Creating A Single Global Electronic Market

Technical Architecture Specification

v1.0.4

Technical Architecture Team

16 February 2001

(This document is the non-normative version formatted for printing, July 2001)

Table of Contents

1 Status of this Document

This document specifies an ebXML Technical Specification for the eBusiness community.

Distribution of this document is unlimited.

The document formatting is based on the Internet Society’s Standard RFC format.

This version:

www.ebxml.org/specs/ebTA.pdf

Latest version:

www.ebxml.org/specs/ebTA.pdf

2 ebXML Technical Architecture Participants

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

Team Lead:

Brian Eisenberg	DataChannel

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
Anders Grangard	EDI France
Peter Kacandes	Sun Microsystems
Kris Ketels	SWIFT
Piming Kuo	Worldspan
Kyu-Chul Lee	Chungnam National University
Henry Lowe	OMG
Matt MacKenzie	XML Global Technologies
Melanie McCarthy	General Motors
Stefano Pagliani	Sun Microsystems
Bruce Peat	eProcessSolutions
John Petit	KPMG Consulting
Mark Heller	MITRE
Scott Hinkelman	IBM

Lynne Rosenthal	NIST
Nikola Stojanovic	Encoda Systems, Inc.
Jeff Sutor	Sun Microsystems
David RR Webber	XML Global Technologies

3 Introduction

3.1 Summary of contents of document

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.

Note Notes are used to further clarify the discussion or to offer additional suggestions and/or resources

3.2 Audience and scope

This document is intended primarily for the ebXML project teams to help guide their work. Secondary audiences include, but are not limited to: 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.

3.3 Related documents

As mentioned above, other documents provide detailed definitions 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/specs

3.4 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.

ISO/IEC 14662: Open-edi Reference Model

ISO 11179/3 Metadata Repository

ISO 10646: Character Encoding

ISO 8601:2000 Date/Time/Number Data typing

OASIS Registry/Repository Technical Specification

RFC 2119: Keywords for use in RFC's to Indicate Requirement Levels

UN/CEFACT Modeling Methodology (UMM)

W3C XML v1.0 Second Edition Specification

4 Design Objectives

4.1 Problem description and 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* enables more open, more flexible business transactions than *EDI*. *XML* might enable more flexible and innovative "eMarketplace" business models than *EDI*. But the challenges of designing M*essages* that meet *Business Process* requirements and standardizing their semantics are independent of the syntax in which the M*essages* 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.

4.2 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 specifications to get the level of detail.

4.3 Design conventions for ebXML specifications

In order to enforce a consistent capitalization and naming convention across all ebXML specifications "Upper Camel Case" (*UCC*) and "Lower Camel Case" (*LCC*) Capitalization styles SHALL be used. *UCC* style capitalizes the first character of each word and compounds the name. *LCC* style capitalizes the first character of each word except the first word.

1. ebXML DTD, XML Schema and *XML* instance documents SHALL have the effect of producing ebXML *XML* instance documents such that:
   - Element names SHALL be in *UCC* convention (example:
   - <UpperCamelCaseElement/>).
   - Attribute names SHALL be in *LCC* convention (example: <UpperCamelCaseElement lowerCamelCaseAttribute="Whatever"/>).
2. When *UML* and *Object Constrained Language (OCL)* are used to specify ebXML artifacts Capitalization naming SHALL follow the following rules:
   - Class, *Interface*, Association, Package, State, Use Case, Actor names SHALL use UCC convention (examples: ClassificationNode, Versionable, Active, InsertOrder, Buyer).
   - Attribute, Operation, Role, Stereotype, Instance, Event, Action names SHALL use LCC convention (examples: name, notifySender, resident, orderArrived).
3. General rules for all names are:
   - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
   - Underscore (_), periods (.) and dashes (-) MUST NOT be used (don't use: header.manifest, stock_quote_5, commercial-transaction, use HeaderManifest, stockQuote5, CommercialTransaction instead).

5 ebXML System Overview

Figure 1 below shows a high-level use case scenario 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 architecture components. These components can 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 *Business Process and Information Meta Models* so they can be shared and 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 the execution of a mutually agreed upon business arrangement which can be derived from information provided by each participant from item 3 above. (*Collaboration Protocol Agreement – CPA*)
6. A standardized business *Messaging Service* framework that enables interoperable, secure and reliable exchange of M*essages* between *Trading Partners*.
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.

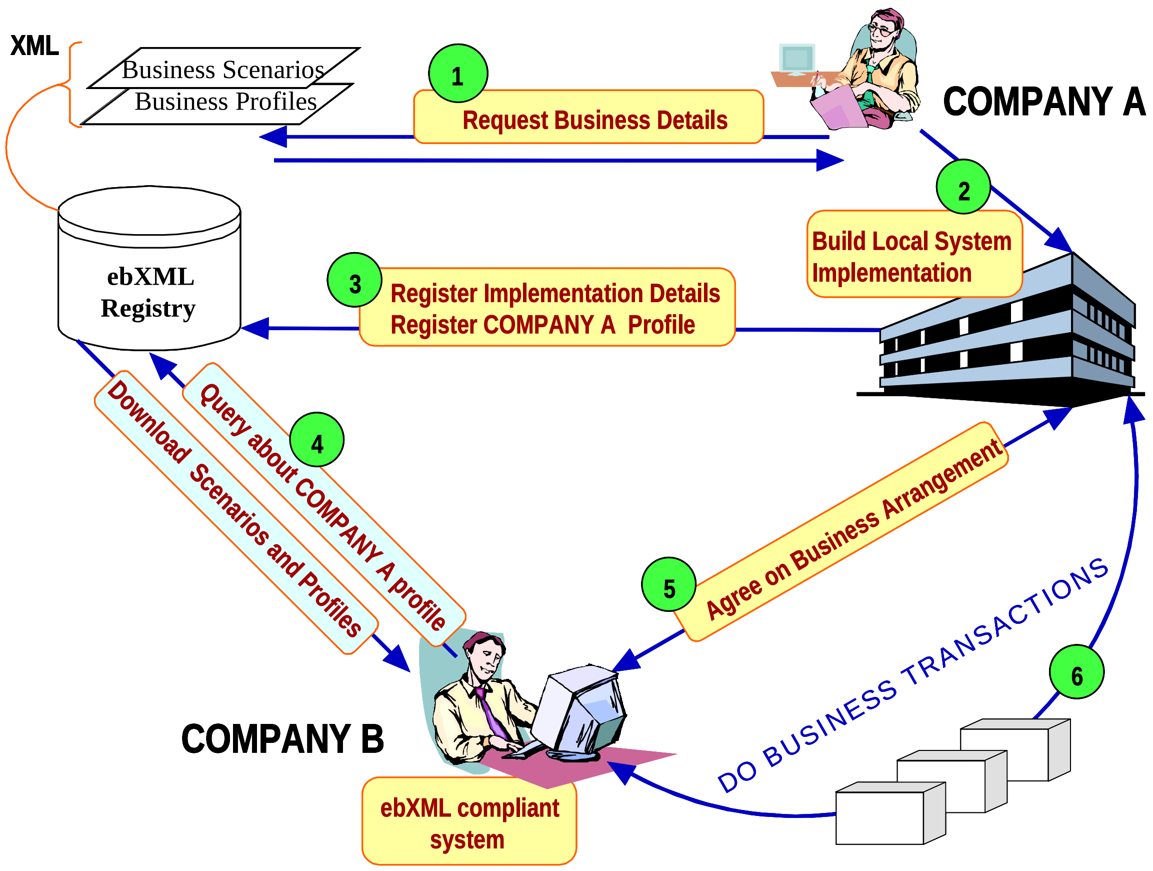

XML
Business Scenarios
Business Profiles
1
Request Business Details
COMPANY A
2
Build Local System Implementation
ebXML Registry
3
Register Implementation Details
Register COMPANY A Profile
Download Scenarios and Profiles
Query about COMPANY A profile
4
5
Agree on Business Arrangement
DO BUSINESS TRANSACTIONS
6
COMPANY B
ebXML compliant system

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). 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 *Business Profile* information (including implementation details and reference links) 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 scenarios. These business scenarios are *XML* versions of the *Business Processes* and associated information bundles (e.g. a sales tax calculation) in which the company is able to engage. After receiving verification that the format and usage of a business scenario is correct, an acknowledgment is sent to Company A (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 scenario using ebXML (Figure 1, step 5). Company B acquires an ebXML compliant shrink-wrapped application.

Before engaging in 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. 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 then accepts the business agreement. Company A and B are now ready to engage in *eBusiness* using ebXML (Figure 1, step 6).

6 ebXML Recommended Modeling Methodology

Business Process and Information Modeling is not mandatory. However, if implementers and users select to model *Business Processes and Information*, then they SHALL use the *UN/CEFACT Modeling Methodology (UMM)* that utilizes *UML*.

6.1 Overview

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 *Business Process and Information Meta Models* that are likely candidates for standardization. The ebXML approach looks for standard reusable components from which to construct interoperable and components.

The *UN/CEFACT Modeling Methodology (UMM)* uses the following two views to describe the relevant aspects of *eBusiness* transactions. This model is based upon the Open-edi Reference Model, ISO/IEC 14662.

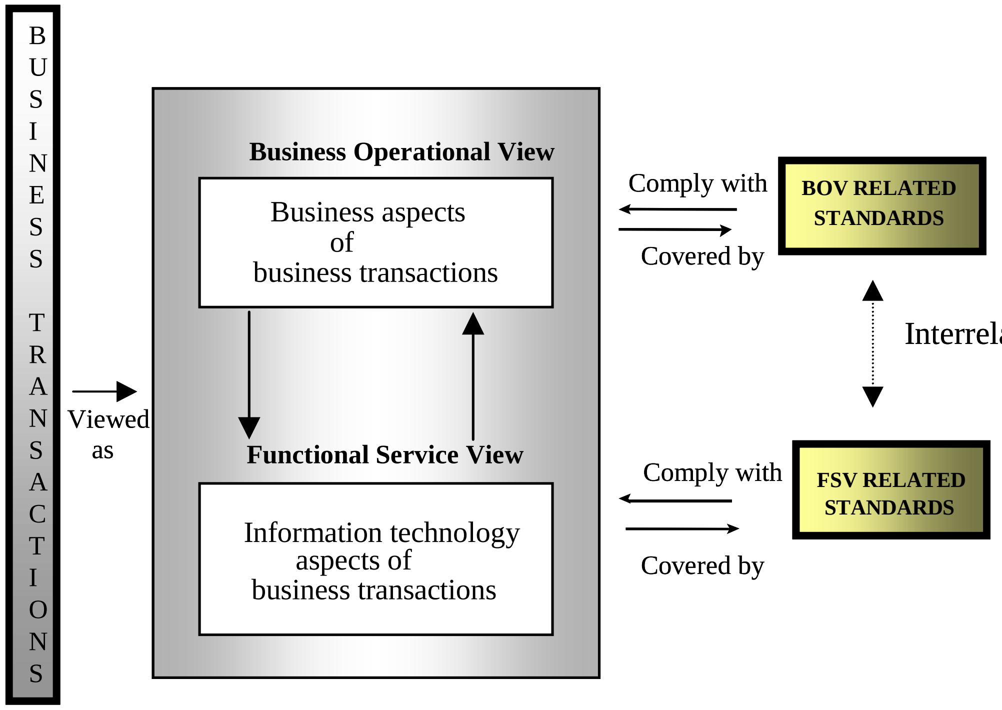


Figure 2 ***ebXML Recommended Modeling Methodology***

The *UN/CEFACT Modeling Methodology (UMM)* 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 *UN/CEFACT Modeling Methodology (UMM)* 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 *UN/CEFACT Modeling Methodology (UMM)* for constructing *Business Process and Information Meta Models* for ebXML compliant applications and components.

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;
- *Business Service Interfaces*;
- Protocols and *Messaging Services.*

This includes, but is not limited to:

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

6.2 ebXML business operational view

The modeling techniques described in this section are not mandatory requirements for participation in ebXML compliant business transactions.

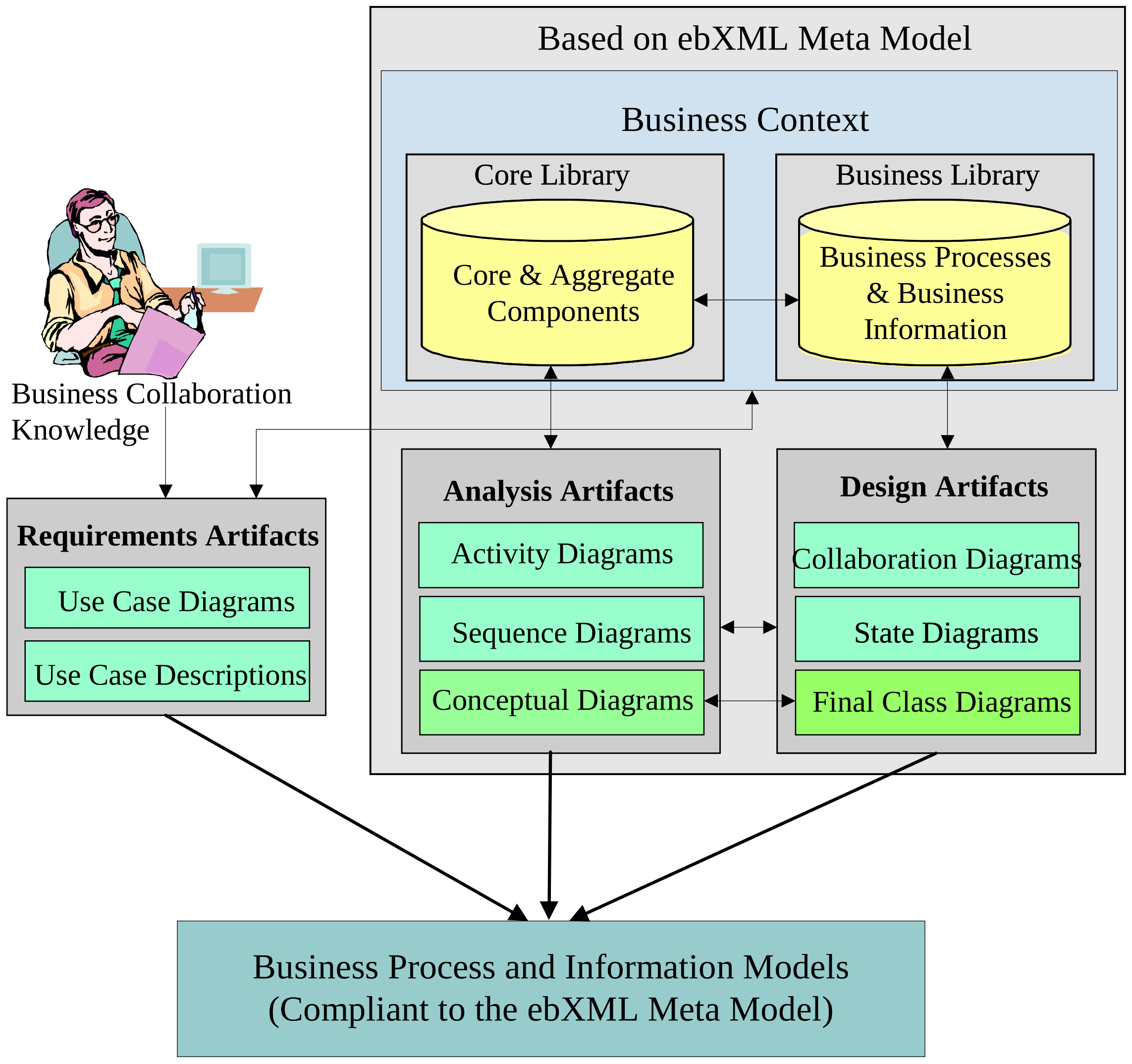


Figure 3: ***detailed representation of the Business Operational View***

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 context 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 (as defined in the *UN/CEFACT Modeling Methodology* specification) describing the *Business Processes*. *Class Diagrams* will capture the associated information parcels (business documents). 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. Common *Business Processes* in the Business Library MAY be referenced during the process of creating analysis and design artifacts.

The design phase is the last step of standardization, which MAY be accomplished by applying object-oriented principles based on the *UN/CEFACT Modeling Methodology*. 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 Information Objects* across all class models. *Business Processes* are created by applying the *UN/CEFACT Modeling Metholodogy (UMM)* which utilizes a common set of *Business Information Objects* and *Core Components.*

6.3 ebXML functional service view

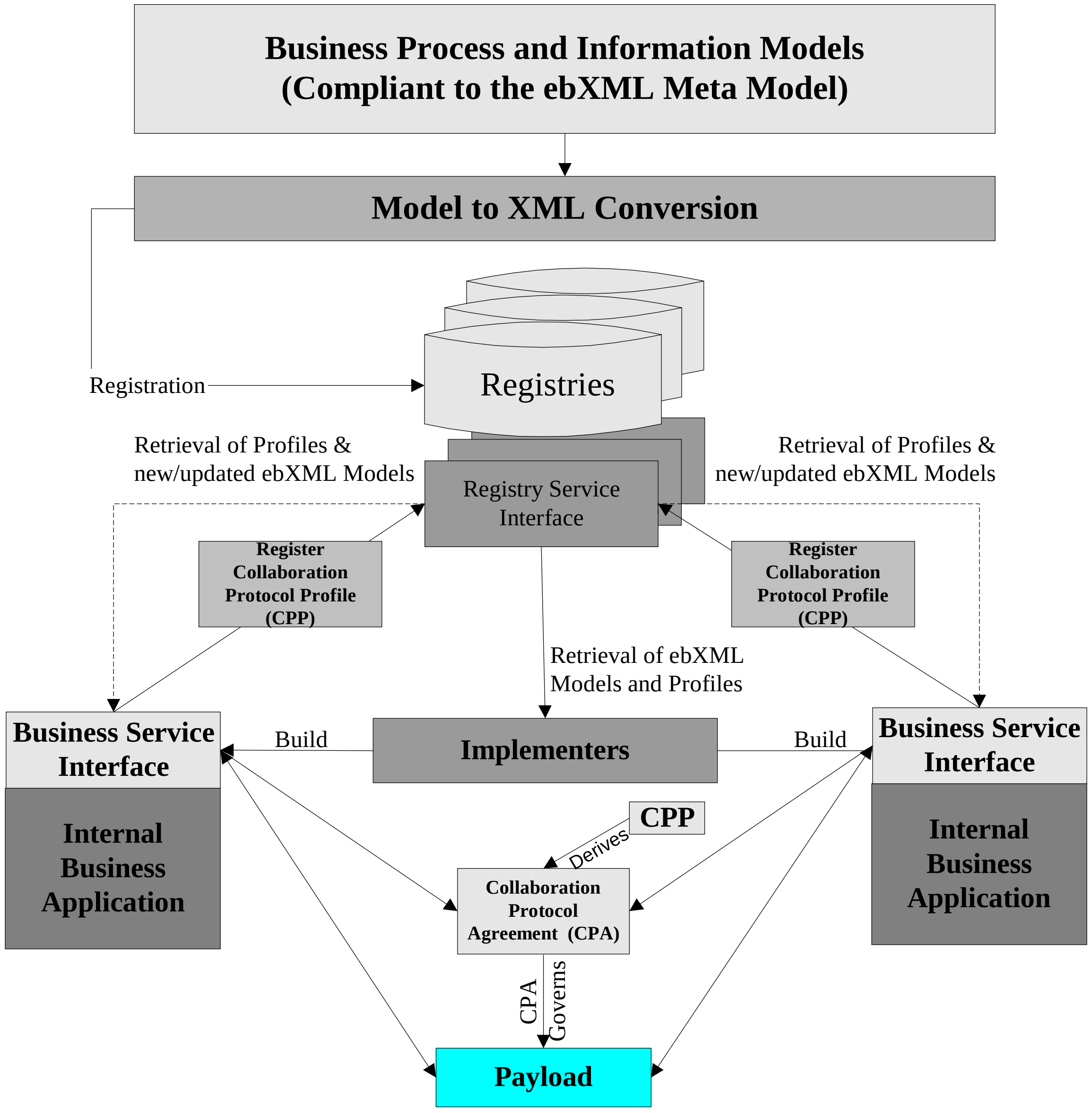

Business Process and Information Models (Compliant to the ebXML Meta Model)
Model to XML Conversion
Registration
Registries
Retrieval of Profiles & new/updated ebXML Models
Registry Service Interface
Retrieval of Profiles & new/updated ebXML Models
Register Collaboration Protocol Profile (CPP)
Register Collaboration Protocol Profile (CPP)
Retrieval of ebXML Models and Profiles
Business Service Interface
Build
Implementers
Build
Business Service Interface
Internal Business Application
CPP
Derives
Collaboration Protocol Agreement (CPA)
Internal Business Application
CPA Governs
Payload

Figure 4: ***ebXML Functional Service View***

As illustrated in Figure 4 above, the ebXML *Registry Service* serves as the storage facility for the *Business Process and Information Models*, the *XML*-based representations of those models, *Core Components*, and *Collaboration Protocol Profiles*. The *Business Process and Information Meta Models* MAY be stored in modeling syntax, however they MAY be also stored as *XML* syntax in the *Registry*. This *XML*-based business information SHALL be expressed in a manner that allows discovery down to the atomic data level via a consistent methodology.

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 Services* (see Registry Functionality, Section 8.4 for details of this architecture).

7 ebXML Functional Phases

7.1 Implementation phase

The implementation phase deals specifically with the procedures for creating an application of the ebXML infrastructure. A *Trading Partner* wishing to engage in an ebXML compliant transaction SHOULD first acquire copies of the ebXML Specifications. The *Trading Partner* studies these specifications and subsequently downloads the *Core Library* and the *Business Library*. The *Trading Partner* MAY also request other *Trading Partners' Business Process* information (stored in their business profile) for analysis and review. Alternatively, the Trading Partner MAY implement ebXML by utilizing 3rd party applications. The *Trading Partner* can also submit its own *Business Process* information to an ebXML compliant *Registry Service*.

Figure 5 below, illustrates a basic interaction between an ebXML *Registry Service* and a *Trading Partner*.

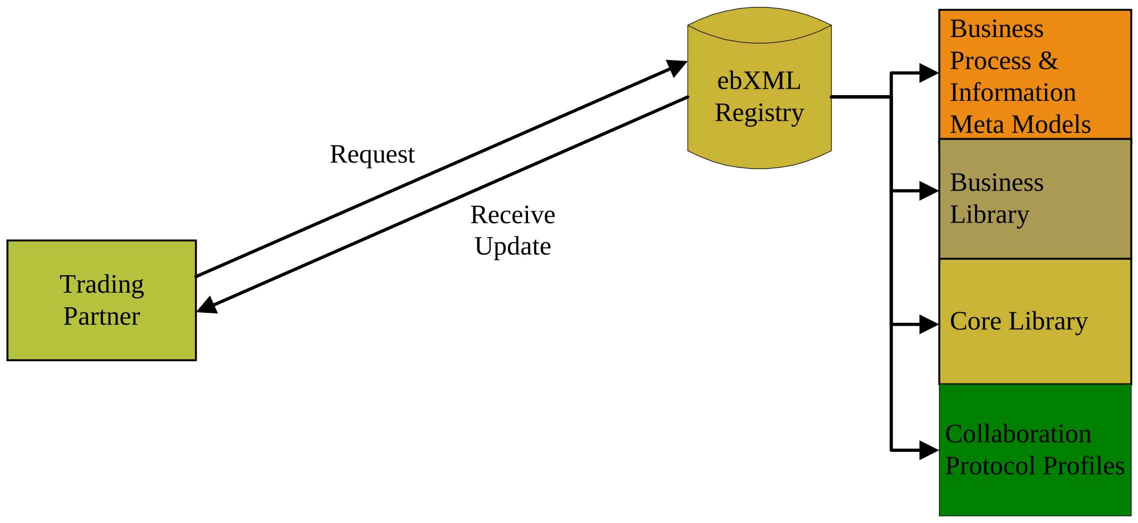


Figure 5: ***Functional Service View: Implementation Phase***

7.2 Discovery and retrieval phase

The Discovery and Retrieval Phase covers all aspects of the discovery of ebXML related resources. 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 *Collaboration Protocol Profile* of another *Trading Partner*. Requests for updates to *Core Libraries*, *Business Libraries* and updated or new *Business Process and Information Meta Models* SHOULD be supported by an ebXML *Business Service Interface*. This is the phase where *Trading Partners* discover the meaning of business information being requested by other *Trading Partners*.

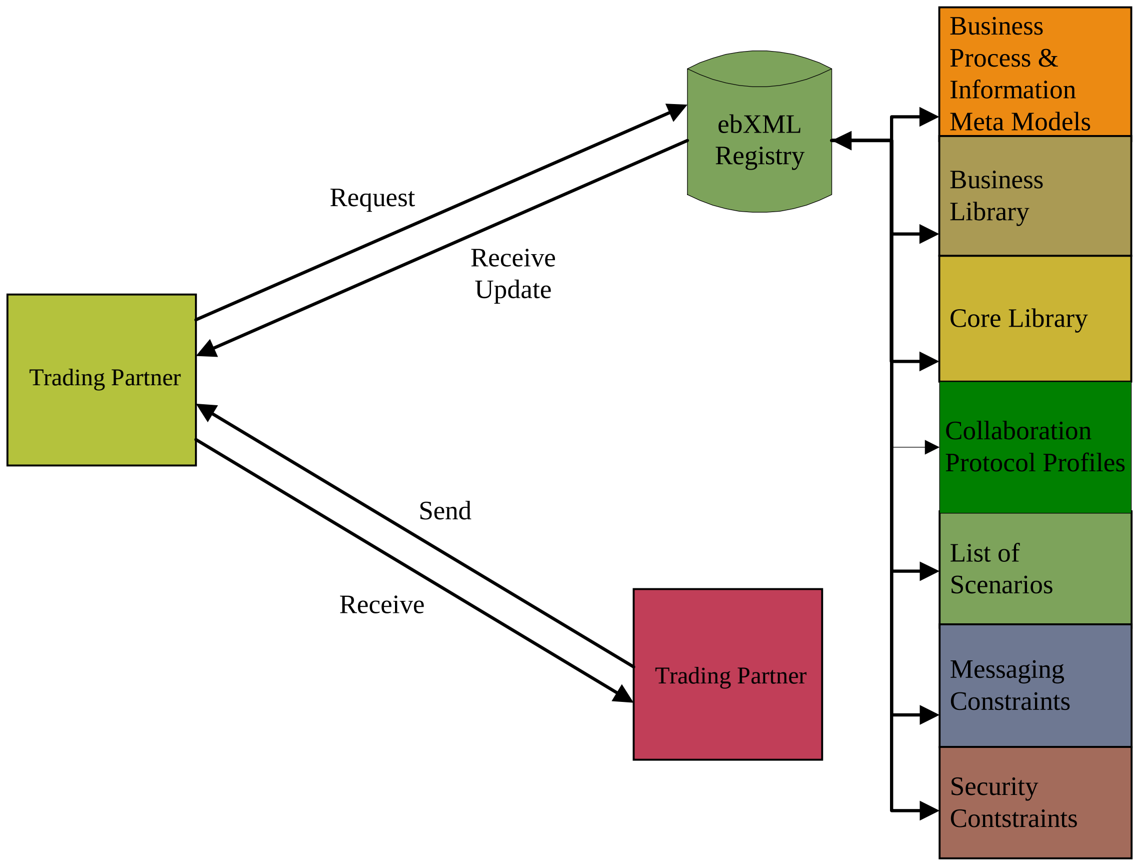


Figure 6: ***Functional Service View: Discovery and Retrieval Phase***

7.3 Run time phase

The run time phase covers the execution of an ebXML scenario with the actual associated ebXML transactions. In the Run Time Phase, ebXML *Messages* are being exchanged between *Trading Partners* utilizing the ebXML *Messaging Service.*

For example, an ebXML *CPA* is a choreographed set of business *Message* exchanges linked together by a well-defined choreography using the ebXML *Messaging Service*.

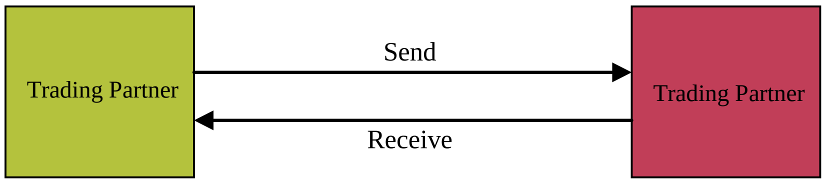


Figure 7: ***Functional Service View: Run Time Phase***

Note There is no run time access to the *Registry*. If it becomes necessary to make calls to the *Registry* during the run time, this SHOULD be considered as a reversion to the Discovery and Retrieval Phase.]

8 ebXML Infrastructure

8.1 *Trading partner information [CPP and CPA's]*

8.1.1 Introduction

To facilitate the process of conducting *eBusiness*, potential *Trading Partners* 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 supported *Business Processes* and *Business Service Interface* requirements in a manner where they can be universally understood by other ebXML compliant *Trading Partners*.

A special business agreement called a *CPA* is derived from the intersection of two or more *CPP's*. The *CPA* serves as a formal handshake between two or more *Trading Partners* wishing to conduct business transactions using ebXML.

8.1.2 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*. The *CPP* contains essential information about the *Trading Partner* including, but not limited to: contact information, industry classification, supported *Business Processes*, *Interface* requirements and *Messaging Service* requirements. *CPP's* MAY also contain security and other implementation specific details. Each ebXML compliant *Trading Partner* SHOULD register their *CPP(s)* in an ebXML compliant *Registry Service*, thus providing a discovery mechanism that allows *Trading Partners* to (1) find one another, (2) discover the *Business Process* that other *Trading Partners* support.

The *CPP* definition SHALL provide for unambiguous selection of choices in all instances where there may be multiple selections (e.g. HTTP or SMTP transport).

8.1.3 CPA formal Functionality

A *Collaboration Protocol Agreement* (*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.

A *CPA* describes: (1) the *Messaging Service* and (2) the *Business Process* 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 capabilities that a *Trading Partner* can support, with a subset of what a *Trading Partner* "will" actually support.

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 Service*, but this is not a mandatory part of the *CPA* creation process.

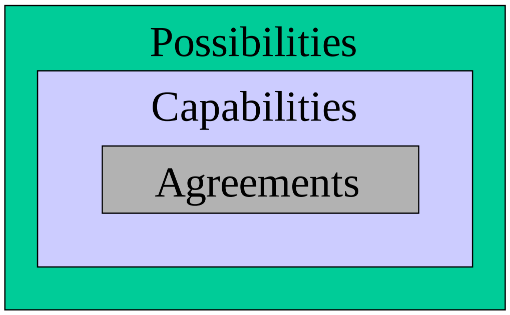


Figure 8: ***Three level view of CPA's***

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.

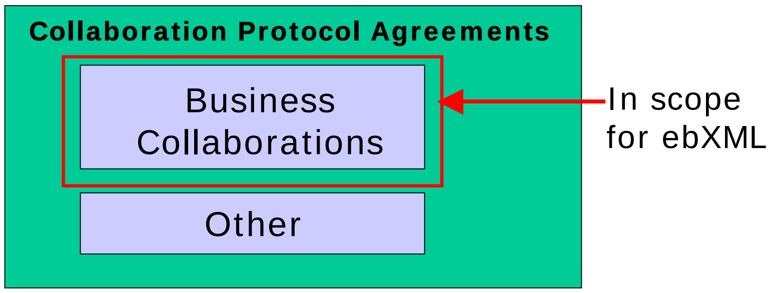


Figure 9: ***Scope for CPA's***

The *CPA-CPP* specification includes a non-normative appendix that discusses *CPA* composition and negotiation and includes advice as to composition and negotiation procedures.

8.1.4 CPP interfaces

8.1.4.1 Interface to business processes

A *CPP* SHALL be capable of referencing one or more *Business Processes* supported by the *Trading Partner* owning the *CPP* instance. The *CPP* SHALL reference the Roles within a *Business Process* that the user is capable of assuming. An example of a Role could be the notion of a "Seller" and "Buyer" within a "Purchasing" *Business Process*.

The *CPP* SHALL be capable of being stored and retrieved from an ebXML *Registry* Mechanism

A *CPP* SHOULD also describe binding details that are used to build an ebXML *Message Header*.

8.1.5 CPA interfaces

A *CPA* governs the *Business Service Interface* used by a *Trading* Partner to constrain the *Business Service Interface* to a set of parameters agreed to by all *Trading Partners* who will execute such an agreement.

CPA's have *Interface*s to *CPP's* in that the *CPA* is derived through a process of mutual negotiation narrowing the *Trading Partners* capabilities (*CPP*) into what the *Trading Partner* "will" do (*CPA*).

A *CPA* must reference to a specific *Business Process* and the interaction requirements needed to execute that *Business Process*.

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

8.1.6 Non-normative implementation details [CPP and CPA's]

A *CPA* is negotiated after the Discovery and Retrieval Phase 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* SHOULD be negotiated between *Trading Partners*.

In some circumstances there may be a need or desire to describe casual, informal or implied *CPA's*.

An eventual goal of ebXML is to facilitate fully automated *CPA* generation. In order to meet this goal, a formal methodology SHOULD be specified for the *CPA* negotiation process.

8.2 Business process and information modeling

8.2.1 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 collaborations. 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*. *Business Documents* MAY be composed from re-useable *Business Information Objects* (see "Relationships to Core Components" under 8.2.3 "*Interface*s" below). At a lower level, *Business Processes* can be composed of re-useable *Core Processes*, and *Business Information 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 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 set of elements required to configure a runtime system in order to execute 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:

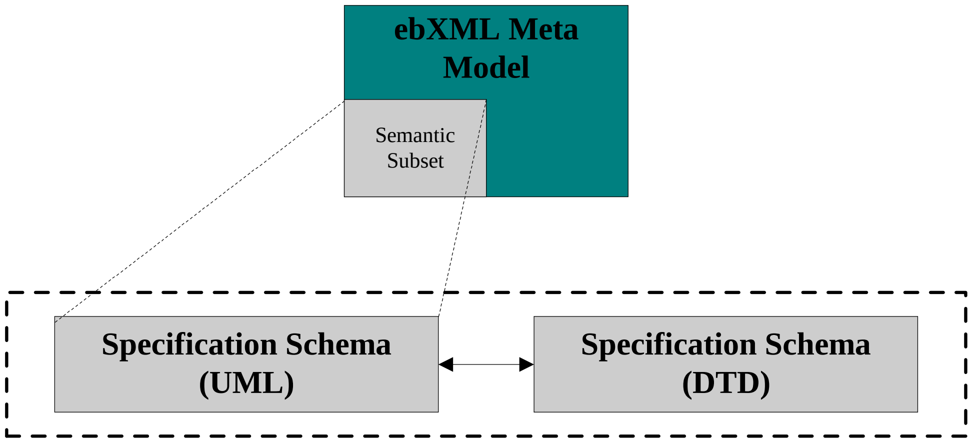


Figure 10 : ***ebXML Meta Model - Semantic Subset***

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 M*essages* and signals between *Trading Partners* to achieve the required electronic 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 patterns. The full specification of a *Business Process* consists of a *Business Process and Information Meta Model* specified against the *Specification Schema* and an identification of the desired pattern(s). This information serves as the primary input for the formation of *Collaboration Protocol Profiles* (*CPP's*) and *CPA's*. This can be shown as follows:

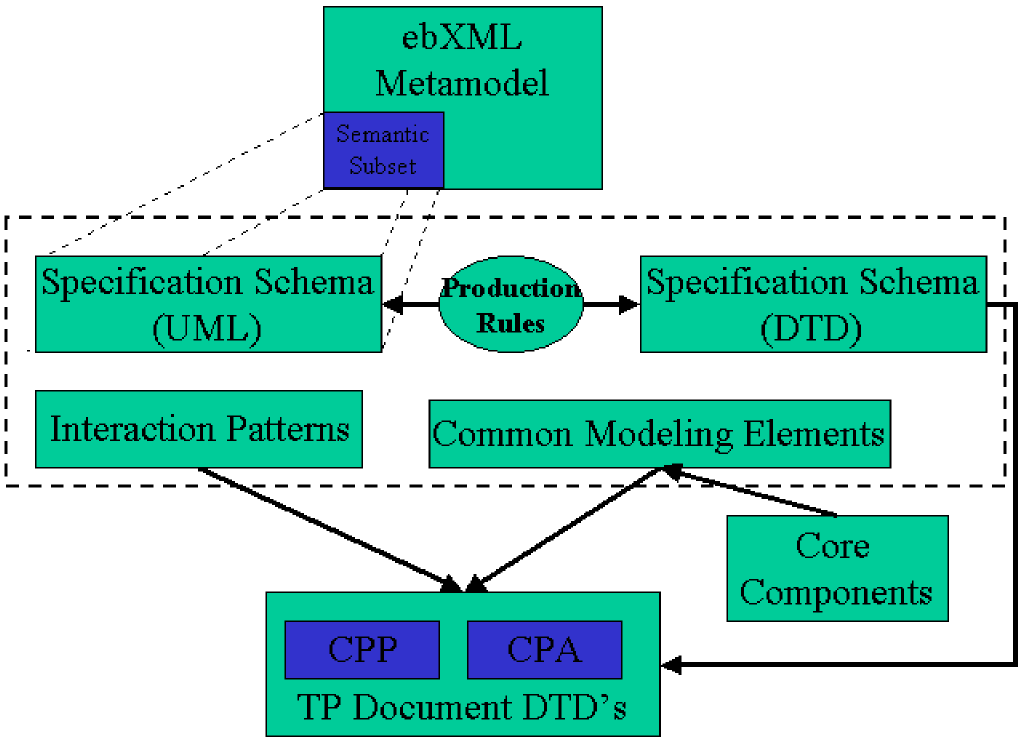


Figure 11: ***ebXML Meta Model***

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*. One of the key benefits of using a single consistent modeling methodology is that it is possible to compare models to avoid duplication of existing *Business Processes*.

To further facilitate the creation of consistent *Business Processes* and information models, ebXML will define a common 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*.

8.2.2 Formal functionality

The representation 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 and retrieval.

To be understood by an application, a *Business Process* SHALL be expressible in *XML* syntax. A *Business Process* MAY be constructed as an *Business Process and Information Meta Model* or an *XML* representation of that model. *Business Processes* are capable of expressing the following types of information:

- Choreography for the exchange of document instances. (e.g. the choreography of necessary *Message* exchanges between two Trading Partners executing a "Purchasing" ebXML transaction.)
- References to *Business Process and Information Meta Model* or *Business Documents* (possibly *DTD's* or *Schemas*) that add structure to business data.
- Definition of the roles for each participant in a *Business Process.*

A *Business Process*:

- Provides the contextual constraints for using *Core Components*
- Provides the framework for establishing *CPAs*
- Specifies the domain owner of a *Business Process,* along with relevant contact information.

Note The above lists are not inclusive.

8.2.3 Interfaces

8.2.3.1 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. The *Interface* between the *Business Process*, its *Information Meta Model,* and the *CPA* is the part of the *Business Process* document. This MAY be instantiated as an *XML* document representing the business transactional and collaboration layers 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.

8.2.3.2 Relationship to core components

A *Business Process* instance SHOULD specify the constraints for exchanging business data with other *Trading Partners*. The business information MAY be comprised of components of the ebXML *Core Library*. A *Business Process* document SHALL reference the *Core Components* directly or indirectly using a *XML* document that references the appropriate Business and Information Models and/or Business Documents (possibly DTD's or Schemas). The mechanism for interfacing with the *Core Components* and *Core Library* SHALL be by way of a unique identifier for each component.

8.2.3.3 Relationship to ebXML messaging

A *Business Process* instance SHALL be capable of being transported from a *Registry Service* to another *Registry Service* via an ebXML *Message*. It SHALL also be capable of being transported between a *Registry* and a users application via the ebXML *Messaging Service*.

8.2.3.4 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 unique identifier.

8.2.4 Non-normative implementation details

The exact composition of *Business Information Objects* 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.

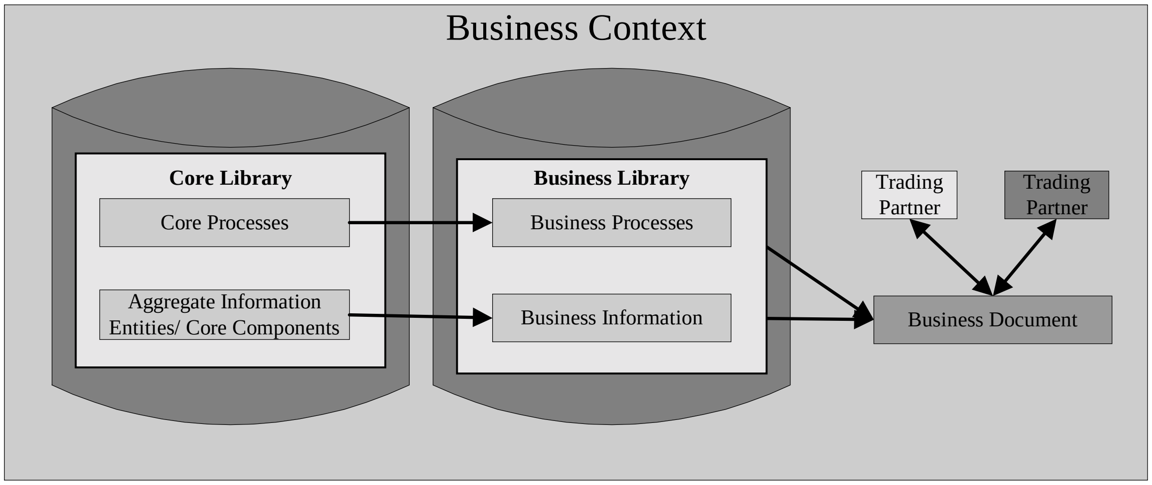


Figure 12: *ebXML Business Process and Information Modeling layer*

ebXML *Business Process and Information Meta Model* MAY be created following the recommended UN/CEFACT *Modeling Methodology* (*UMM*), or MAY be arrived at in any other way, as long as they comply with the ebXML *Business Process and Information Meta Model*.

8.3 Core components and core library functionality

8.3.1 Introduction

A *Core Component* captures information about a real world business concept, and the relationships between that concept, other *Business Information Objects,* and a contextual description that describes how a *Core* or *Aggregate Information Entity* 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 Objects* that may be assembled into *Aggregate Information Entities*.

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*.

8.3.2 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 *eBusiness* needs.

Core Components SHALL be capable of being expressed 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 Objects*.
- Other *Core Components* in combination with zero or more individual pieces of *Business Information Objects*.

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

8.3.3 Interfaces

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

A *Core Component* SHALL 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.

8.3.4 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 Information Entity*. This is referred to as *Business Context*.

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

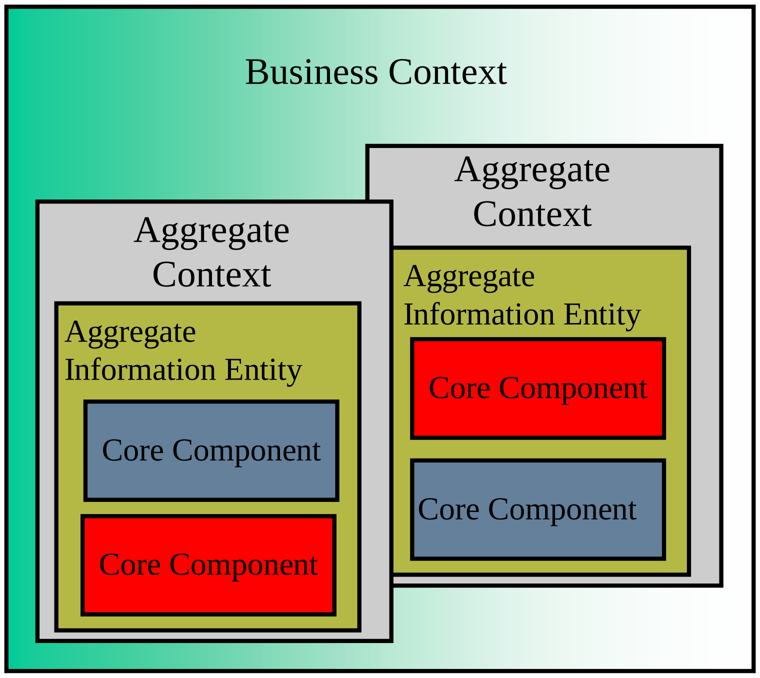


Figure 13: ***Business Context defined in terms of Aggregate Context, Aggregate Information Entities, and Core Components***

The pieces of *Business Information Objects*, 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.

8.4 Registry functionality

8.4.1 Introduction

An ebXML *Registry* provides a set of services that enable the sharing of information between *Trading Partners*. 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*.

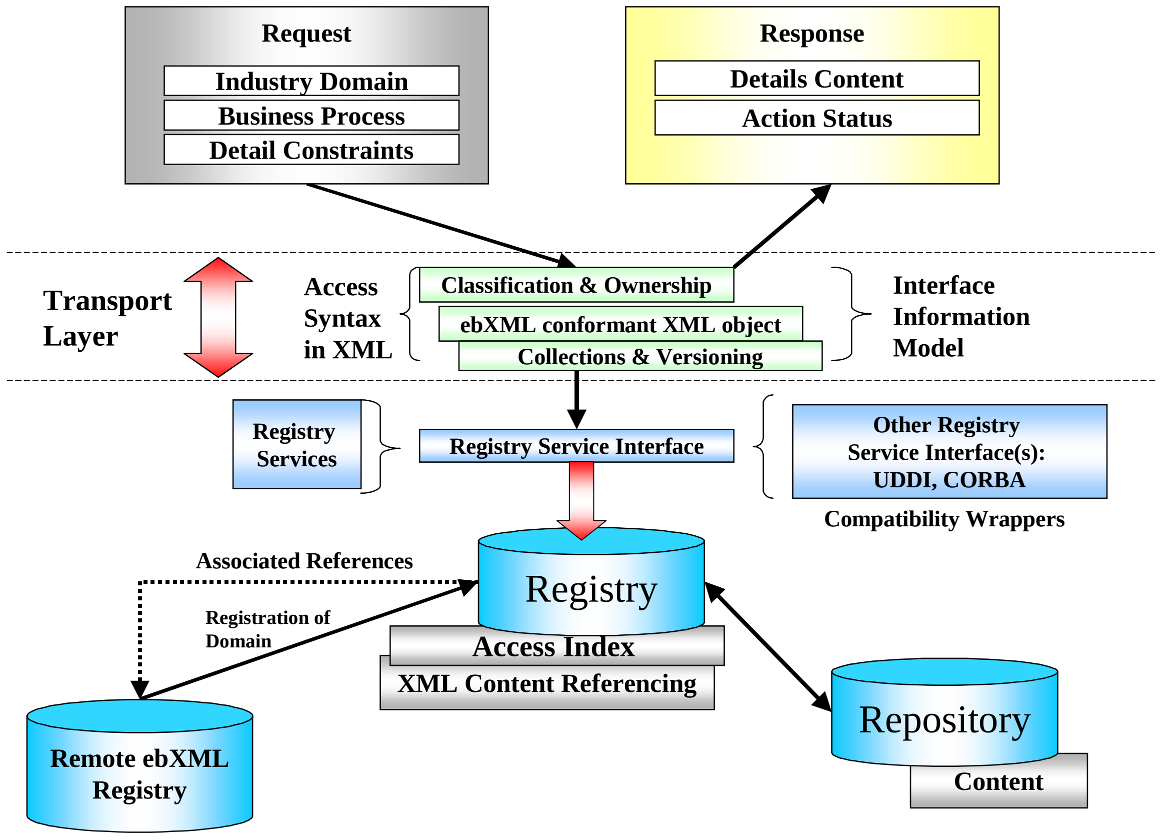


Figure 14: ***Overall Registry Architecture.***

8.4.2 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 as 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 unique identifier. 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 that identify, name, describe it, 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* serves as an application-to-registry access mechanism. Human-to-registry interactions SHALL be built as a layer over a *Registry Interface* (e.g. a Web browser) and not as a separate *Interface*.

The *Registry Interface* SHALL be designed to be independent of the underlying network protocol stack (e.g. HTTP/SMTP over TCP/IP). Specific instructions on how to interact with the *Registry Interface* MAY be contained in the payload of the ebXML *Message*.

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 SHOULD be able to browse and traverse the content based on the available *Registry* classification schemes.

Registry Services 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*.

Unique Identifiers (*UIDs*) SHALL be assigned to all items within an ebXML *Registry System*. *UID* keys are REQUIRED references for all ebXML content. *Universally Unique Identifiers (UUIDs)* MAY be used to ensure that *Registry* entries are truly globally unique, and thus when systems query a *Registry* for a *UUID*, one and only one result SHALL be retrieved.

To facilitate semantic recognition of *Business Process and Information Meta Models*, the *Registry Service* SHALL provide a mechanism for incorporating human readable descriptions of *Registry* items. Existing *Business Process and Information Meta Models* (e.g. RosettaNet PIPs) and *Core Components* SHALL be assigned *UID* keys when they are registered in an ebXML compliant *Registry Service*. These *UID* keys MAY be implemented in physical *XML* syntax in a variety of ways. These mechanisms MAY 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.

8.4.3 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* MAY serve as the transport mechanism for all communication into and out of the *Registry*.

Business process:

Business Processes are published and retrieved via ebXML *Registry Services*.

Core components:

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

8.4.4 Non-normative implementation details

The *Business Process and Information Meta Models* 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 includes, but is not limited to 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 *Business Process and Information Meta Model* for an ebXML *Registry Service* may be an extension of the existing OASIS Registry/Repository Technical Specification, 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.

8.5 Messaging service functionality

8.5.1 Introduction

The ebXML *Message Service* mechanism provides a standard way to exchange business M*essages* among ebXML *Trading Partners*. The ebXML *Messaging Service* provides a reliable means to exchange business M*essages* without relying on proprietary technologies and solutions. An ebXML *Message* contains structures for a *Message Header* (necessary for routing and delivery) and a *Payload* 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 15 below.

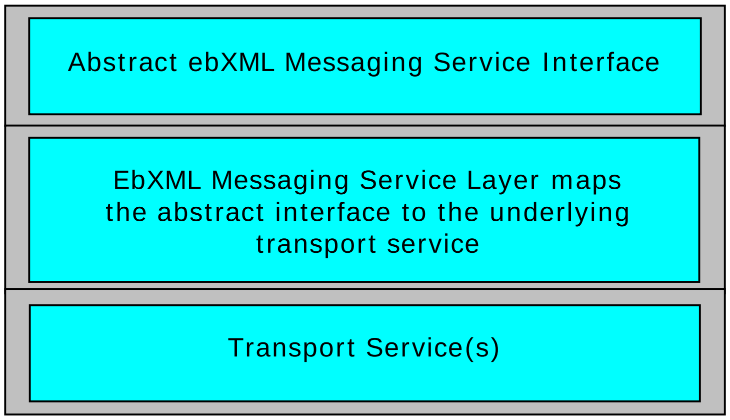

Abstract ebXML Messaging Service Interface
EbXML Messaging Service Layer maps the abstract interface to the underlying transport service
Transport Service(s)

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.

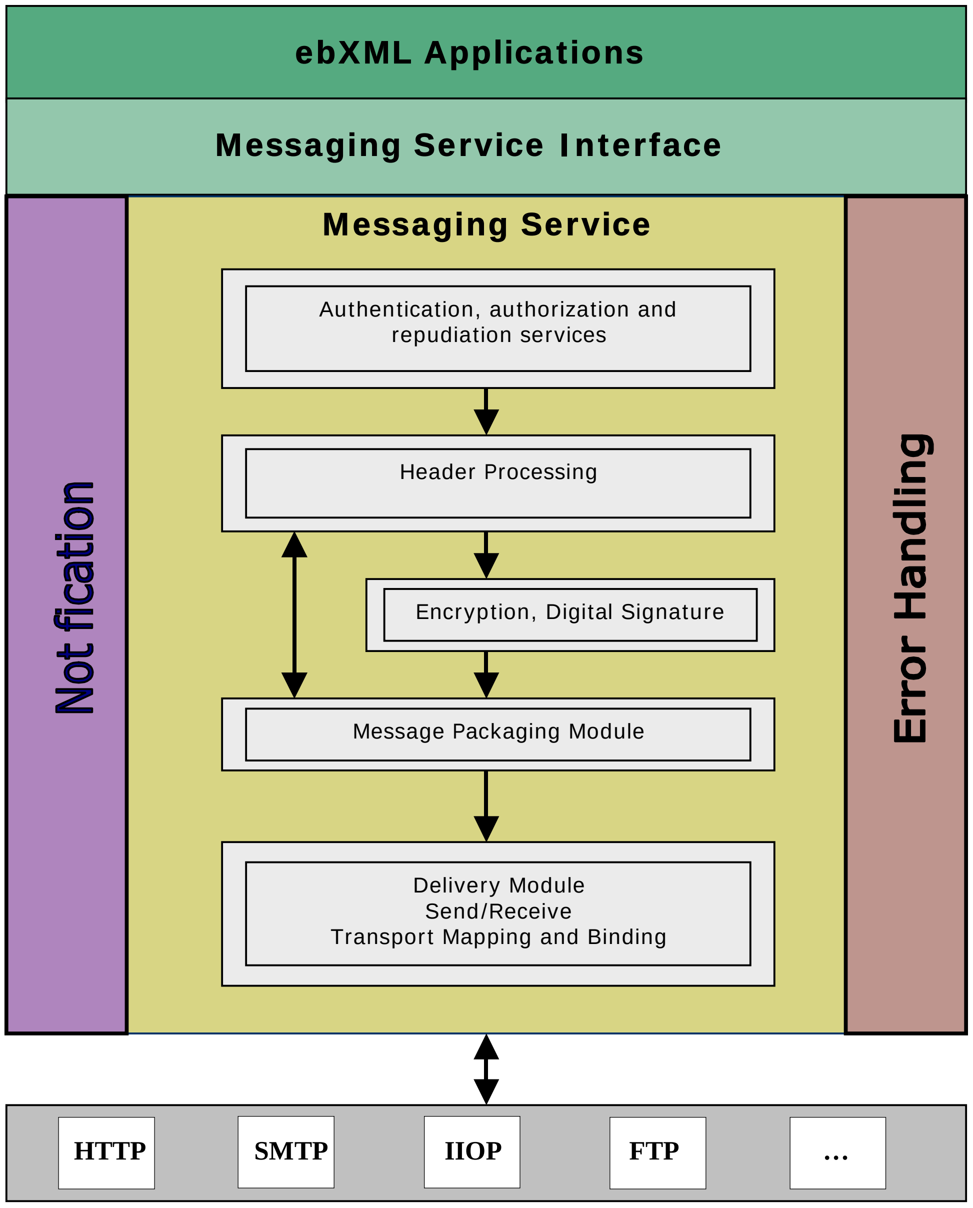


Figure 16: ***The Messaging Service Architecture***

8.5.2 Formal functionality

The ebXML *Messaging Service* provides 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* prescribes formats for all *Messages* between distributed ebXML *Components* including *Registry* mechanisms and compliant user *Applications*.

The ebXML *Messaging Service* does not place any restrictions on the content of the payload.

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

The ebXML *Messaging Service* supports sequencing of payloads in instances where multiple payloads or multiple *Messages* are exchanged between *Trading Partners*.

The ebXML *Messaging Service Layer* enforces the "rules of engagement" as defined by two *Trading Partners* in a *Collaboration Protocol Agreement* (including, but not limited to security and *Business Process* functions related to *Message* delivery). The *Collaboration Protocol Agreement* defines the acceptable behavior by which each *Trading Partner* 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* performs all security related functions including:

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

8.5.3 Interfaces

The ebXML *Messaging Service* provides ebXML with an abstract *Interface* whose functions, at an abstract level, include:

- <u>Send</u> – send an ebXML *Message* – values for the parameters are derived from the ebXML *Message Headers*.
- <u>Receive</u> – indicates willingness to receive an ebXML *Message*.
- <u>Notify</u> – provides notification of expected and unexpected events.
- <u>Inquire</u> – 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

The ebXML *Messaging Service* SHALL help facilitate the *Interface* to an ebXML *Registry*.

8.5.4 Non-normative implementation details

8.5.4.1 ebXML message structure and packaging

Figure 17 below illustrates the logical structure of an ebXML *Message*.

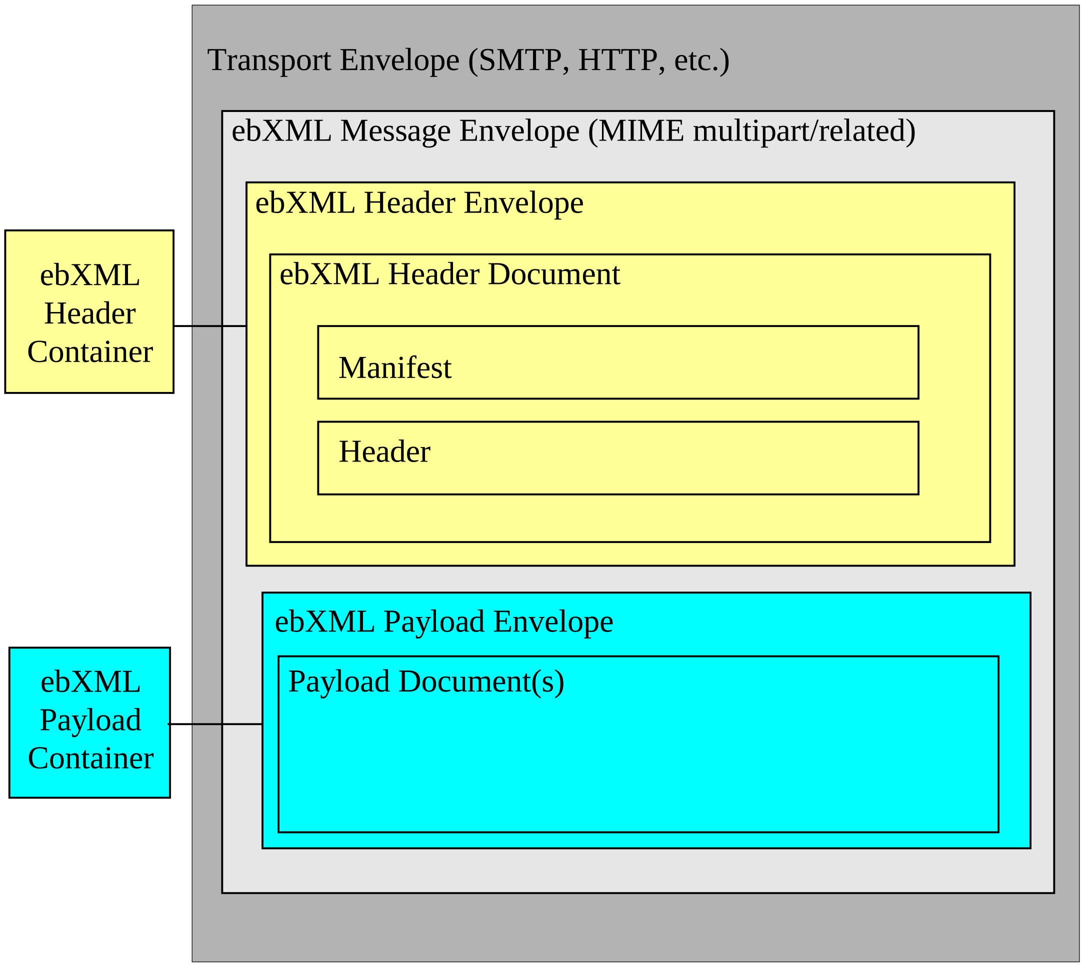


Figure 17: *ebXML Message Structure*

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 *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 Information.

9 Conformance

9.1 Introduction

This clause specifies 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*. Successful interoperability and open interchange is more likely to be achieved if implementations conform to the requirements in the ebXML specifications.

9.2 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.

9.3 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).

9.4 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* SHOULD be tested to available test suites to verify their conformance to ebXML Specifications as soon as test suites are available.

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.

10 Security Considerations

10.1 Introduction

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

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

11 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.

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 the ebXML specifications as a convenient mean for companies to properly run electronic business 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 that defines a *Business Process* involving the use of external business services.

d.) Three or more *Trading Partners* conduct business using shared *Business Processes* 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 Profile (*CPP*).

  Each Profile references:

  - One or more existing *Business Process* found in the ebXML *Registry*
  - One of more *Message* Definitions. Each *Message* definition is built from reusable components (*Core Components*) found in the ebXML *Registry*

  Each Profile (*CPP*) defines:

  - The business transactions that the *Trading Partner* is able to engage into

    - The Technical protocol (like HTPP, SMTP etc) and the technical properties (such as special encryption, validation, authentication) that the *Trading Partner* supports in the engagement
    - The *Trading Partners* acknowledge each other profile and create a *CPA.*
- 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

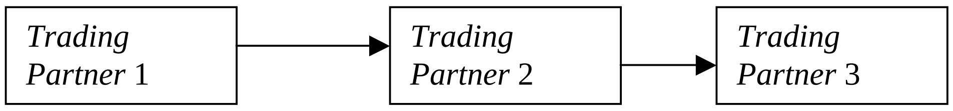
  In both cases, this step is about :

    - Plugging the Legacy into the ebXML technical infrastructure as specified by the *Messaging Service*.
    - 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 parties set-up a business process implementing a supply-chain and run the associated exchanges

The simple case of a supply-chain involving two *Trading Partners* can be redefined in terms of the Scenario 1.

Here we are dealing with situations where more *Trading Partners* 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 local portion of the *Business Process* is managed by each *Trading Partner*, i.e. that each *Trading Partner* is fully responsible of the Business 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 Profile (*CPP*). Each Profile (*CPP*) references:

- One or more existing *Business Process* found in the ebXML *Registry*
- One of more *Message* Definitions. Each *Message* definition is built from reusable components (*Core Components*) found in the ebXML *Registry*

  Each Profile (*CPP*) 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 Technical protocol (like HTPP, SMTP etc) and the technical properties (such as special encryption, validation, authentication) that the *Trading Partner* supports in the engagement. As to "*Trading Partner* 2", 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 profile and create the relevant *CPA*. (at least 2 in this Scenario).
  - "*Trading Partner* 2" 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:

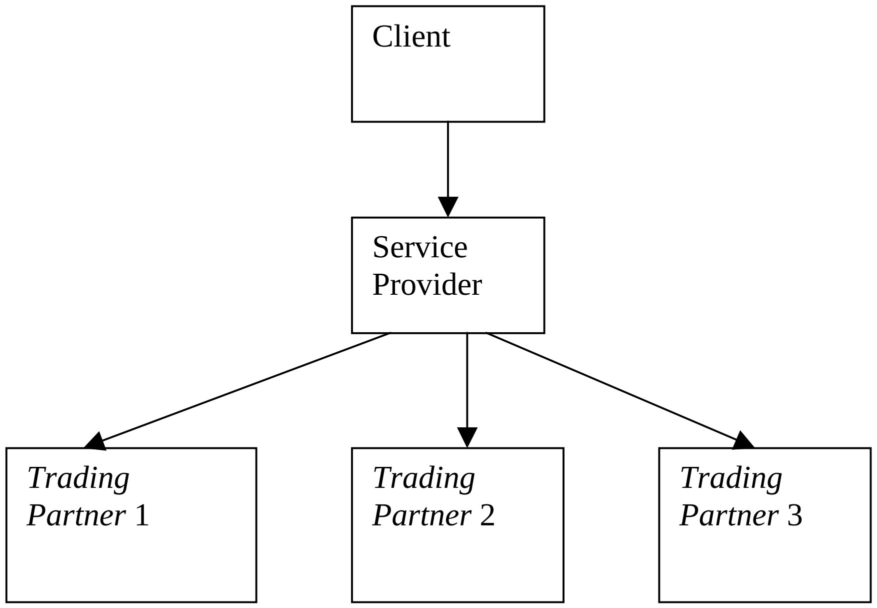
  - Plugging the Legacy into the ebXML technical infrastructure as specified by the *Messaging Service*
  - Granting that the software is able to properly engage the stated conversations
  - Granting that the exchanges semantically conform to the agreed upon ebXML *Message* definitions
  - Granting that 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 performing the agreed upon commercial transactions.
  - "*Trading Partner* 3" places an order at "*Trading Partner* 2"
  - "*Trading Partner* 2" (eventually) places an order with "*Trading Partner* 1"
  - "*Trading Partner* 1" fulfills the order
  - "*Trading Partner* 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 *Trading Partners* that provide information to build the final answer.

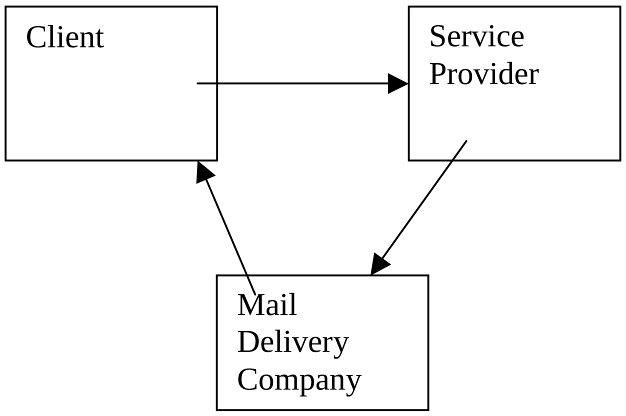
In the simplest case, this Scenario could be modeled as follows :



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

Scenario 4: Three or more trading partners conduct business using shared business processes and run the associated exchanges

This Scenario is about 3 or more *Trading Partners* having complex 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 or good that is ordered by the *Client* with a Service Provider is delivered by a 3rd party.

In this scenario:

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

  Each Profile (*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 special encryption, validation, authentication) that the *Trading Partner* supports in the engagement.

      In case the technical infrastructure underlying the different exchanges differes, 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 *CPA.* Each *Trading Partner,* in this Scenario, must be able to negotiate at least 2 Agreements.

      Each *Trading Partner* is enagaged in 2 Agreements (*CPA*).
- 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 Legacy into the ebXML technical infrastructure as specified by the *Messaging Service.*
- 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 technical conform with the underlying ebXML *Messaging Service.*
- 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 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.