchange formats in new *eBusiness* applications on the Internet. Many people have interpreted the XML groundswell as evidence that "*EDI* is dead" – made completely obsolete by the XML upstart -- but this view is naïve from both business and technical standpoints.

EDI implementations encode substantial experience in business processes, and companies with large investments in *EDI* integration will not abandon them without good reason. XML might enable more open, more loosely-coupled, and more object- or component-oriented systems than *EDI*. XML might enable more flexible and innovative "eMarketplace" business models than *EDI*. But the challenges of designing messages that meet business process requirements and standardizing their semantics are independent of the syntax in which the messages are encoded.

The ebXML specifications provide a framework in which *EDI's* substantial investments in business processes can be preserved in an architecture that exploits XML's new technical capabilities.

5.2 Caveats and Assumptions

This specification is designed to provide a high level overview of ebXML, and as such, does not provide the level of detail required to build ebXML applications, components, and related services. Please refer to each of the respective ebXML Project Team Specifications to get the level of detail.

5.3 Design Conventions for ebXML Specifications

[NOTE: waiting for wording from Nikola]

6 ebXML System Overview

Figure 1 below shows a high level conceptual model for two *Trading Partners*, first configuring and then engaging in a simple business transaction and interchange. This model is provided as an example of the process and steps that MAY be REQUIRED to configure and deploy ebXML applications and related system components. These components MAY be implemented in an incremental manner. The ebXML specifications are not limited to this simple model, provided here as quick introduction to the concepts. Specific ebXML implementation examples are described in Appendix A.

The conceptual overview described below introduces the following concepts and underlying architecture:

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 Meta 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* that are exchanged between their respective *Business 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 mutually agreed upon business arrangement which MAY be derived from information provided by each participant from item 3 above.
6. A standardized business *Messaging Service* that 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 business arrangement.

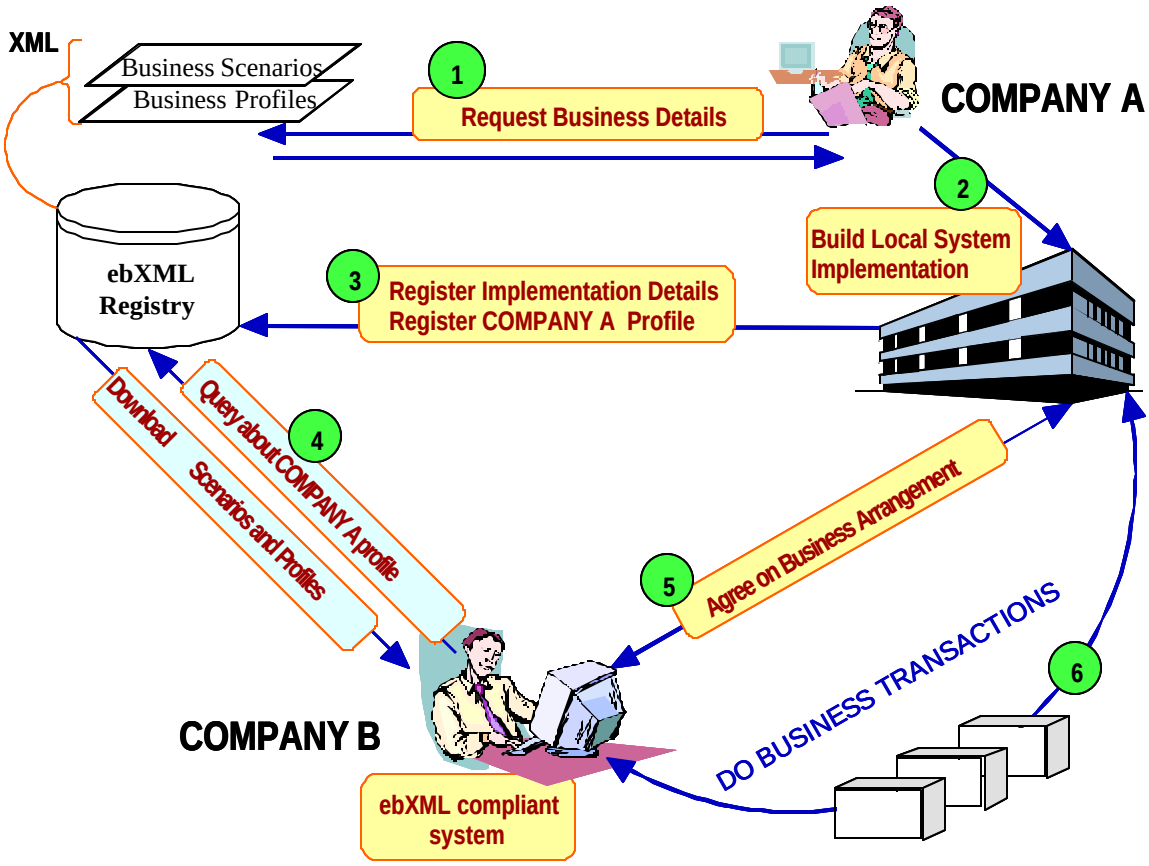


Figure 1 *- a high level overview of the interaction of two companies conducting eBusiness using ebXML.*

In Figure 1, Company A has become aware of an *ebXML Registry* that is accessible on the Internet (Figure 1, step 1). Company A, after reviewing the contents of the *ebXML Registry*, decides to build and deploy its own ebXML compliant application (Figure 1, step 2). It SHOULD be noted that custom software development is not a necessary prerequisite for ebXML participation. ebXML compliant applications and components MAY also be commercially available as shrink-wrapped solutions.

Company A then submits its own implementation details, reference links, and *Business Profile information* to the *ebXML Registry* (Figure 1, step 3). The business profile submitted to the *ebXML Registry* describes the company's ebXML capabilities and constraints, as well as its supported business processes. These business scenarios are *XML* versions of the *Business Processes* and associated information parcels (e.g. a sales tax calculation) that the company is able to engage in. After receiving verification that the format and usage of a business scenario is correct, an acknowledgment is sent to Company A by the *ebXML Registry* (Figure 1, step 3).

Company B discovers the business scenarios supported by Company A in the ebXML Registry (Figure 1, step 4). Company B sends a request to Company A stating that they would like to engage in a business transaction using ebXML (Figure 1, step 5). Company

B acquires a shrink-wrapped application that is ebXML compliant. Company A knows that its business scenarios and profiles are compliant with the ebXML infrastructure based on the information available in the ebXML specifications.

Before engaging in that the scenario Company B submits a proposed business arrangement directly to Company A's ebXML compliant software interface. The proposed business arrangement outlines the mutually agreed upon business scenarios and specific agreements on who it wants to conduct business transactions with Company A. The business arrangement also contains information pertaining to the messaging requirements for transactions to take place, contingency plans, and security-related requirements (Figure 1, step 5). Company A accepts the business agreement which then triggers an acknowledgement message that is sent directly to Company B's ebXML software application (Figure 1, step 5). Company A and B are now ready to engage in *eBusiness* using ebXML (Figure 1, step 6).

7 ebXML Architecture Reference Model

7.1 Overview

The *ebXML Architecture Reference Model* uses the following two views to describe the relevant aspects of *eBusiness* transactions. This model is based upon the Open-edi Reference Model, ISO 14662.

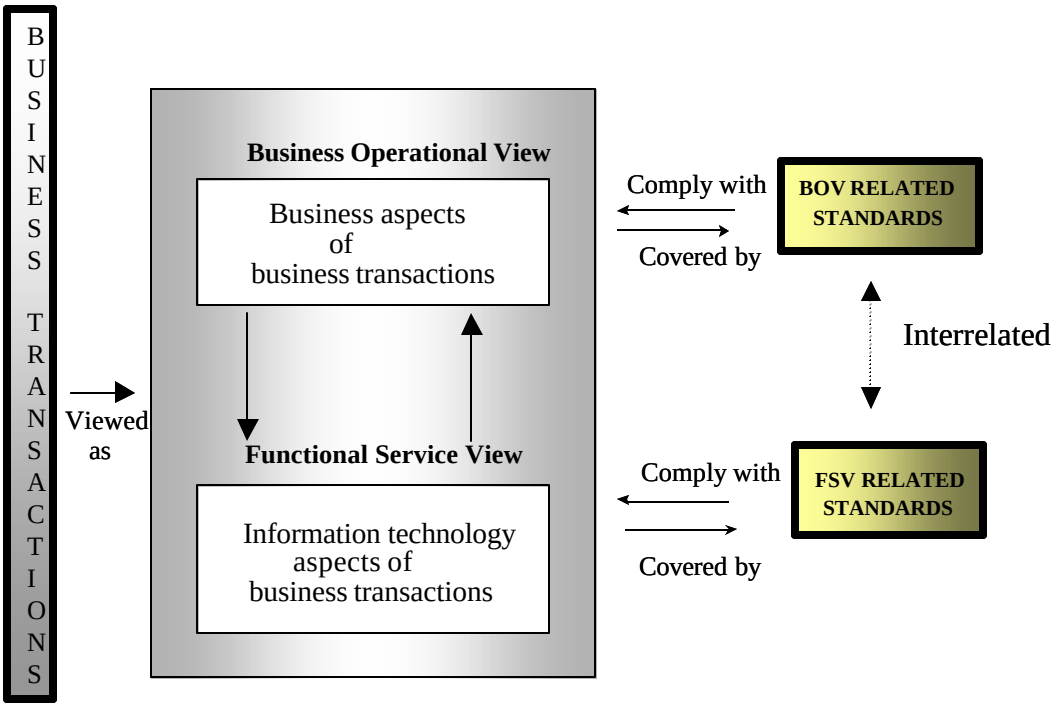


***Figure 2** - ebXML Reference Model*

The ebXML architecture is broken down into the *Business Operational View (BOV)* and the supporting *Functional Service View (FSV)* described above. The assumption for

ebXML is that the *FSV* serves as a reference model that MAY be used by commercial software vendors to help guide them during the development process. The underlying goal of the *ebXML Reference Model* is to provide a clear distinction between the operational and functional views, so as to ensure the maximum level of system interoperability and backwards compatibility with legacy systems (when applicable). As such, the resultant *BOV*-related standards provide the business and object class models needed to construct ebXML compliant applications and components.

While business practices from one 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 interoperable ebXML applications and components. The *BOV* and *FSV* are described in more detail below.

The *BOV* addresses:

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

b) The architecture for business transactions, including:

- operational conventions;
- agreements and arrangements;
- 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 and *Messaging Services.*

This includes, but is not limited to:

- Capabilities for implementation, discovery, deployment and run time scenarios;
- User *Application* interfaces;
- Data transfer infrastructure interfaces;
- *Protocols* for enabling interoperability of XML vocabulary deployments from different organizations.

7.2 ebXML Business Operational View

The modeling techniques described in this section are not mandatory requirements for participation in ebXML compliant business transactions. Figure 3 below provides a detailed view of the ebXML *BOV*.

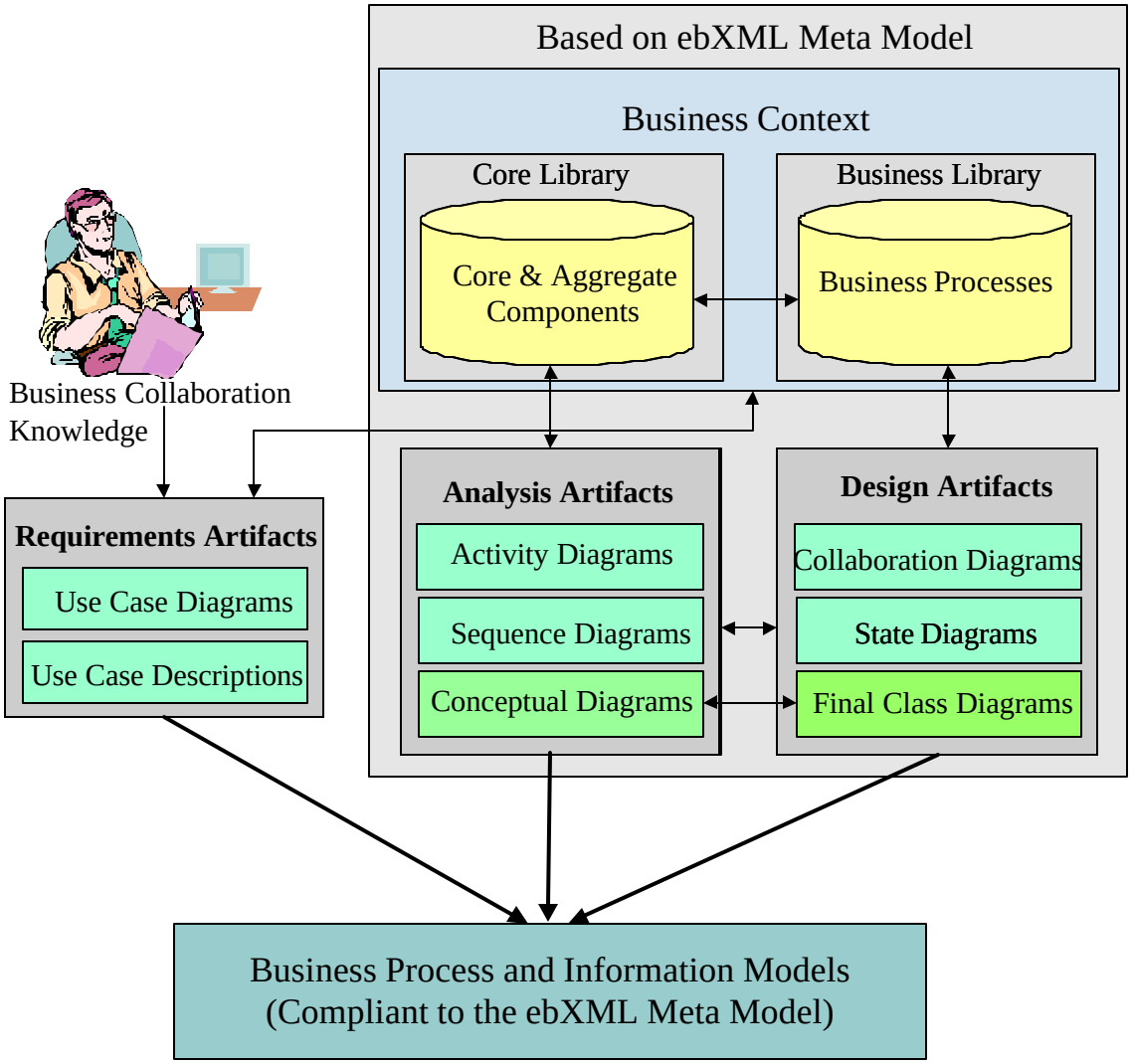


Figure 3 *- the Business Operational View*

In Figure 3 above, *Business Collaboration Knowledge* is captured in a *Core Library*. The *Core Library* contains data and process definitions, including relationships and cross-references, as expressed in business terminology that MAY be tied to an accepted industry classification scheme or taxonomy. The *Core Library* 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 that describe the problem using *Use Case Diagrams and Descriptions*. If *Core Library* entries are available from an ebXML

compliant *Registry* they will be utilized, otherwise new *Core Library* entries will be created and registered in an ebXML compliant *Registry*.

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 *Core Library*. 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 class view diagram from the analysis phase will undergo harmonization to align it with other models in the same industry and across others.

In ebXML interoperability is achieved by applying *Business Objects* across all class models. The content of the *Business Library* is created by analyzing existing *Business Objects* as used by many industries today in conjunction with the *Core Library* content and ebXML selected modeling methodology.

7.3 ebXML Functional Service View

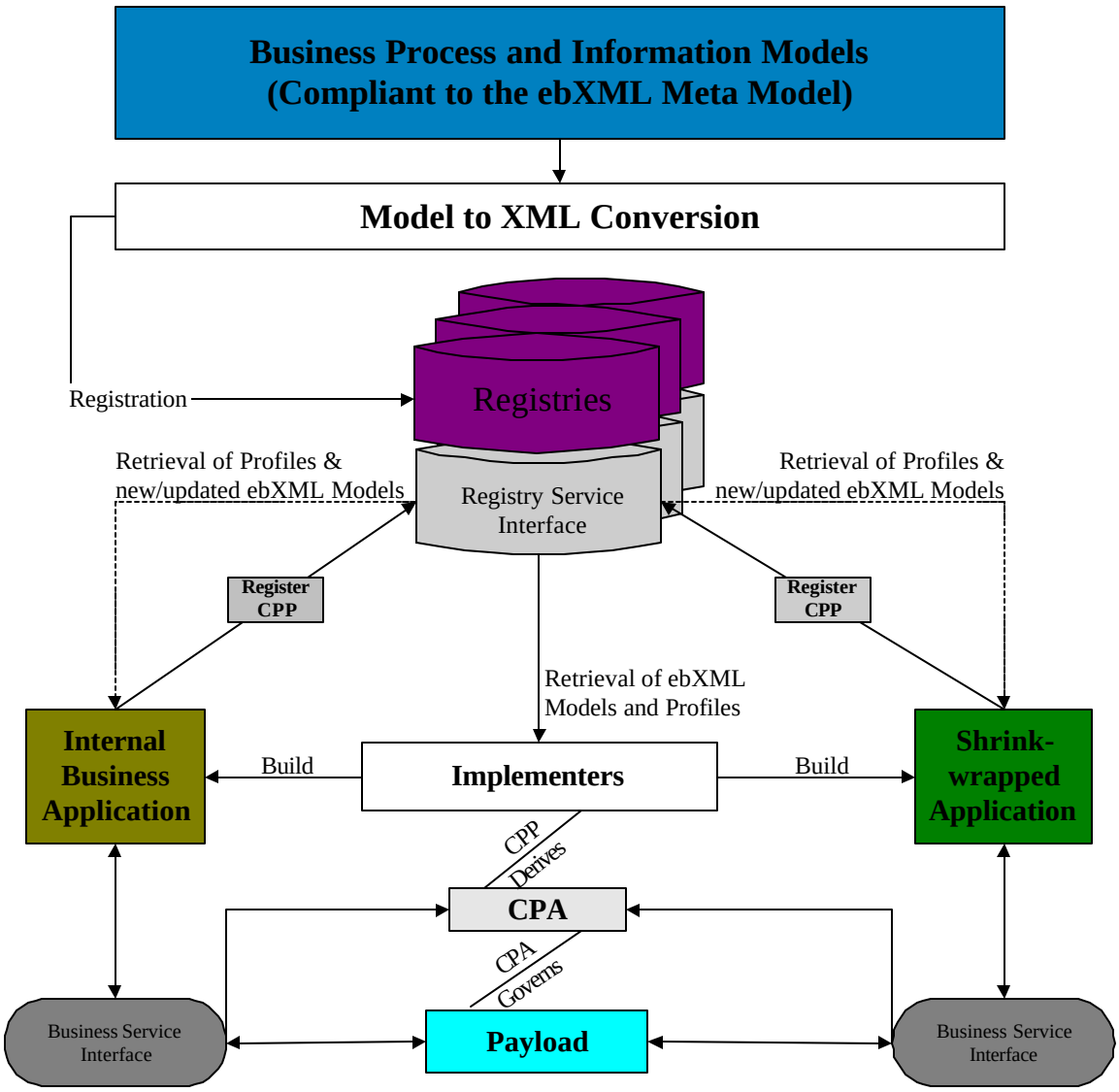


Figure 4 *- ebXML Functional Service View*

As illustrated in Figure 4 above, the *ebXML Registry* system is an important part of ebXML. It serves as the storage facility for the *Business Process and Information Meta Models* developed by industry groups, *SMEs*, and other organizations. In order to store the models in the *Registry*, they are converted to XML (e.g. XML DTD, Schema, etc.). This *XML*-based business information SHALL be expressed in a manner that will allow discovery down to the atomic data 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 System*. *UID* keys are REQUIRED references for all ebXML content. *Globally Unique Identifiers* (DC 128 - GUID) MAY be used to ensure that *Registry* entries are truly globally unique, and thus when systems query a *Registry* for a *GUID*, one and only one result SHALL be retrieved.

To facilitate semantic recognition of *Business and Information Meta Models*, the *Registry* system SHALL provide a mechanism for incorporating human readable descriptions for *Registry* items. Existing *Business Process and Information Models* (e.g. RosettaNet PIPs) SHALL be assigned *UID* keys when they are registered in an ebXML compliant *Registry* system. These *UID* keys MAY be implemented in physical *XML* syntax in a variety of ways. These mechanisms include, but are not limited to:

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

Components in ebXML MUST facilitate multilingual support. A *UID* reference is particularly important here as it provides a language neutral reference mechanism. To enable multilingual support, the ebXML specification SHALL 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 9.4 for details of this architecture).

8 ebXML Functional Phases

8.1 Overview

8.1.1 The Implementation Phase

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

8.1.2 The Discovery and Retrieval Phase

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

8.1.3 The Run Time Phase

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

8.2 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(via HTTP, FTP, etc.). The *Trading Partner* studies the ebXML specification and subsequently requests to download the *Core Library* and the *Business Object Library*. The *Trading Partner* MAY also request other *Trading Partners*' *Business Process* information (stored in it business profile) for analysis and review. The *Trading Partner* MAY also submit its own *Business Process* information to an ebXML compliant *Registry* system.

Figure 5 below, illustrates a basic interaction between an ebXML *Registry* system and a *Business Service Interface*.

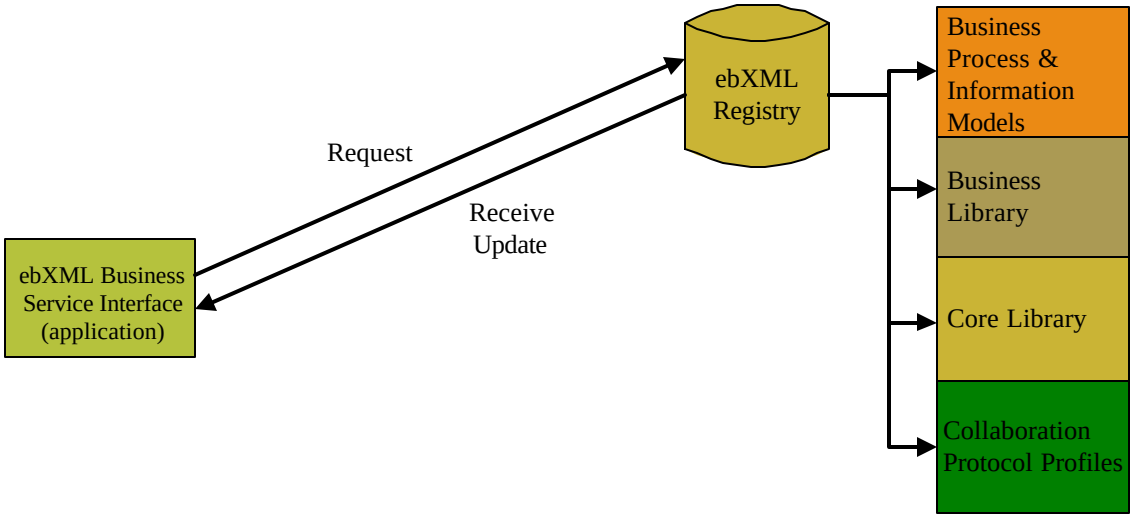


Figure 5 *- Functional Service View: Implementation Phase*

8.3 Discovery and Retrieval Phase

A *Trading Partner* who has implemented an ebXML *Business Service Interface* can now begin the process of discovery and retrieval (Figure 6 below). One possible discovery method MAY be to request the *Trading Partner Profile* of another *Trading Partner*. Requests for updates to *Core Libraries*, *Business Object Libraries* and updated or new *Business Process* and information models are also methods that 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*.

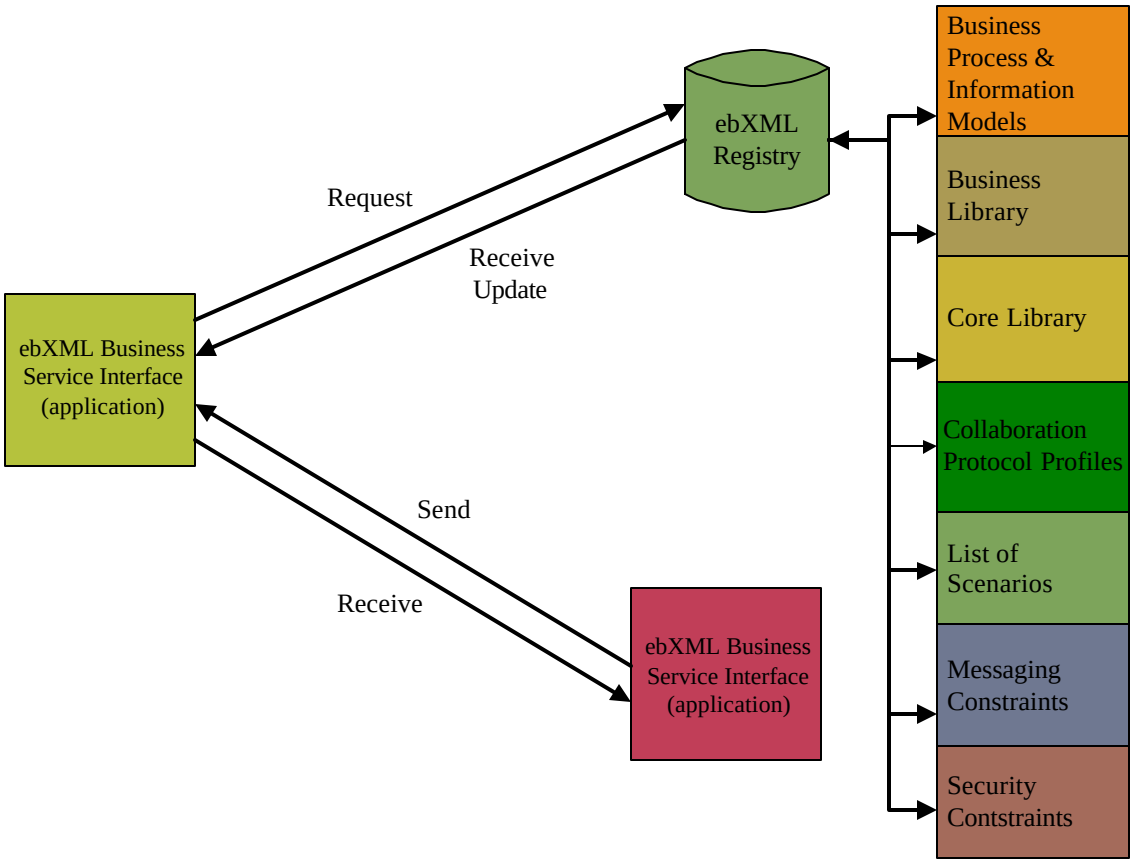


Figure 6 - *Functional Service View: Discovery and Retrieval Phase*

8.4 Run Time Phase

In the Run Time Phase, ebXML messages are being exchanged between *Trading Partners* utilizing the *ebXML Messaging Service*. If it becomes necessary to make calls to the *Registry* during the Run Time, this will be considered as a reversion to the Discovery and Retrieval Phase.

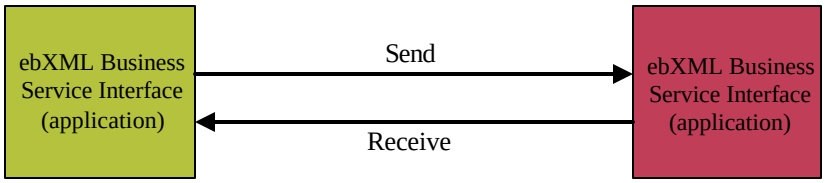


Figure 7 - *Functional Service View: Run Time Phase*

9 ebXML Infrastructure

9.1 Trading Partner Information [CPP and CPA's]

9.1.1 Introduction

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 through the use of a *Collaboration Protocol Profile (CPP).* The CPP 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*.

To facilitate the process of conducting electronic business, organizations also 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 *Collaboration Protocol Agreement (CPA).* A *CPA* is a document that represents the intersection of two CPP's and is mutually agreed upon by both Trading Partners who wish to conduct *eBusiness* using ebXML.

9.1.2 CPP Formal Functionality

The *CPP* describes the specific capabilities that a *Trading Partner* supports as well as the *Service Interface* requirements that need to be met in order to exchange business documents with that *Trading Partner*. Each *Trading Partner* SHALL register one and only one *CPP* in an ebXML compliant Registry system. The CPP contains essential information about the Trading Partner, which MAY include (but is not limited to): contact information, industry classification, supported business processes, and interface requirements.

CPP's describe the *Business Processes* that a given *Trading Partner* supports, plus the *Messaging Service* interface requirements that the given *Trading Partner* will use as a support mechanism for such collaborations. *CPP's* MAY optionally include security and other implementation specific details. Each ebXML compliant *Trading Partner* SHALL register their *CPP* in an ebXML compliant *Registry* system, thus providing a discovery mechanism that allows *Trading Partners* to (1) find one another, (2) discover the *Business Process* that other *Trading Partners* support.

9.1.3 CPA Formal Functionality

A *Collaboration Protocol Agreement* (*CPA*) describes: (1) the *Messaging Service* (technology), and (2) the *Business Process* (application) requirements that are agreed upon by two or more *Trading Partners*. Conceptually, ebXML supports a three level view of narrowing subsets to arrive at *CPA's* for transacting *eBusiness.* The outer-most scope relates to all of the possibilities that a *Trading Partner* could do, with a subset of that of what a *Trading Partner* is capable of doing, with a subset of what a *Trading Partner* "will" do.

A *CPA* contains the *Messaging Service* interface requirements as well as the implementation details pertaining to the mutually agreed upon *Business Processes* that both *Trading Partners* agree to use to conduct *eBusiness*. *Trading Partners* MAY decide to register their *CPA's* in an ebXML compliant *Registry* system, but this is not a mandatory part of the *CPA* creation process.

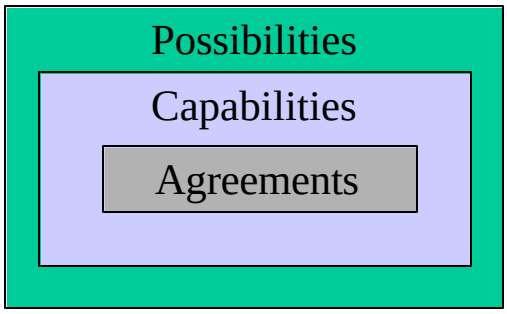


Figure 8 *- Three level view of CPA's*

Business Collaborations are the first order of support that can be claimed by ebXML *Trading 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, such as an ebXML *Registry* or other available service. Figure 9 below outlines the scope for *Collaboration Protocol Agreements* within ebXML.

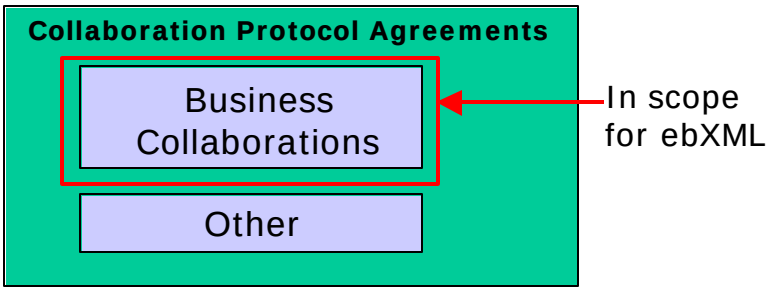


Figure 9 *- Scope for CPA's*

9.1.4 CPP Interfaces

Business Process: The CPP must be capable of referencing one or more business processes supported by the entity owning the CPP instance. The CPP must also reference the Roles within that BP that the user is capable of assuming.

The CPP must be capable of referencing, either directly or indirectly, the CPA for each supported Business Process.

The CPP must be capable of being stored and retrieved from a Registry Mechanism

The CPP must be capable of being carried in the payload of the ebXML Messaging service. A CPP may also describe binding details that are used to build an ebXML message header.

9.1.5 CPA Interfaces

A CPA has an Interface to a software component used by a Trading Partner via the ebXML Messaging mechanism.

A CPA has an interface to the CPP by the fact it must narrow down the Trading Partners Capabilities into what the Trading Partner "will" do. What a Trading Partner "will" do must be within that Trading Partners' capabilities hence an abstract interface between the two documents.

A CPA has an interface to a Business Process document by the fact it may be reached and referenced for each Business Process.

A CPA also may be stored in a Registry mechanism, hence an implied ability to be stored and retrieved is present.

9.1.6 Non-Normative Implementation Details [CPP and CPA's]

A *CPA* is negotiated after the discovery process and is essentially a snapshot of the *Messaging Services* and *Business Process* related information that two or more *Trading Partners* agree to use to exchange business information. If any parameters contained within an accepted *CPA* change after the agreement has been executed, a new *CPA* SHALL be negotiated between all parties.

9.2 Business Process and Information Modeling

9.2.1 Introduction

The ebXML *Business Process and Information Meta Model* is a mechanism that allows *Trading Partners* to capture the details for a specific business scenario using a consistent modeling methodology. A *Business Process* describes in detail how *Trading Partners* take on roles, relationships and responsibilities to facilitate interaction with other *Trading Partners* in shared *Business Processes*. The interaction between roles takes place as a choreographed set of *Business Transactions*. Each *Business Transaction* is expressed as an exchange of electronic *Business Documents.* The sequence of the exchange is defined by the *Business Process*, messaging and security considerations. *Business Documents* are composed from re-useable business information components (see "Relationships to Core Components" under 9.2.3 "Interfaces" below). At a lower level, *Business Processes* can be composed of re-useable *Core Processes*, and *Business Objects* can be composed of re-useable *Core Components.*

The ebXML *Business Process and Information Meta Model* supports requirements, analysis and design viewpoints that provide a set of semantics (vocabulary) for each viewpoint and forms the basis of specification of the objects and artifacts that are required to facilitate business process and information integration and interoperability.

An additional view of the Meta Model, the *Specification Schema*, is also provided to support the direct specification of the nominal set of elements necessary to configure a runtime system in order to executive a set of ebXML business transactions. By drawing out modeling elements from several of the other views, the *Specification Schema* forms a semantic subset of the ebXML *Business Process and Information Meta Model.* The *Specification Schema* is available in two stand-alone representations, a UML profile, and a DTD.

The relationship between the ebXML *Business Process and Information Meta Model* and the ebXML *Specification Schema* can be shown as follows:

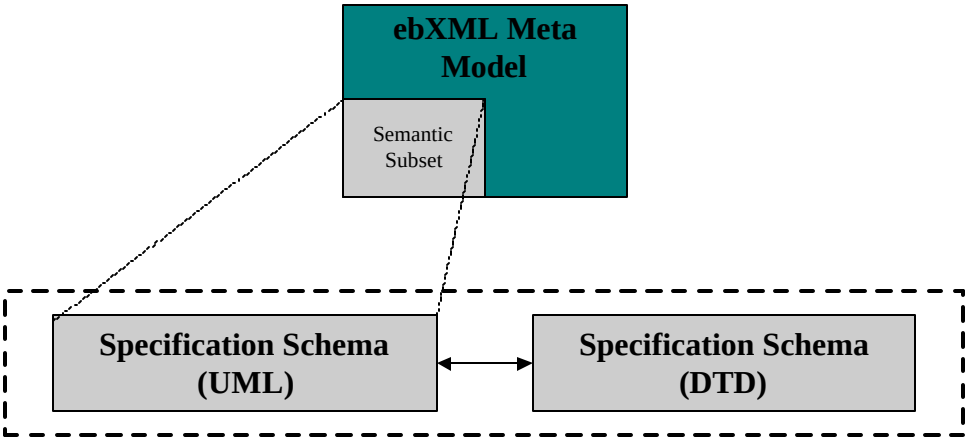


Figure 10 - *ebXML Meta Model - Semantic Subset*

The *Specification Schema* supports the specification of *Business Transactions* and the choreography of *Business Transactions* into *Business Collaborations*. Each *Business Transaction* can be implemented using one of many available standard patterns. These patterns determine the actual exchange of messages and signals between the partners to achieve the required legally binding electronic commerce transaction. To help specify the patterns *the Specification Schema* is accompanied by a set of standard patterns, and a set of modeling elements common to those standard patterns. The full specification, thus, of a business process consists of a business process model specified against the *Specification Schema* and an identification of the desired pattern(s). This full specification is then the input to the formation of *Trading Partner Collaboration Profiles* and *Collaboration Agreements.* This can be shown as follows:

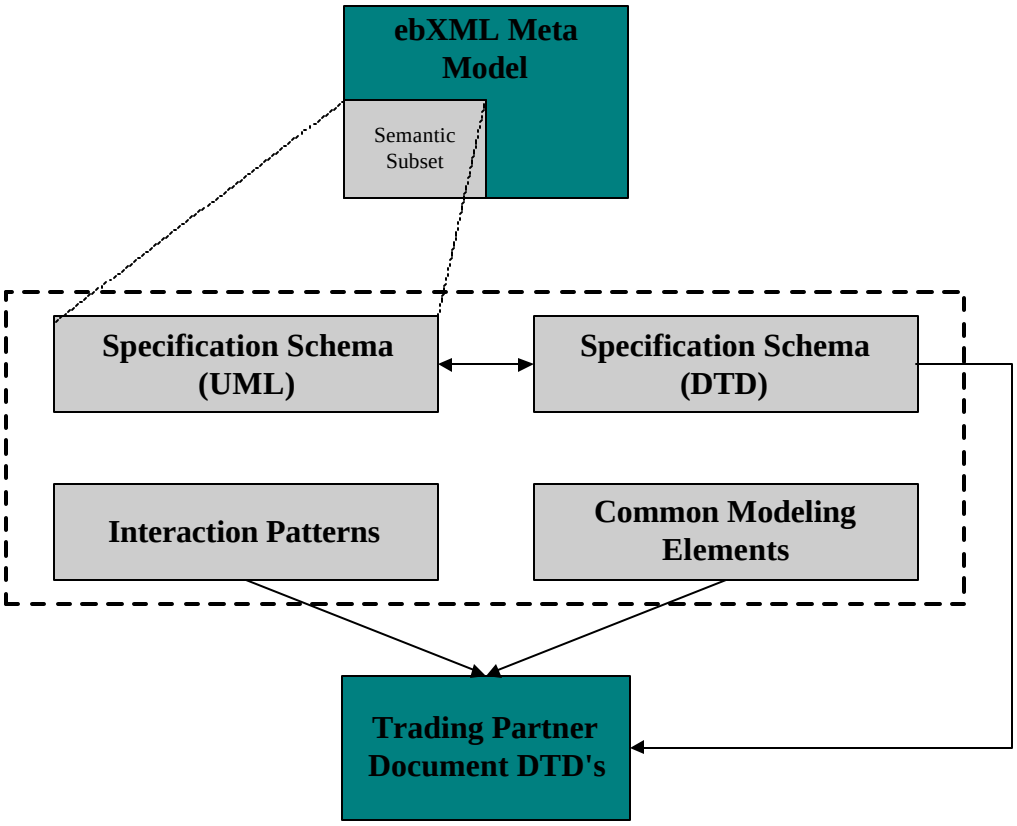


Figure 11 *- ebXML Meta Model*

There are no formal requirements to mandate the use of a modeling language to compose new *Business Processes*, however, if a modeling language is used to develop *Business Processes*, it SHALL be the *Unified Modeling Language (UML)*. This mandate ensures that a single, consistent modeling methodology is used to create new *Business Processes*.

To further facilitate the creation of consistent business processes and information models, ebXML will define a core set of *Business Processes* in parallel with a *Core Library*. It is possible that users of the ebXML infrastructure MAY wish to extend this set or use their own *Business Processes*.

9.2.2 Formal Functionality

The interpretation of a *Business Process* document instance SHALL be in a form that will allow both humans and applications to read the information. This is necessary to facilitate a gradual transition to full automation of business interactions.

The *Business Process* SHALL be storable and retrievable in a *Registry* mechanism. *Business Processes* MAY be registered in an ebXML *Registry* in order to facilitate discovery.

To be understood by an application, a *Business Process* SHALL be expressible in XML syntax. A *Business Process* SHALL be comprised of an information model or XML-

based representation of that model, that is capable of expressing the following types of information:

- Choreography for the exchange of document instances.
- References to *Metadata* (possibly *DTD's* or *Schemas*) that add structure to business data.
- Definition of the roles for each participant in a *Business Process.*
- May reference supporting *Metadata.*
- Provide a context constraint that affects *Core Components*
- Provide the framework for establishing *CPAs*
- The domain owner of the *Business Process* and contact information.

[NOTE: this is not an inclusive list.]

9.2.3 Interfaces

The interface from a *Business Process* to its associated *Business Process and Information Meta Model* to other parts of the ebXML Architecture, or to other tools and environments is outside the scope of the ebXML specifications.

Relationship to CPP and CPA

The *CPP* instance of a *Trading Partner* defines that partner's functional and technical capability to support zero, one, or more *Business Processes* and one or more roles in each process.

The agreement between two *Trading Partners* defines the actual conditions under which the two partners will conduct business transactions together. Accordingly, the interface between the *Business Process* and its associated *Business Process and Information Meta Model* and the *CPA* is the part of the *Business Process* document that is instantiated as an XML document that represents the business transactional layer of the *Business Process and Information Meta Model*. The expression of the sequence of commercial transactions in XML is shared between the *Business Process* and *Trading Partner Information* models.

Relationship to Core Components

A *Business Process* instance MAY specify the constraints for exchanging business data with other *Trading Partners*. The business data MAY be comprised of components of the ebXML *Core Library*. Accordingly, a *Business Process* document SHALL reference the *Core Components* directly or indirectly using a XML document with metadata (possibly DTD's or Schemas) that can be uniquely referenced. The mechanism for interfacing with the *Core Components* and *Core Library* SHALL be by way of a *UID* or *GUID* for each component.

Relationship to ebXML Messaging

A *Business Process* instance SHALL be capable of being transported from a *Registry* mechanism to another *Registry* mechanism via an *ebXML Message*. It SHALL also be

capable of being transported between a *Registry* and a users application via the *ebXML Messaging Service*.

Relationship to a Registry System

A *Business Process* instance intended for use within the ebXML infrastructure SHALL be retrievable through a Registry query, and therefore, each *Business Process* SHALL contain a *UID* or *GUID*.

9.2.4 Non-Normative Implementation Details

The exact composition of a *Business Object* or a *Business Document* is guided by a set of contexts derived from the *Business Process*. The modeling layer of the architecture is highlighted in green in Figure 12 below.

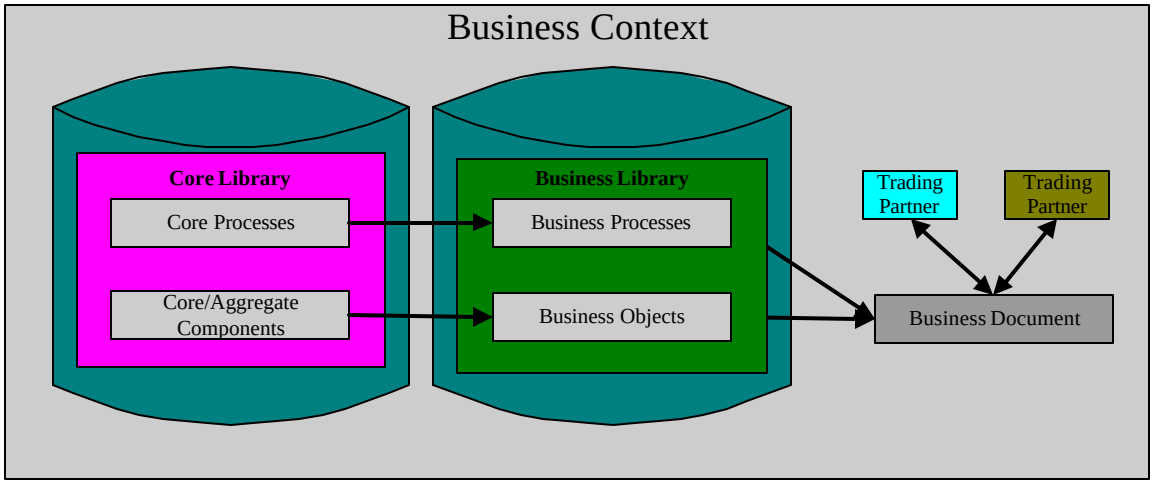


Figure 12 – *ebXML Business Process and Information Modeling layer*

ebXML *Business Process and Information Models* MAY be created following the RECOMMENDED ebXML *Modeling Methodology* (*UML*), or MAY be arrived at in any other way, as long as they comply with the ebXML *Business Process and Information Meta Models*.

9.3 Core Components and Core Library Functionality

9.3.1 Introduction

A *Core Component* captures information about a real world business concept, and the relationships between that concept, other *Business Information*, and a contextual description that describes how a *Core* or *Aggregate Component* may be used in a particular ebXML *eBusiness* scenario.

A *Core Component* can be either an individual piece of business information, or a natural "go-together" family of *Business Information* that may be assembled into *Aggregate Components.*

The *ebXML Core Components Project Team* SHALL define an initial set of *Core Components.* ebXML users may adopt and/or extend components from the ebXML *Core Library.*

9.3.2 Formal Functionality

As a minimum set of requirements, *Core Components* SHALL facilitate the following functionality:

Core Components SHALL be storable and retrievable using an ebXML *Registry* Mechanism.

Core Components SHALL capture and hold a minimal set of information to satisfy both business and technical needs.

Core Components SHALL be expressible in XML syntax.

A *Core Component* SHALL be capable of containing:

- Another *Core Component* in combination with one or more individual pieces of *Business Information*.
- Other *Core Components* in combination with zero or more individual pieces of *Business Information*.

A *Core Component* SHALL be able to be uniquely identified.

9.3.3 Interfaces

A *Core Component* MAY be referenced indirectly or directly from a *Business Process* instance. The *Business Process* MAY specify a single or group of core components as required or optional information as part of a *Business Process*.

A *Core Component* MAY interface with a *Registry* mechanism by way of being storable and retrievable in such a mechanism.

A *Core Component* MAY interface with an XML Element from another XML vocabulary by the fact it is bilaterally or unilaterally referenced as a semantic equivalent.

9.3.4 Non-Normative Implementation Details

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

The process of aggregating *Core Components* for a specific business context, shall include a means to identify the placement of a *Core Component* within another *Core Component*. It MAY also be a combination of structural contexts to facilitate *Core Component* re-use at different layers within another *Core Component* or *Aggregate Component*. This is referred to as *Business Context*.

Context MAY also be defined using the *Business Process and Information Meta Model*, which defines the instances in which the *Business Object* occurs.

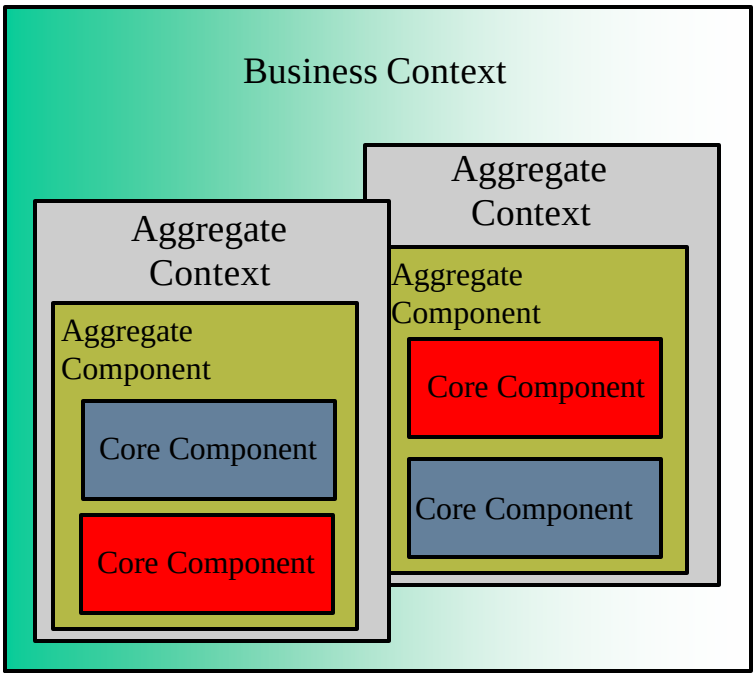


Figure 13 *- Business Context defined in terms of Aggregate Context and Aggregate and Core Components*

The pieces of *Business 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 (aggregate or business context) may alter the fundamental mandatory/optional cardinality.

9.4 Registry Functionality

9.4.1 Introduction

An *ebXML Registry* provides a set of services that enable the sharing of information between users. A *Registry* is a component that maintains an interface to metadata for a registered item. Access to an ebXML *Registry* is provided through interfaces (APIs) exposed by *Registry Services*.

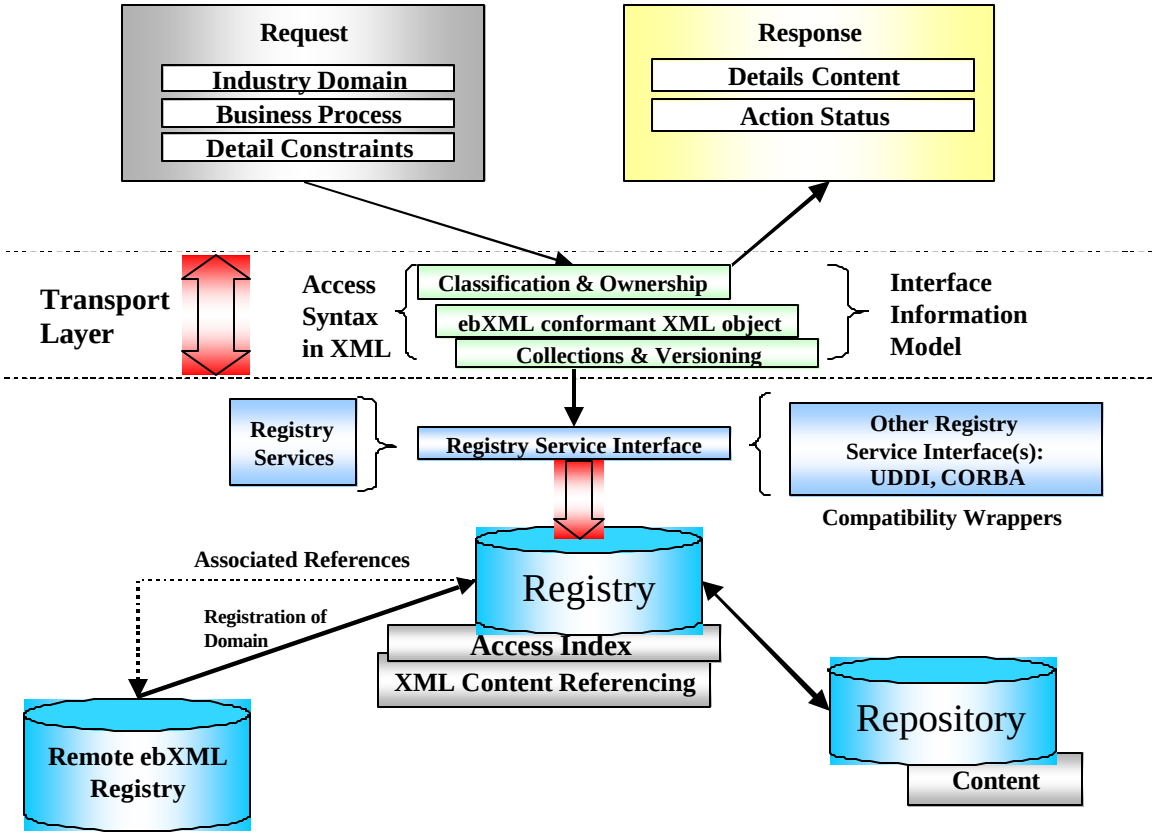


Figure 14 *- Overall Registry / Repository Architecture.*

9.4.2 Formal Functionality

A *Registry* SHALL accommodate the storage of items expressed in syntax using multi-byte character sets.

Each *Registry Item*, at each level of granularity is defined by the *Submitting Organization*, MUST be uniquely identifiable. This is essential to facilitate application-to-Registry queries.

A *Registry* SHALL return either zero or one positive matches in response to a contextual query for a *UID* or *GUID*. In such cases where two or more positive results are displayed for such queries, an error message SHOULD be reported to the *Registry Authority.*

A *Registry Item* SHALL be structured to allow information associations to identify, name, and describe each registered item, give its administrative and access status, define its persistence and mutability, classify it according to pre-defined classification schemes, declare its file representation type, and identify the submitting and responsible organizations.

The *Registry Interface* provides an application-to-registry automated access. Human-to-Registry interactions SHALL be built as a layer over a *Registry Client* (e.g. a Web browser) and not as a separate interface.

The *Registry* interface SHALL be designed to be transport layer neutral.

The processes supported by the *Registry* MAY also include:

- A special *CPA* between the *Registry* and Registry Clients.
- A set of 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 Process*.
- A set of primitive interface mechanisms to support the *Business Messages* and associated query and response mechanisms.
- A special *CPA* for orchestrating the interaction between ebXML compliant Registries.
- A set of functional processes for *Registry*-to-*Registry* interactions.
- A set of error responses and conditions with remedial actions.

To facilitate the discovery process, browse and drill down queries MAY be used for human interactions with a *Registry* (e.g. via a Web browser). A user SHALL be able to browse and traverse the content based on the available *Registry* classification schemes.

Registry messages SHALL exist to create, modify, and delete *Registry Items* and their metadata.

Appropriate security protocols MAY be deployed to offer authentication and protection for the *Repository* when accessed by the *Registry*.

9.4.3 Interfaces

ebXML Messaging:
The query syntax used by the *Registry* access mechanisms is independent of the physical implementation of the backend system. The ebXML *Messaging Service* serves as the transport mechanism for all communications into and out of the Registry.

Business Process:
Business Processes MAY be published and retrieved via ebXML *Registry* services.

Core Components:
Core Components MAY be published and retrieved via ebXML *Registry* services.

Any item with metadata: XML elements provide standard metadata about the item being managed through ebXML *Registry* services. [NOTE: The metadata is separate from the item itself, thus allowing the *ebXML Registry* to catalog arbitrary items.] Since

ebXML Registries are distributed each *Registry* MAY interact with and cross-reference another ebXML *Registry*.

9.4.4 Non-Normative Implementation Details

The *Business Process and Information Model* within a *Registry* MAY be stored according to various classification schemes.

The existing ISO11179/3 work on *Registry* implementations MAY be used to provide a model for the *ebXML Registry* implementation.

Registry Items and their metadata MAY also be addressable as *XML* based URI references using only HTTP for direct access.

Examples of extended Registry services functionality MAY be deferred to a subsequent phase of the ebXML initiative. This MAY include: transformation services, workflow services, quality assurance services and extended security mechanisms.

A *Registry* service MAY have multiple deployment models as long as the *Registry* interfaces are ebXML compliant.

The assignment of a *GUID* to *Registry Items* MAY benefit from the use of a standard algorithm such as the DC 128 GUID algorithm.

The *Business Process and Information Model* for an *ebXML Registry* service MAY be an extension of the existing *OASIS Registry Information Model*, specifically tailored for the storage and retrieval of business information, whereas the OASIS model is a superset designed for handling extended and generic information content.

9.5 Messaging Service Functionality

9.5.1 Introduction

The *ebXML Message Service* mechanism SHALL provide a standard way to exchange business messages among ebXML *Trading Partners*. The *ebXML Messaging Service* provides a reliable means to exchange business messages without relying on proprietary technologies and solutions. An *ebXML Message* contains structures for a *Header* (necessary for routing and delivery) and a *Payload* section (necessary for transport).

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 15 below.

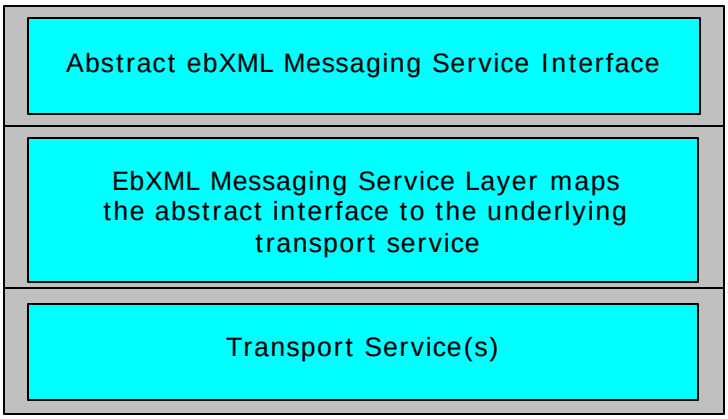


Figure 15 *- ebXML Messaging Service*

The following diagram depicts a logical arrangement of the functional modules that exist within the ebXML *Messaging Services* 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.

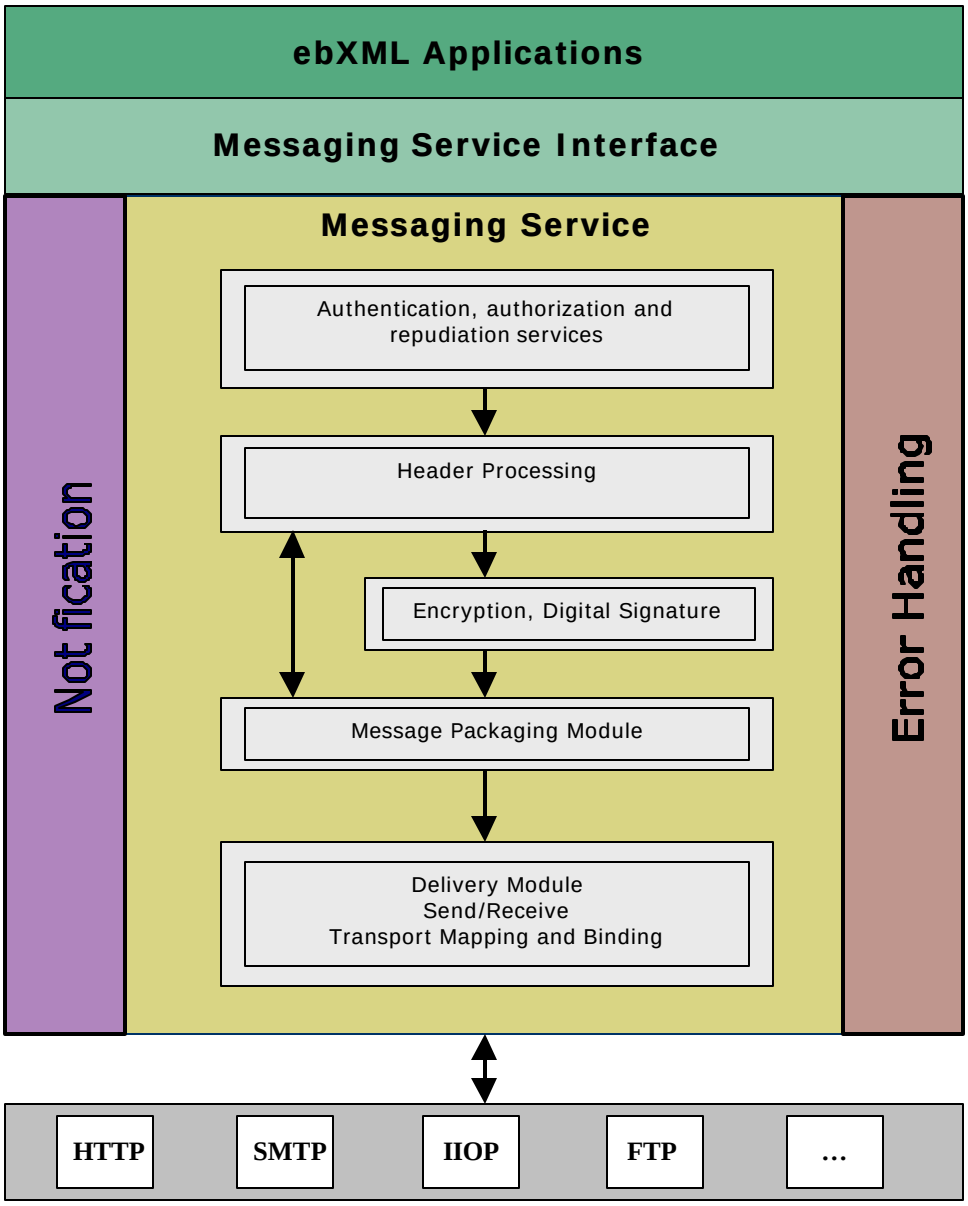


Figure 16 *- The Messaging Service Architecture*

9.5.2 Formal Functionality

The *ebXML Messaging Service* SHALL provide a secure, consistent and reliable mechanism to exchange *ebXML Messages* between users of the ebXML infrastructure over various transport *Protocols* (possible examples include SMTP, HTTP/S, FTP, etc.).

The *ebXML Messaging Service* SHALL prescribe formats for all messages between distributed ebXML *Components* including *Registry* mechanisms and compliant user *Applications*. It SHALL also utilize and enforce the "rules of engagement" defined in a *Collaboration Protocol Agreement* (*CPA*).

The *ebXML Messaging Service* SHALL NOT place any restrictions on the content of the payload.

The *ebXML Messaging Service* SHALL support simplex (one-way) and request/response (either synchronous or asynchronous) message exchanges.

The *ebXML Messaging Service* SHALL meet business needs, consequently it SHALL support sequencing of payloads in instances where multiple payloads or multiple messages are being used.

The ebXML *Messaging Service Layer* SHALL enforce the "rules of engagement" as defined by two parties in a *Collaboration Protocol Agreement* (including security and *Business Process* functions related to message delivery). The *Collaboration Protocol Agreement* defines the acceptable behavior by which each *Party* agrees to abide. The definition of these ground rules can take many forms including formal *Collaboration Protocol 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.

The *ebXML Messaging Service* SHALL perform all security related functions including:

- Identification
- Authentication (verification of identity)
- Authorization (access controls)
- Privacy (encryption)
- Integrity (message signing)
- Non-repudiation
- Logging

9.5.3 Interfaces

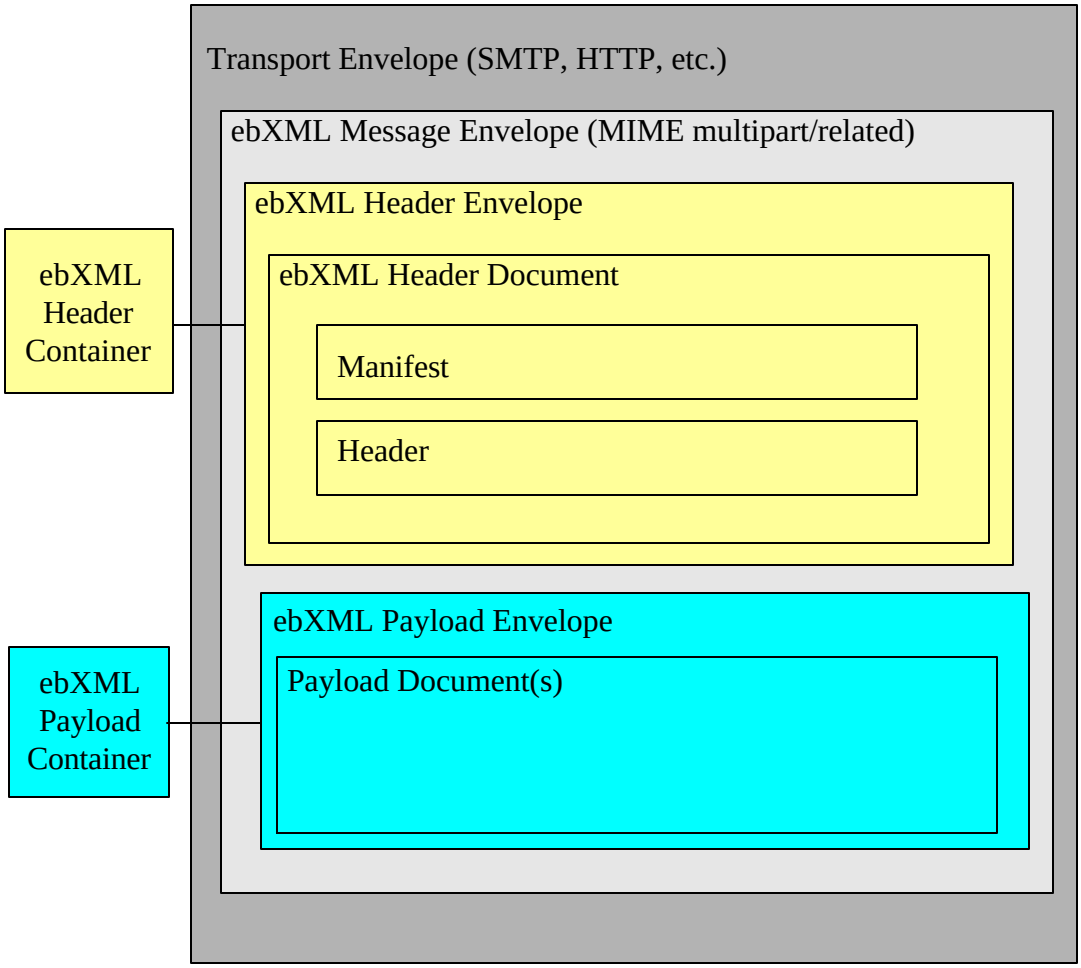
The *ebXML Message Service* provides ebXML with an abstract interface whose functions, at an abstract level, include:

- 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.

The *ebXML Messaging Service* SHALL interface with internal systems including:

- Routing of received messages to internal systems
- Error notification

998 The *ebXML Messaging Service* SHALL help facilitate the interface to an ebXML
999 *Registry*.
1000
1001 #### 9.5.4 Non-Normative Implementation Details
1002
1003 **ebXML Message Structure and Packaging**
1004
1005 Figure 17 below illustrates the logical structure of an *ebXML Message*.



1006
1007
1008 ***Figure 17 -*** *ebXML Message Structure*
1009
1010 An *ebXML Message* MAY consist of an OPTIONAL transport *Protocol* specific outer
1011 *Communication Protocol Envelope* and a *Protocol* independent *ebXML Message*
1012 *Envelope*. The *ebXML Message Envelope* MAY be packaged using the MIME
1013 multipart/related content type. MIME is used as a packaging solution because of the
1014 diverse nature of information exchanged between *Partners* in *eBusiness* environments.
1015 For example, a complex B2B business transaction between two or more *Trading Partners*
1016 might require a payload that contains an array of business documents (*XML* or other
1017 document formats), binary images, or other related *Business Object*s.

10 Conformance

10.1 Introduction

This clause specified the general framework, concepts and criteria for *Conformance* to ebXML, including an overview of the conformance strategy for ebXML, guidance for addressing conformance in each ebXML technical specification, and the conformance clause specific to the Technical Architecture specification. Except for the Technical Architecture Specification, this clause does not define the conformance requirements for each of the ebXML technical specifications – the latter is the purview of the technical specifications.

The objectives of this section are to:

a) Ensure a common understanding of conformance and what is required to claim conformance to this family of specifications;
b) Ensure that conformance is consistently addressed in each of the component specifications;
c) Promote interoperability and open interchange of *Business Processes* and messages;
d) Encourage the use of applicable conformance test suites as well as promote uniformity in the development of conformance test suites.

Conformance to ebXML is defined in terms of conformance to the ebXML infrastructure and conformance to each of the technical specifications for ebXML. The primary purpose of conformance to ebXML is to increase the probability of successful interoperability between implementations and the open interchange of XML business documents and messages. While conformance is a necessary condition, it is not on its own a sufficient condition to guarantee interoperability. Successful interoperability and open interchange is more likely to be achieved if implementations conform to the requirements in the ebXML specifications.

10.2 Conformance to ebXML

ebXML Conformance is defined as conformance to an ebXML system that is comprised of all the architectural components of the ebXML infrastructure and satisfies at least the minimum conformance requirements for each of the ebXML technical specifications, including the functional and interface requirements in this Technical Architecture specification.

In the context of ebXML, an implementation is said to exhibit conformance if it complies with the requirements of each applicable ebXML technical specification. The conformance requirements are stated in the conformance clause of each technical specification of ebXML. 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.

10.3 Conformance to the Technical Architecture Specification

This section details the conformance requirements for claiming conformance to the Technical Architecture specification.

In order to conform to this specification, each ebXML technical specification:

a) SHALL support all the functional and interface requirements defined in this specification that are applicable to that technical specification;
b) SHALL NOT specify any requirements that would contradict or cause non-conformance to ebXML or any of its components;
c) MAY contain a conformance clause that adds requirements that are more specific and limited in scope than the requirements in this specification;
d) SHALL only contain requirements that are testable.

A conforming implementation SHALL satisfy the conformance requirements of the applicable parts of this specification and the appropriate technical specification(s).

10.4 General Framework of Conformance Testing

The objective of conformance testing is to determine whether an implementation being tested conforms to the requirements stated in the relative ebXML specification. Conformance testing enables vendors to implement compatible and interoperable systems built on the ebXML foundations. EbXML *implementations* and *Applications* SHALL be tested to available test suites 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. Open source reference implementations MAY be available to allow vendors to test their products for interface compatibility, conformance, and interoperability.

11.0 Security Considerations

11.1 Introduction

A comprehensive *Security Model* for ebXML will be expressed in a separate document. The *Security Model* SHALL be applied to the entire ebXML Infrastructure, with the underlying goal of best meeting the needs of users of ebXML.

The Security Model SHALL comply with security needs specified in the *ebXML Requirements Document*.

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 SME's to properly conduct *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

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

Scenario 1

Two *Trading Partners* set-up an agreement and run the associated exchange.

In this scenario:

- Each *Trading Partner* defines its own *Trading Partner Profile (TPP)*. Each *TPP* references:
  - One or more existing *Business Process* found in an ebXML *Registry* system.
  - One of more *Business Message* definitions. Each definition is built from reusable *Components* (*Core and/or Aggregate Components*) found in the ebXML *Registry* system.

Each *TPP* defines:

- The commercial transactions that the *Trading Partner* is able to support.
- The underlying protocol (like HTPP, SMTP etc) and the technical properties (such as encryption, validation, authentication, digital signing) that the *Trading Partner* supports in the engagement.
- The *Trading Partners* acknowledge each other's *TPP* and create a *Collaboration Protocol Agreement (CPA)*. The *CPA* references:
  - The relevant *TPP's*.
  - 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 *ebXML Messaging Services Specification*.
  - 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 Messaging Service*.
  - The *Trading Partners* start exchanging messages and performing the agreed upon commercial transactions.

Scenario 2:

Three or more *Trading Partners* set-up a *Business Process* implementing a supply-chain *eBusiness* scenario.

The simple case of a supply-chain involving two *Trading Partners* 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 *Trading Partner* 2 is engaged at the same time with two different *Trading Partners*. The assumption is that the "state" of the entire *Business Process* is managed by each *Trading Partner*, i.e. that each *Trading Partner* is fully responsible of the commercial transaction involving it (*Trading Partner* 3 only knows about *Trading Partner* 2, *Trading Partner* 2 knows about *Trading Partner* 3 and *Trading Partner* 1, *Trading Partner* 1 knows about *Trading Partner* 2).

In this scenario:

- Each *Trading Partner* defines its' own *TPP*. Each *TPP* references:
  - One or more existing *Business Process* found in the ebXML *Registry* system.
  - One of more *Business Message* definitions. Each definition is built from reusable *Components* (*Core and/or Aggregate Components*) found in the ebXML *Registry*.
- Each *TPP* defines:
  - The commercial transactions that the *Trading Partner* is able to engage into
    Trading Partner 2 must be able to support at least 2 commercial transactions.

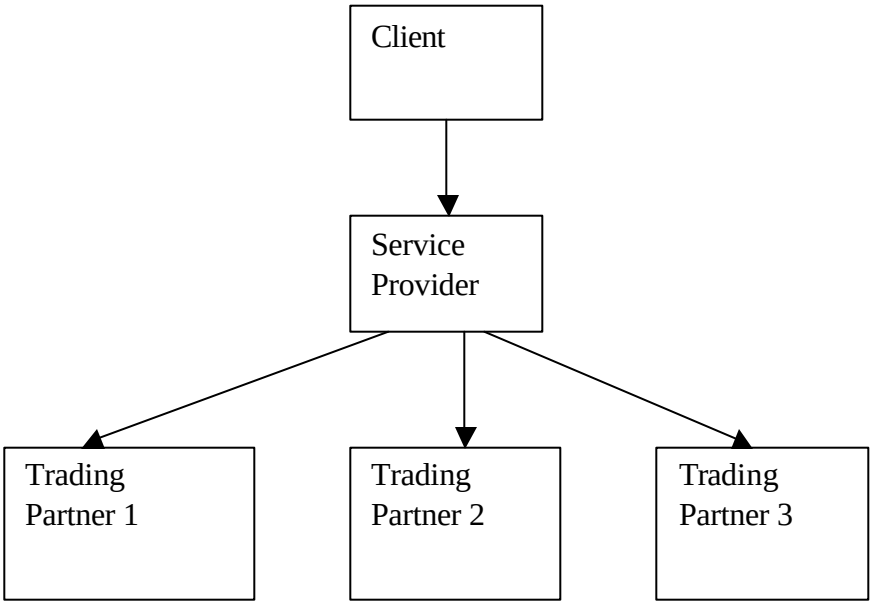
- The underlying protocol (like HTPP, SMTP etc) and the technical properties (such as encryption, validation, authentication, and digital signing) that the *Trading Partner* supports in the engagement.
  - The technical requirements for the exchanges with *Trading Partner* 1 and *Trading Partner* 3 MAY be different. In such case, *Trading Partner* 2 must be able to support different protocols and/or properties.
- The *Trading Partners* acknowledge each other *TPP* and create the relevant *CPA's* (at least 2 in this scenario).

- Each *CPA* references:
  - The relevant *CPP's* for each respective *Trading Partner*.
  - The terms and conditions related to the mutually agreed upon business exchange
- *Trading Partner 2* is engaged in 2 *CPA's*. Each Trading Partner implements their own respective part of each CPA. 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 *Business Message* definitions
    - Granting that the exchanges technically conform with the underlying ebXML Messaging Service.
    - *Trading Partner* 2 MAY need to implement a complex *Business Service Interface* in order to be able to engage with different *Trading Partners*.
    - The *Trading Partners* start exchanging messages and perform the agreed upon commercial transactions.
    - *Trading Partner* 3 places an order with Trading Partner 2.
    - *Trading Partner* 2 (eventually) places an order with *Trading Partner* 1.
    - *Trading Partner* 1 fulfills the order.
    - *Trading Partner* 2 fulfills the order.

Scenario 3

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

This a 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:

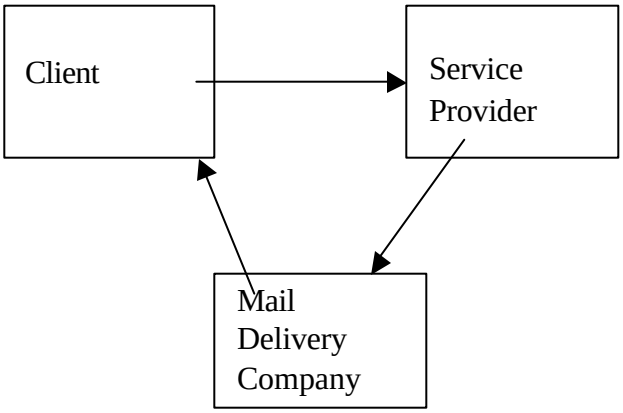


This is an evolution of Scenario 2. The Description of this scenario is omitted to minimize the duplication of processes explained in detail in Scenario 2.

Scenario 4

Three or more *Trading Partners* engage in *eBusiness* using *Business Processes* that were created by each respective *Trading Partner* and run the associated business exchanges.

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



In this Scenario, each *Trading Partner* is involved with more than one other *Trading Partner* but the relationship is not linear. The product ordered by the client from the Service Provider will be delivered by a 3rd *Trading Partner.*

In this scenario:

- Each *Trading Partner* defines its own *CPP*.
  Each *CPP* references:
- One or more existing *Business Process* found in the ebXML *Registry*
- One of more *Business Message* definitions. Each definition is built from reusable *Components* (*Core and/or Aggregate Components*) found in the ebXML *Registry*

Each *CPP* defines:

- The Commercial Transactions that the *Trading Partner* is able to engage into
  In this case, each *Trading Partner* must be able to support at least 2 commercial transactions.
- The technical protocol (like HTPP, SMTP etc) and the technical properties (such as encryption, validation, authentication, and digital signing) that the *Trading Partner* supports in the engagement.
  In case the technical infrastructure underlying the different exchanges differs, each *Trading Partner* 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 *Trading Partners* acknowledge each other profile and create a Partner *CPA*.
  Each *Trading Partner*, in this Scenario, must be able to negotiate at least 2 *CPA's*.
- The *CPA* references:
  - The relevant *CPP's*
  - The terms and conditions related to the exchange

  - Each *Trading Partner* is engaged in 2 *CPA's*.
    - The Trading Partners 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 application into the ebXML technical infrastructure as specified by the *ebXML Messaging Service*.
    - Granting that the software is able to properly engage the stated business scenarios.
    - Granting that the exchanges semantically conform to the agreed upon *Business Message* definitions
    - Granting that the exchanges technically conform with the underlying *ebXML Messaging Services Specification.*

- All *Trading Partners* MAY need to implement complex *Business Service Interfaces* to accommodate the differences in the *CPA's* with different *Trading Partners*.
- The *Trading Partners* start exchanging messages and performing the agreed upon commercial business 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 product to be delivered at the Client
- The Mail Delivery Service delivers the product to the Client
- The Clients notifies the Service Provider that the product 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