ebXML: Global Standard for Electronic Business

Sujatha Babu

ebXML: Global Standard for Electronic Business

A Thesis-Equivalent Project

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

in Partial Fulfillment

of the Requirements for the Degree

Master of Science

University of Nebraska at Omaha

by

Sujatha Babu

December, 2004

Thesis-Equivalent Project Acceptance

Acceptance for the faculty of the Graduate College, University of Nebraska, in partial fulfillment of the requirements for the degree (name the degree), University of Nebraska at Omaha.

Committee

Name	Signature
K. L. Dick	[illegible]
S. QURESHI	[illegible]
Hesham Ali	[illegible]

Chairperson K. L. Dick

Date ______________________

ebXML: Global Standard for Electronic Business

Sujatha Babu, MS

University of Nebraska, 2004

Advisor: Ken Dick, Ph.D.

Business-to-business integration is transforming the market and has already begun to increase the efficiency of those companies involved. EDI (Electronic Document Interchange) became very popular during 1970's; Today EDI transactions total about $750 billion year. EDI is being used by 90% of Fortune 1000 companies. It has indeed become a dominant technology for the largest companies, on the other hand it has been adopted by less than 5% of small and medium sized companies in general and, of these, many use EDI only because their larger customers require it. The reason behind is that EDI is a difficult, complex technology to implement usually comes with high transactional cost. Hence it is suitable for large companies with large volume of transactions. EDI uses fixed, rigid and compressed data format that is difficult to decipher and debug. The data exchange in EDI happens in proprietary VAN (value added network) which is an expensive solution.

EbXML (Electronic Business XML) envisioned creating a single global electronic marketplace where enterprises of any size and in any geographic location can

meet and conduct business with each other through exchange of xml based messages. The XML (the Extensible Markup Language) has rapidly imposed itself as a popular format for exchange of information on the web. The very nature of XML is that it is a structured document format, in that it represents not only the information to be exchanged, but the metadata encapsulating its meaning. XML technology has potential to solve the existing problems in current EDI systems. Using ebXML, companies have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes. EbXML is designed to provide a simple way for companies to find one another and conduct business over the Web, allowing those with different platforms to speak a common language. EbXML targets to provide low cost solutions for small and medium enterprises as well as complex solution for large enterprises. This project attempts to implement a prototype of ebXML messaging service as per ebXML specification to obtain the insight look of feasibility and suitability of XML solution for EDI.

Table of Contents

List of Figures

Chapter 1 Introduction and Problem Definition

1.1 Introduction

With the revolution of computers business has adopted the usage of new technologies to advance. Today companies and organizations are heavily depended on software infrastructure to make fast and successful business transactions in order to withstand the pressure and competition in the market. The key aspects for any two companies who want to conduct a business are the communication and the exchange of business documents. The exchange of business information between enterprises became the focus and standards like Electronic Document Interchange emerged.

1.2 Background

1.2.1 Electronic Document Interchange

Electronic Document Interchange (EDI) was developed in the 1970's to enable computer to computer exchange of information between companies, using an industry standard format. Two major standards: American National Standards Institute X12 (ANSI X12) and United nations/Electronic Data Interchange for Administration Commerce and Transport (UN/EDIFACT). ANSI X12 standard is primarily North American Standard. UN/EDIFACT is followed by rest of the globe. Industry specific EDI standard was followed by all major industries today. EDI transactions total about $750 billion year and it is being used by 90% of Fortune 1000 companies.

EDI has been very successful; there are some major drawbacks to this system. First of all, an EDI system is expensive, not affordable by small or medium sized companies. Money is saved during the transactions, so EDI is only interesting if a company does a lot of transactions. For an EDI system specialized middleware is needed and a company that implements EDI has to agree upon a message format with each of its trading partners that it wants to use EDI with. The system offers no flexibility towards the message: the format is fixed. Finally, EDI only offers the possibility of sending messages and receiving. Other information, like business profiles or business processes or binary data cannot be exchanged.

EDI trading partners exchange business data between their respective computer systems provided fast and accurate data that was otherwise subject to processing and handling delays inherent in surface mail, faxes and data entry.

Sample ANSI X12 EDI transaction of a medical claim:

```
ISA*00* *00* *ZZ*0079
*ZZ*ENCOUNTER*030101*1220*U*00401*000000001*0*T*:~GS*HC*0079*ENCOUNTER*2
0040712*1220*11122233*X*004010X098A1~ST*837*100001~BHT*0019*00*EP190*20030
101*1220*RP~REF*87*004010X098A1~NM1*41*2*SUBMITTOR CLEARINGHOUSE
CORP*****46*0079~PER*IC*SALLY SMITH*TE*2484890000~NM1*40*2*MDCH
MEDICAID*****46*D00111~HL*1**20*1~NM1*85*2*PROFESSIONAL
PROVIDER*****24*123456789~N3*145 PLYMOUTH RD~N4*ANN
ARBOR*MI*48105~REF*1D*771234567~PER*IC*RUBY ORTHOPEDIC
GROUP*TE*7346669990~HL*2*1*22*0~SBR*S*18*******MC~NM1*IL*1*DOE*JOHN*JAY***MI
*12345678~N3*123 MY STREET~N4*MT
CLEMENS*MI*48043~DMG*D8*19900101*M~REF*SY*123456789~NM1*PR*2*MDCH
MEDICAID*****PI*D00111~CLM*98MH1001*175***11::1*Y*A*Y*Y*B*EM~DTP*304*D8*20020
905~CN1*05~REF*D9*98MH1001~REF*EA*MREC~NTE*ADD*CONSULTATION~HI*BK:805
4~NM1*82*1*KYLE*PHYSICIAN*JONATHON***34*756334111~PRV*PE*ZZ*207RS0010X~R
EF*1D*908321456~SBR*P*18*PG10010*MEDICAID*HM****HM~AMT*D*80~AMT*B6*80~DM
G*D8*19900101*M~OI***N*P**Y~NM1*IL*1*DOE*JOHN*JAY***MI*12345678901~N3*123 MY
STREET~N4*MT CLEMENS*MI*48043~NM1*PR*2*MEDICAID HEALTH
PLAN*****PI*171111112~REF*F8*98MH1001~LX*1~SV1*HC:99244*175*UN*1***1**Y~DTP*
472*D8*20040701~
REF*6R*98MH100101~SVD*171111112*80*HC:99244**1~CAS*CO*42*95*1~DTP*573*D8*
20040701~SE*48*100001~GE*1*111222333~IEA*1*000000001~
```

EDI works by encoding data in a very specific, standard format. EDI communication consists of one or more messages, each conforming to a subset of the standard format. Within each message, delimiters define a series of segments, the first three characters of which contains the code for the type of the segment. Within each segment, a series of codes and numbers (separated by another delimiter) define the data being transmitted. EDI messages are highly compressed and based on codes as seen in the example above. Due to the extensive use of very compact codes and concise data structures, EDI messages are almost impossible to humanly create and interpret without extensive documentation.

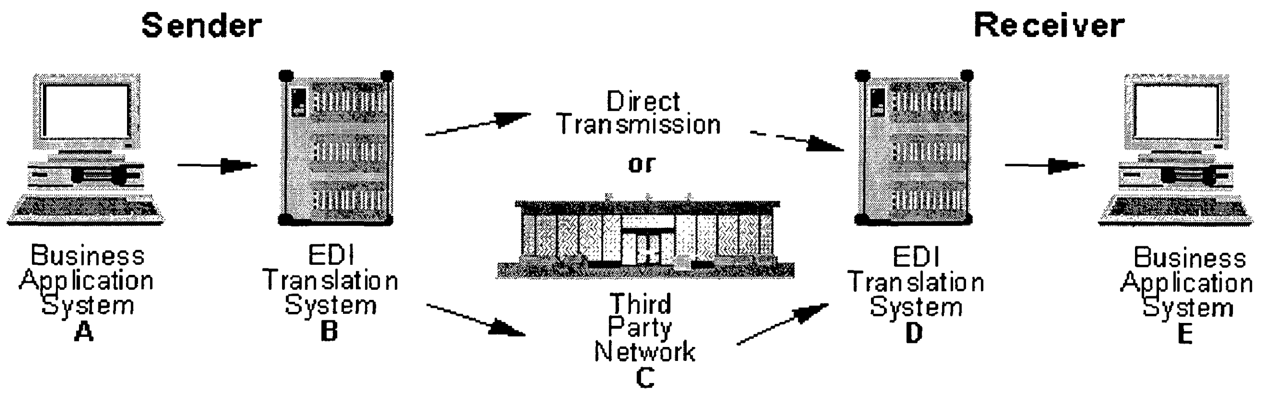


Figure 1-1 EDI Transaction [Source: http://www.fritolay.com/edi/pages/edi]

The above figure explains how an EDI transaction takes place between two business applications belongs to the different trading partners.

A. The sender assembles the data using its own business application system.

B. This data is translated into an EDI standard format (i.e. transaction set)

C. The transaction set is transmitted either through a Value Added Network (VAN) or directly to the receiver's EDI translation system.

D. The transaction set, in EDI standard format, is translated into files that are usable by the receiver's business application system.

E. The files are processed using the receiver's business application system.

EDI messages are never used "as is" in EDI practice. Because the standard messages have evolved through accretion of optional data elements to handle the information requirements of every conceivable business relationship, they contain vastly more information than is typically necessary in any particular case. As a result, the messages that are exchanged between trading partners are always substantially reduced subsets that are heavily customized to that relationship.

EDI has a self-fulfilling bias against the kind of spontaneous commerce to be enabled in open trading communities; because of the historically high cost of EDI integration, companies don't use it unless they have entered into a long term, high volume or high value business arrangement.

EDI standards are large, complex, difficult to implement, and often have high price tags attached for software, networks, and consultants. Traditional EDI refers to the use of rigid transaction sets with business rules embedded in them. This model simply does not work in today's rapidly changing business environment.

This problem is compounded by the fact that companies have chosen to interpret these transaction set standards in ways that suit their unique business requirements. As a result, vendors who engage in EDI with multiple customers typically must create a unique solution to handle the transaction sets for each company. This makes the implementation of EDI far too expensive and took too long to implement, especially for Small and Medium size Enterprises [1].

While North America may have its X12 standard, the rest of the world uses the UN/EDIFACT standard for EDI. UN/EDIFACT resembles X12 in many ways but still has many differences that require companies doing business internationally to carry at least two sets of electronic formats for each transaction.

To make matters worse, each industry defines its implementation guidelines for the X12 standard differently. In many respects, one cannot avoid this situation since each industry has its own set of business rules and practices.

In addition to different EDI standards around the world and in different industries, the X12 standards change every year. The most basic change is the addition of new transaction sets. Every company using EDI in the industry had to change their code tables as a result.

Using EDI day-to-day gets pricey. Translator software that takes data from legacy systems and formats them in the X12 syntax and back again needs to change with the growing and ever changing X12 standard. Therefore it often has a high initial price tag and maintenance costs.

EDI has indeed become a dominant technology for the largest companies, on the other hand it has been adopted by less than 5% of small and medium sized companies in general and, of these, many use EDI only because their larger customers require it. It may seem clear that EDI had failed to serve Small to Medium Sized enterprises (SME) over the preceding 25 years.

1.2.2 Electronic Business XML

Electronic Business XML (ebXML) can be viewed as a next generation of EDI. The ebXML began to emerge during 1999 as an effort by United Nations Center for Trade Facilitation and Electronic Business (CEFACT) and Organization for the Advancement of Structured Information Standards (OASIS). EbXML vision is to create a single global electronic marketplace where enterprises of any size and in any geographic location can meet and conduct business with each other through exchange of xml based messages.

The ebXML aims to lower the initial costs so that SME's can also participate in the process of electronic business. EbXML has some advantages over EDI. First of all, information is stored and exchanged in the XML format. Secondly, the internet is used to exchange documents. Since nowadays most companies are connected to the internet, a large infrastructure is available [3].

Companies can use protocols like http, ftp, SMTP etc for the exchange. Communication consists of exchanging XML document. XML is basically a metadata language - it contains the information about the data being relayed. XML is a language for the description of structured documents and data. Information coded in XML is easy to read and understand, plus it can be processed easily by computers.

In XML, there is no fixed set of tags; new tags can be created based on need. Information publishers have the ability to create and define new tags and attribute names at will. In XML one can work with Document Type Definitions (DTD) and Schema to define any number of documents that form a language of their own. To meet specific needs of various industries - financial, legal, publishing, etc., vast resource of specific document vocabularies have already been created using XML. Since there is no processing specification or limitation (XML does not specify how the data should be processed) with XML, its documents can be exchanged across multiple platforms, database and applications. The only condition is that the subscriber data stores and applications should be XML-aware.

XML messages include embedded metadata. The metadata provides business context, which cryptic EDI messages lack. Therefore, XML messages are less prone to errors in interpretation. XML is potentially more flexible and is truly an Internet format, so it requires no special kind of network infrastructure. In the long run, XML will prove to be easier for small and midsize businesses to use for Business-to-Business (B2B) [2].

It will be easier to incorporate components such as digital signature, smartcard authorization, routing instructions, spreadsheet, graph into an XML message, thus making use of the intrinsic flexibility and extensibility of XML.

The major benefit of XML is that virtually all software and service suppliers are committed to providing XML support for their products. Increasingly, XML is being used within companies as the way to provide inter-process communication. Inadequacies within XML are being overcome by the huge investment in XML-based tools, many of which are available at no charge.

XML documents can contain any possible data type - from multimedia data (image, sound and video) to active components (Java applets, ActiveX). Mapping existing data structures like file systems or relational databases to XML is simple. XML supports multiple data formats and can cover all existing data structures. It would be clear from the benefits of XML; it would quickly and easily overcome the problems with EDI transactions.

EbXML is targeting to provide low cost solutions for small and medium enterprises as well as complex solution for large enterprises. Eventually ebXML will become dominant technology and may replace EDI.

1.3 Purpose of the project

The objective of this master's project is to study how ebXML standard can be used between two companies for electronic business transactions. The objective would be achieved by developing a prototype for a business scenario. This project will have a detailed discussion about framework, various concepts in the ebXML proposed architecture. Main concepts include: Registry, Business Processes, Collaboration protocol profile and Messaging.

To summarize the following aspects will be covered in this project:

- To identify problems, limitations of existing EDI technology.
- To investigate the emerging ebXML framework and it's potential to solve these problems.
- To analyze the EBXML messaging specifications to build a prototype of exchanging messages between two trading partners.
- To identify the impact and future trends in the development of ebXML.

1.4 Scope of the project

The main scope of this project is to create a prototype of ebXML messaging process between two trading partners. The transactions will be implemented as per messaging standards of ebXML specifications. Implementation involves use of client side programming language with extensive use of Simple Object Access Protocol (SOAP) based XML messages.

Because the ebXML is such a recent development in electronic commerce, by studying ebXML implementation, the results of this project can be beneficial for companies considering implementation of ebXML.

1.5 Related Work

In general, ebXML is an evolving technology. There are many business high level ideas how this specification works but there are not enough concrete implementations which demonstrate the advantage of ebXML. Therefore there are limited resources available in the context of real project environment.

1.6 Structure of the project

The project paper is organized as follows:

Chapter 1: This chapter provides the introduction, background, purpose and scope of the project and related work.

Chapter 2: This chapter provides an introduction and background information about XML, ebXML and ebXML scenario.

Chapter 3: This chapter provides detailed information about ebXML framework such as Registry/repository, Business Process Specification Schema, Collaboration Protocol Profile, Collaboration Protocol Agreement and ebXML messaging services.

Chapter 4: This chapter provides details of implementation of ebXML messaging service prototype.

Chapter 5: This chapter provides future trends and the conclusion we achieved.

Chapter 2 Overview of ebXML

2.1 Overview of XML

XML (extensible Markup Language) is a markup language for documents containing structured information. XML is a standard from the World Wide Web consortium. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

XML has basically two parts: the definition of a structure and one or many instances of it. First the structure is defined. XML Schema provides data type definition and inheritance. Structured information can contain words, pictures, etc. XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there's no predefined tag set, there can't be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by style sheets.

XML is platform-independent and well-supported open standard. Because of its nature to define data structure it is platform and programming language independent. The XML language is a flexible data formatting language for the messages. Both the client and the server have to use a standard messaging format. XML solves a common problem in data interchange, which is defining how to write and use data and documents as flat files in a standard format. With XML, we have a common language in a flat file format that is both human and machine-readable for communicating between systems. A DTD or schema sets the semantic rules for the elements and attributes in an XML document. The language has flexibility to adapt to system requirements by allowing different vocabularies and semantics in each business context. The parties using or exchanging an XML document can validate using a DTD that their copies of XML documents follow the same common rules.

The true power of XML to improve business processes is evident when multiple documents all use the same public data format. A single software application can process the set of a document. If the format is publicly available, anyone can generate a document that can be processed by the software. The flexibility in the basic structure of XML application that adapts to different requirement and business scenarios is perfect for exchanging data between heterogeneous systems.

ebXML.org says "ebXML is an evolutionary change from EDI to XML technologies. It opens a migration path from EDI and developing XML standards, while providing support for multi-lingual, national, and international trade requirements. As a pragmatic compromise between XML-centric and EDI-centric worldviews, it combines ideas from both into an open integration framework. Within this cross-industry framework, EDI investments in business processes can be preserved in an architecture that leverages the technical capability of XML".

2.2 Overview of ebXML

In contrast to EDI which is only a messaging standard, ebXML is a framework that contains a number of elements that can be used for doing e-business. It is not required that organizations utilize all of the ebXML specifications. Organizations can pick and choose the specifications that best serve their business needs. The ebXML framework is a set of building blocks. The ebXML technical architecture document has been written as a general guideline for describing at a high level how the ebXML components or building blocks fit together.

EbXML is designed on three basic concepts:

- Provide an infrastructure that ensures data communication interoperability. It is provided through standard message transport mechanism with a well defined interface, packaging rules and a predictable delivery and security model. A business service interface that handles incoming and outgoing messages at either end of the transport.
- Provide a semantic framework that ensures business interoperability. It is provided through a meta-model for defining business process and information models. A set of re-usable core components that reflect common business semantics and xml vocabularies.
- Provide a discovery mechanism that allows enterprises to find each other, agree to become trading partners and conduct business with each other. Shared repository network where enterprises can register and discover

each other's business services via partner profile information. Process for defining and agreeing to a formal Collaboration Protocol Agreement (CPA), if desired, which can be based on the intersection of individual business Collaboration Protocol Profile (CPP). Shared repository for company profiles, business process models and related message structures.

2.3 ebXML Scenario

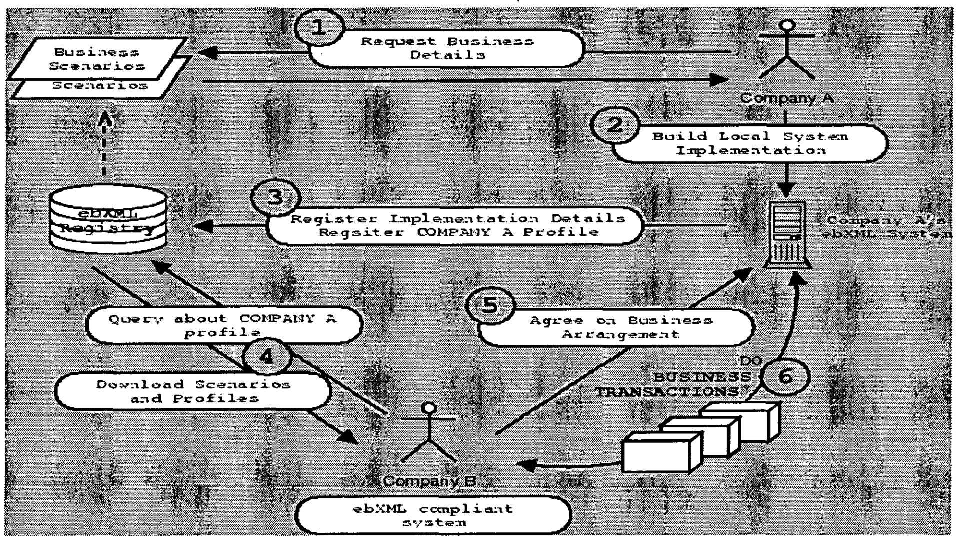

1
Request Business Details
Business Scenarios
Scenarios
Company A
2
Build Local System Implementation
3
ebXML Registry
Register Implementation Details Register COMPANY A Profile
Company A's ebXML System
Query about COMPANY A profile
4
Download Scenarios and Profiles
5
Agree on Business Arrangement
DO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

Figure 2-1 Overview of the interaction of two companies conducting e-Business using ebXML. (Source: ebxml.org)

(1) Company A decides to conduct electronic business based on ebXML. They consult the registry on the internet to check to see any pre-determined common business process that match their core business process, that way they can re-use the existing business processes. In case of un-availability, Company A develops a new business processes according to the ebXML business process specification schema.

(2) Company A implements a Collaboration protocol profile (CPP, their business profile information) and submits to an ebXML registry. CPP contains business process capabilities, constraints and technical ebXML information. This information is used by other companies to discover new trading partners from registry listing.

(3) Company A submits its own business profile information to the ebXML registry. The business profile submitted to the ebXML registry describes the company's ebXML capabilities and constraints, as well as its supported business scenarios.

(4) ebXML compliant Company B discovers company A, who is also ebXML compliant and also identifies the business scenario supported by company A.

(5) Company B receives the CPP of company A. Company B has then two CPP's: Company A's CPP and company B's CPP. ebXML then derives a third document from the intersection of the two CPP's called Collaboration Protocol Agreement (CPA). The two companies negotiate technical details and functionality overrides and outlines the mutually agreed upon business scenarios and specific agreements in the form of a CPA.

(6) Company A and B are now ready to engage in e-business using ebXML. The two companies (software) can send and receive ebXML messages containing ebXML business documents, over the secure and reliable ebXML Messaging Service.

Chapter 3 The ebXML Framework

3.1 Introduction

ebXML can be viewed as a comprehensive treatment for out-standing technical challenges in b2b electronic commerce evolution. ebXML is comprised of five components that provide all the functionality to carry out existing electronic commerce as well as to automate the process which are manually done today. Five components are: Messaging Service, Registry and Repository, Core components, Collaboration protocol profile (CPP) and Business process specification schema.

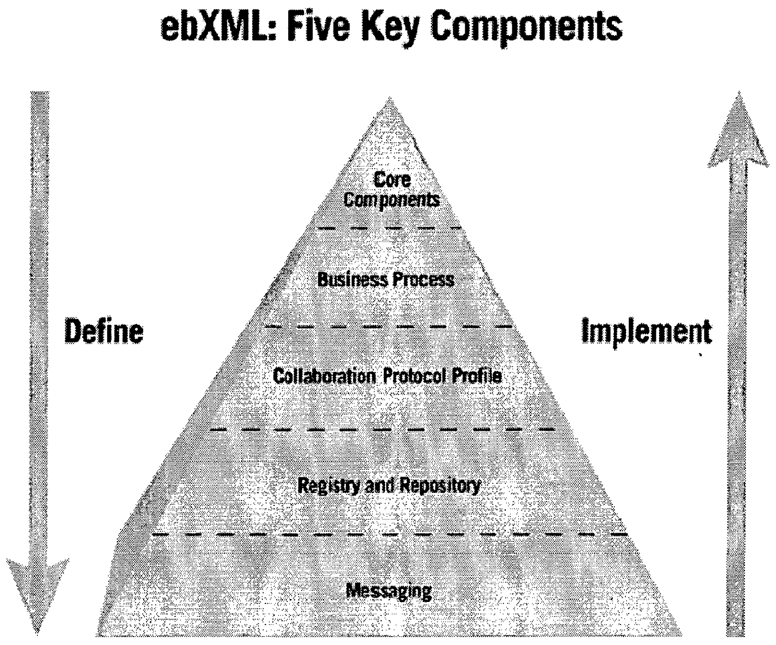


Figure 3-1 ebXML five key components

3.2 ebXML Registry/Repository

A registry is an open directory where companies list a very detailed description of their electronic capabilities. The registry can be viewed as a database of items that support doing business electronically. Technically speaking, a registry stores information about items that actually reside in a repository. Registry act as an interface to access the information of ebXML registry/repository, repository is a storage mechanism where the objects are stored, managed by ebXML registry services. The ebXML registry may be thought of as warehouse (repository) and a catalog (registry) for this warehouse. Similar to a catalog (which contains meta-information about contents in the warehouse), the registry contains information on objects in the repository. Items in the repository are created, updated, or deleted through requests made to the registry. An original goal of the ebXML registry was to support a fully distributed, networked set of interacting registries that would provide transparent interaction to any ebXML compliant registry by interfacing with only one of them. In other words, an interface for accessing and discovering company profiles, business semantics, core components, trading partner specifications and other objects shared by companies. It is important to note that ebXML registry stores metadata about specification not the actual documents.

The ebXML Registry architecture is based on client-server architecture. The communication can be based on ebXML Messaging Service or by HTTP. Client can be designed in any platform but the communication between server and registry should follow the ebXML registry interface. As per ebXML specification, registry/repository is an optional component that can be used to support public sharing of b2b artifacts.

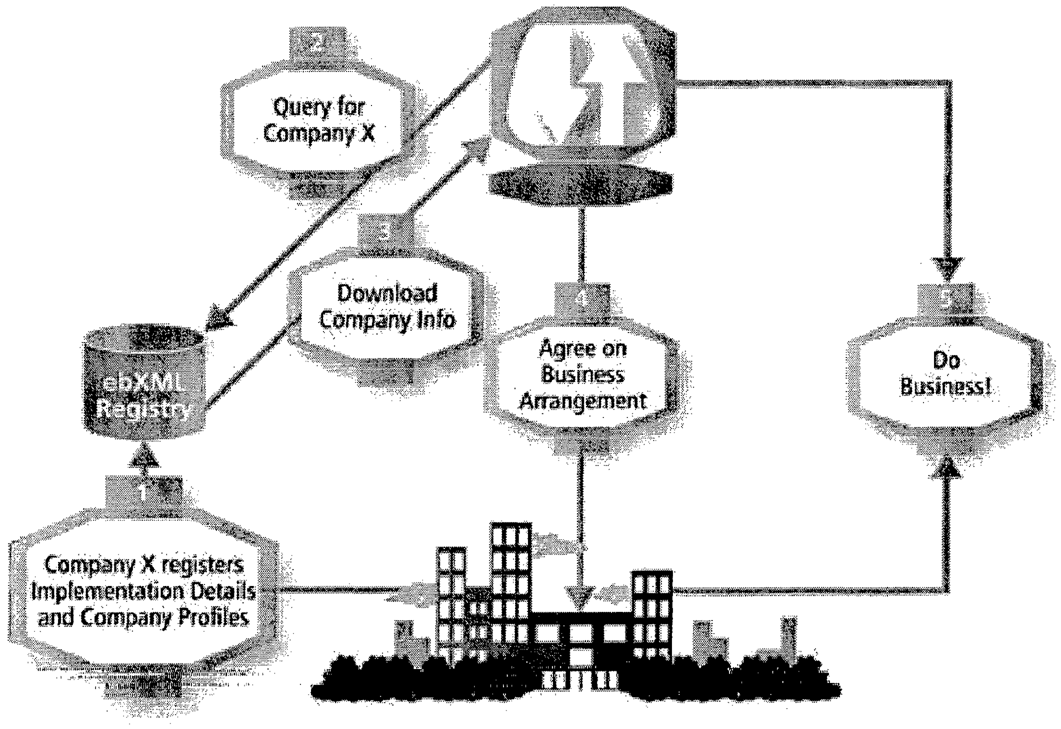

2
Query for Company X
3
Download Company Info
4
Agree on Business Arrangement
5
Do Business!
ebXML Registry
1
Company X registers Implementation Details and Company Profiles

Figure 3-2 Companies Interaction with ebXML registry

Two main objects of registry are: the RegistryEntry and the ClassificationNode. the RegistryEntry may consists of ebXML object, such as a Collaboration Protocol Profile (CPP), a Collaboration Protocol Agreement, core components etc., ClassificationNodes are meant for creating an hierarchical tree structure to define the classification of objects. New ClassificationNodes can be created, queried and updated.

There are two interfaces available in ebXML registry specification: ObjectManager and the ObjectQueryManager.

ObjectQueryManager interface provides the way to search the registry and retrieve objects based on the query. ObjectManager interface creates new object and change the state on the existing objects. ebXML objects undergo four different states. When the object is newly created in the registry the initially it will be "submitted state". Once an object was transitioned to "approved state", it can be accessed and updated by the business parties. After a while, the object may get retired and reach "deprecated state". In this state object is accessible but can't be updated any more. Eventually it will be moved to "removed state", the object will be removed from registry and no longer will be accessible by the business parties.

ObjectQueryManager supports various methods for accessing the objects. Three basic methods are: the Browse and drill down Query, the Filter query and SQL query. Browse and drill down Query has three methods to locate a particular service. getRootClassificationNodes method returns all ClasssificationNodes in the registry that don't have a parent. getClassificationTree drills down into one of these nodes by retrieving all of its children. getClassifiedObjects looks at the RegistryEntries associated with that ClassificationNode.

Filter Query supports more difficult queries such as XML syntax describing a set of class filters. Each of these class filters is a predicate clause intended to restrict the result set, so that you can specify various class filters to restrict the result set.

```
<AdhocQueryRequest>
   <ResponseOption returnType = "RegistryObject"/>
   <FilterQuery>
     <RegistryObjectQuery>
       <ClassifiedByBranch>
          <ClassificationNodeQuery>
             <DescriptionBranch>
               <LocalizedStringFilter>
                  <Clause>
                    <SimpleClause leftArgument = "value">
                       <StringClausestringPredicate= "Equal">transistor</StringClause>
                    </SimpleClause>
                  </Clause>
               </LocalizedStringFilter>
             </DescriptionBranch>
          </ClassificationNodeQuery>
       </ClassifiedByBranch>
   </FilterQuery>
</AdhocQueryRequest>
```

Figure 3-3 Sample Filter Query of Registry object

SQL Query interface supports a basic subset of the SELECT statement. It is also been extended to allow for the calling of stored procedures. The specification defines a binding between the RIM and a set of fictional database tables that are used in SQL Query. When querying ClassificationNodes, the primary identifier is a unique ID created by the registry, and using this ID, you can query other nodes by parents and so on. Methods discussed above don't return actual data rather a unique identifier of the Repository-Entries returns. In order to get the real data getContent method is used.

The ebXML registry defines measures for ensuring the integrity of information stored in it. To allow access to authorized users only, it also allows specification of access-control policies. Every object registered and stored in the registry has a Universal unique ID (UUID). The UUID may be assigned by the registry or provided while submitting an object to the registry [5].

Several companies are working to provide commercial implementations of the ebXML Registry. It is expected that ebXML registry will be available free of cost in near future.

3.3 Business Processes specification Schema

The process used to do business among business partners involves many steps. The process to issue a purchase orders is an example of such a process. Business Processes Specification Schema provides the definition in the form of an XML DTD that describes how an organization conducts its business. The advantage of an XML-specific business process specification is that it can be processed easily by the computers.

It is roughly a business scenario about what happens, who the trading partners are, what the roles of the trading partners are, which documents the trading partners exchange, in what order the documents are exchanged and what information is in the documents (structure of the documents). The specification for business process definition enables an organization to express its business processes in a specific scenario so that they are understandable by other organizations. This enables the integration of business processes within a company or between companies.

While the CPA/CPP deals with the technical aspects of how to conduct business electronically, the Specification Schema deals with the actual business process. It identifies such things as the overall business process, the roles, transactions, identification of the business documents used (the DTDs or schemas), document flow, legal aspects, security aspects, business level acknowledgments, and status. A Specification Schema can be used by a software application to configure the business details of conducting business electronically with another organization.

```
<BusinessTransaction name="Create Order">

  <RequestingBusinessActivity
   name=""

      isNonRepudiationRequired="true"
      timeToAcknowledgeReceipt="P2D"
      timeToAcknowledgeAcceptance="P3D">

    <DocumentFlow
      isSuccess="true"
      documentType="Purchase Order" />

 </RequestingBusinessActivity>
  <RespondingBusinessActivity
   name=""

     isNonRepudiationRequired="true"
     timeToAcknowledgeReceipt="P5D" >

   <DocumentFlow
    isSuccess="true"
    documentType="PO Acknowledgement" />

  </RespondingBusinessActivity>
</BusinessTransaction>
```

Figure 3-4 Sample Business Process Specification

3.4 Business Transactions

Business transactions can be defined as the actual transfer of documents between trading partners. There are exactly two roles: requestor and responder. The requestor may send a request; in that case responder will sends back the information. Or it can be one way notification. Choreography defines the sequence of business transactions. It is expressed in terms of states and the transition of business transactions.

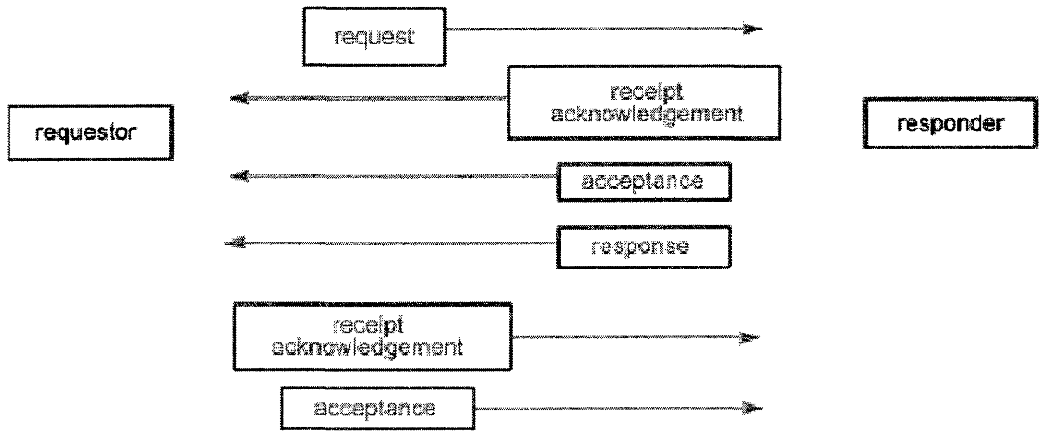


Figure 3-5 Business Transactions

3.5 Business Document Flows

Each Business Transaction consists of one or two predefined Business document flows.

Business documents are composed of business information objects, or smaller chunks of information that have previously been identified. These components are merely structures such as XML schema or DTD that define the presentation of information in the document. These business documents are not defined within the business-process specification; rather they are referenced. The process described in the ebXML core components specification is used to create these business documents. EDI, XML, or any other type of document can be directly used in the business transactions.

A Business Transaction may be additionally supported by one or more Business Signals. Each business transaction has one requesting (incoming) document and an optional responding (outgoing) document. BPSS also supports business signals, or application-level documents that signal a business transaction's current state—for example, an acknowledgment document.

3.6 Business Collaboration

The Business Process Specification Schema defines "business collaboration" as follows: Business collaboration is a choreographed set of business transaction activities in which two trading partners exchange documents. Two or more business partners participate in the business collaboration through roles. The roles interact with each other through Business Transactions.

A Business Collaboration consists of choreographed Business Transactions. That choreography is expressed in terms of states and the transitions between them. In fact, a Business Activity is known as an abstract state, with Business Collaborations and Business Transaction Activities known as concrete states. Auxiliary states include start, fork, synchronization, and completion (which take the form of either success or failure).

As the collaboration proceeds, it transitions from one state to the next. In some cases in which a particular requirement exists (such as document receipt or validation), a guard gates the transition to control whether or not it takes place

Business collaboration is a set of Business Transactions between business partners. Each partner plays one or more roles in the collaboration [6]. Two common variation of business collaborations are: Binary collaboration and multi-party collaboration. Binary collaboration consists of a number of transactions between two trading partners. Binary business collaboration binds a set of business transactions to two roles. For example, the process of issuing a purchase order requires the business partners to play a set of roles, such as customer and supplier. Each activity in a binary collaboration is either a business transaction or a nested binary collaboration. A business activity may consist of a business transaction or a collaboration activity that can define another binary collaboration. Thus, recursive composition of business collaborations is permitted within BPSS.

The sample XML source code explains the Binary collaboration. This code has one transaction activity “Create Order”. There are two authorized roles: buyer and Seller.

```
<BinaryCollaboration name="Company Order" timeToPerform="P5D">
   <Documentation>
      timeToPerform = Period: 5 days from start of transaction
   </Documentation>
      <InitiatingRole name="buyer"/>
      <RespondingRole name="seller"/>
   <BusinessTransactionActivity name="Create Order"
      businessTransaction="Create Order"
      fromAuthorizedRole="buyer"
      toAuthorizedRole="seller"/>
</BinaryCollaboration>
```

Figure 3-6 Sample Binary Collaboration

A multi-party collaboration describes collaboration between multiple companies. A multiparty collaboration is constructed by combining multiple binary collaborations, in other words same information flowing between two parties at all time.

The difference between business processes and business collaborations is that business processes describe the activities from the point of the view of only one company.

A business process consists of a set of steps that need to be executed in specified order. The choreography of the steps to carry out a business process is expressed as a state machine in BPSS. A state is represented by a business transaction. When a business process transition into a state, an action corresponding to the business-transactions specification has to be carried out. For example: while in a check credit business transaction a credit request document is delivered to a responding business activity.

```
<BusinessTransaction name="Create Order">
 <RequestingBusinessActivity
   name=""
   isNonRepudiationRequired="true"
   timeToAcknowledgeReceipt="P2D"
   timeToAcknowledgeAcceptance="P3D"
  >
  <DocumentFlow
   isSuccess="true"
   documentType="Purchase Order"
  />
 </RequestingBusinessActivity>
 <RespondingBusinessActivity
  name=""
  isNonRepudiationRequired="true"
  timeToAcknowledgeReceipt="P5D"
 >
  <DocumentFlow
   isSuccess="true"
   documentType="PO Acknowledgement"
  />
 </RespondingBusinessActivity>
</BusinessTransaction>
```

Figure 3-7 Sample Business Transaction with two document flows and three business signals

A business transaction is atomic in the sense that there is no partially successful execution: it either succeeds or fails completely. If failed its effects must be reversed. A set of such states (business transactions) and the transition rules from the states are described in business collaboration [5].

The above business transaction requests acknowledgement receipt in the period of 2 days (P2D) and acknowledgement acceptance in the period of 3 days (P3D). The receiving party must make sure that a requesting document is not garbled [isNonRepudiationRequired="true"] before sending acknowledgement of receipt.

3.7 Collaboration Protocol Profile

The Collaboration Protocol Profile (CPP) is a formal description file that lists capabilities of company in terms of ebXML operations, which will engage in electronic business with other companies. It will identify capabilities such as transport protocol (e.g. http, SMTP), delivery channel, security constraints (e.g. digital signature, certificates) and bindings to a business process that an organization supports. Depending on the level of security desired, this document can be digitally signed. CPP is stored in an ebXML registry with a unique id called globally unique Identifier (GUID) so that it can be queried by other potential trading partners. Then company calculates the intersection of the retrieved CPP with its own CPP that results in a CPA.

CPP includes:

- Process-specification layer
- Delivery-channel specification layer
- Document Exchange layer
- Transport layer.

The process-specification layer - It is the heart of business agreement between companies. It provides the information about a party, the services (business transactions) and transition rules that defines the order of communication. It also defines the role of the party is going to play in the business process.

The delivery-channel layer - It provides the specification of the messaging channels available for message delivery so that the services described in the business-process specification can be invoked. The delivery channel layer also defines whether an acknowledgement is required for the message delivered. It also defines the security characteristics. Many delivery channels can be defined in same CPP.

Document-exchange layer – It defines the way the business documents will be processed. The document exchange represents the messaging protocol, such as ebMS. It also defines security and reliability properties for messages that include encryption, digital signature and reliable messaging specifications.

Transport layer – It defines the details of underlying transport protocol used for sending messages. It contains end point addresses in the network also various other properties of the protocol.

3.7.1 Structure of CPP

The root element of CPP is the Collaboration Protocol Profile element that contains one or more party info elements, Packaging and Simple part. Also have optional elements: Signature and comment elements. The contents of CPP can be protected using an optional “signature” element.

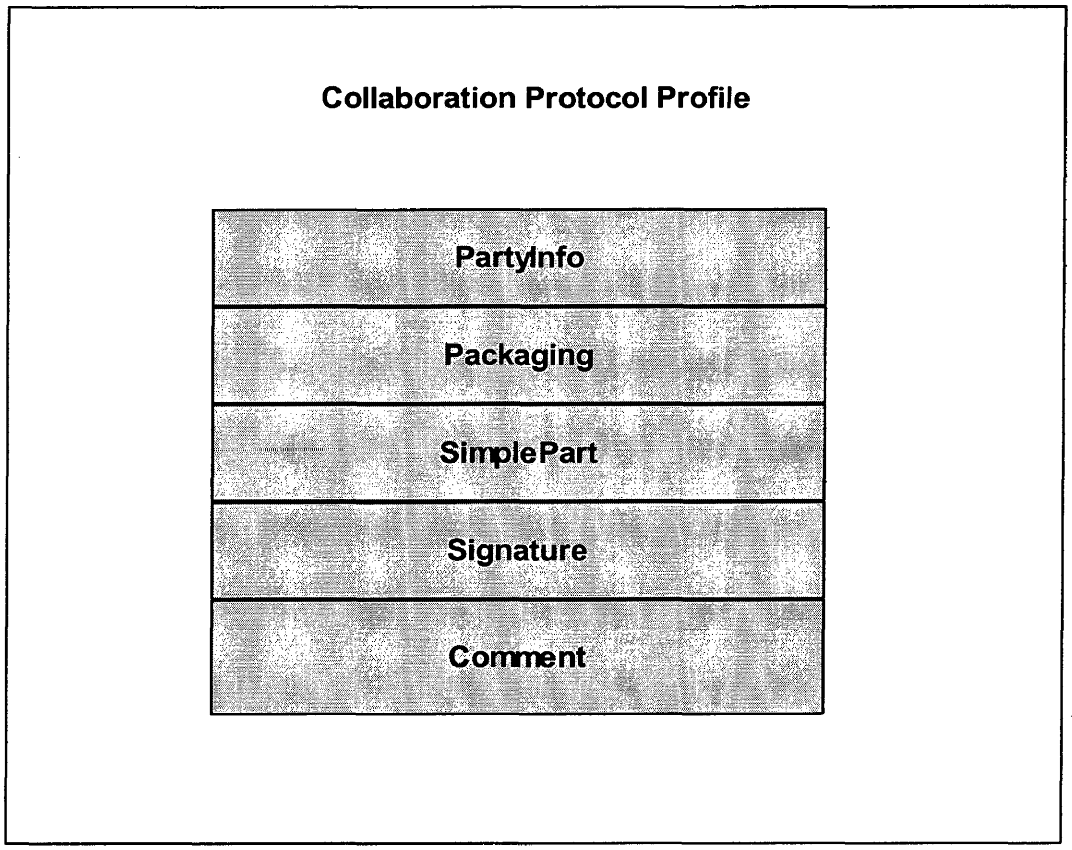


Figure 3-8 Structure of Collaboration Protocol Profile (source: Implementing B2b E-commerce with ebXML)

- *PartyInfo* element identifies the business and technical capabilities described in the CPP. CPP can contain one or more *PartyInfo*.

  The *PartyInfo* element includes:

- One or more *PartyId* elements. These elements provide a logical identifier for the organization, such as a DUNS number.
- One *PartyRef* element. This element points to an external resource with more information about the organization.
- One or more *CollaborationRole* elements. These elements are the heart of the CPP, providing information on the Business Processes in which the party engages, and the roles it plays within those processes. The *CollaborationRole* element directly references a Business Process Specification stored in the registry.
- One or more *Certificate* elements. These elements identify the party's security certificates.
- One or more *DeliveryChannel* elements. These elements define the ways in which the party can receive messages, including references to both a document exchange, or message protocol, and a transport protocol layer described below.
- One or more *Transport* elements. These elements provide specifics for the transport layers referenced in the DeliveryChannel elements. Transport layers may include HTTP, SMTP, or other transport protocols.
- One or more *DocExchange* elements. These elements provide specifics for the document exchanges referenced in the *DeliveryChannel* elements. The document exchange represents the messaging protocol, such as ebMS.

Each of these elements has its own child elements.

- The Packaging element provides information about the way in which messages are actually constructed. Messages are processed as SOAP Messages with Attachments and the *Packaging* element provides information on how these messages are organized. The payload may consist of multiple documents with different content structure such as XML, text or binary. CPP can contain one or more *Packaging* element.

  The *Packaging* element has three potential child elements:

  - The *ProcessingCapabilities* element is an empty element with two required attributes, generate and parse, which indicate whether the system is capable of creating or reading messages.
  - The *SimplePart* element defines message pieces that consist of a certain Multipurpose Internet Mail Extensions (MIME) type. Pieces are identified so that they can be referenced within the CompositeList element. *SimplePart* describes the composition of a single document, which may be referred from multiple locations in *Packaging*. CPP can contain one or more *SimplePart*.
  - The *CompositeList* element provides information about composites or encapsulations of *SimpleParts*. This element is optional, and will not appear if parts are sent individually.
- *Signature* element provides the security for the contents of CPP by incorporating the digital signature feature. It is an optional element.

- CPP can have zero or more *comment* element.

```
<CollaborationProtocolProfile

     xmlns="http://www.ebxml.org/namespaces/tradePartner"
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.1">

    <PartyInfo>
          ...
          <!--REQUIRED, Repeatable-->
          ...
    </PartyInfo>

    <Packaging id="ID">
          ...
          <!--REQUIRED-->
          ...
    <Packaging>

    <ds:Signature>
          ...
          <!--OPTIONAL-->
          ...
     </ds:Signature>

    <Comment>
          ...
          <!-- OPTIONAL -->
          ...
     </Comment>

</CollaborationProtocolProfile>
```

Figure 3-9 Sample structure of CPP

The above sample structure of a CPP consists of a root element: CollaborationProtocolProfile that requires three namespace declarations:

http://www.ebxml.org/namespaces/tradePartner is the default namespace. http://www.w3.org/2000/09/xmldsig#, is the namespace for XML Digital Signature, and is included to allow signing of CPPs. http://www.w3.org/1999/xlink is the XLink namespace, which allows the CPP to reference external information. Other elements present in the structure are: PartyInfo, Packaging, Signature, and Comment elements.

```
<?xml version="1.0" encoding="UTF-8" ?>
<tp:CollaborationProtocolProfile
xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd"
tp:version="1.1">
<tp:PartyInfo>
<tp:PartyId tp:type="DUNS">123456789</tp:PartyId>
<tp:PartyRef tp:href="http://example.com/about.html" />
<tp:CollaborationRole tp:id="N00">
<tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
xlink:type="simple"
xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
<tp:Role tp:name="buyer" xlink:type="simple"
xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>
<tp:CertificateRef tp:certId="N03" />
<tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402">
<tp:Service tp:type="uriReference"
>uri:example.com/services/buyerService</tp:Service>
<tp:Override tp:action="orderConfirm" tp:channelId="N07"
tp:packageId="N0402" xlink:type="simple"
xlink:href="http://ebxml.org/processes/buySell.xml#orderConfirm"/>
</tp:ServiceBinding>
</tp:CollaborationRole>
```

```
<tp:Certificate tp:certId="N03">
<ds:KeyInfo />
</tp:Certificate>
<tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
tp:docExchangeId="N06">
<tp:Characteristics tp:syncReplyMode="none"
tp:nonrepudiationOfOrigin="true"
tp:nonrepudiationOfReceipt="false"
tp:secureTransport="true" tp:confidentiality="true"
tp:authenticated="true" tp:authorized="false" />
</tp:DeliveryChannel>
<tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
tp:docExchangeId="N06">
<tp:Characteristics tp:syncReplyMode="none"
tp:nonrepudiationOfOrigin="true" tp:confidentiality="true"
tp:nonrepudiationOfReceipt="false"
tp:secureTransport="false" tp:authenticated="true"
tp:authorized="false" />
</tp:DeliveryChannel>
<tp:Transport tp:transportId="N05">
<tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
<tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
<tp:Endpoint tp:uri="https://www.example.com/servlets/ebxmlhandler"
tp:type="allPurpose" />
<tp:TransportSecurity>
<tp:Protocol tp:version="3.0">SSL</tp:Protocol>
<tp:CertificateRef tp:certId="N03" />
</tp:TransportSecurity>
</tp:Transport>
```

```
<tp:Transport tp:transportId="N08">
<tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
<tp:ReceivingProtocol tp:version="1.1">SMTP</tp:ReceivingProtocol>
<tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
tp:type="allPurpose" />
</tp:Transport>
<tp:DocExchange tp:docExchangeId="N06">
<tp:ebXMLBinding tp:version="0.98b">
<tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
tp:idempotency="true"
tp:messageOrderSemantics="Guaranteed">
<tp:Retries>5</tp:Retries>
<tp:RetryInterval>30</tp:RetryInterval>
<tp:PersistDuration>P1D</tp:PersistDuration>
</tp:ReliableMessaging>
<tp:NonRepudiation>
<tp:Protocol
>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
<tp:HashFunction
>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
<tp:SignatureAlgorithm
>http://www.w3.org/2000/09/xmldsig#dsa-sha1</tp:SignatureAlgorithm>
<tp:CertificateRef tp:certId="N03" />
</tp:NonRepudiation>
<tp:DigitalEnvelope>
<tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
<tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
<tp:CertificateRef tp:certId="N03" />
</tp:DigitalEnvelope>
</tp:ebXMLBinding>
</tp:DocExchange>
</tp:PartyInfo>
```

```
<tp:Packaging tp:id="N0402">
<tp:ProcessingCapabilities tp:parse="true" tp:generate="true" />
<tp:SimplePart tp:id="N40" tp:mimetype="text/xml">
<tp:NamespaceSupported
tp:location=
"http://ebxml.org/project_teams/transport/messageService.xsd"
tp:version="0.98b"
>http://www.ebxml.org/namespaces/messageService</tp:NamespaceSupported>
<tp:NamespaceSupported tp:location=
"http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"
tp:version="1.0"
>http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>
</tp:SimplePart>
<tp:SimplePart tp:id="N41" tp:mimetype="text/xml">
<tp:NamespaceSupported tp:version="1.0"
tp:location="http://ebxml.org/processes/buysell.xsd"
>http://ebxml.org/processes/buysell.xsd</tp:NamespaceSupported>
</tp:SimplePart>
<tp:CompositeList>
<tp:Composite tp:id="N42" tp:mimetype="multipart/related"
tp:mimeparameters="type=text/xml;">
<tp:Constituent tp:idref="N40" />
<tp:Constituent tp:idref="N41" />
</tp:Composite>
</tp:CompositeList>
</tp:Packaging>
<tp:Comment tp:xml_lang="en-us">buy/sell agreement between example.com and
contrived-example.com</tp:Comment>
</tp:CollaborationProtocolProfile>
```

Figure 3-10 Sample CPP

3.8 Collaboration-Protocol Agreement (CPA)

A Collaboration-Protocol Agreement (CPA) describes the capabilities that two Partners agree to use in doing electronic business with each other. In other words, CPA is a trading partner agreement. CPA is used to configure the systems of both trading partners. CPA can also be added to the registry for reference. CPA is essentially XML documents that encode parties' e-business agreements. It is simply an intersection of matching capabilities of two CPPs, usually it is followed by a series of negotiations among the parties. Once an agreement was made, each party will take an electronic copy of the same CPA and configure their systems.

3.8.1 Overall structure of a CPA

The structure of the CPA is similar to that of the CPP.

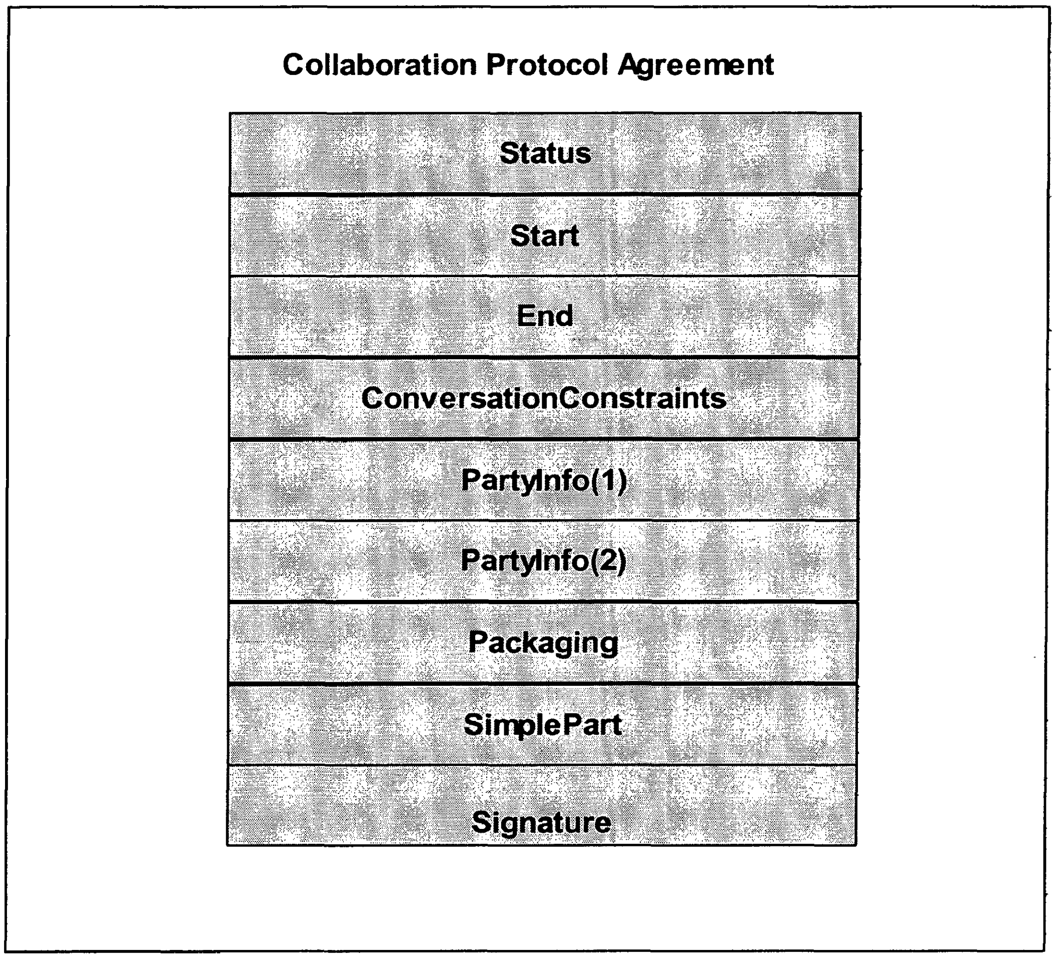


Figure 3-11 Structure of CPA

CPA defines namespaces on its root element: CollaborationProtocolAgreement element and a version to distinguish any subsequent changes. The CPA also includes a cpaid attribute that both parties use. CPA contains almost all the elements and attributes of CPP. In addition, the CPA has some additional elements and attributes.

```
<CollaborationProtocolAgreement
   xmlns="http://www.ebxml.org/namespaces/tradePartner"
   xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
   xmlns:xlink = "http://www.w3.org/1999/xlink"
   cpaid="http://www.example.com/cpas/clipCPA"
   version="1.7">
<Status value = "proposed"/>
<Start> 1988-04-07T18:39:09 </Start>
<End> 1990-04-07T18:40:00 </End>
<ConversationConstraints invocationLimit = "250"
concurrentConversations = "5"/>
<PartyInfo>
...
<!--REQUIRED, repeatable-->
...
</PartyInfo>
<PartyInfo>
...
<!--REQUIRED, repeatable-->
...
</PartyInfo>
<Packaging id="N20">
...
<!--REQUIRED, repeatable-->
...
</Packaging>
<ds:Signature> <!--OPTIONAL--> </ds:Signature>
<Comment xml:lang="en-gb"> <!--OPTIONAL--></Comment>
</CollaborationProtocolAgreement>
```

Figure 3-12 Sample structure of CPA

PartyInfo – CPA always between two parties, which are mentioned in two PartyInfo elements. The elements and value reflect the agreement between the each party.

Status - The Status element records the state of the composition/negotiation process that creates the CPA. Typically, one party generates a CPA and offers it to the other party for approval, so the Status element shows where the document is in this process. The possible values are proposed, agreed, and signed.

The lifetime of the CPA is given by the Start and End elements. The Start element specifies the starting date and time of the CPA. The End element specifies the ending date and time of the CPA. The Start and End elements represent, in Coordinated Universal Time, the beginning and end of the period during which this CPA is active.

Finally, the optional CoversationConstraints element defines the finite number of conversations that may be held under this CPA, and the number that may be held concurrently. In other words, ConversationConstraints element places limits on the number of conversations under the CPA.

The Packaging, Signature, and Comment elements have the same meaning they have for a CPP. CPA can be digitally signed to ensure the integrity of these documents, which imply that once signed these documents, must not be changed during processing or while being stored in ebXML registry.

```
<?xml version="1.0" ?>
<tp:CollaborationProtocolAgreement
xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
tp:cpaid="http://www.example.com/cpas/clipCPA"
tp:version="1.2">
<tp:Status tp:value="proposed" />
<tp:Start>2001-05-20T07:21:00Z</tp:Start>
<tp:End>2002-05-20T07:21:00Z</tp:End>
<tp:ConversationConstraints tp:invocationLimit="100"
tp:concurrentConversations="100"/>
<tp:PartyInfo>
<tp:PartyId tp:type="DUNS">123456789</tp:PartyId>
<tp:PartyRef xlink:href="http://example.com/about.html"/>
<tp:CollaborationRole tp:id="N00">
<tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
xlink:type="simple"
xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
<tp:Role tp:name="buyer" xlink:type="simple"
xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>
<tp:CertificateRef tp:certId="N03" />
<tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402">
<tp:Service tp:type="uriReference"
>uri:example.com/services/buyerService</tp:Service>
<tp:Override tp:action="orderConfirm" tp:channelId="N08"
tp:packageId="N0402" xlink:type="simple" xlink:href=
"http://ebxml.org/processes/buySell.xml#orderConfirm"/>
</tp:ServiceBinding>
</tp:CollaborationRole>
<tp:Certificate tp:certId="N03">
<ds:KeyInfo />
</tp:Certificate>
```

```
<tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
tp:docExchangeId="N06">
<tp:Characteristics tp:syncReplyMode="none"
tp:nonrepudiationOfOrigin="true"
tp:nonrepudiationOfReceipt="false" tp:secureTransport="true"
tp:confidentiality="true" tp:authenticated="true"
tp:authorized="false" />
</tp:DeliveryChannel>
<tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
tp:docExchangeId="N06">
<tp:Characteristics tp:syncReplyMode="none"
tp:nonrepudiationOfOrigin="true" tp:secureTransport="false"
tp:nonrepudiationOfReceipt="false" tp:confidentiality="true"
tp:authenticated="true" tp:authorized="false" />
</tp:DeliveryChannel>
<tp:Transport tp:transportId="N05">
<tp:SendingProtocol
tp:version="1.1">HTTP</tp:SendingProtocol>
<tp:ReceivingProtocol
tp:version="1.1">HTTP</tp:ReceivingProtocol>
<tp:Endpoint tp:type="allPurpose"
tp:uri="https://www.example.com/servlets/ebxmlhandler"/>
<tp:TransportSecurity>
<tp:Protocol tp:version="3.0">SSL</tp:Protocol>
<tp:CertificateRef tp:certId="N03" />
</tp:TransportSecurity>
</tp:Transport>
<tp:Transport tp:transportId="N18">
<tp:SendingProtocol
tp:version="1.1">HTTP</tp:SendingProtocol>
<tp:ReceivingProtocol
tp:version="1.1">SMTP</tp:ReceivingProtocol>
<tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
tp:type="allPurpose" />
</tp:Transport>
```

```
<tp:DocExchange tp:docExchangeId="N06">
<tp:ebXMLBinding tp:version="0.98b">
<tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
tp:idempotency="true"
tp:messageOrderSemantics="Guaranteed">
<tp:Retries>5</tp:Retries>
<tp:RetryInterval>30</tp:RetryInterval>
<tp:PersistDuration>P1D</tp:PersistDuration>
</tp:ReliableMessaging>
<tp:NonRepudiation>
<tp:Protocol
>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
<tp:HashFunction
>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
<tp:SignatureAlgorithm
>http://www.w3.org/2000/09/xmldsig#dsa-sha1</tp:SignatureAlgorithm>
<tp:CertificateRef tp:certId="N03" />
</tp:NonRepudiation>
<tp:DigitalEnvelope>
<tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
<tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
<tp:CertificateRef tp:certId="N03" />
</tp:DigitalEnvelope>
</tp:ebXMLBinding>
</tp:DocExchange>
</tp:PartyInfo>
<tp:PartyInfo>
<tp:PartyId tp:type="DUNS">987654321</tp:PartyId>
<tp:PartyRef xlink:type="simple"
xlink:href="http://contrived-example.com/about.html" />
<tp:CollaborationRole tp:id="N30">
<tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
xlink:type="simple"
xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
<tp:Role tp:name="seller" xlink:type="simple"
xlink:href="http://ebxml.org/processes/buySell.xml#seller" />
<tp:CertificateRef tp:certId="N33" />
<tp:ServiceBinding tp:channelId="N34" tp:packageId="N0402">
<tp:Service tp:type="uriReference">uri:example.com/services/sellerService</tp:Service>
</tp:ServiceBinding>
</tp:CollaborationRole>
```

```
<tp:Certificate tp:certId="N33">
<ds:KeyInfo />
</tp:Certificate>
<tp:DeliveryChannel tp:channelId="N34" tp:transportId="N35"
tp:docExchangeId="N36">
<tp:Characteristics tp:nonrepudiationOfOrigin="true"
tp:nonrepudiationOfReceipt="false"
tp:secureTransport="true" tp:confidentiality="true"
tp:authenticated="true"
tp:authorized="false"/>
</tp:DeliveryChannel>
<tp:Transport tp:transportId="N35">
<tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
<tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
<tp:Endpoint
tp:uri="https://www.contrived-example.com/servlets/ebxmlhandler"
tp:type="allPurpose" />
<tp:TransportSecurity>
<tp:Protocol tp:version="3.0">SSL</tp:Protocol>
<tp:CertificateRef tp:certId="N33" />
</tp:TransportSecurity>
</tp:Transport>
<tp:DocExchange tp:docExchangeId="N36">
<tp:ebXMLBinding tp:version="0.98b">
<tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
tp:idempotency="true"
tp:messageOrderSemantics="Guaranteed">
<tp:Retries>5</tp:Retries>
<tp:RetryInterval>30</tp:RetryInterval>
<tp:PersistDuration>P1D</tp:PersistDuration>
</tp:ReliableMessaging>
```

```
<tp:NonRepudiation>
<tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
<tp:HashFunction
>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
<tp:SignatureAlgorithm
>http://www.w3.org/2000/09/xmldsig#dsa-sha1</tp:SignatureAlgorithm>
<tp:CertificateRef tp:certId="N33" />
</tp:NonRepudiation>
<tp:DigitalEnvelope>
<tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
<tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
<tp:CertificateRef tp:certId="N33" />
</tp:DigitalEnvelope>
</tp:ebXMLBinding>
</tp:DocExchange>
</tp:PartyInfo>
<tp:Packaging tp:id="N0402">
<tp:ProcessingCapabilities tp:parse="true" tp:generate="true" />
<tp:SimplePart tp:id="N40" tp:mimetype="text/xml">
<tp:NamespaceSupported
tp:location=
"http://ebxml.org/project_teams/transport/messageService.xsd"
tp:version="0.98b"
>http://www.ebxml.org/namespaces/messageService</tp:NamespaceSupported>
<tp:NamespaceSupported tp:version="1.0"
tp:location=
"http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"
>http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>
</tp:SimplePart>
```

```
<tp:Simplepart tp:id="N41" tp:mimetype="text/xml">
<tp:NamespaceSupported tp:version="1.0"
tp:location="http://ebxml.org/processes/buysell.xsd"
>http://ebxml.org/processes/buysell.xsd</tp:NamespaceSupported>
</tp:SimplePart>
<tp:CompositeList>
<tp:Composite tp:id="N42" tp:mimetype="multipart/related"
tp:mimeparameters="type=text/xml;">
<tp:Constituent tp:idref="N40" />
<tp:Constituent tp:idref="N41" />
</tp:Composite>
</tp:CompositeList>
</tp:Packaging>
<tp:Comment xml:lang="en-us">buy/sell agreement between example.com and
contrived-example.com</tp:Comment>
</tp:CollaborationProtocolAgreement>
```

Figure 3-13 Sample CPA

3.9 Overview of Simple Object Access Protocol (SOAP)

The ebXML Message Service is defined as a set of layered extensions to the Simple Object Access Protocol (SOAP) and SOAP Messages with Attachments – which is itself an extension of SOAP. ebXML infrastructure can be used for the secure, reliable exchange of information. It is independent of transaction vocabulary, encoding and the choice of vendor solution.

Simple Object Access Protocol (SOAP) is a lightweight specification protocol used to access methods on servers, components and objects in a platform independent manner so as to exchange information in a decentralized, distributed environment. It is a protocol that acts as the glue between heterogeneous software components by facilitating interoperability. In other words, SOAP is a lightweight mechanism for exchanging XML over the World Wide Web.

SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC. The advantages of SOAP: simplicity, extensibility and its ability to pass through firewalls.

According to the specification, the SOAP 1.1 protocol consists of three parts:

- The SOAP envelope describes an overall structure for expressing the content; processing party and whether it is optional or mandatory.
- The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.
- Soap's Remote Protocol Call defines a convention, which can be used to represent remote procedure calls and their responses.

The XML part of every SOAP message contains particular tags and attributes. It consists of:

- The SOAP envelope: this is the first element in the XML document representing the message. It identifies the XML as being a SOAP message and must be the root element of the message
- The (optional) SOAP header: this is a generic mechanism that adds characteristics to the SOAP message. SOAP defines several attributes

that can be used to indicate who must process the characteristics, and whether this process is optional or mandatory.

- The SOAP body: this contains message payload for the mandatory information being sent to the message endpoint.

3.9.1 Structure of SOAP

SOAP message structure consists of HTTP header, the SOAP envelope, SOAP header and the SOAP body.

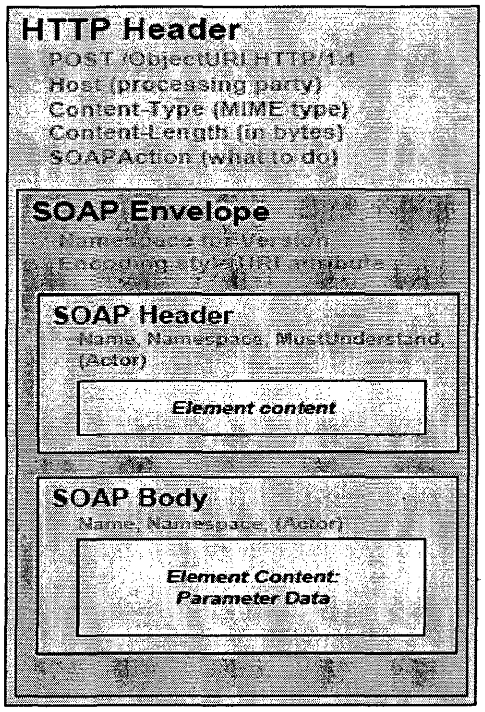


Figure 3-14 Basic Structure of SOAP (Source: techmetrix.com)

3.9.2 HTTP Header

The HTTP header is at the beginning of the message and used by the HTTP server software to handle the SOAP message. Post method is for sending the request via the network.

In the first line, the post method, URI that is the target request and protocol version are defined.

```
POST /Computer HTTP/1.1
```

Figure 3-15 HTTP header SOAP Post method

The next line gives the target host:

```
Host: www.example.com
```

Figure 3-16 HTTP header target host

The next three lines are used to define the MIME format for message display, the HTTP coding and the length of the message.

```
Content-Type: text/xml;
charset="utf-8"
Content-Length: 10
```

Figure 3-17 HTTP header MIME format

Then, methods are added such as SOAPAction (also known as SOAPMethodName) which determines the intention of the HTTP request. The identifier following the # sign must match the name of the first tag in the SOAP message body.

```
SOAPAction="http://www.example.com#EventManager"
```

Figure 3-18 HTTP header SOAP Action

3.9.3 SOAP Envelope

It is mandatory and the first part of the SOAP message. It contains the name of the element (Envelope), followed by a namespace defining the SOAP version being used, and the optional encodingStyle attribute which points to a link where the serialization (tree structure) and encoding rules are defined. The SOAP envelope consists of an optional SOAP header and the SOAP body. The envelope is presented as follows:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
http://schemas.xml.org/soap/envelope/" SOAP-ENV
:encodingStyle="http://schemas.xml.org/soap/
encoding/"/> ... </SOAP:Envelope>
```

Figure 3-19 SOAP Envelope

Namespaces are used to provide a context and guarantee the uniqueness of elements associated in this way.

3.9.4 SOAP Header

This is an optional part of the SOAP message encapsulated in the SOAP envelope. It carries information to intermediaries, and is made up of one of more entries. These bear a local name, a full name, a namespace and the two actor attributes which designate the endpoint of the entry, and mustUnderstand, which indicates the optional nature of the process. A SOAP application must include a correct SOAP namespace for all the elements and attributes defined in the message generated. This is a URI which points to a description of the message information in order to guarantee the uniqueness of the message. DTDs are never used. Faults occur when processing a message, and they may be caused by an unrecognized header field, a message that cannot be authenticated, or errors that occur when invoking a method to process a message.

```
<SOAP-ENV:Header>
<t:new Eventxmlns:t="http://w w w .techmetrix.com/
eventmanager"
SOAP-ENV :actor="http://schemas.xml.org/soap /actor/next/" SOAP-ENV :mustUnderstand="1">
Christmas Event
</t:new Event>
</SOAP-ENV:Header>
```

Figure 3-20 SOAP Header

3.9.5 SOAP Body

The information to be processed by the endpoint is found in the body of the SOAP message. This can contain a set of entries which are all kept in the root of the message body.

```
<SOAP-ENV:Body>
<m:NewCustomer xmlns:m="Some-URI">
<Name>Dumser</Name>
<Surname>Johann</Surname>
<City>Cambridge</City>
<ZipCode>01800</ZipCode>
<State>MA</State>
<Country>USA</Country>
</m:NewCustomer>
</SOAP-ENV:Body>
```

Figure 3-21 SOAP Body

The Body element contains the message payload. In the case of a request message the payload of the message is processed by the receiver of the message and is typically a request to perform some service and, optionally, to return some results. In the case of a response message the payload is typically the results of some previous request or a fault.

3.9.6 Encoding Rules

SOAP can be seen as the sum of HTTP and XML. A SOAP message is simply the HTTP request or response in which the payload data are in XML format. SOAP defines a serialization mechanism for the body of a SOAP message. The resulting XML schema represents the structure of the object data to be passed. The encoding rules describe a standard method to do this, including using “Element Normal Form” where all values are represented as elements. SOAP is called payload-neutral which means it does not impose any limitations on or make any assumption about the contents [7].

3.9.7 Example SOAP request

Below is an example of the SOAP message request code:

```
POST /EventManager HTTP/1.1
Host: www.example.com
Content-Type: text/xml;
charset="utf-8"

Content-Length: 60 SOAPAction="http://www.example.com/Event
#New Customer"

<SOAP-ENV:Envelope xmlns:SOAP-ENV=" http://schemas.xml.org/soap/envelope/"
SOAP-ENV :encodingStyle="http://schemas.xml.org/soap/
encoding/"/> <SOAP-ENV:Header>
<t:Name
xmlns:t="www.example.com/EventManager"
SOAP-ENV :actor="http://schemas.xml.org/soap/actor /next/" SOAP-ENV
:mustUnderstand="1"> Dumser
</t:Name >
</SOAP-ENV:Header>
<SOAP:Body> <m:NewCustomer xmlns:m="www.example.com/Event">
<Entreprise>SQLI</Entreprise>
<Address>Paris</Address> </m:NewCustomer> </SOAP:Body>
</SOAP:Envelope>
```

Figure 3-22 SOAP message request code

3.9.8 SOAP transport architecture

The SOAP transport architecture and mechanism is based on a Web Service application, but it is also applicable for data and document interchange.

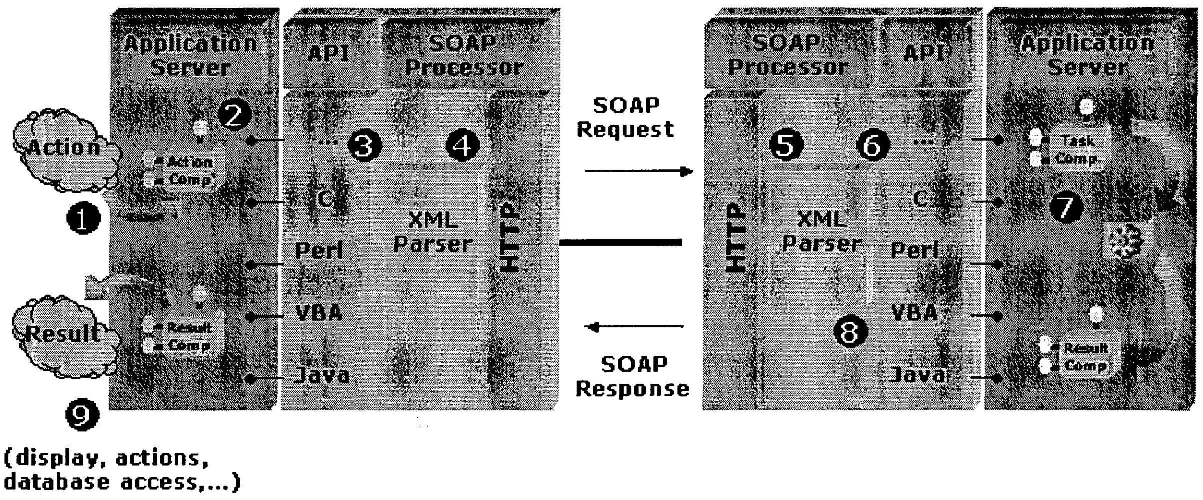


Figure 3-23 SOAP transport Architecture (source: techmetrix.com)

1) Station1 executes a command which creates an action on the associated application server. For example, suppose a customer wants to use a purchase order Web service that is exposed on the application server of another company. It issues a command which is passed to the API that provides access to the SOAP processor. API serializes the call using a schema provided by the other party and sends the result as an XML document to the XML parser in the SOAP processor. After the document is checked to see that is well formed, the order document is packaged as a SOAP request and sent over HTTP to the other party.

2) This command generates a process within the application and the result arrives in the application interface. After smoothly passing any firewalls between the two companies, the request arrives at the receiving party's SOAP processor.

3) The message is translated into XML format by the implementation and is then sent to the Web server.

4) The XML parser checks the coherence of the XML document and sends the SOAP message via HTTP. The message is parsed and checked to ensure it is well formed and valid. The SOAP application does additional verifications, including identifying the message parts that are addressed to it.

5) The XML parser checks the validity of the message using the HTTP and XML headers, and accepts or rejects it.

6) The message is then routed to the relevant application server and translated by the implementation so that the task is meaningful for the application. The SOAP application which receives the SOAP message must proceed as follows to translate the message:

- Identify the parts of the SOAP message which correspond to the application
- Check that all the mandatory parts of are supported by the application or discard the message. The application must check if all mandatory parts with mustUnderstand = "1" are supported, or it must respond with a fault message.
- Remove all the parts before transferring he message if the application is not the endpoint

7) The application then executes the task. A result is produced.

8) The return is done in the same way: implementation and then sending by HTTP.

9) The result of the action may be different: display in a browser, actions, access to a database, and so on.

3.10 Overview of ebMS (ebXML Messaging Service)

EbXML messaging service specification defines the set of services and protocols that enables electronic business applications to exchange data. The specification allows any application-level protocol to be used. The ebMS uses existing technology as much as possible. It specifies SOAP 1.1 for message structure, SOAP with attachments and multipurpose internet mail extensions (MIME) specifications for packaging and XML signature for the digital signature. MIME is an extension of the original Internet e-mail standard that allows users to exchange text, audio or visual files.

The ebXML message is built using extensions to SOAP 1.1 header and body. In the case of ebXML, the message header and body exist to deliver the attachments. These attachments are called payloads.

Packaging the payloads, which can be any type of data (for example text, binary, EDI or XML) with the SOAP header and body requires the use of MIME for packaging.

The ebMS specifies binding to HTTP, SMTP, FTP protocols for carrying the message. The message communication can be synchronous and asynchronous, and is defined independently of the underlying transport protocols. In a synchronous message communication, the requesting (sending) party waits for the response and/or acknowledgement before making another request to the same party. In asynchronous messaging, there are no restrictions.

While the ebXML Message Service was designed to work within the overall context of the ebXML initiative, due to the modular nature of the ebXML Technical Architecture, the ebXML Message Service can be used independently of ebXML as a whole. Software vendors can easily integrate ebXML Message Service functionality into their existing enterprise solutions.

A complete message, referred to as the Message Package, is a MIME multipart/related object. MIME types are used throughout to describe all of the contents of the Message Package. The Message Package contains two principal parts: a SOAP Message container and zero or more payload containers. The SOAP Message contains the ebXML SOAP extension elements routing information, trading partner information, message identification, and delivery semantics information. The payload is optional, and can contain any type of information that is to be exchanged between parties.

The ebXML Message Service introduces a manifest along with each message. The manifest contains references to each of the payload objects along with schema location and version information about the payload. This versioning of inner and outer layers permits the ebXML Message Service to be truly payload neutral. Because the versions are separated, an older version of the ebXML Message Service software can still route messages with newer version numbers without having to upgrade the ebXML Message Service software. Conversely, when the ebXML Message Service is versioned, the payload objects that it carries are not affected. Other standards that have not separated the payload from the message envelope and headers suffer from potential version problems. This limits the flexibility of these standards in a global context where uniform versioning is highly unrealistic.

To guarantee reliable message delivery, positive acknowledgement and persistent storage is required. Prior to sending a message, the sending ebXML Message Service will save the message in persistent storage. Once the message has been correctly received, the receiving ebXML Message Service will save the message in persistent storage and send an acknowledgement message to the sending ebXML Message Service. After the sending ebXML Message Service receives the acknowledgement, it may delete the message from persistent storage if no longer needed. If the sending ebXML Message Service does not receive acknowledgement, it can resend the message or notify the sending application that the message was unable to be delivered. The entire messaging operation is asynchronous, meaning that transmission of one message need not be completed before additional messages are sent.

If the sending ebXML Message Service does not receive acknowledgement, it will attempt a recovery sequence. The sending service will resend the message and again wait for acknowledgement. This cycle is repeated a number of times. The number of retries and retry interval can be defined as needed.

The ability to exchange any type of information, including XML, binary data, or EDI messages makes the ebXML Message Service highly flexible. This flexibility permits new businesses to use XML messages for documents while businesses with existing EDI infrastructure can continue to leverage their legacy systems.

3.10.1 EbXML message service handler

The ebXML Messaging Service (ebXML MS) is divided into three parts: abstract Message service interface, functions provided by the message service handler (MSH) and the mapping to underlying transport service interface.

- **Abstract Message service interface** – It connects the messaging service to ebXML application and from there to company's information system.
- **Transport service interface –** It works on the other end to connect MSH to the internet by means of mapping to underlying transport protocol such as HTTP, SMTP, FTP etc.
- **Functional modules** – these contains functions provided by the messaging service handler (MSH)

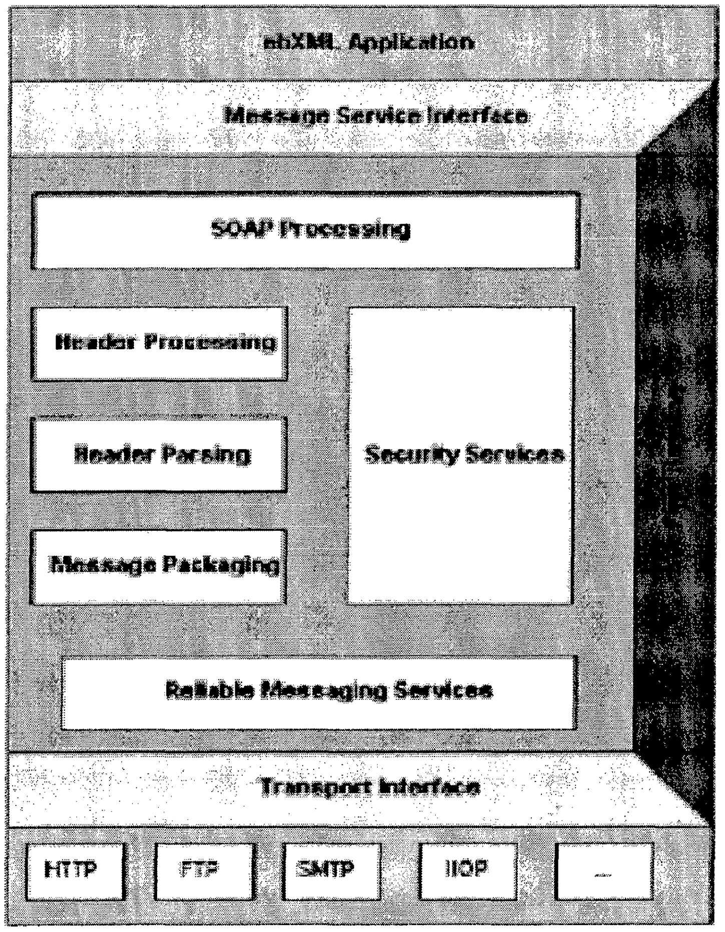


Figure 3-24 ebXML Message Service Handler Components

SOAP Processing – it is not really considered as part of the Message Service, as it provides the functionality described in the SOAP specification. Basically, it contains an XML Parser to handle incoming and outgoing messages according to SOAP rules.

Header Processing - deals with the outgoing messages by creating SOAP header elements for an ebXML message, based on the application input that is passed through the service interface and the rule prescribed by the CPA, and adds some extra, message-specific information such as digital signatures.

Header Parsing - extracts the incoming messages from the SOAP header to make it available to the other MSH modules.

Message Packaging – this component takes care of packaging the data in the ebXML message structure. The outermost envelope consists of headers of respective protocols such as HTTP, SMTP etc. The final ebXML message is structured as SOAP with Attachments envelope wrapped around the SOAP envelope itself and the ebXML payload.

Reliable Messaging Services – this component deals with all issues regarding reliable delivery, such as persistence, acknowledgement, retry and error notification.

Security Services - may create digital signatures, and control authentication and authorization when requested by other components.

Error Handling - responds to errors that may be reported by several other components. ebXML message consists of headers of respective protocols, for example HTTP headers.

3.10.2 EbXML Message Structure

The Packaging Specification deals with how the data has to be organized or packaged. An ėbXML Message Package has basically a Header Container and zero or more Payload Container. Header Container has ebXML and SOAP specific information whereas the Payload Container has the "real" data of the message. The ebXML specification defines a set of namespace-qualified SOAP header and Body element extensions within the SOAP envelope. As the figure explains, there are two MIME parts within the message package:

- Header Container - It contains XML document message complaint with SOAP 1.1
- Payload Container - It may contain application level payload.

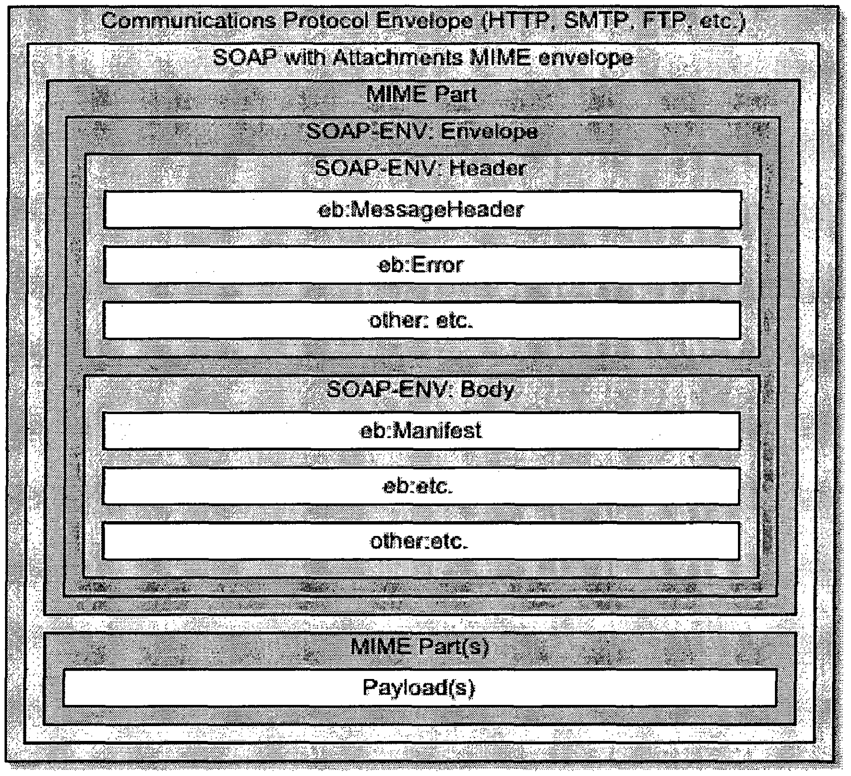


Figure 3-25 EbXML messaging service structure

3.10.3 Header Container

SOAP message is an XML document which has a root element called SOAP Envelope. The *SOAP:envelope* contains *SOAP:header* and *SOAP:body* elements. The *SOAP:header* contains the *eb:MessageHeader* and other elements. The eb namespace is defined as xmsns:eb=http://www.oasisopen.org/committees/ebxml-msg/schema/msg-header-2_0.xsd.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
      <SOAP-ENV:Header> ... </SOAP-ENV:Header>
      <SOAP-ENV:Body> ... </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3-26 A simple SOAP Envelope example

The *SOAP:body* contains the *eb:Manifest* element, which contains a meta description of payloads stored in *eb:reference* elements, where each reference element contains meta information corresponding to one payload available in the MIME message as a MIME body part.

The communication between MSHs (messaging service handler) happens under the context of an agreement between the MSHs. Every ebXML message has a required *CPAId* element in the *eb:MessageHeader*. This element has a unique identifier that points to the CPA. The value of *CPAId* must be agreed upon by the MSHs prior to starting any communications. The value of *CPAId* must be unique within the namespace mutually agreed upon by the two MSHs.

3.10.4 Payload container

Zero or more payload containers must have identified by the ebXML message manifest element within the SOAP body. Payloads can be in any data format. If the message package contains an application payload, then it should be enclosed in payload container. If application payload is not used, then payload container must not be present.

A simple Payload container example:

```
Content-ID: <domainame.example.com>
Content-type: application/xml

<invoice>
   <invoicedata>
   .....
   .....
   </invoicedata>
</invoice>
```

Figure 3-27 A simple Payload container example

3.10.5 ebXML SOAP Header

The message header includes routing information about who is sending the message, who the message is intended for, which CPA the message conforms to, the message ID, a conversation ID and so on. The header also includes the optional reliable messaging parameters, such as whether an acknowledgement is required, how many times a message needs to be retried in case a message is not received and so on. The header also contains optional digital signatures to ensure its integrity.

SOAP header contains: *MessageHeader, TraceHeaderList, ErrorList, Signature, Acknowledgment* and *Via*. The *MessageHeader* element contains routing information for the message. It has child elements: *From, To, CPAId, ConversationId, Service, Action, MessageData, QualityOfServiceInfo, SequenceNumber, and Description.*

Elements of SOAP Header are:

- MessageHeader

  The *MessageHeader* element goes into the SOAP Header element and has the following child elements: *From, To, CPAId, ConversationId, Service, Action, MessageData, QualityOfServiceInfo, SequenceNumber, and Description.* The *MessageHeader* element has a further two attributes: *mustUnderstand* and *Version.* This element plays an important role. The child elements of the MessageHeader will be described next.

  - *From* and *To*:

    This is a required element. In other words *From* element identifies the party that originated the message. *To* element identifies the party who is the recipient of the message.

    The *From* and *To* element contains:

    - *PartyID* - This element has the *PartyId* which can contain an internationally known identifier, like a DUNS number or a simple web site address. *From* and *To* elements can have multiple *PartyID* elements.
    - *Role* - Role element identifies the authorizing party when sending and receiving the message.

- *CPAId* - This is a required element. This element is also quite obvious. The element references the CPA of the two trading partners.
- *ConversationId* - This is a required element. This element represents a unique ID for the current conversation between two trading partners. The initiating MSH creates this ID and from that moment this ID will be used in the proceeding messages.
- *Service* - This is a required element. The Service element relates back to the Business Process Specification, where authorized roles within a business process are used. The service element names the activity of a party for this message. For example a service could be: “SupplierOrderProcessing”.
- *Action* - This is a required element. This element identifies a process within the Service. This could be a ``NewOrder" or Acknowledgment of reliable messaging purpose.
- *Messagedata* - This required element is a means of uniquely identifying an ebXML message. This element has further child elements: *MessageId, Timestamp, RefToMessageId, and TimeToLive.*
    - *MessageId* - It is a required element, an identifier for each Message to uniquely identify for further referencing.
    - *Timestamp* - It represents the time when the message header was created.

        - *RefToMessageId* - If the message is an Acknowledge Message (used for reliable messaging) this element references a previous message by its ID.
        - *TimeToLive* - This element is used for the functionality of reliable messaging and sets a time frame for the delivery of the message.
    - *QualityOfServiceInfo* - This element deals with reliable messaging. The *QualityOfServiceInfo* element has three child elements: *deliverySemantics, messageOrderSemantics and deliveryReceiptRequested*. The *deliverSemantics* has an example value of *OnceAndOnlyOnce* which indicates the importance of this element.
    - *Description* - This is a simple human readable description of the message and is optional.

- *TraceHeaderList(optional)* - The *TraceHeaderList* element consists of one or more *TraceHeaders* elements. If a message is transferred to a remote ebXML MSH via multiple MSH, each MSH adds a *TraceHeader* to the *TraceHeaderList*. It is typically used in a multi-hop scenario.

- *ErrorList (optional)* – It contains a list of errors for reporting an error on a previous message.

- *Signature (optional)* – It is an optional element for digital signature.

- *Acknowledgement (optional)* – it is used for sending acknowledgement by receiving message handler to acknowledge sending message handler.

- *Via (optional)* – The *Via* element indicates the way a message goes from point A to point B.

3.10.6 ebXML SOAP Body

SOAP Body elements includes: *Manifest, StatusRequest, StatusResponse, and DeliveryReceipt.*

- *Manifest* - The *Manifest* element is a very important element. It is an element for pointing to any data present in the Payload container. The *Manifest* element is a composite element consists of one or more Reference elements. As mentioned above an ebXML Message has a Header Container and zero or more Payload Container. Each Reference element references one payload in the Payload Container. The Reference element itself has a Schema element which can link to the schema of the referenced payload.
- *StatusRequest (optional)* - The *StatusRequest* element is used to request status of a message, referenced by the *RefToMessageId* element of the *StatusReqeust* element.

- The *StatusResponse (optional)* - element is the return message to a *StatusRequest* message. This element has child elements and attributes: *RefToMessageId, TimeStamp, version, messageStatus* and *an id*. The *MessageStatus* can be: *UnAuthorized, NotRecognized or Received.*
- *DeliveryReceipt(optional)*: The *DeliveryReceipt* element is used to indicate, that a previous message (referenced by *RefToMessageId*) was received [11].

3.10.7 Reliable Messaging

Reliability is a key ebMS extension of SOAP. The reliability module of ebMS is designed to guarantee a sending service handler can deliver a specific message once and only once to a recipient message service handler. The reliable messaging module consists of a number of extensions to the SOAP message format and a reliable messaging protocol that specifies behavior of ebMS handlers.

At the message format level, ebMS defines an optional *<eb:AckRequested>* extension element for the *<SOAP:Header>.* If specified, the responding message handler can send a message containing another *<eb:Acknowledgment>* extension element, with an *<eb:RefToMessageId>* element to specify which message is being acknowledged. The reliability module interacts with the security module: the *eb:AckRequested* has a *signed* attribute that can request the responding message handler to sign the acknowledgment digitally in order to provide non-repudiation of receipt.

The Reliable Messaging Protocol specifies a mechanism for resending lost messages or lost acknowledgments. The maximum number of times, or the maximum time interval, for resending these messages or acknowledgments may be configured differently for different business partners.

3.10.8 Error handling

Finally, the Error Handling component deals with reporting errors detected in a message to another MSH. This component can be considered as an "application-level handler" on top of the SOAP processor layer, which is necessary because errors can occur anywhere in the MSH (SOAP-related, security, reliability) or in the application. *ErrorList* element is used to send information about the error(s) to the other MSH

3.10.9 Security

No technology, regardless of how advanced it might be, is an adequate substitute to the effective application of security management policies and practices. The ebXML Message Service is no exception. The ebXML Message Service specification and the ebXML Technical Architecture Security specification provide some guidance as to how security management policies and practices can and should be used to minimize risks that are introduced when doing business electronically via the Internet. A primary solution in ebXML is using digital signature to verify the identity of sender and recipient. It can be used to ensure the integrity of the message and to verify that it was sent or received. The digital signature enables secure transactions, ensuring the integrity and authenticity of origin for business documents. A digital signature is an electronic identifier created by a computer. It is intended to have the same force and effect as the use of a manual signature. The security features for an exchange of business information used by ebXML requirements:

- Confidentiality: Only sender and receiver can interpret document contents.
- Authentication of sender: the sender's identity is verified.
- Authentication of receiver: the receiver's identity is verified.
- Integrity: The message contents have not been altered.
- Non-repudiation of origin: The sender cannot deny having sent the message.
- Non-repudiation of receipt: The receiver cannot deny having received the message.
- Archiving: A document can be reconstructed for a certain time period after its creation.

Because the ebXML Message Service has been designed to be transport and network protocol neutral, a variety of network protocol security standards such as SSL and IPSEC may be used to provide confidentiality, authentication and message integrity, thus enhancing the security countermeasures that are defined in the ebXML Message Service Specification.

Chapter 4 Implementation of ebMS prototype

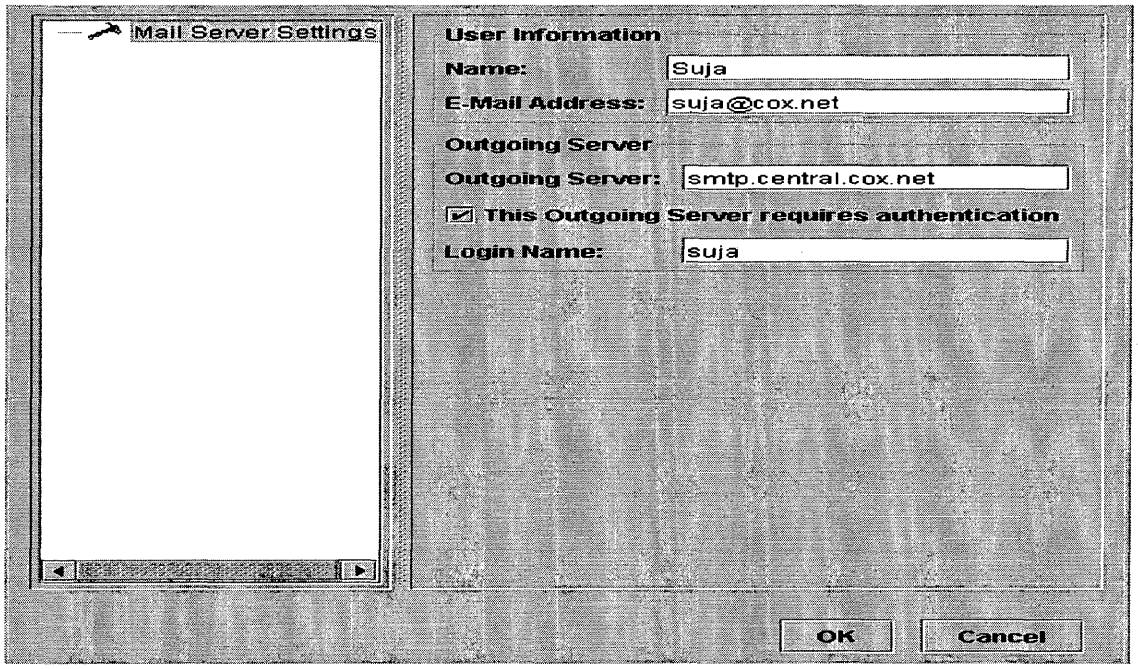
4.1 The ebXML messaging prototype implementation

The ebXML messaging prototype has built in Java Swing. Java Swing is a GUI Toolkit that provides many components that allow building sophisticated Java user interfaces. Java is an object-oriented programming (OOP) language used in web development to provide system services not available through HTML. Java allows a single application to run on multiple platforms. Ant compiler is used (java based build tool from Apache) for building, packaging and installing java applications. Ant engine translates to OS specific commands. Thus, ant files work cross platforms.

4.2 User interface Overview

This prototype involves a client-based java application. The software has implemented with packaging and transporting features over Simple Mail transport Protocol (SMTP) based on ebXML messaging service specification. Graphical user Interface (GUI) captures the data from the user, transport documents through internet.

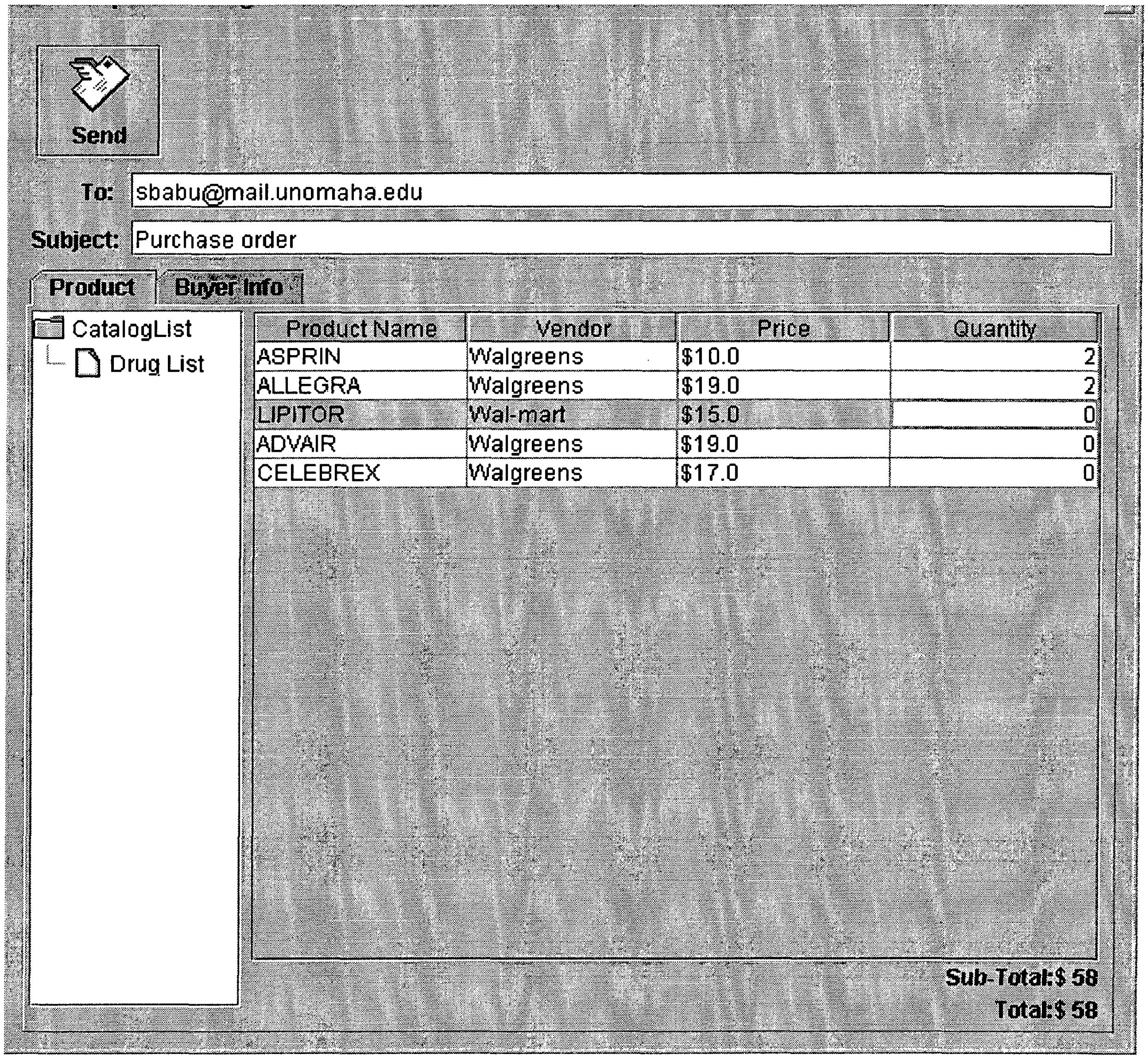
There are three main user interface screens where the data entry by the user was captured. “Mail Server Settings” screen have all necessary information of sender’s mailing profile. It stores the information about outgoing server name, email address, email user name etc. This information is used for ebXML message header.



User Information
Name: Suja
E-Mail Address: suja@cox.net
Outgoing Server
Outgoing Server: smtp.central.cox.net
[✓] This Outgoing Server requires authentication
Login Name: suja

Figure 4-1 Screen shot of Java Prototype – Mail server settings

We use purchase order as an example for payload data. "Send Messaging" screen consists of two tab pages. "Product" tab page has information about the product for purchase order. "Buyer's Info" has all the details about delivery and payment info of the buyer.



To: sbabu@mail.unomaha.edu

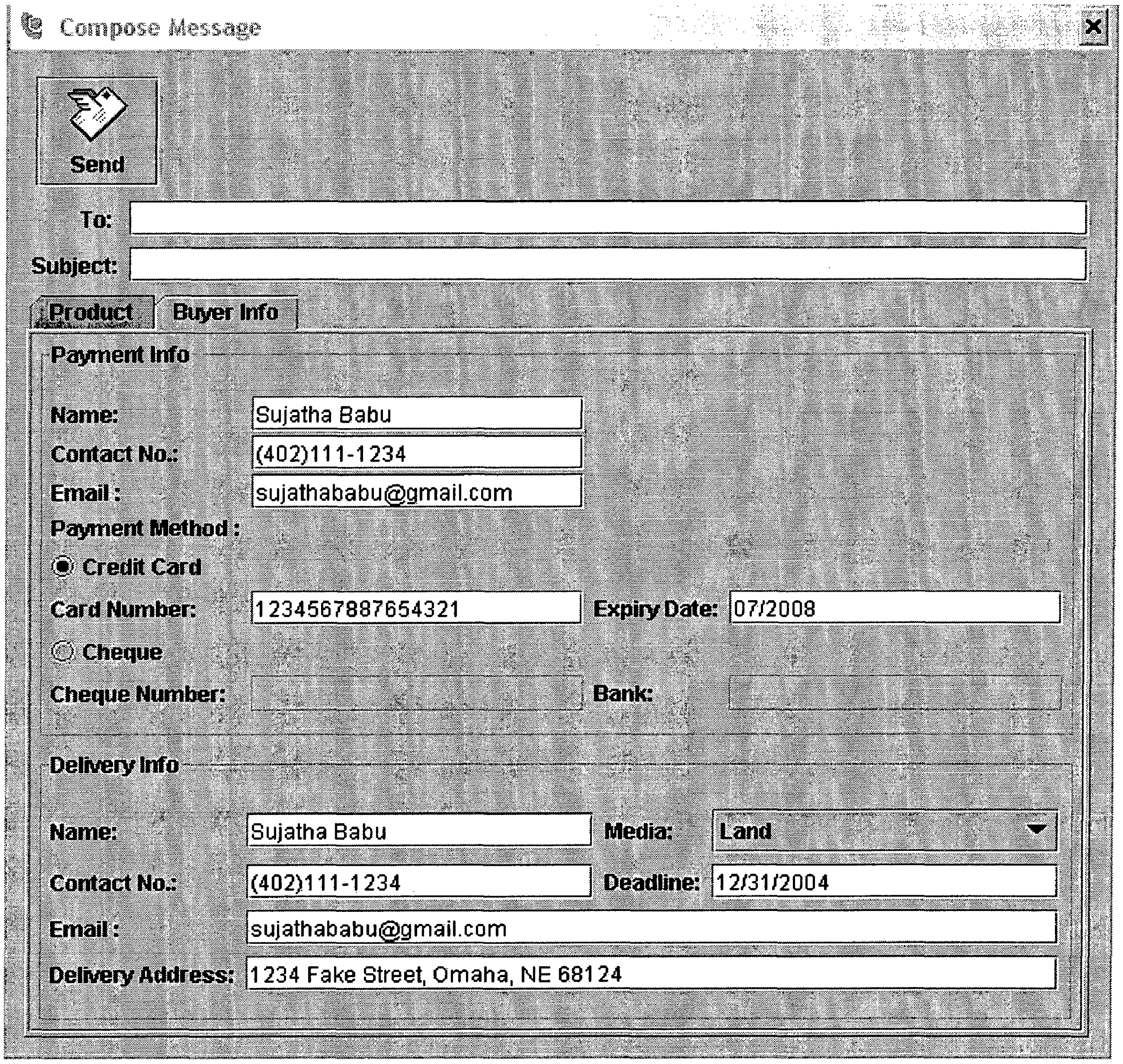
Subject: Purchase order

Product Name	Vendor	Price	Quantity
ASPRIN	Walgreens	$10.0	2
ALLEGRA	Walgreens	$19.0	2
LIPITOR	Wal-mart	$15.0	0
ADVAIR	Walgreens	$19.0	0
CELEBREX	Walgreens	$17.0	0

Sub-Total:$ 58

Total:$ 58

Figure 4-2 Screen shot of Java Prototype – Product Info



To:

Subject:

Payment Info

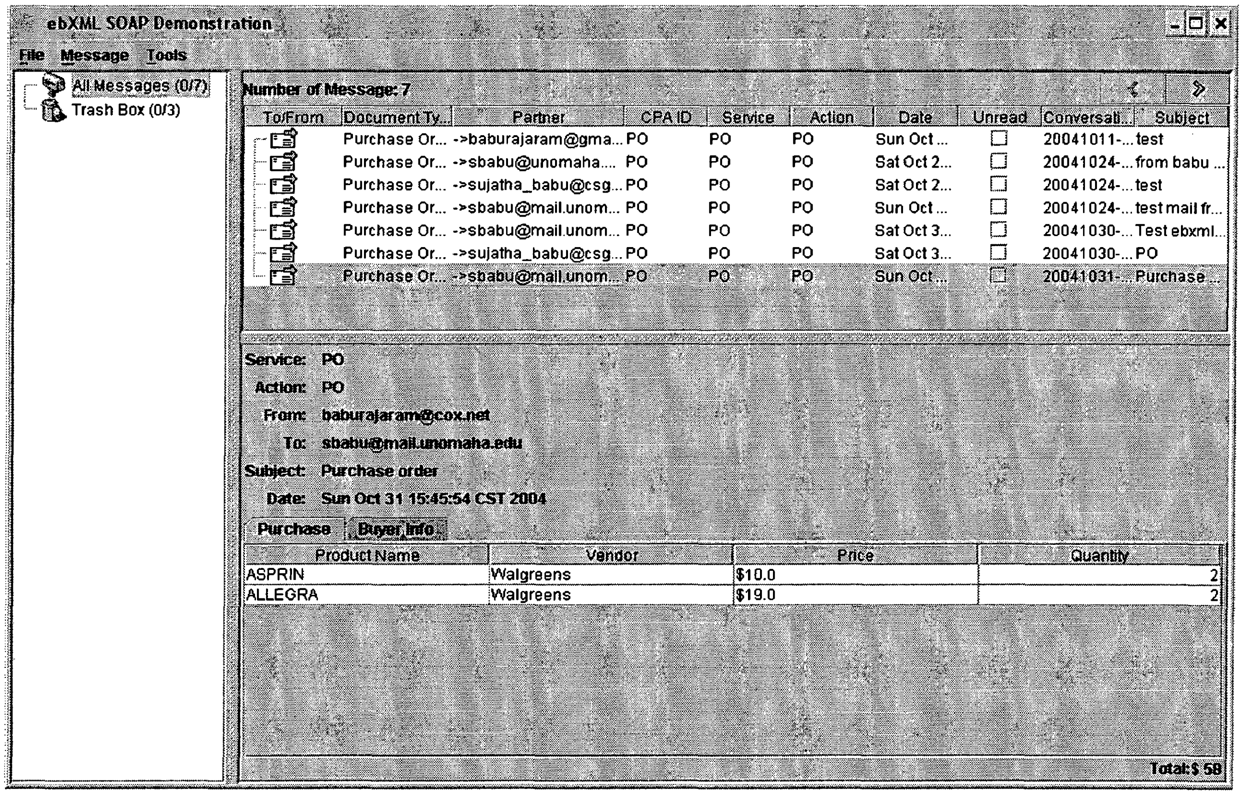
Name:	Sujatha Babu		
Contact No.:	(402)111-1234		
Email :	sujathababu@gmail.com		
Payment Method :			
● Credit Card			
Card Number:	1234567887654321	Expiry Date:	07/2008
○ Cheque			
Cheque Number:		Bank:	

Delivery Info

Name:	Sujatha Babu	Media:	Land
Contact No.:	(402)111-1234	Deadline:	12/31/2004
Email :	sujathababu@gmail.com		
Delivery Address:	1234 Fake Street, Omaha, NE 68124		

Figure 4-3 Screen shot of Java Prototype – Buyer Info

After the data entry, when the user click “send” button. The message will be send to the email address with attachments containing ebXML message header and payload data. That information can be stored and viewed in the screen.



Number of Message: 7

To/From	Document Ty...	Partner	CPA ID	Service	Action	Date	Unread	Conversati...	Subject
	Purchase Or...	->baburajaram@gma...	PO	PO	PO	Sun Oct ...	☐	20041011-...	test
	Purchase Or...	->sbabu@unomaha....	PO	PO	PO	Sat Oct 2...	☐	20041024-...	from babu ...
	Purchase Or...	->sujatha_babu@csg...	PO	PO	PO	Sat Oct 2...	☐	20041024-...	test
	Purchase Or...	->sbabu@mail.unom...	PO	PO	PO	Sun Oct ...	☐	20041024-...	test mail fr...
	Purchase Or...	->sbabu@mail.unom...	PO	PO	PO	Sat Oct 3...	☐	20041030-...	Test ebxml...
	Purchase Or...	->sujatha_babu@csg...	PO	PO	PO	Sat Oct 3...	☐	20041030-...	PO
	Purchase Or...	->sbabu@mail.unom...	PO	PO	PO	Sun Oct ...	☐	20041031-...	Purchase ...

Service: PO
Action: PO
From: baburajaram@cox.net
To: sbabu@mail.unomaha.edu
Subject: Purchase order
Date: Sun Oct 31 15:45:54 CST 2004

Product Name	Vendor	Price	Quantity
ASPRIN	Walgreens	$10.0	2
ALLEGRA	Walgreens	$19.0	2

Total:$ 58

Figure 4-4 Screen shot of Java Prototype – Messages

4.3 Technical Overview

This section explains how the prototype works technically. The figure below shows the implementation of Java application interacts with MSH (message handler) that provides the payloads and the underlying application level communication protocol (SMTP) which delivers the message to the other MSH. It also depicts the interaction between two ebXML applications through a layered SMTP handler. The solid arrow between SMTP handlers signifies the real connection through which bytes of data are exchanged. The dotted arrow between MSHs and the ebXML applications signify the virtual connection.

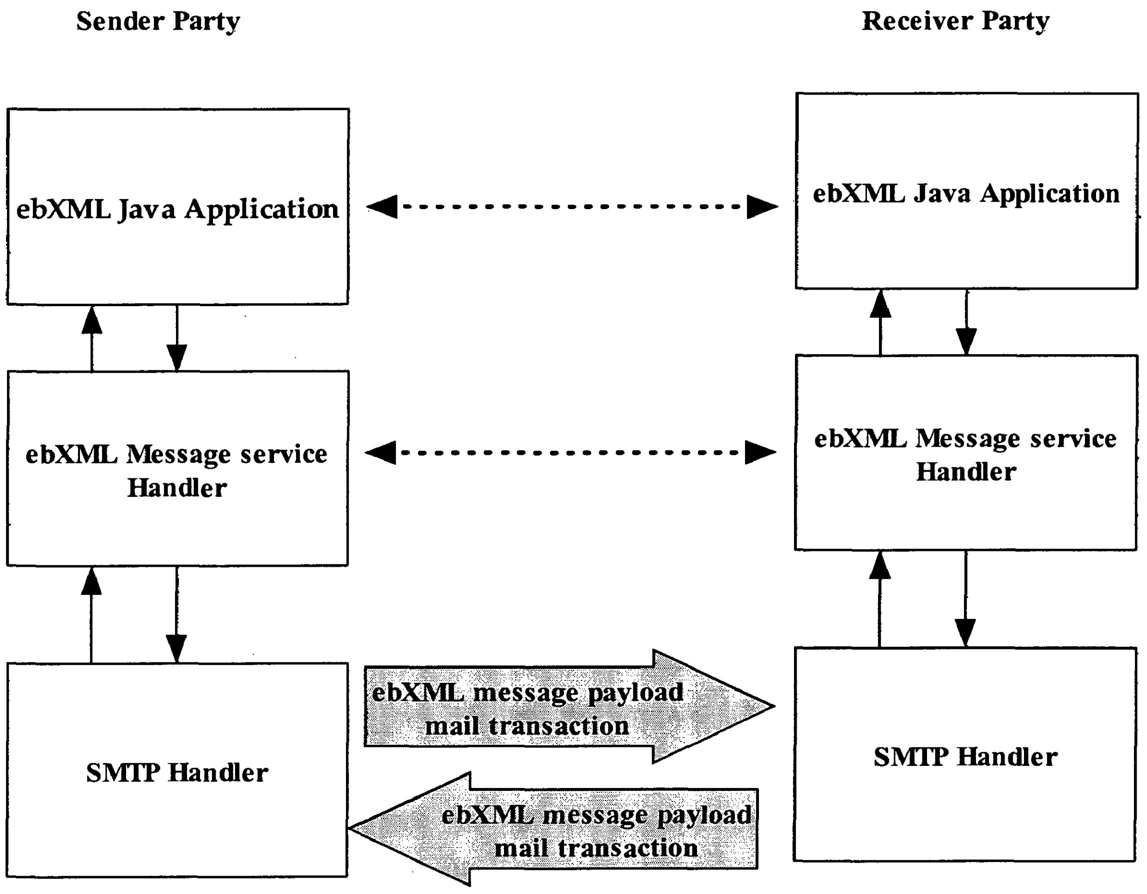

Sender Party
Receiver Party
ebXML Java Application
ebXML Java Application
ebXML Message service Handler
ebXML Message service Handler
SMTP Handler
SMTP Handler
ebXML message payload mail transaction
ebXML message payload mail transaction

Figure 4-5 ebXML messaging prototype SMTP transaction

The message structure consists of an envelope and zero or more payloads transported as attachments to the envelope. ebMS defines the required elements in the message, this prototype will produce the message which contains only required elements as given by ebXML specification.

4.4 ebXML Header Container

The ebXML Java application provides valid values of CPAId, Service, Action elements and the payloads to be sent including all meta-data related to each payload. In real scenario Service and Action element contents is retrieved from the stored CPA. In this project dummy CPAId, Service and Action elements are used because creating actual CPA is not in the scope of the project. MSH creates the ebXML header (SOAP header with SOAP extensions) contents based on CPA.

```
<eb:From>
        <eb:PartyId>mailto:babu@cox.net</eb:PartyId>
    </eb:From>

<eb:To>
        <eb:PartyId>mailto:suja@csgsystems.com</eb:PartyId>
</eb:To>

<eb:CPAId>PO</eb:CPAId>

<eb:ConversationId>
20041030-202132031-PO.PO.PO.1@192.168.0.2
</eb:ConversationId>

    <eb:Service>PO</eb:Service>
    <eb:Action>PO</eb:Action>
```

Figure 4-6 Creation of From, To, Service and Action elements

The To and From elements are filled with information from the CPA. The message header “From” and “To” elements must contain the SMTP compliant email address as per ebXML SMTP specification.

In real world environment, the message context should inform MSH whether a new Conversation context is desired or an existing conversation context is to be used. In the later case, message context provides the ConversationId of an existing conversation. The ConversationId associated with the Conversation context is inserted in this message. If the new message is associated with a previously received or sent message, then the MSH inserts a RefToMessageId element with the value of that MessageID. As per the specification ConversationId should be unique. Our program works in such a way it always considers a message as the first conversation.

A new messageId is created for each message. The program generates the MessageId as combination of current date, time, service, action element along with IP address of the network.

```
<eb:MessageData>
<eb:MessageId>20041030-202132031-PO.PO.PO.1@192.168.0.2</eb:MessageId>
        <eb:Timestamp>2004-10-30T20:21:32</eb:Timestamp>
 </eb:MessageData>
```

Figure 4-7 Creation of MessageData elements

As shown above, the MSH inserts the required TimeStamp element in the MessageData element with the current time at the Sending MSH, where the format of the element conforms to XML schema.

The eb:Manifest element within the SOAP body element is populated with the payload meta-information cid reference which is used to refer the payload. XLink is a linking mechanism that is somewhat similar to HTML links. However, unlike HTML links, XLink permits bidirectional links and/or one link to connect many documents together.

```
<eb:Manifest
xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2
_0.xsd" eb:version="2.0">
<eb:Reference eb:id="0" xlink:type="simple" xlink:href="cid:0">
        <eb:Description xml:lang="en-US">description</eb:Description>
        </eb:Reference>
 </eb:Manifest>
```

Figure 4-8 Creation of Manifest elements

```
<soap-env:Envelope xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
    <soap-env:Header
xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-heade
r-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:MessageHeader eb:version="2.0" soap-env:mustUnderstand="1">
    <eb:From>
<eb:PartyId>mailto:baburajaram@cox.net</eb:PartyId>
</eb:From>
    <eb:To>
<eb:PartyId>mailto:suja_babu@hotmail.com</eb:PartyId>
</eb:To>
<eb:CPAId>PO</eb:CPAId>
<eb:ConversationId>20040912-183828546-PO.PO.PO.1@192.168.0.2</eb:ConversationI
d>
<eb:Service>PO</eb:Service>
<eb:Action>PO</eb:Action>
    <eb:MessageData>
<eb:MessageId>20040912-183828546-PO.PO.PO.1@192.168.0.2</eb:MessageId>
<eb:Timestamp>2004-09-12T18:38:28</eb:Timestamp>
</eb:MessageData>
<eb:Description xml:lang="en-US">Test from ebXML</eb:Description>
</eb:MessageHeader>
</soap-env:Header>
    <soap-env:Body
xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-heade
r-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:Manifest eb:version="2.0">
    <eb:Reference eb:id="0" xlink:type="simple" xlink:href="cid:0">
<eb:Description xml:lang="en-US">description</eb:Description>
</eb:Reference>
</eb:Manifest>
</soap-env:Body>
</soap-env:Envelope>
```

Figure 4-9 ebXML message header generated by the prototype

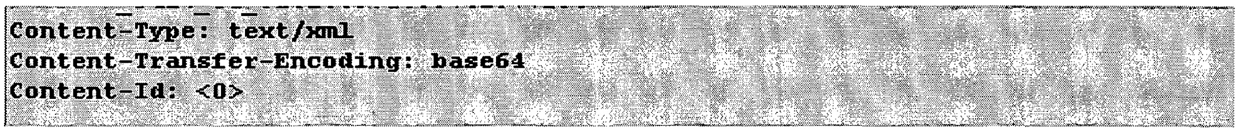
4.5 ebXML Payload container

The ebXML payload contains the actual document to be sent. The ebXML payload can be simple plain text object or a complex multipart object. The ebXML does not define the structure or content of application payloads. The specification of the structure and composition of payload objects are defined by the organization that defines the business process. The ebXML message service is payload-neutral, meaning that any kind of information can be reliably routed. This information can include XML documents, binary data, or EDI messages. Business can incorporate ebXML technology to leverage their existing infrastructure. Our prototype has the capability to develop an XML message.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<purchaseOrder accept="false" orderDate="Sun Oct 31 19:41:05 CST 2004"
isFollowUp="false">
<deliveryInfo deadline="12/31/2004" media="Land">
  <receiver>Sujatha Babu</receiver>
  <address>1234 Fake Street, Omaha, NE 68124</address>
  <contact>(402)111-1234</contact>
  <email>sujathababu@gmail.com</email>
  </deliveryInfo>
<paymentInfo>
  <name>Sujatha Babu</name>
  <contact>(402)111-1234</contact>
  <email>sujathababu@gmail.com</email>
 <paymentMethod>
<creditCard>
  <cardNumber>1234567887654321</cardNumber>
  <expireDate>07/2008</expireDate>
  </creditCard>
  </paymentMethod>
  <total>0</total>
  </paymentInfo>
<purchase orderDate="Sun Oct 31 19:40:35 CST 2004">
<product productID="3">
  <productName>LIPITOR</productName>
  <vendor>Wal-mart</vendor>
  <property name="Net Weight">50g</property>
  <price>15.0</price>
  <quantity>3</quantity>
  </product>
<product productID="4">
  <productName>ADVAIR</productName>
  <vendor>Walgreens</vendor>
  <property name="Net Weight">200g</property>
  <price>19.0</price>
  <quantity>3</quantity>
  </product>
<product productID="1">
  <productName>ASPRIN</productName>
  <vendor>Walgreens</vendor>
  <property name="Net Weight">110g</property>
  <price>10.0</price>
  <quantity>1</quantity>
  </product>
  </purchase>
  </purchaseOrder>
```

Figure 4-10 ebXML payload generated by the prototype

Payload contains the purchase order data generated from the user entry on “Compose message” screen. In this project, an XML payload is generated; but it is not necessarily to have XML payload, it can be any data format. Receiver, address, contact, delivery info, payment info, Credit card XML elements get the appropriate data from “compose message” screen as entered by the user. Other purchase order data gets populated and filled in the appropriate XML elements.



```
Content-Type: text/xml
Content-Transfer-Encoding: base64
Content-Id: <0>
```

Figure 4-11 Email message property containing content id from SOAP manifest reference element

Email message send by the prototype has a header which has information referenced from the <eb:Reference> element in the <eb:Manifest> that has the eb:cid attribute set to the value 0. Content type explains the type of data being used which is an XML file.

Chapter 5 Conclusion

5.1 Conclusion

The ebXML provides an open framework for global electronic business in the form of good specifications which still actively get updated and enhanced. It is a set of layered specifications that together enable modular electronic business framework. It facilitates global trade by expanding electronic business to new and existing trading partners. It is complex, it employs business process definitions and global registries of potential business partners, and it implements a robust messaging specification with a good security model. And ebXML is not proprietary; it supports the Web services standards SOAP, UDDI, and Web Services Description Language (WSDL).

Productive live ebXML systems in the real world show that ebXML gets adopted by companies around the world and the few, very promising; ebXML open source implementations show the growing interest to provide ebXML technologies on an open source basis. The loosely coupled ebXML components allow integrating ebXML into other frameworks by deploying one component by one.

EbMS is internet friendly, has the reliability and security features that enterprise users require, is suited for XML business payloads, and most importantly is getting endorsements from major industry associations.

Cheaper bandwidth will allow more information to be exchanged between companies for any given transaction. XML drives down implementation costs, especially in light of the well-established commodity pricing for EDI VAN that cannot go lower. Also, because many EDI implementations require significant custom development, XML conversion can reduce the amount of code written to make applications at different companies operate together seamlessly.

Because ebXML is platform and vendor neutral, provides support for different protocols such as HTTP/S and SMTP, offers a flexible payload independent architecture, provides security based on digital signatures, supports attachments and high performance, many consider it a future-proof investment and a solid foundation on which to build. Even with the progress ebXML has achieved, the initiative is still considered to be in its infancy.

Trends suggest that EDI implementations will remain a key technology for processing high volume transaction-based information between large organizations. However, these organizations will add-in XML support to process these same types of transactions with their SME suppliers. As SMEs discover the benefits of integrating XML messages from large customers directly into their back office systems, so they will encourage their own customers and suppliers to communicate using XML. Some agencies have been doing EDI for a long time, and it might be more cost-effective to stay on EDI than migrate away from EDI. They may have trading partners who don't want to migrate away from EDI.

Industry-wide DTD will still be needed, in the same way as the different EDIFACT subsets used now in different sectors, i.e. there cannot be an Invoice DTD both universal and simple. There will still be the need for partners' agreements to refer to specific repositories, sets of DTD, sets of codes (currency, country, etc.). For these reasons, the market transformation from EDI to XML is estimated to require several years.

The evolution has only just begun, and the transition from EDI to ebXML systems will not occur overnight. Many companies will migrate systems to XML-driven infrastructure only as standards consolidate and the technology stabilizes as the mainstay of business computing. One of the challenges facing companies looking to integrate EDI and XML-based systems is linking and synchronizing the business content, including the documents, policies and procedures that form the foundation of the two types of infrastructure.

The original specifications were immature and untried. This young technology needs time to mature. As the specifications mature and software and tools are developed to enable ebXML transactions organizations and industries may adopt some or all of the ebXML specifications within their standards. There are a lot of companies doing electronic inter enterprise business today not only with EDI but with the availability of the Internet and it is a matter of time when these companies embrace ebXML.

Bibliography

[1] Alan Kotok, XML and EDI Lessons Learned and Baggage to Leave Behind, August 1999, www.xml.com/pub/a/1999/08/edi/

[2] John Moore, Organizations can manage both EDI and XML to get the best of both worlds, Nov 2002, http://www.fcw.com/fcw/articles/2002/1118/tec-mixed-11-18-02.asp

[3] Dennis Krukkert, Matching of ebXML business processes, Oct 2003

[4] Kristian Cibulskis, The ebXML Registry, http://www.syscon.com/xml/article.cfm/ Id=315&count=15093&tot=5&page=4

[5] Brian Gibb & Suresh Damodaran, ebXML concepts and application, 2003

[6] ebXML.org, ebxml technical architecture specification v.1.0.4. http://www.ebxml.org/specs/ebBPSS.pdf, May 2001

[7] www.techmetrix.com, 2004

[8] Eric Chiu, Eric Chiu, ebXML Simplified:A Guide to the New Standard for Global E Commerce, 2002

[9] Aaron E.Walsh, UDDI, SOAP and WSDL The web services specification reference book, 2000

[10] Kennard Scribner and Mark C. Stiver, Understanding SOAP – the authoritative solution, 2000

[11] Sacha Schlegel, **ebXML (electronic business XML)** http://www.schlegel.li/ebXML/postgraduate_report/www/node1.html, 2002