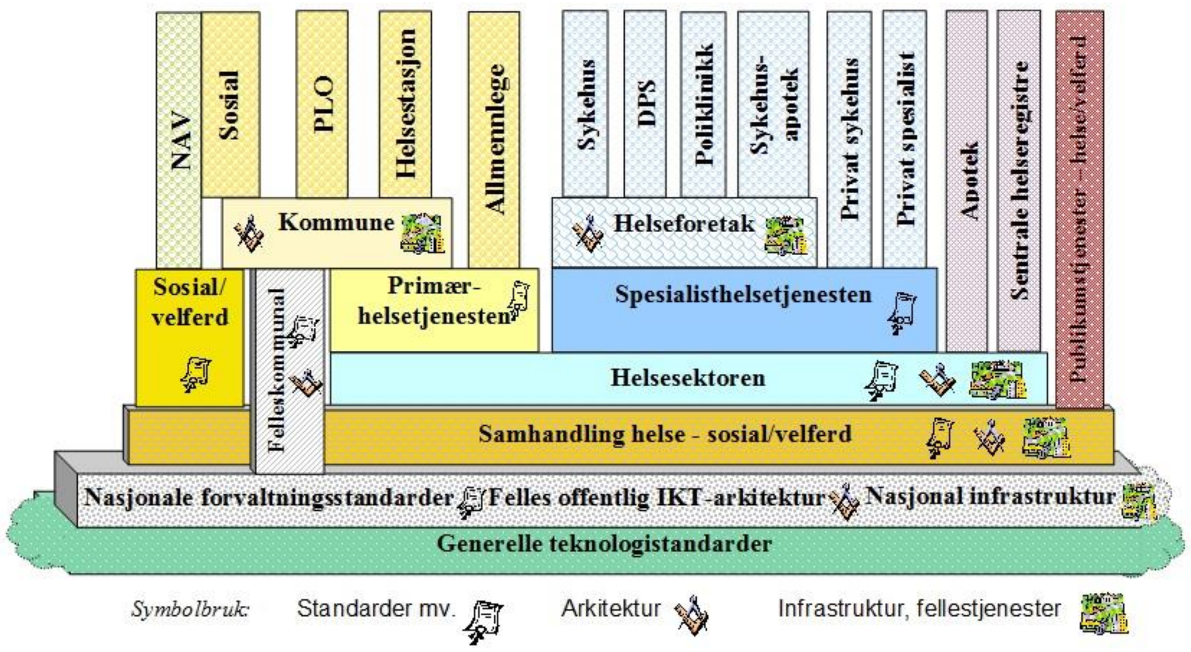
# SDP Dokument 4 MB

Dette er et testdokument for Sikker digital posttjeneste.



## 1 Tabell

| Kolonne 1 | Kolonne 2 | Kolonne 3 | Kolonne 4 | Kolonne 5 |
|---|---|---|---|---|
| 23 | | | | |
| 18 | | | | |
| 53 | | | | |
| 66 | | | | |
| 75 | | | | |
| 93 | | | | |
| 44 | | | | |

| Kolonne 1 | Kolonne 2 | Kolonne 3 | Kolonne 4 | Kolonne 5 |
|---|---|---|---|---|
| 23 | | | | |
| 18 | | | | |
| 53 | | | | |
| 66 | | | | |
| 75 | | | | |
| 93 | | | | |
| 44 | | | | |

| Kolonne 1 | Kolonne 2 | Kolonne 3 | Kolonne 4 | Kolonne 5 |
|---|---|---|---|---|
| 23 | | | | |
| 18 | | | | |
| 53 | | | | |

| | | | | |
|---|---|---|---|---|
| 66 | | | | |
| 75 | | | | |
| 93 | | | | |
| 44 | | | | |

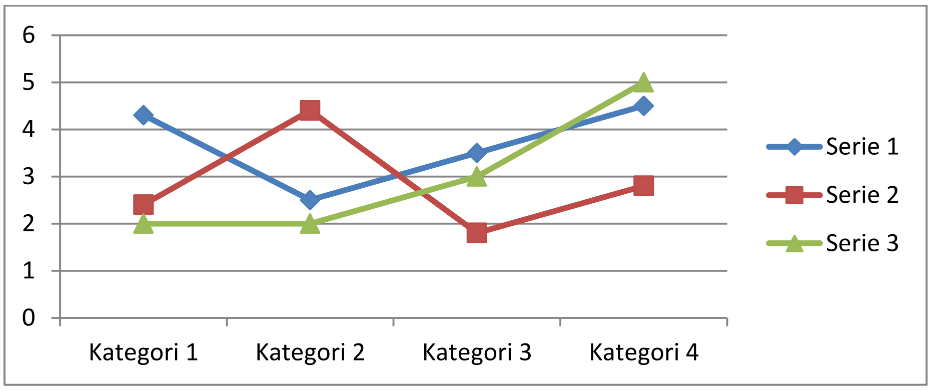
| Kolonne 1 | Kolonne 2 | Kolonne 3 | Kolonne 4 | Kolonne 5 |
|---|---|---|---|---|
| 23 | | | | |
| 18 | | | | |
| 53 | | | | |
| 66 | | | | |
| 75 | | | | |
| 93 | | | | |
| 44 | | | | |

| Kolonne 1 | Kolonne 2 | Kolonne 3 | Kolonne 4 | Kolonne 5 |
|---|---|---|---|---|
| 23 | | | | |
| 18 | | | | |
| 53 | | | | |
| 66 | | | | |
| 75 | | | | |
| 93 | | | | |
| 44 | | | | |

| Kolonne 1 | Kolonne 2 | Kolonne 3 | Kolonne 4 | Kolonne 5 |
|---|---|---|---|---|
| 23 | | | | |
| 18 | | | | |
| 53 | | | | |
| 66 | | | | |
| 75 | | | | |
| 93 | | | | |
| 44 | | | | |

| Kolonne 1 | Kolonne 2 | Kolonne 3 | Kolonne 4 | Kolonne 5 |
|---|---|---|---|---|
| 23 | | | | |
| 18 | | | | |
| 53 | | | | |
| 55 | | | | |
| 56 | | | | |
| 58 | | | | |
| 66 | | | | |
| 75 | | | | |
| 93 | | | | |
| 44 | | | | |

## 2 Grafer



## 3 Tekst

### Table (database)

A table is a collection of related data held in a structured format within a database. It consists of fields (columns), and rows.[1]

In relational databases and flat file databases, a **table** is a set of data elements (values) using a model of vertical columns (which are identified by their name) and horizontal rows, the cell being the unit where a row and column intersect.[2] A table has a specified number of columns, but can have any number of rows.[3] Each row is identified by the values appearing in a particular column subset which has been identified as a unique key index.

Table is another term for relation; although there is the difference in that a table is usually a multiset (bag) of rows where a relation is a set and does not allow duplicates. Besides the actual data rows, tables generally have associated with them some metadata, such as constraints on the table or on the values within particular columns.[*dubious* – *discuss*]

The data in a table does not have to be physically stored in the database. Views are also relational tables, but their data are calculated at query time. Another example are nicknames, which represent a pointer to a table in another database.[4]

### Comparisons

In non-relational systems, hierarchical databases, the distant counterpart of a table is a structured file, representing the rows of a table in each record of the file and each column in a record. This structure implies that a record can have repeating information, generally in the child data segments. Data are stored in sequence of records, which are equivalent to table term of a relational database, with each record having equivalent rows.

Unlike a spreadsheet, the datatype of field is ordinarily defined by the schema describing the table. Some SQL systems, such as SQLite, are less strict about field datatype definitions.

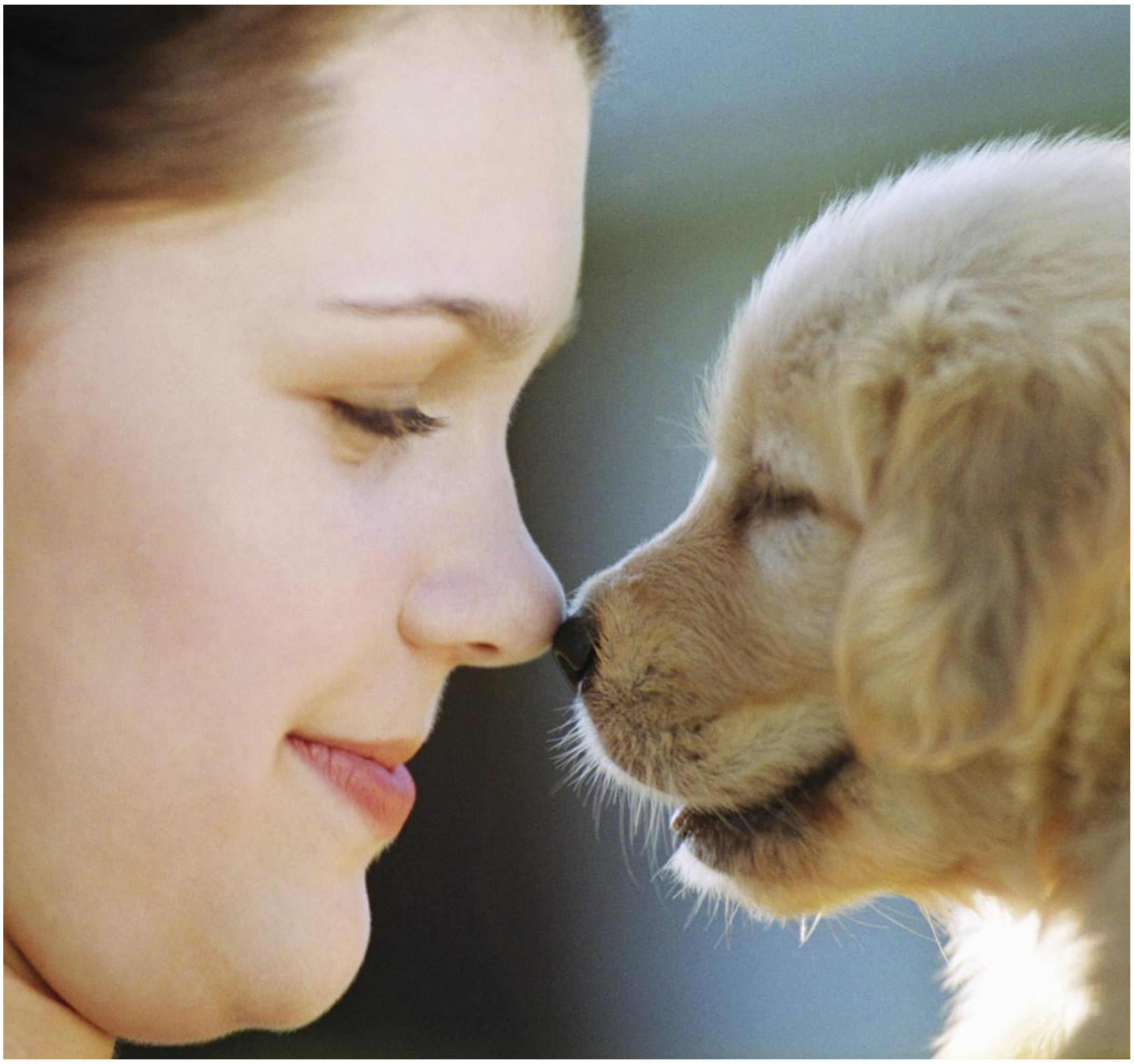
### Tables versus relations

In terms of the relational model of databases, a table can be considered a convenient representation of a relation, but the two are not strictly equivalent. For instance, an SQL table can potentially contain duplicate rows, whereas a true relation cannot contain duplicate tuples. Similarly, representation as a table implies a particular ordering to the rows and columns, whereas a relation is explicitly unordered. However, the database system does not guarantee any ordering of the rows unless an `ORDER BY` clause is specified in the `SELECT` statement that queries the table.

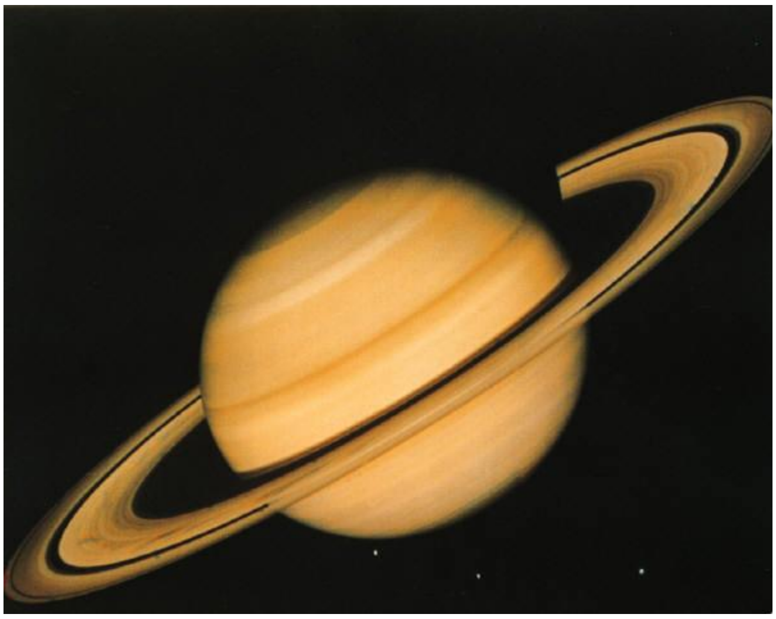
An equally valid representations of a relation is as an *n*-dimensional chart, where *n* is the number of attributes (a table's columns). For example, a relation with two attributes and three values can be represented as a table with two columns and three rows, or as a two-dimensional graph with three points. The table and graph representations are only equivalent if the ordering of rows is not significant, and the table has no duplicate rows.

## 4 Bilde

## 5 Bilde



92
TAXI



## 6 Tekst

### World

From Wikipedia, the free encyclopedia

*For other uses, see World (disambiguation).*

"The Blue Marble" photograph of Earth.

The flag of the World Health Organizationcombines a modern world map (azimuthal equidistant projection) with the Rod of Asclepius, in origin a symbol of the axis mundi[1]

**World** is a common name for the whole of human civilization, specifically human experience, history, or the human condition in general, *worldwide*, i.e. anywhere on Earth[2] or pertaining to anywhere on earth.

In a philosophical context it may refer to:

1. the whole of the physical Universe, or
2. an ontological world (*see world disclosure*).

In a theological context, *world* usually refers to the material or the profane sphere, as opposed to the celestial, spiritual, transcendent or sacred. The "end of the world" refers to scenarios of the final end of human history, often in religious contexts.

World history is commonly understood as spanning the major geopolitical developments of about five millennia, from the first civilizations to the present.

World population is the sum of all human populations at any time; similarly, world economy is the sum of the economies of all societies (all countries), especially in the context of globalization. Terms like world championship, gross world product, world flags etc. also imply the sum or combination of all current-day sovereign states.

In terms such as world religion, world language, world government, and world war, *world* suggests international or intercontinental scope without necessarily implying participation of the entire world.

In terms such as world map and world climate, *world* is used in the sense detached from human culture or civilization, referring to the planet Earth physically.

## Etymology and usage

The English word *world* comes from the Old English *weorold (-uld), weorld, worold (-uld, -eld)*, a compound of *wer* "man" and *eld* "age," which thus means roughly "Age of Man."[3] The Old English is a reflex of theCommon Germanic **wira-alđiz*, also reflected in Old Saxon *werold*, Old High German *weralt*, Old Frisian *warld* and Old Norse *verǫld* (whence the Icelandic *veröld*).[4]

The corresponding word in Latin is *mundus*, literally "clean, elegant", itself a loan translation of Greek *cosmos* "orderly arrangement." While the Germanic word thus reflects a mythological notion of a "domain of Man" (compare Midgard), presumably as opposed to the divine sphere on the one hand and the chthonic sphere of the underworld on the other, the Greco-Latin term expresses a notion of creation as an act of establishing order out of chaos.

'World' distinguishes the entire planet or population from any particular country or region: *world affairs* pertain not just to one place but to the whole world, and *world history* is a field of history that examines events from a global (rather than a national or a regional) perspective. *Earth*, on the other hand, refers to the planet as a physical entity, and distinguishes it from other planets and physical objects.

'World' was also classically used to mean the material universe, or the cosmos: "The worlde is an apte frame of heauen and earthe, and all other naturall thinges contained in them." [5] The earth was often described as 'the center of the world'.[6]

'*World*' can also be used attributively, to mean 'global', 'relating to the whole world', forming usages such as world community or world canonical texts.[7]

By extension, a '*world*' may refer to any planet or heavenly body, especially when it is thought of as inhabited, especially in the context of science fiction or futurology.

'*World*', in original sense, when qualified, can also refer to a particular domain of human experience.

- The *world of work* describes paid work and the pursuit of career, in all its social aspects, to distinguish it from home life and academic study.
- The *fashion world* describes the environment of the designers, fashion houses and consumers that make up the fashion industry.
- historically, the *New World* vs. the *Old World*, referring to the parts of the world colonized in the wake of the age of discovery. Now mostly used in zoology and botany, as New World monkey.

## Philosophy

*The Garden of Earthly Delights* triptych byHieronymus Bosch (c. 1503) shows the "garden" of mundane pleasures flanked by Paradise and Hell. The exterior panel shows the world before the appearance of humanity, depicted as a disc enclosed in a sphere.

In philosophy, the term world has several possible meanings. In some contexts, it refers to everything that makes up reality or the physical universe. In others, it can mean have a specific ontological sense (seeworld disclosure). While clarifying the concept of world has arguably always been among the basic tasks of Western philosophy, this theme appears to have been raised explicitly only at the start of the twentieth century[8] and has been the subject of continuous debate. The question of what the world is has by no means been settled.

### Parmenides

The traditional interpretation of Parmenides' work is that he argued that the every-day perception of reality of the physical world (as described in *doxa*) is mistaken, and that the reality of the world is 'One Being' (as described in aletheia): an unchanging, ungenerated, indestructible whole.

### Plato

In his Allegory of the Cave, Plato distinguishes between forms and ideas and imagines two distinct worlds : the sensible world and the intelligible world.

### Hegel

In Hegel's philosophy of history, the expression *Weltgeschichte ist Weltgericht* (World History is a tribunal that judges the World) is used to assert the view that History is what judges men, their actions and their opinions. Science is born from the desire to transform the World in relation to Man; its final end is technical application.

### Schopenhauer

*The World as Will and Representation* is the central work of Arthur Schopenhauer. Schopenhauer saw the human will as our one window to the world behind the representation; the Kantian thing-in-itself. He believed, therefore, that we could gain knowledge about the thing-in-itself, something Kant said was impossible, since the rest of the relationship between representation and thing-in-itself could be understood by analogy to the relationship between human will and human body.

### Wittgenstein

Two definitions that were both put forward in the 1920s, however, suggest the range of available opinion. "The world is everything that is the case," wrote Ludwig Wittgenstein in his influential *Tractatus Logico-Philosophicus*, first published in 1922. This definition would serve as the basis of logical positivism, with its assumption that there is exactly one world, consisting of the totality of facts, regardless of the interpretations that individual people may make of them.

### Heidegger

Martin Heidegger, meanwhile, argued that "the surrounding world is different for each of us, and notwithstanding that we move about in a common world".[9] The world, for Heidegger, was that into which we are always already "thrown" and with which we, as beings-in-the-world, must come to terms. His conception of "world disclosure" was most notably elaborated in his 1927 work *Being and Time*.

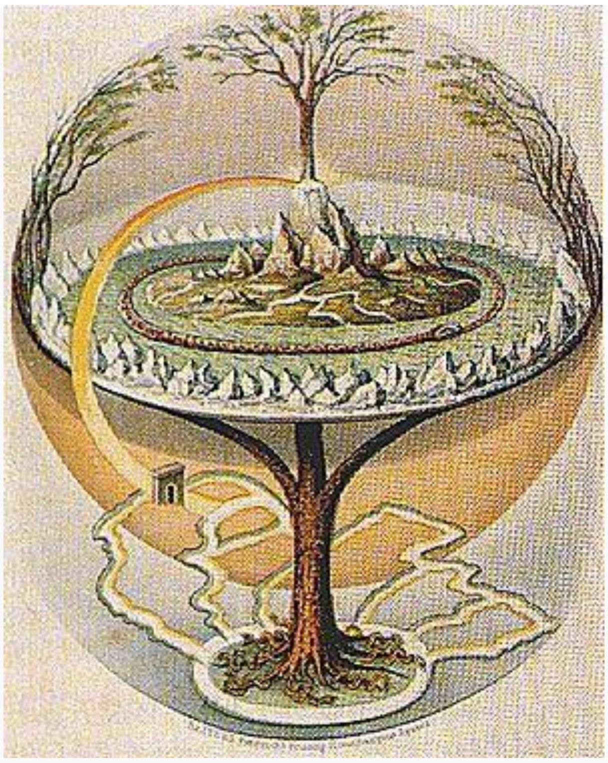
**Freud**

In response, Sigmund Freud proposed that we do not move about in a common world, but a common thought process. He believed that all the actions of a person are motivated by one thing: lust. This led to numerous theories about reactionary consciousness.

**Other**

Some philosophers, often inspired by David Lewis, argue that metaphysical concepts such as possibility, probability and necessity are best analyzed by comparing *the* world to a range of possible worlds; a view commonly known as modal realism.

## Religion and mythology



Yggdrasil, a modern attempt to reconstruct the Norse world tree which connects the heavens, the world, and the underworld.

Mythological cosmologies often depict the world as centered around an axis mundi and delimited by a boundary such as a world ocean, a world serpent or similar. In some religions, worldliness (also called carnality[*citation needed*]) is that which relates to this world as opposed to other worlds or realms.

### 6.1.1 Buddhism

In Buddhism, the world means society, as distinct from the monastery. It refers to the material world, and to worldly gain such as wealth, reputation, jobs, and war. The spiritual world would be the path toenlightenment, and changes would be sought in what we could call the psychological realm.

### 6.1.2 Christianity

In Christianity, the term often connotes the concept of the fallen and corrupt world order of human society, in contrast to the World to Come. The world is frequently cited alongside *the flesh* and *the Devil* as a source of temptation that Christians should flee. Monks speak of striving to be "*in* this world, but not *of* this world"—as Jesus said, and the term "worldhood" has been distinguished from "monkhood", the former being the status of merchants, princes, and others who deal with "worldly" things.

This view is clearly expressed by king Alfred the Great of England (d. 899) in his famous Preface to the *Cura Pastoralis*:

"Therefore I command you to do as I believe you are willing to do, that you free yourself from worldly affairs *(Old English: woruldðinga)* as often as you can, so that wherever you can establish that wisdom that God gave you, you establish it. Consider what punishments befell us in this world when we neither loved wisdom at all ourselves, nor transmitted it to other men; we had the name alone that we were Christians, and very few had the practices."

Although Hebrew and Greek words meaning "world" are used in Scripture with the normal variety of senses, many examples of its use in this particular sense can be found in the teachings of Jesus according to the Gospel of John, e.g. 7:7, 8:23, 12:25, 14:17, 15:18-19, 17:6-25, 18:36. For contrast, a relatively newer concept is Catholic imagination.

*Contemptus mundi* is the name given to the recognition that the world, in all its vanity, is nothing more than a futile attempt to hide from God by stifling our desire for the good and the holy.[10] This view has been criticized as a "pastoral of fear" by modern historian Jean Delumeau.[11]

During the Second Vatican Council, there was a novel attempt to develop a positive theological view of the World, which is illustrated by the pastoral optimism of the constitutions *Gaudium et Spes*, *Lumen Gentium*, *Unitatis Redintegratio* and *Dignitatis Humanae*.

#### 6.1.2.1 Eastern Christianity

In Eastern Christian monasticism or asceticism the world of mankind is driven by passions. Therefore the passions of the World are simply called "the world". Each of these passions are a link to the world of mankind or order of human society. Each of these passions must be overcome in order for a person to receive salvation (theosis). The process of theosis is a personal relationship with God. This understanding is taught within the works of ascetics like Evagrius Ponticus, and the most seminal ascetic works read most widely by Eastern Christians, the Philokalia and the Ladder of Divine Ascent (the works of Evagrius and John Climacus are also contained within the Philokalia). At the highest level of world transcendence is hesychasm which culminates into the Vision of God.

#### 6.1.2.2 Orbis Catholicus

*Orbis Catholicus* is a Latin phrase meaning *Catholic world*, per the expression Urbi et Orbi, and refers to that area of Christendom under papal supremacy. It is somewhat similar to the phrases secular world, Jewish world and Islamic world.

Design Report: Midcentury Icons
By Philip Ward
The chair isn't just an essential everyday object. It is also a medium for designers to express their higher ideals.
airs

THE LEGO MOVIE

## Architectural Overview of ebXML

The experiences of EDI taught us that in the real world, for business-to-business (B2B) collaboration to work successfully a lot more is required than just exchanging documents. One has to deal with issues surrounding business process semantics, negotiating terms and conditions, interoperability, security, reliability, and so on. The inner circle in Figure 7.1 shows the different areas surrounding business collaboration that two partners need to address and the steps the partners must go through to realize the B2B collaboration:

1. **Process definition.** A community or consortium of trading partners defines the business processes to be used in the community according to a well-known domain model and describes them in agreed-upon formats. In ebXML, this is realized using UML and XML. Examples of such consortiums are the Open Travel Alliance*(www.opentravel.org)*, which includes more than one hundred key players in the travel industry; the Global Commerce Initiative *(www.globalcommerceinitiative.org)*, a group of manufacturers and retailers of consumer goods; and the Automotive Industry Action Group *(www.aiag.org)*, of key players in the automotive industry.
2. **Partner discovery.** For two partners to engage, there must be some form of discovery about each other's services and business processes. In ebXML, this is realized using a *registry-repository.*
3. **Partner sign-up.** Partners negotiate their business level and transaction level agreements. In ebXML, these are realized as a Collaboration-Protocol Profile (CPP) and Collaboration-Protocol Agreement (CPA).
4. **Electronic plug-in.** The trading partners configure their interfaces and software according to the agreed-upon business processes and details of the collaborating partner.
5. **Process execution.** Business services collaborate to do and execute the agreed-upon business processes.
6. **Process management.** The business processes defined in the process definition phase (1) and agreed upon in the partner sign-up phase (3) are monitored and facilitated by process management services.
7. **Process evolution.** The partners evaluate their existing processes, improve them through process re-engineering if necessary, and create new processes to meet the needs of the market. Process evolution brings us back to process definition, since the new processes will be defined according to the domain model and published in the community.

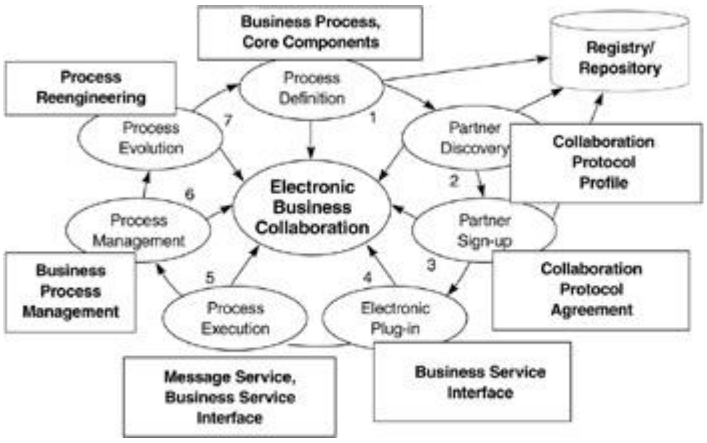

Business Process, Core Components
Registry/ Repository
Process Reengineering
Process Definition
1
Process Evolution
7
Partner Discovery
Collaboration Protocol Profile
Electronic Business Collaboration
6
2
Process Management
Partner Sign-up
Business Process Management
5
4
3
Collaboration Protocol Agreement
Process Execution
Electronic Plug-in
Message Service, Business Service Interface
Business Service Interface

**Figure 7.1:** ebXML frameworks (adapted from the ebXML Business Process Specification Schema)

All of the above can be broadly categorized into two groups as Figure 7.2 shows: design time (things that need to be done before the actual collaboration can be realized) and runtime-(things that are involved in the physical B2B exchange). Let us look at this architecture in further detail.

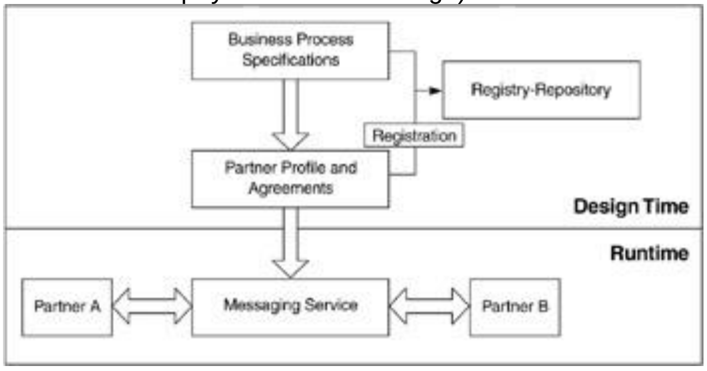


**Figure 7.2:** Design-time and runtime components

## 7 Business Process Specifications

In an electronic exchange, a business document conveys complete intent about the product. For example, when you order books online, the request implicitly or explicitly specifies

- Who are the parties involved (the buyer and the seller).
- What commodity or product is involved.
- When events occur (e.g., credit checks, inspections, shipping).
- Where the product is located.
- How it is going to be handled, shipped, packaged, and so on.

The ebXML business-process model specifies how this information can be captured in an XML format partners can use to configure their ebXML services that execute the business process. The ebXML Business Process Specification Schema (BPSS) specifies this schema that is captured through some business process modeling. To model the business process using object-oriented principles, ebXML defines a complete methodology, called UN/CEFACT modeling methodology (UMM), based on UML and XML. In the grand scheme of things, BPSS is expected to be produced as a result of business modeling using UMM, but UMM is not *required.* Any business process editor is capable of producing the standard schema, using its own modeling techniques. A few tools already support this. Figure 7.3 shows the Bind Studio product that graphically models the collaboration between Flute Bank and OfficeMin (in which Flute Bank uses OfficeMin as its vendor for office supplies-see Chapter 11) to generate the BPSS and CPP/CPA documents.

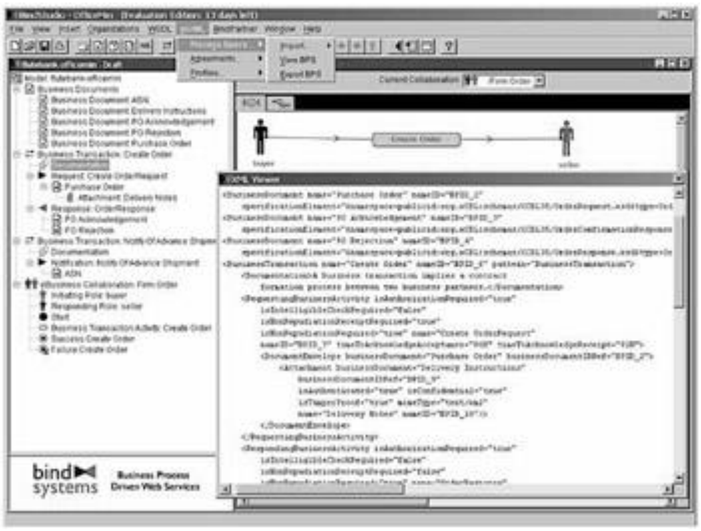
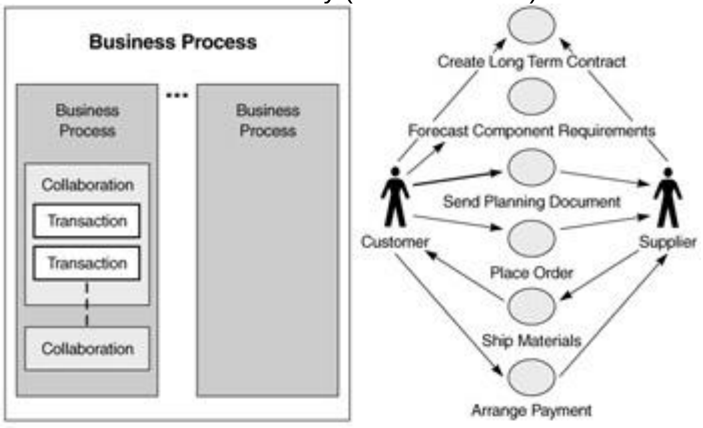

[illegible]

**Figure 7.3:** Graphic process modeling and the Business Process Specification Schema

The BPSS document (schema instance) generated is an XML representation of the use cases. As Figure 7.4 shows it models a business process as a set of *collaborations* composed of discrete *transactions.* For example the document in Listing 7.1 shows how a collaboration is composed of a create order transaction between Flute Bank and OfficeMin, their roles (buyer, seller), and the states in that activity (success/failure).



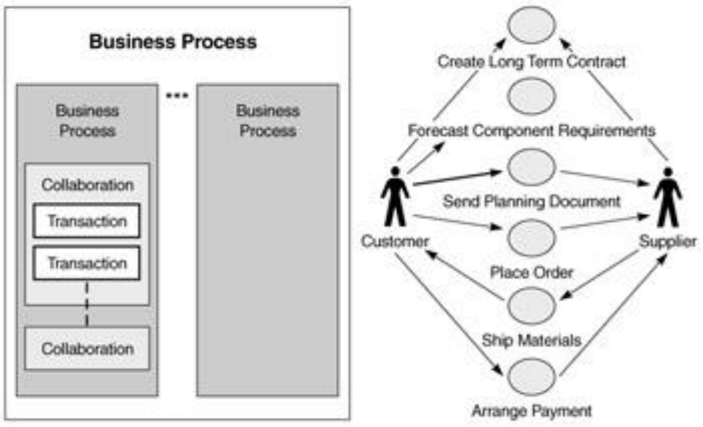


**Figure 7.4:** Use cases map to collaborations grouped as discrete transactions (adapted from the ebXML Business Process Specification Schema)

**Listing 7.1: The BPSS document**

```
<?xml version="1.0" encoding="UTF-8"?>

<ProcessSpecification name="flutebank-officemin" uuid="[1234-5678-901234]"
version=
"1.0">
    <BusinessDocument name="Purchase Order"
nameID="BPID_2"specificationElement=

"$namespace=publicid:org.xCBL:schemas/XCBL35/OrderRequest.xsd$type=OrderReq
uest"

specificationLocation="http://www.flutebank.com/db/schemalibrary/xCBL/XSDL3
.5/

OrderRequest.xsd"/>
    <BusinessTransaction name="Create Order" nameID="BPID_6"

pattern="BusinessTransaction">
       <RequestingBusinessActivity isAuthorizationRequired="true"
            isIntelligibleCheckRequired="false"
isNonRepudiationReceiptRequired="true"
            isNonRepudiationRequired="true" name="Create OrderRequest"
nameID="BPID_7"
                     timeToAcknowledgeAcceptance="P6H"
timeToAcknowledgeReceipt="P2H">
          <DocumentEnvelope businessDocument="Purchase Order"
businessDocumentIDRef=

"BPID_2">
             <Attachment businessDocument="Delivery Instructions"

businessDocumentIDRef="BPID_9"
                     isAuthenticated="true" isConfidential="true"
isTamperProof="true"
                     mimeType="text/xml" name="Delivery Notes"
nameID="BPID_10"/>
          </DocumentEnvelope>
       </RequestingBusinessActivity>
       <RespondingBusinessActivity isAuthorizationRequired="true"

isIntelligibleCheckRequired="false"
               isNonRepudiationReceiptRequired="false"
isNonRepudiationRequired="true"
               name="OrderResponse" nameID="BPID_8"
timeToAcknowledgeReceipt="P2H">
          <DocumentEnvelope businessDocument="PO Acknowledgement"

businessDocumentIDRef="BPID_3"
                     isAuthenticated="false" isConfidential="false"
                     isPositiveResponse="true"   isTamperProof="false"/>
          <DocumentEnvelope businessDocument="PO Rejection"
businessDocumentIDRef=
"BPID_4"
                     isAuthenticated="false" isConfidential="false"
                     isPositiveResponse="false" isTamperProof="false"/>
       </RespondingBusinessActivity>
    </BusinessTransaction>

<BinaryCollaboration name="Firm Order" nameID="BPID_11">
       <InitiatingRole name="buyer" nameID="BPID_12"/>
       <RespondingRole name="seller" nameID="BPID_13"/>
       <BusinessTransactionActivity businessTransaction="Create Order"

businessTransactionIDRef="BPID_6"
                     fromAuthorizedRole="buyer"
fromAuthorizedRoleIDRef="BPID_12"
                     isConcurrent="false" isLegallyBinding="false"
name="Create Order"
                     nameID="BPID_14" timeToPerform="P1D"
toAuthorizedRole="seller"
                     toAuthorizedRoleIDRef="BPID_13"/>
 <Start toBusinessState="Create Order" toBusinessStateIDRef="BPID_14"/>
  <Success conditionGuard="Success" fromBusinessState="Create Order"

fromBusinessStateIDRef="BPID_14"/>
 <Failure conditionGuard="AnyFailure" fromBusinessState="Create Order"

fromBusinessStateIDRef="BPID_14"/>
</BinaryCollaboration>
</ProcessSpecification>
```

# 8 Partner Profiles and Agreements

For an organization to communicate with a business partner, it needs to know what the other end is capable of doing. We looked at how a business process can be modeled as business *collaboration.* The Collaboration-Protocol Profile describes an organization's functional and technical capabilities, such as:

- The business processes in which it can participate
- The roles played in the processes (e.g., seller, shipper)
- The transport protocols (e.g., HTTP, SMTP)
- The messaging protocols (ebXML messaging)
- The security aspects (digital signatures, etc.)

The concept of a Collaboration-*Protocol* Profile is a bit of a misnomer, since it doesn't refer to a protocol (as in networking protocol) but describes the information necessary for another partner to engage in a collaboration with it.

A CPP is an XML document that is governed by the schema defined in the ebXML collaboration protocol profile and agreement specifications. All partners register their CPP documents in an ebXML registry, so that other partners can discover them, understand the supported processes, and make choices where necessary (e.g., to HTTP or SMTP) on their side. Listing 7.2 shows a sample CPP used by OfficeMin.

**Listing 7.2: The CPP document**

```
<?xml version="1.0" encoding="UTF-8"?>
<CollaborationProtocolProfile
xmlns="http://www.ebxml.org/namespaces/tradePartner"
    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" >
    <PartyInfo>
       <PartyId>urn:www.officemin.com</PartyId>
       <PartyRef xlink:href="/" xlink:type="simple"/>
       <CollaborationRole id="CR1">
          <ProcessSpecification xlink:href="flutebank-officemin"
                                 name="Firm Order" xlink:type="simple"
version="1.0"/>
          <Role xlink:href="flutebank-officemin#seller" name="seller"
xlink:type=

"simple"/>
          <ServiceBinding channelId="C1" packageId="P1">
             <Service type="uriReference">/flutebank-officemin/Firm
Order/seller

</Service>
          </ServiceBinding>
       </CollaborationRole>
       <Certificate certId="CRT1">
          <ds:KeyInfo/>
       </Certificate>
       <DeliveryChannel channelId="C1" docExchangeId="DE1"
transportId="T0">
          <Characteristics authenticated="false" authorized="false"
confidentiality=

"false"
                        nonrepudiationOfOrigin="false"
nonrepudiationOfReceipt="false"
                        secureTransport="false" syncReplyMode="none"/>
       </DeliveryChannel>
       <Transport transportId="T0">
          <SendingProtocol version="1.0">HTTP</SendingProtocol>
          <ReceivingProtocol version="1.0">HTTP</ReceivingProtocol>
          <Endpoint type="allPurpose"
uri="http://MACHINEB:80/bindpartner/servlet/
```

```
BindMessageRouter"/>
       </Transport>
       <DocExchange docExchangeId="DE1">
          <ebXMLBinding version="1.0">
             <ReliableMessaging deliverySemantics="OnceAndOnlyOncet"

idempotency="false"
                        messageOrderSemantics="NotGuaranteed">
                <Retries>0<</Retries>
                <RetryInterval>0</RetryInterval>
                <PersistDuration>P</PersistDuration>
             </ReliableMessaging>
          </ebXMLBinding>
       </DocExchange>
    </PartyInfo>
    <Packaging id="P1">
       <ProcessingCapabilities generate="true" parse="true"/>
       <SimplePart id="SP0" mimetype="text/xml"/>
    </Packaging>
</CollaborationProtocolProfile>
```

Figure 7.5 shows the key elements of the CPP. These are:

- PartyInfo. Specifies the organization described in the CPP.
- PartyId. A unique identifier, such as a D-U-N-S code, or an industry-specific identifier, such as an airline carrier code.
- PartyRef. Describes the business partner. It can be a URL or point to an item in the repository.
- CollaborationRole. Describes the business process supported and the roles derived from the BPSS XML document (see Listing 7.1).
- Certificate. Defines the digital certificate, such as an X.509 certificate, used by the organization for nonrepudiation or authentication.
- DeliveryChannel. Defines the transport and message protocols organization supports.
- Transport. Describes the details of the messaging transport protocol, such as type, version, and endpoint.
- DocExchange. Describes the semantics of the messaging service, such as how the message is encoded, retries, deliveries, digital envelope for encryption, and namespaces.

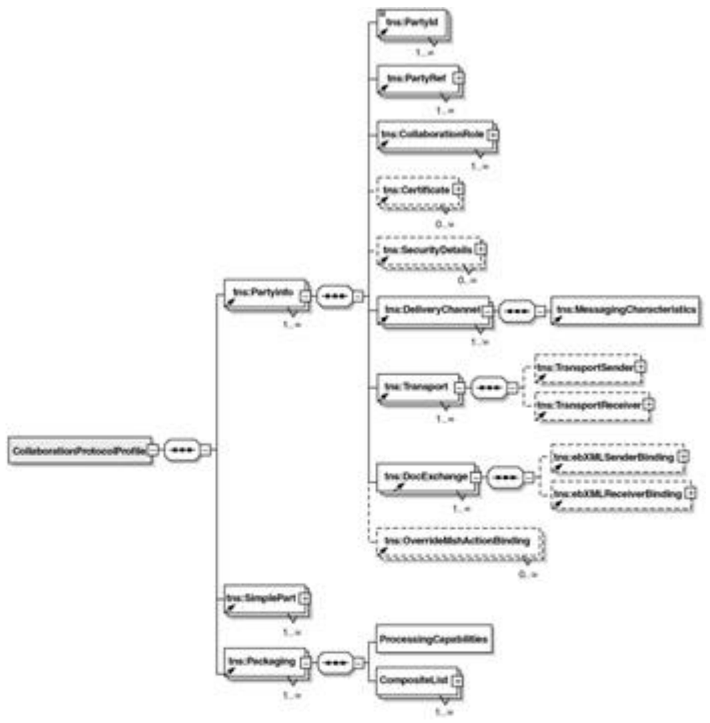


**Figure 7.5:** The XML elements in a Collaboration-Protocol Profile

A CPP is an XML document that defines the capabilities of a particular organization. An organization can be represented by multiple CPP documents.

The second aspect of collaboration deals with how CPP documents for two organizations intersect. The Collaboration-Protocol Agreement defines the system level agreement for data interchange between partners, in the sense that it narrows down a subset from what both partners *can* support to what both partners *will* actually support in the exchange. A good example is the transport protocol. OfficeMin may specify HTTP and SMTP in its Collaboration-Protocol Profile, but Flute Bank can do only HTTP. The Collaboration-Protocol Agreement specifies that the ebXML exchange between the two organization will occur using the messaging service over HTTP, based on the requirements for business processes that both partners mutually agree upon.

As Figure 7.6 shows, the CPA serves, in essence, as a sort of service level agreement that, once agreed to by both parties, can be enforced by the ebXML systems on both ends of the communication. Listing 7.3 shows a sample CPA between OfficeMin and Flute Bank.

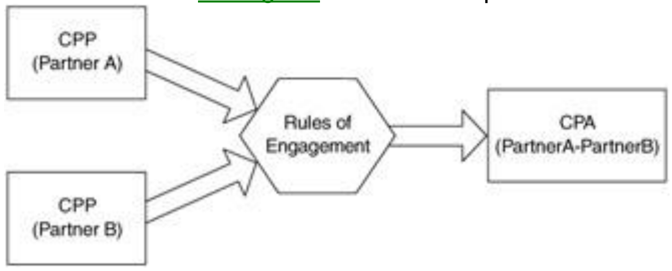


**Figure 7.6:** A Collaboration-Protocol Agreement is agreed upon based on the Collaboration-Protocol Profile documents

**Listing 7.3: The CPA document**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CollaborationProtocolAgreement SYSTEM
"http://ebxml.org/project_teams/trade_
                                                          partner/cpp-
cpa-v1_0.dtd">
```

```
<CollaborationProtocolAgreement cpaid="flute-officemin-cpa" xmlns="http://

www.ebxml.org/namespaces/tradePartner"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <Status value="proposed"/>
    <Start>2001-05-15T17:08:03.062</Start>
    <End>2032-09-27T09:16:06.124</End>
    <ConversationConstraints concurrentConversations="100"
invocationLimit="10000"/>
    <PartyInfo>
       <PartyId>urn:www.officemin.com</PartyId>
       <PartyRef xlink:href="" xlink:type="simple"/>
       <CollaborationRole id="CR1">
          <ProcessSpecification xlink:href="flutebank-officemin" name="Firm
Order"
                              xlink:type="simple" version="1.0"/>
          <Role xlink:href="flutebank-officemin#seller" name="seller"
xlink:type=

"simple"/>
          <ServiceBinding channelId="C1" packageId="P1">
             <Service type="uriReference">/flutebank-officemin/Firm
Order/seller

</Service>
          </ServiceBinding>
       </CollaborationRole>
       <Certificate certId="CRT1">
          <ds:KeyInfo/>
       </Certificate>
       <DeliveryChannel channelId="C1" docExchangeId="DE1"
transportId="T1">
          <Characteristics authenticated="false" authorized="false"
                           confidentiality="false"
nonrepudiationOfOrigin="false"
                           nonrepudiationOfReceipt="false"
secureTransport="false"
                           syncReplyMode="none"/>
       </DeliveryChannel>
       <Transport transportId="T1">
          <SendingProtocol version="1.0">HTTP</SendingProtocol>
          <ReceivingProtocol version="1.0">HTTP</ReceivingProtocol>
          <Endpoint type="allPurpose"
uri="http://machine2:80/bindpartner/servlet/

BindMessageRouter"/>
```

```
       </Transport>
       <DocExchange docExchangeId="DE1">
          <ebXMLBinding version="1.0">
             <ReliableMessaging deliverySemantics="BestEffort"
idempotency="false"
                                  messageOrderSemantics="NotGuaranteed">
                <Retries>0</Retries>
                <RetryInterval>0</RetryInterval>
                <PersistDuration>P</PersistDuration>
             </ReliableMessaging>
          </ebXMLBinding>
       </DocExchange>
    </PartyInfo>
    <PartyInfo>
       <PartyId>urn:www.flutebank.com </PartyId>
       <PartyRef xlink:href="" xlink:type="simple"/>
       <CollaborationRole id="CR2">
          <ProcessSpecification xlink:href="flutebank-officemin" name="Firm
Order"
                                  link:type="simple" version="1.0"/>
          <Role xlink:href="flutebank-officemin#buyer" name="buyer"
xlink:type=

"simple"/>
          <ServiceBinding channelId="C2" packageId="P1">
             <Service type="uriReference">/flutebank-officemin/Firm
Order/buyer

</Service>
          </ServiceBinding>
       </CollaborationRole>
       <Certificate certId="CRT2">
          <ds:KeyInfo/>
       </Certificate>
       <DeliveryChannel channelId="C2" docExchangeId="DE2"
transportId="T2">
          <Characteristics authenticated="false" authorized="false"
                           confidentiality="false"
nonrepudiationOfOrigin="false"
                           nonrepudiationOfReceipt="false"
secureTransport="false"
                           syncReplyMode="none"/>
       </DeliveryChannel>
       <Transport transportId="T2">
          <SendingProtocol version="1.0">HTTP</SendingProtocol>>
          <ReceivingProtocol version="1.0">HTTP</ReceivingProtocol>
          <Endpoint type="allPurpose"
uri="http://machineA:80/bindpartner/servlet/

BindMessageRouter"/>
       </Transport>
       <DocExchange docExchangeId="DE2">
          <ebXMLBinding version="1.0">
             <ReliableMessaging deliverySemantics="BestEffort"
idempotency="false"
                                messageOrderSemantics="NotGuaranteed">
                <Retries>0</Retries>
                <RetryInterval>0</RetryInterval>
                <PersistDuration>P</PersistDuration>
             </ReliableMessaging>
          </ebXMLBinding>
       </DocExchange>
    </PartyInfo>
    <Packaging id="P1">
       <ProcessingCapabilities generate="true" parse="true"/>
       <SimplePart id="SP0" mimetype="text/xml"/>
    </Packaging>
</CollaborationProtocolAgreement>
```

A CPA is an XML document that describes the agreement on the business conversation between two partners based on their CPP documents.

Figure 7.7 shows the key XML elements of a Collaboration-Protocol Agreement:

- Status. One partner generates a Collaboration-Protocol Agreement and offers it to the other for approval. The status element describes the stage of agreement the CPA has reached between them. It can take only the discrete values of proposed, agreed, and signed.
- Start and End. The Start and End elements represent the beginning and end of the period during which this Collaboration-Protocol Agreement is active.
- ConversationConstraints. This optional element describes the number of *conversations* that may be held under this CPA and the number that may be held concurrently.
- PartyInfo. This is the same as the PartyInfo in the Collaboration-Protocol Profile and is used to describe the information for each partner. Note that there must be two and only two PartyInfo elements in a CPA, because a CPA is between two business partners.
- PackagingInfo. Describes the ebXML (SOAP) message headers and the attachment and its associated properties for security, MIME content, namespaces, and so on.
- Signature. Describes the digital signature as per the XML-DSIG specifications and namespace (at *www.w3.org/2000/09/xmldsig#*).

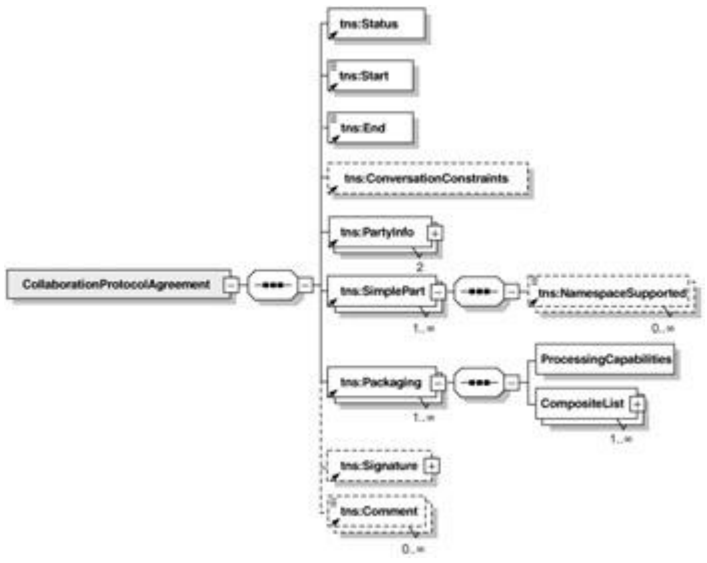


**Figure 7.7:** The XML elements in a Collaboration-Protocol Agreement

JWSDL JSR-157, lead by Sybase and Cyclone Commerce, is a work in progress to provide a standard set of Java APIs for representing and manipulating CPP/CPA documents.

Having taken a look at the BPSS and CPP-CPA, let us redraw Figure 7.2 with more detail in it to elaborate on the design-time and runtime aspects of this collaboration. This is shown in Figure 7.8. Clearly, the registry is a central interaction point between the two partners. Let us now look at this in more detail.

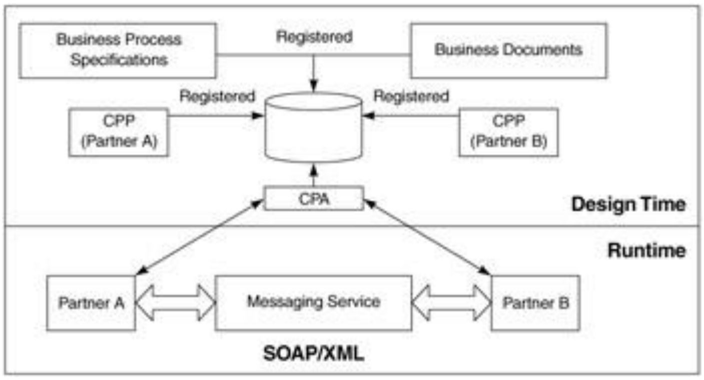


**Figure 7.8:** Design-time and runtime components in detail

# 9 ebXML Registry-Repository Service

In Chapter 6, we talked about registries and their role in a business exchange. In this chapter, we will focus on the ebXML registry service and model. An ebXML registry is similar in concept to the UDDI registry but much broader in scope. In general, a registry is an important block in a business collaboration, because it serves as a central point where an organization can describe itself and its services, business semantics, and processes for other partners to retrieve.

The ebXML a registry is composed of two different concepts-*registry* and *repository.* Because of this, the ebXML registry is often referred to as the *reg-rep.* A registry is what we discussed when we were talking about UDDI, in that a registry stores information *about* items, not the items themselves. A repository, on the other hand, is where the items are physically stored; it serves as a database that is exposed by the services of a registry. Together, they provide key functions, including discovery-storage of information and discovery-storage of information assets. This is the fundamental difference between UDDI and ebXML.

The ebXML registry service is described by two specifications that separate the static and dynamic structures of the registry into two discrete, object-oriented views:

- The *ebXML Registry Information Model* (RIM) describes the registry's static representation in terms of a model or the blueprints for the registry's logical design.
- The *ebXML Registry Services Specification* describes the registry's dynamic structure in terms of its interfaces and API.

ebXML registry architecture is specified using UML notation and can be implemented in any programming language. This chapter gives you an overview of what is involved in the *Registry Information Model*. We will take a closer look at RIM and registry services in Chapter 12.

The Registry Information Model describes in terms of metadata the structures and relationships that can be stored in the registry. It is not a database schema or the registry content; it is simply an object-oriented roadmap of the data. To understand this further, recall the structures discussed about UDDI; namely, the tModels, businessEntity,businessService, and so on. That was the information model for UDDI. The ebXML information model is actually richer than the UDDI model and also more intuitive in terms of its terminology, as Figure 7.9 shows.

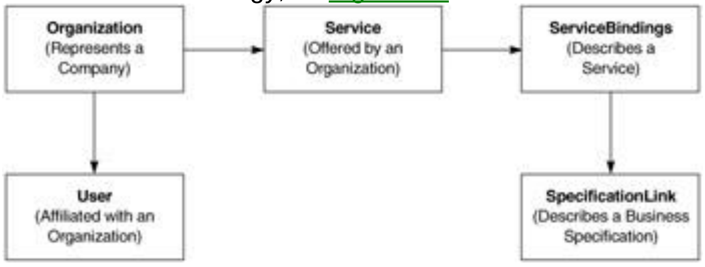


**Figure 7.9:** Relationships in the Registry Information Model

Figure 7.10 shows a more detailed view of the information model. Organizations have Users and Service objects that may be classified based on a ClassificationScheme. All items that represent stored metadata in the registry are subclasses of a RegistryObject. A RegistryObject can have AuditableEvents and Classificationsassociated with it.

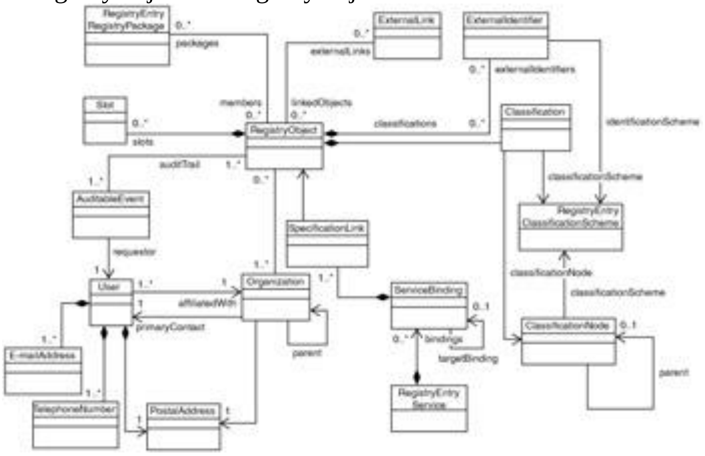


**Figure 7.10:** ebXML Registry Information Model (source- ebXML RIM specifications)

The ebXML Registry Services Specification defines the registry service in abstract terms, using:

- A set of interfaces that must be exposed by the registry
- The set of operations that must be supported in each interface
- The parameters and responses that must be supported by each operation

These interfaces and operations are not defined in a particular programming language but are specified in abstract terms and use XML to describe the interactions. The ebXML registry is abstracted using the notion of a LifeCycleManager and a QueryManager interface, as Figure 7.11 shows. The LifeCycleManager is responsible for exposing operations relating to the creation and management of registry objects defined in the RIM. The QueryManager interface specifies operations for querying the registry and the underlying content in the repository. The client is abstracted using a RegistryClient interface that the registry can use for callbacks.

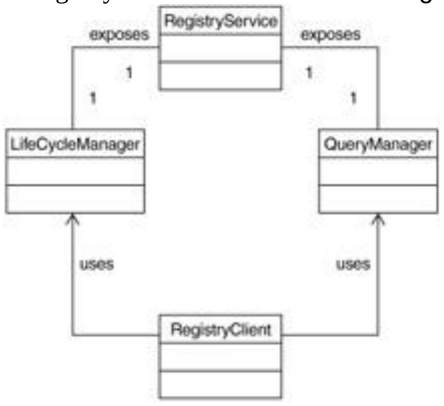


**Figure 7.11:** Registry interfaces

To understand how the abstractions are used, let us look at an example. The LifeCycleManager defines an operation called SubmitObjects, which clients can use to submit data to the registry, as Figure 7.11 shows. The operation takes a SubmitObjectRequest as an argument, and the registry responds by invoking the onResponse method in theRegistryClient. Each of these abstractions (SubmitObjectsRequest, RegistryResponse etc) is defined using an XML schema. The registry service itself can be implemented in any programming language, as long as the interface supports this schema.

---

The registry service interface schema is defined at *www.oasis-open.org/committees/regrep/documents/2.0/schema/rs.xsd*.

---

The implementation has to expose the services of the registry using concrete protocol bindings and a wire protocol, like any other Web service. The ebXML Registry Services Specification defines two bindings. The implementation is free to use either or both:

- SOAP bindings using the HTTP protocol, shown in Figure 7.12

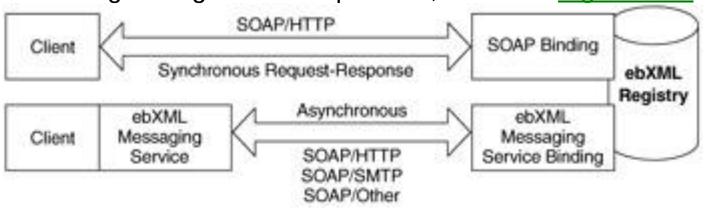


**Figure 7.12:** Registry bindings

- ebXML Messaging Service binding

The SOAP/HTTP bindings specify the services using a document-literal binding and work similar to UDDI. The ebXML messaging service, covered in the next section, facilitates asynchronous SOAP/XML messaging between two points. When using this binding for the interaction, clients can realize the benefits of the messaging service (e.g., reliability, asynchronous invocation) but need to go though an ebXML messaging implementation on their side.

The registry service is itself a Web service. In Chapter 5, we described how a WSDL service description can be separated into abstract and concrete representations. The ebXML registry uses this concept to break up the registry service into abstract and concrete descriptions:

- The abstract representation of the ebXML registry service describes its operations and messages using the WSDL at *www.oasis-open.org/committees/regrep/documents/2.0/services/Registry.wsdl*.
- The concrete description of the registry describes the bindings to SOAP using the WSDL at *www.oasis-open.org/committees/regrep/documents/2.0/services/RegistrySOAPBinding.wsdl*.

We will break off the topic of the ebXML registry at this point, hoping to have provided sufficient information without too much detail. We revisit ebXML registries in Chapter 12, when we talk about working with registries from Java. As a final thought, we will address an often-asked question by architects on the comparison between UDDI and ebXML registries.

## 10 UDDI or ebXML

Functionally, the ebXML registry is a superset of the UDDI registry, even though UDDI may seem to have a greater momentum at present. Both specifications are now under the umbrella of the OASIS consortium and offer overlapping functionality and feature sets. A bit of confusion often surrounds the two, and in this section, we will compare and contrast them.

**Information Model** In Chapter 6, we looked at the UDDI information model, and in Figure 7.10, we looked at the ebXML information model. Both models came about as a result of identifying use cases for the registry, which differ in their fundamental approach. The UDDI use cases are focused more on publishing organization and service information, whereas the ebXML use cases seek to address the broader issues of B2B collaboration. This difference is reflected in the models.

The forte of the UDDI Registry Information Model is the focus on business and service *listings.* However, this model does not address some of the interactions involved in the collaboration. The ebXML RIM is described in more intuitive terms and supports the *storage* of arbitrary content, represented by a RegistryObject. This is a powerful concept, because the physical content (such as XML documents describing the WSDL, Collaboration-Protocol Profile, Collaboration-Protocol Agreement, and Business Process Specification Schema) can be stored and retrieved from the registry.

- The UDDI Registry Information Model supports a fixed set of relationships between its primary entities (the company and the service). ebXML supports the creation of arbitrary associations between any two objects, which map well to the UML associations that may be described in a business model. So, for example, two XML schemas can be stored in the ebXML registry, and an association representing a version change (i.e., supersede) can be indicated.

**Taxonomies and Classifications** In Chapter 6, we showed how a taxonomy is represented by a tModel in a UDDI registry. tModels are overloaded, in that they have other uses (e.g., in Chapter 12, we show how a tModel can represent the WSDL service interface). UDDI inherently supports three taxonomies (NAICS, UNSPSC, and ISO 3166), and users may submit new tModels to proxy for additional taxonomies. The issue is that since there is no way to clearly represent this taxonomy, there is no way for clients to either browse or validate the usage. ebXML, on the other hand, supports internal and external taxonomies and browsing from the client.

**Registry Queries** The inquiry API for both registries has been derived from the information model, which is derived from the core use cases. This is reflected in how the client can query the registry. The UDDI client API (in XML) is simple and allows for search on a business, service, or tModel. The ebXML registry, on the other hand, supports the fixed set of queries on key objects in the information model as well as declarative queries, where SQL syntax may be retrieved to search, query, and retrieve content.

**Security** UDDI supports a simple password-based authentication scheme over HTTP for the publishing API. HTTPS is supported in UDDI v. 2.0. The ebXML registry, on the other hand, uses digital certificates for authentication and maintains audit trails on content (e.g., who changed specific objects, and when).

Internal and external taxonomies, queries for UDDI and declarative queries on ebXML, password-based user authentication in UDDI and digital-certificate-based authentication using JAXR are all covered further in Chapter 12.

**Protocol Support** A registry exposes itself with an XML interface and API, which need to be accessed using some transport protocol. UDDI supports the use of these XML APIs over SOAP/HTTP and SOAP/HTTPS. The ebXML registry supports the use of these API using SOAP/HTTP directly, as UDDI does, or the ebXML messaging service. The messaging service, discussed in the next section, is layered on SOAP 1.1 with attachments and HTTP. This also allows the client to interact asynchronously with the service and the service to asynchronously respond to the client.

The options provided by the ebXML registry give business partners greater flexibility in the mechanism and infrastructure to use.

**WSDL for Registry** The registry itself is a Web service and is capable of being described by a WSDL. The ebXML registry clearly describes its service interface using a WSDL, as mentioned earlier. This capability of describing the UDDI registry using WSDL has been added in UDDI v. 3.0.

An organization can expose an ebXML registry and can even register this as a Web service in UDDI- see www.ebxml.org/specs/rrUDDI.pdf.

## 11 ebXML Message Service

Successful collaboration between business partners must include a mechanism to handle the flow of requests from one end to another and take context (e.g., who is the requestor, committed service levels, privacy, prior fulfillment, and personalization) into account before determining the flow. This is the runtime component identified in Figure 7.8. The ebXML messaging service specification defines a reliable means to exchange business messages using SOAP without relying on proprietary technologies or solutions. It is a critical piece in the ebXML architecture, because it solves many of the problems inherent in the EDI messaging systems we described earlier in this chapter.

The messaging service defines two aspects of messaging:

1. How the message is packaged.
2. What different components are present in the messaging system to support the transport and processing of the message.

In Chapter 4, we talked about the concept of vertical extensibility in SOAP, in which other specifications extend SOAP by specifying schemas for the SOAP header. This is precisely what the ebXML messaging service does, by defining a packaging scheme using the SOAP envelope, as Figure 7.13 shows.

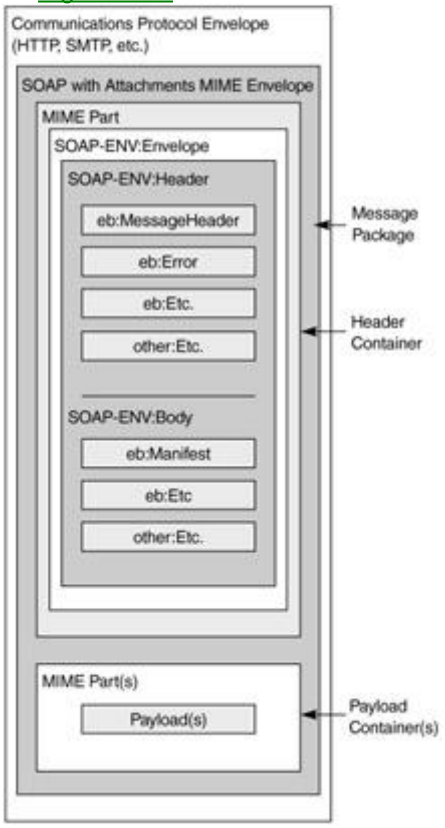


**Figure 7.13:** The ebXML message (source- ebXML Message Service Specification)

The message package is based on SOAP with attachments and consists of a MIME-multipart structure. The package contains the SOAP envelope and the payload, which may be any business document in any format. The specifications define the ebXML specific headers extensions contained in the header element of the SOAP message and processed by the messaging server on the receiving side. The MessageHeader is added to the message with a mustUnderstand=1 attribute, so that it is processed by the other end and contains the following subelements:

- From and To. Describe the PartyId and Role of the two parties involved in the message exchange. The PartyId and Role are based on the Collaboration-Protocol Agreement the parties have agreed upon for this exchange.
- CPAId. The Collaboration-Protocol Agreement between the two partners. It is a URI that points to the location in the reg-rep of the Collaboration-Protocol Agreement or another URL where the CPA may be mutually accessed.
- ConversationId. Collaborating business partners will typically exchange a group of messages creating a conversation. This element identifies the message as belonging to a particular conversation.
- Service. Refers to the service targeted by this message.
- Action. Refers to a particular task or action in the targeted service.
- MessageData. Contains metadata, such as the message ID, a timestamp, and time to live information about the message.
- DuplicateElimination. Tells the receiver that it should check if the message is a duplicate.
- Description. Provides a human-readable description of the message.

For example, Listing 7.4 shows the SOAP message containing the ebXML message headers from Flute Bank to OfficeMin. What is not shown is the actual message payload, the business document. The payload would be part of the MIME attachment in the SOAP message.

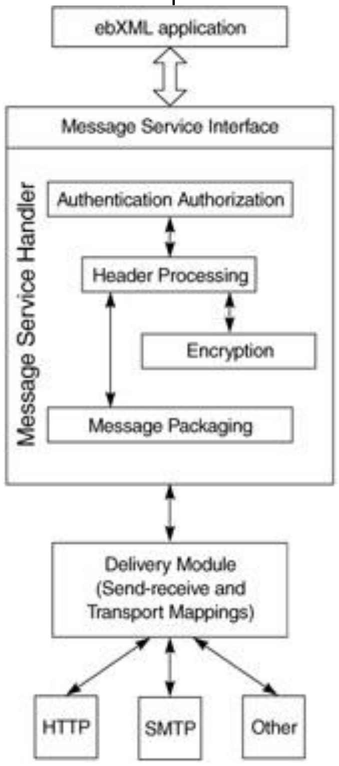
**Listing 7.4: The ebXML SOAP message**

```
<?xml version="1.0" encoding="UTF-8"?>

<soap-env:Envelope xmlns:soap-
env="http://schemas.xmlsoap.org/soap/envelope/">

<soap-env:Header>

    <eb:MessageHeader
xmlns:eb=http://www.ebxml.org/namespaces/messageHeader

        eb:version="1.0" soap-env:mustUnderstand="1">

       <eb:From>

            <eb:PartyId eb:type="URI">

                http://www.flutebank.com/ordersupplies

            </eb:PartyId>

       </eb:From>

       <eb:To>

           <eb:PartyId eb:type="URI">

               http://www.officemin.com/processorders

           </eb:PartyId>

       </eb:To>

    <eb:CPAId>

             http://www.flutebank.com/agreements/agreementwithofficemin.xml

    </eb:CPAId>

       <eb:ConversationId>www.flute.com/orders/829202</eb:ConversationId>

       <eb:Service eb:type="">purchaseorderservice</eb:Service>

       <eb:Action>Purchaseorder</eb:Action>

       <eb:MessageData>

          <eb:MessageId>
```

```
                    89fcfba5-fac8-4ddd-94b1-ba74339d42de
             </eb:MessageId>
             <eb:Timestamp>1031591106992</eb:Timestamp>
         </eb:MessageData>
     </eb:MessageHeader>
</soap-env:Header>
<soap-env:Body/>
</soap-env:Envelope>
```

The ebXML messaging specifications define three logical architecture levels between the business application and the network protocols that carry this SOAP message, as Figure 7.14 shows:

- The message service interface
- The message service handler
- The transport interface


ebXML application
Message Service Interface
Message Service Handler
Authentication Authorization
Header Processing
Encryption
Message Packaging
Delivery Module (Send-receive and Transport Mappings)
HTTP
SMTP
Other
**Figure 7.14:** ebXML messaging system modules

The message service interface is the portion of the service that applications interact with. It forms an application interface for business applications to invoke the message handler. The message service handler (MSH) is the software system that handles the message and contains basic services, such as authentication, header parsing, encryption, and message packaging. The messaging specifications define these abstract areas of functionality, how they must act, and how the service as a whole must act. For example, the specifications define how authentication information can be included in the ebXML message headers using digital signatures and how the service must provide reliable messaging functionality.

The specifications do not define any particular implementation of these abstractions. What they define is the behavior of the messaging service and the functional areas it must support. For example, the specifications do not define an API for the message service interface. A vendor who provides the messaging software will provide the API to interact with the service. (For Java applications, this is where JAXM comes in, which we cover in Chapter 11). However, what the specifications define is the bindings with HTTP and SMTP, as Figure 7.15 shows. If two vendors conform to these bindings for the SOAP message, which is an XML document and an attachment, they will not need to bother about what technology the other is implemented in. The message package, the transport, the format on the wire, and the communication, which together form an *ebXML reliable messaging protocol,* are standardized by the specifications.

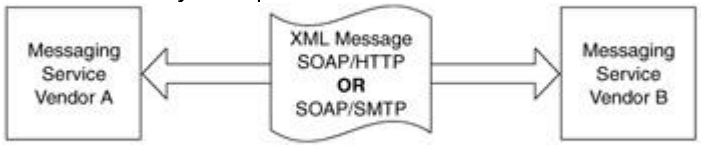

Messaging Service Vendor A
XML Message SOAP/HTTP OR SOAP/SMTP
Messaging Service Vendor B

**Figure 7.15:** ebXML messaging

Earlier, we mentioned how the Collaboration-Protocol Agreement forms an agreement between two business partners, and the messaging service is required to enforce it at runtime. Typically, the implementation will be configured to point to the location of the CPA between the parties in an ebXML registry-repository. When the service is invoked to send the message, it will validate that the message conforms to the agreement between the two partners.

As with registries, we will again pause in our discussion on the messaging service and resume it in Chapter 11, where we talk about implementing XML messaging.

## The ebXML Messaging Service

by Pim van der Eijk
March 18, 2003

The ebXML Messaging Service specification (ebMS) extends the SOAP specification to provide the security and reliability features required by many production enterprise and e-business applications. As an OASIS Open standard, ebMS is a mature specification, supported by a variety of commercial and open source software implementations. The interoperability of many of these implementations has been demonstrated in a number of ongoing projects internationally. This makes ebMS a strong complement or even alternative to other web service specifications.

### Web Services and the ebXML Messaging Service

Concerns about security and reliability -- as Rich Salz's recent article, "Securing Web Services" made clear -- are among the biggest impediments to widespread adoption of web services. This is particularly true for business-to-business integration applications but also for application integration projects within the enterprise. Several ongoing efforts, in both industry standards organizations and ad hoc vendor consortia, are looking to create additional specifications beyond the core web services specifications to remedy this.

In this article we examine a specification that deserves serious attention in this context, namely, the the ebXML Messaging Service specification (ebMS). ebMS, endorsed in August 2002 as anOASIS standard, adds security and reliability extensions to SOAP and aligns e-business messaging with other e-business integration standards (like trading partner agreements and business process specification). After three years of development, ebMS has become a mature specification. While it is designed to support the ebXML framework, ebMS is a standalone specification that can be used independently of other ebXML specifications. Prospective users evaluating message service or web services technology have a choice of commerical and open source implementations of ebMS, many of which have been verified for interoperability in implementation pilots.

### Web Services, ebXML and ebMS

The term "web services" is often defined narrowly as describing applications built using SOAP, WSDL and UDDI, but there are good reasons to interpret the term more broadly. Web services may include a larger range of applications which are built from a larger range of specifications and designed to meet a broader range of requirements. Taking this view, one way to classify web services applications is to distinguish the following types:

- *Integration web services* that aim at exposing application interfaces or business services to remote systems. An integration web service may provide direct RPC-style, synchronous or asynchronous access to a single application. It may also provide access to several backend systems and provide simple routing and rule-based transformations, but even then it only implements some business logic. The majority of the business logic is still provided by the backend applications.

- *Collaborative web services* that aim at managing shared business processes between business partners. A collaborative web service often needs to manage business transactions with many different business partners. These business transactions will be role-based, subject to sequencing rules (choreography), and tied to specific message formats. Choreography and message formats are often defined by vertical industry standards bodies. A collaborative web service will need a system to manage and validate these transactions and to maintain some state information to correlate asynchronous responses and check deadlines. It will often be configured for each specific partner according to service-level agreements.

Integration web services are a natural extension of RPC application access or Enterprise Application Integration (EAI) beyond an organization's boundary. They are often referred to as *simple web services*, although EAI is not necessarily a simpler problem than e-business integration. Similarly, collaborative web services are sometimes called *complex Web Services*. While simple web services -- like Google search, stock quote retrieval, or parcel tracking -- have either limited requirements for security and reliability or none at all, this is certainly not the case for other integration web services. For example, as web services start to be used for financial applications like credit card verification, security clearly becomes a major issue. The security and reliability features designed primarily for collaborative web services are becoming important for integration web services as well.

The ebXML framework for e-business was a joint initiative of UN/CEFACT and OASIS. The first phase of ebXML was executed as a joint project between 1999 and 2001. After successful delivery of a series of specifications, reports, and whitepapers in May 2001, further development of the ebXML specification has continued in the two organizations. The complete set of specifications in the ebXML framework is too complex to summarize in a few pages (see Professional ebXML Foundations for an introduction); fortunately, the specifications have been designed to be modular and independently useful. This is certainly true for the ebXML messaging specification, currently maintained by the OASIS ebXML Messaging Service Technical Committee. In August 2002, version 2.0 of this specification was endorsed by the OASIS members as an OASIS Open standard.

The collaborative web services category as defined here obviously covers the business-to-business integration scenarios that ebXML was designed for from the start. In fact, if the ebXML project were to start today rather than in 1999, it probably would have been called something like "ebWS", rather than ebXML. The various specifications that make up the ebXML framework and their implementations are of great potential interest to developers of collaborative web services applications and of integration web services with high-end security or reliability requirements. Taking this view, let's see how the ebXML Messaging Service builds on the core SOAP specification, and how it provides additional functionality.

## Architecture and Requirements of ebMS

The ebXML Message Service defines both a message format and the behavior of software that exchanges ebXML messages. That software can be implemented as a standalone messaging software product, or it can be part of the functionality of e-business integration products or application servers. Conceptually, the ebXML messaging service can be thought of as having three layers:

1. an abstract service interface, which applications use to access the messaging service;
2. message service handler functions; and
3. a mapping to underlying transport services (for example, HTTP or SMTP).

The following diagram provides an overview of the various functions provided by the message service handler.

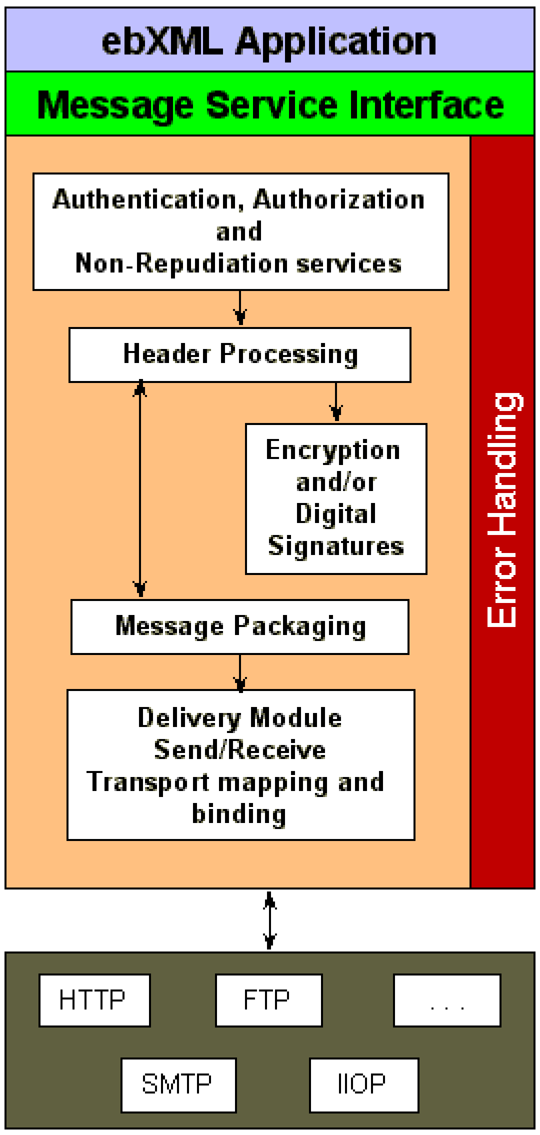

ebXML Application
Message Service Interface
Authentication, Authorization and Non-Repudiation services
Header Processing
Encryption and/or Digital Signatures
Message Packaging
Delivery Module Send/Receive Transport mapping and binding
Error Handling
HTTP
FTP
. . .
SMTP
IIOP

**Figure 1: Components in the ebXML Message Handler**

Functional requirements for the message service include:

- *header processing* (building a header from parameters passed from the invoking application);
- *header parsing* (the reverse of header processing),
- *security services;* (creating and verifying digital signatures, authentication and authorization);
- *reliable messaging* (message persistence, retries, error notification, receipt acknowledgment);
- *packaging* payloads into a SOAP with Attachments message package; and
- *error handling*.

The ebMS specifies protocol bindings to HTTP and SMTP and allows implementations to support other protocols. For instance, one European car manufacturer uses ebMS over IBM MQSeries in an after-sales support application which exchanges information between a call center and dealer CRM systems. Interoperability is less crucial since this is an internal application.

The ebMS is also payload neutral: it imposes no restrictions on the content it carries, which can be based on any XML vocabulary, EDI, or binary data. Content can also be referenced by hyperlinks only and not transported itself, which is an important advantage that makes ebMS suitable for a very wide range of applications.

## Anatomy of an ebXML message

The ebXML Message Service is layered over SOAP with Attachments. SOAP with Attachments (also discussed in another Rich Salz XML.com column) embeds a SOAP envelope as a first part of a MIME container, which contains all information needed for routing and thus allows for very efficient message processing. Other content (possibly multi megabyte EDI or UBL messages) can be appended as additional MIME parts and passed on to the receiving application. The general structure of an ebXML message is displayed in the following diagram:

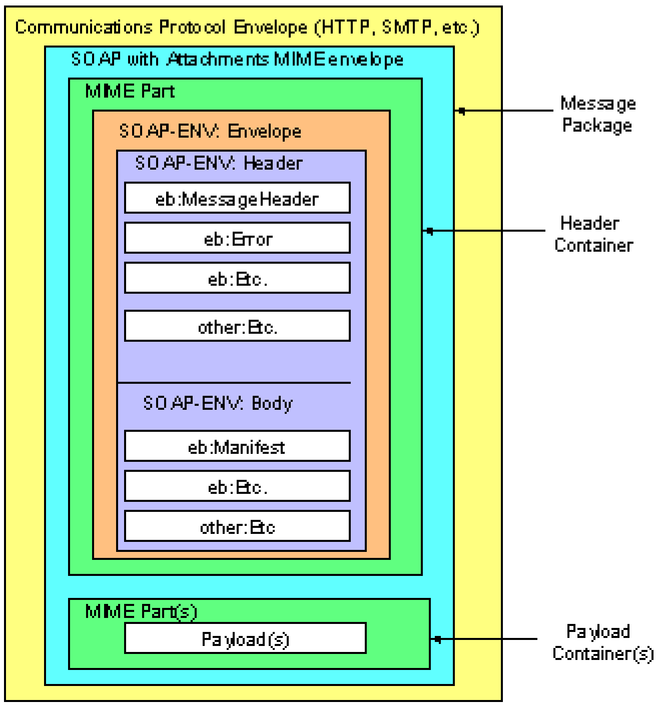


**Figure 2: Message Anatomy**

**The ebXML Messaging Service**
by Pim van der Eijk

Let's walk through the structure of a specific message, sent on 23 January 2003 from a prototype ebMS implementation developed by Seeburger, one of the participants in a European ebXML interoperability pilot project. This message corresponds to test case 101 from the test set defined for this project. The full message includes three payload documents: a small XML document, a 20 MB EDIFACT message, and a 14 MB PDF file.

The following listing shows the first part of the MIME container, which contains a `<SOAP:Envelope>` element with ebXML extensions. We've omitted the content of the `<SOAP:Header>` and`<SOAP:Body>` elements for clarity; we'll get back to them shortly. We've also omitted all MIME content after the content separator for the first payload element.

This listing has been wrapped for readability. The original listing is available here: listing1.txt

```
SOAPAction: "ebXML"
Content-Type: multipart/related; type="text/xml";
boundary="--20030121-144231102-0017----"; start="ebXMLHeader"

----20030121-144231102-0017----
Content-Type: text/xml
Content-Transfer-Encoding: 7bit
Content-ID: ebXMLHeader
Content-Length: 2757
```

```
<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
   <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/
ebxml-msg/schema/msg-header-2_0.xsd"
     xsi:schemaLocation="http://www.oasis-open.org/committees/
ebxml-msg/schema/msg-header-2_0.xsd http://www.oasis-open.org/
committees/ebxml-msg/schema/msg-header-2_0.xsd">
      <!-- ... SOAP header content, see below ... -->
   </SOAP:Header>
   <SOAP:Body xmlns:eb="http://www.oasis-open.org/committees/
ebxml-msg/schema/msg-header-2_0.xsd"
     xsi:schemaLocation="http://www.oasis-open.org/committees/
ebxml-msg/schema/msg-header-2_0.xsd http://www.oasis-open.org/
committees/ebxml-msg/schema/msg-header-2_0.xsd">
      <!-- ... SOAP body content, also see below ... -->
   </SOAP:Body>
</SOAP:Envelope>

----20030121-144231102-0017----
```

The `<SOAP:Header>` element must contain an `<eb:MessageHeader>` element that provides the main routing information. The `<eb:MessageHeader>` has subelements which identify the message's sender (`<eb:From>`) and recipient (`<eb:To>`). ebXML does not define its own party identification scheme; it's designed to leverage identification schemes such as EAN.UCCGlobal Location Numbers, which provides a unique, unambiguous, and efficient identification of locations. The following examples uses a simple, ad hoc identification scheme.

Listing wrapped for readability, original: listing2.txt.

```
<SOAP:Header
   xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/
schema/msg-header-2_0.xsd"
   xsi:schemaLocation="http://www.oasis-open.org/committees/
ebxml-msg/schema/msg-header-2_0.xsd
     http://www.oasis-open.org/committees/ebxml-msg/schema/
msg-header-2_0.xsd">
   <eb:MessageHeader eb:id="messageHeader" eb:version="2.0"
      SOAP:mustUnderstand="1">
      <eb:From>
         <eb:PartyId>http://ebxml.seeburger.de:80/ebXMLMSH?async?
response</eb:PartyId>
         <eb:PartyId eb:type="name">SEEBURGER AG, Christopher
           Frank (c.frank@seeburger.de)</eb:PartyId>
         <eb:Role>Requestor</eb:Role>
      </eb:From>
      <eb:To>
         <!-- eb:PartyID omitted, similar to preceding eb:From -->
         <eb:Role>Responder</eb:Role>
      </eb:To>
      <!-- continuation follows ... -->
```

The optional `<eb:Role>` element identifies the role of the sender and participant in an `<eb:Action>` invoked on a `<eb:Service>`. In ebXML, the semantics of this service and action can be defined in a Collaboration Protocol Agreement (CPA), which sender and recipient and have agreed to and which is identified here in the `<eb:CPAId>` element. CPAs are XML documents defined in a separate ebXML-related OASIS Standard, Collaboration Protocols and Agreements. A CPA in turn references a separate business process specification document (expressed in a Business Process Specification Schema) that provides partner-independent descriptions of e-business collaborations.

We don't need to understand these other ebXML specifications here: the test service of the receiving ebXML message handler (`urn:services:ipp:tcipp101`) is instructed to return a message containing exact copies of the payload documents contained in this message (`Reflector`). That reply message will have the same `<eb:ConversationId>` and can use the value`20030121-144230993-0016` in a `<eb:RefToMessageId>` element to allow the original sender to correlate request and response.

```
      <!-- continued from preceding ... -->
      <eb:CPAId>cpa_00</eb:CPAId>
      <eb:ConversationId>cpa_00-20030121-144230993-0015</eb:ConversationId>
      <eb:Service>urn:services:ipp:tcipp101</eb:Service>
      <eb:Action>Reflector</eb:Action>
      <eb:MessageData>
         <eb:MessageId>20030121-144230993-0016</eb:MessageId>
         <eb:Timestamp>2003-01-21T13:42:30</eb:Timestamp>
         <eb:TimeToLive>2003-01-26T13:42:30</eb:TimeToLive>
      </eb:MessageData>
      <eb:Description xml:lang="en-US">Initial ebXML message for
             initialization of Test Step 1 in IPP Test Case
TCIPP101.</eb:Description>
   </eb:MessageHeader>
</SOAP:Header>
```

The `<SOAP:Body>` element contains `<eb:Manifest>` elements which reference MIME parts attached to the ebXML message or data elsewhere on the Web. In this case, there are three payload documents.

Listing wrapped for readability, original available as listing3.txt.

```
<eb:Manifest eb:id="manifest" eb:version="2.0">
   <eb:Reference eb:id="pl-0000" xlink:type="simple"
      xlink:href="cid:mspld_00.xml">
      <eb:Description xml:lang="en-US">Simple
        small XML file.</eb:Description>
   </eb:Reference>
   <eb:Reference eb:id="pl-0001" xlink:type="simple"
      xlink:href="cid:mspld_02.edifact_ORDERS_UN_D93A_small">
      <eb:Description xml:lang="en-US">Large EDIFACT
        file (20MB).</eb:Description>
   </eb:Reference>
   <eb:Reference eb:id="pl-0002" xlink:type="simple"
      xlink:href="cid:mspld_03.pdf_papiNetVR200_small">
      <eb:Description xml:lang="en-US">Large binary
        file (15MB)</eb:Description>
   </eb:Reference>
</eb:Manifest>
```

The following shows the very first part of the second one, with id `<mspld_02.edifact_ORDERS_UN_D93A_small>`, referenced from the `<eb:Reference>` element in the `<eb:Manifest>` that has the `eb:id` attribute set to the value `pl-0001`. This payload component is a fragment of a very large EDIFACT message.

Listing wrapped for readability, original available as listing4.txt.

```
----20030121-144231102-0017----
Content-Type: text/plain
Content-Transfer-Encoding: 7bit
Content-ID: <mspld_02.edifact_ORDERS_UN_D93A_small>
Content-Length: 368

UNB+UNOA:2+ABSENDER9012345678901234567890ABCDE:CD-A:
WEITERLEITNG-A+EMPFAENGER:CD-E:WEITERLEITNG-E+940812:
```

```
0235+T940812023504A++0026-Anwendung'
UNH+M940812023504A+ORDERS:D:93A:UN:EAN007'
BGM+220+5211229'
DTM+002:940815'
NAD+SU+4004353000000::9'
NAD+BY+4306517005214::9'
LIN+1++4004353054099:EN'
QTY+21:9'
UNS+S'
UNT+51+M940812023504A'
UNZ+3+T940812023504A'
```

This example shows how ebMS builds on SOAP with Attachments to transport arbitrary data between two ebXML Message Service handlers. It also shows how additional ebXML-related business information is encoded, allowing message associations, ongoing conversations, business partner agreements, and business process definitions. And yet ebMS is a self-contained specification that can be deployed without an additional ebXML infrastructure.

## Security

One of the areas in which ebMS extends SOAP is *security*. The ebMS specification distinguishes nine different security features. Implementations of ebMS are required to implement one of these, *persistent digital signatures*, to provide non-repudiation of origin. The precise configuration of these features is partner-dependent and can be agreed between partners in a CPA or otherwise. These signatures are to be applied to the message by the sending message service handler and are constructed using a separate specification, the XML-Signature Sytax and Processing specification. This specification defines a `<Signature>` element that can be added to the `<SOAP:Header>`. The use of XML Digital Signatures in the ebXML messaging specification is similar to the use in the more recent draft Web Services Security (WSS) specification. As this has also recently been discussed at XML.com (by Rich Salz, again!), we will limit ourselves to pinpointing some differences:

- WSS only allows one `<Signature>`. EbMS allows multiple `<Signature>` elements and specifies that the first of these represents the digital signature as signed by the *From Party MSH*. However, it does not define the purpose of any other signatures.
- In WSS, the signature is applied to the `<SOAP:Body>`. The ebMS specification describes a way to sign the complete `<SOAP:Envelope>` and any relevant attached business documents, omitting the `Signature` element from the `<SOAP:Header>` and any information not intended for the final recipient but only for intermediate nodes.

All security features other than persistent digital signing by the sending MSH depend upon specifications that were still under development at the time of ebMS's completion. Thus, interoperability between parties using these features is not guaranteed. The specification describes the following security features.

- Non repudiation of receipt can be expressed by returning an acknowledgment message signed using the certificate of the recipient.
- XML encryption can be used to provide *persistent confidentiality* of payload containers and/or the SOAP header.
- Persistent authorization could leverage the Security Assertion Markup Language (SAML), recently adopted as an OASIS Open Standard.
- A secure network protocol like TLS (SSL) can be used to provide non-persistent authentication, confidentiality, and authorization.

## Reliability

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. The *Collaboration Protocols and Agreements* specification that accompanies ebMS provides a standard way to encode this type of message service configuration information rigorously.

Reliable messaging requires a message service handler to detect duplicate messages. For example, a sender may resend a message that has been delivered to a recipient and for which an acknowledgment message has been created and returned but not yet delivered to the original sender. Duplicate message elimination and the requirement of resilience to system failure generally require ebMS handlers to maintain persistent copies of sent messages.

## Interoperability of ebMS Implementations

Before the ebXML messaging service, those of us who needed secure and reliable exchange of XML-based messages over the public Internet or an Intranet had few alternatives to message queuingsoftware. Although the Java Messaging Service (JMS) offers a standard programming interface to message queuing products, the message formats and wire protocols used by these products are proprietary, requiring sender and recipient to use implementations from the same vendor or to use JMS-to-JMS bridges. One promise of the ebXML Messaging Service is to provide some of the benefits of message queuing products, while offering users a choice of ebMS implementations, provided by different vendors, including open source implementations. Thus, the interoperability of those implementations needs to be verified. There are currently several ongoing projects addressing this topic. We will mention four of them here.

The OASIS Implementation, Interoperability and Conformance TC addresses the requirements for ebXML interoperability, with an initial practical focus on ebMS. The committee is chartered to provide a conformance plan, a set of reference implementation guidelines, a set of base line interoperability tests, and guidelines and direction for third-party creation of conformance laboratories. Many participants in this TC are also active in other interoperability projects. One of the deliverables of this TC is a test framework that allows automation of interoperability testing of ebMS implementations. The ebXML IIC is currently finalizing its Basic Interoperability test suite, which aims at bringing together the various ebMS interoperability test initiatives, thus providing a common interoperability platform. The IIC has also developed the concept of a Deployment Template, first developed for ebXML Messaging. This is an important tool in defining interoperability at the business or usage level. It has been used by EAN-UCC to specify its deployment and implementation guidelines (which include the use of GLN mentioned earlier) in a more systematic way, which is easier to map to ebMS requirements, for users and vendors.

The Drummond Group has been conducting a series of ebMS interoperability tests sponsored by the Uniform Code Council. The most recent series was recently concluded and involved ebMS implementations provided by bTrade, CDC, Cyclone Commerce, eXcelon, Fujitsu, GXS, IPNet Solutions, Sun Microsystems, Sybase, Sterling Commerce, Tibco, and XML Global.

The European ebXML Interoperability Pilot Project is an ongoing joint project of CEN ISSS and OASIS. The project tested the interoperability of ebMS implementations developed by some participants and developed a realistic demonstration scenario for ebXML applications. The project is to deliver a (partial) implementation of the demonstration scenario using software implementations developed by project participants, which will be demonstrated at the upcoming XML Europe 2003 ebXML conference. The demonstration scenario has been developed with input from people active in industry associations like CIDX, EAN International, and Eurofer. The demonstration uses steel industry e-business integration scenarios from an actual ebMS implementation by Steel 24-7, a marketplace for the steel industry that today uses a production implementation of ebMS version 1.0. The current design of the demonstration scenario involves steel and automotive business partners and messages drawn from two XML vocabularies and EDIFACT. OASIS members Seeburger, Software AG, Sonic Software (Excelon), and Sun Microsystems, as well as individuals, contribute to the interoperability effort.

Adoption of ebXML is very strong in Asia. A recent interoperability project involving Fujitsu Limited, Hitachi Limited, NEC Corporation, Infoteria Corporation, and Nippon Telegraph and Telephone Corporation was successfully completed in October 2002. Several similar projects are ongoing.

## Conclusion

ebMS is a complete and mature specification that provides useful extensions to SOAP. It has been implemented by a wide range of vendors, and many of these implementations have been verified for interoperability. People looking for web services to provide a secure and reliable messaging service for e-business applications or enterprise application integration today should take a close look at the specification and its implementations.