Applicazioni Java – ebXML

Laurea Specialistica in Ingegneria Informatica per la Gestione d'Azienda
Corso di Sistemi Informativi per le Aziende

sistema di tracciabilità agroalimentare

Ing. Mario G.C.A. Cimino

Dicembre 2004

Terminologia

- ✓ **business** = attività di tipo *(i)* industriale oppure *(ii)* commerciale
- ✓ **processo di business** = unità di suddivisione funzionale del business = qualsiasi metodo o sistema per *(i)* fare un prodotto/servizio oppure *(ii)* acquistarlo/venderlo. Es. approvigionamenti, pagamenti, spedizioni.
- ✓ **specifica** = come la tecnologia deve funzionare: cosa dovrebbe fare (metodi) e non fare (vincoli)
- ✓ **standard** = specifica approvata dalla maggioranza di una comunità del settore. Es. VHS (standard **aperto** per il formato dei dati video sui nastri magnetici per i videoregistratori), MPEG, SOAP, XML,…
  … WINDOWS (standard **proprietario** "de facto" per la gestione delle risorse HW dei personal computer, con la maggioranza del mercato mondiale).

Concetti introduttivi

- ✓ Lo sviluppo di un' applicazione *Java* distribuita che adoperi *ebXML* comporta l'utilizzo di diverse componenti API (*Application Programming Interface*) e l'integrazione con soluzioni di terze parti (*Database Management System, File System, Web Server, Firewall,…*).

- ✓ La tecnologia ***Java***, i cui componenti sono eseguiti sulla *Java Virtual Machine*, permette di realizzare applicazioni ***platform independent***.

- ✓ *Java* fornisce anche un valido supporto alla *serializzazione* degli oggetti, ossia alla loro codifica in un formato lineare che possa viaggiare in uno stream. Uno stream può essere a sua volta collegato ad un file per rendere persistente l'oggetto, oppure ad un socket per trasmettere dati ad un processo residente su un host remoto.

- ✓ Tuttavia, questa codifica si basa su un formato binario legato a *Java*. Ciò significa la necessità di N interfacce per N applicazioni "non Java" e – dualmente – altrettante interfacce per ogni formato legato ad altri framework di sviluppo o applicativi.

- ✓ Il linguaggio ***XML***, derivato da una famiglia di linguaggi di markup nati per codificare i documenti web, consente di strutturare i dati in documenti *plain text* e trasportarli su *HTTP*.

- ✓ Lo standard XML del W3C permette a tali documenti di essere processati da applicazioni di qualsiasi natura, definendone la struttura mediante *XML Schema*, realizzando quindi applicazioni ***device independent***.

- ✓ Un documento XML può descrivere entità di qualsiasi livello, es. relative alle modalità di espletamento del rapporto tra due attori del web in un processo di business. Un documento XML, al pari di un protocollo di intesa, relativamente ad uno specifico scenario, può definire il modo in cui devono essere adoperati altri documenti XML.

- ✓ Gli scenari organizzativi dei rapporti tra attori del web possono essere raggruppati in classi, quindi è possibile pensare ad uno standard per fornire un supporto (entità UML, schemi XML) alla progettazione ed alla formulazione di protocolli di intesa (es. disciplinari tecnici di procedura della filiera).

- ✓ Tale standard, ***electronic business XML-based*** (***ebXML***), tende al raggiungimento della ***interoperabilità fra le organizzazioni***.

Modello dei dati e documenti XML

- ✓ Supponiamo di avere un sistema di tracciabilità con il semplice modello dei dati raffigurato di seguito in UML.

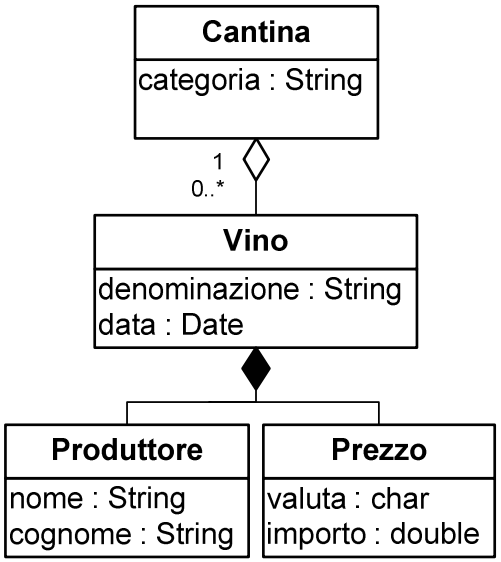


Fig.1 Diagramma delle classi

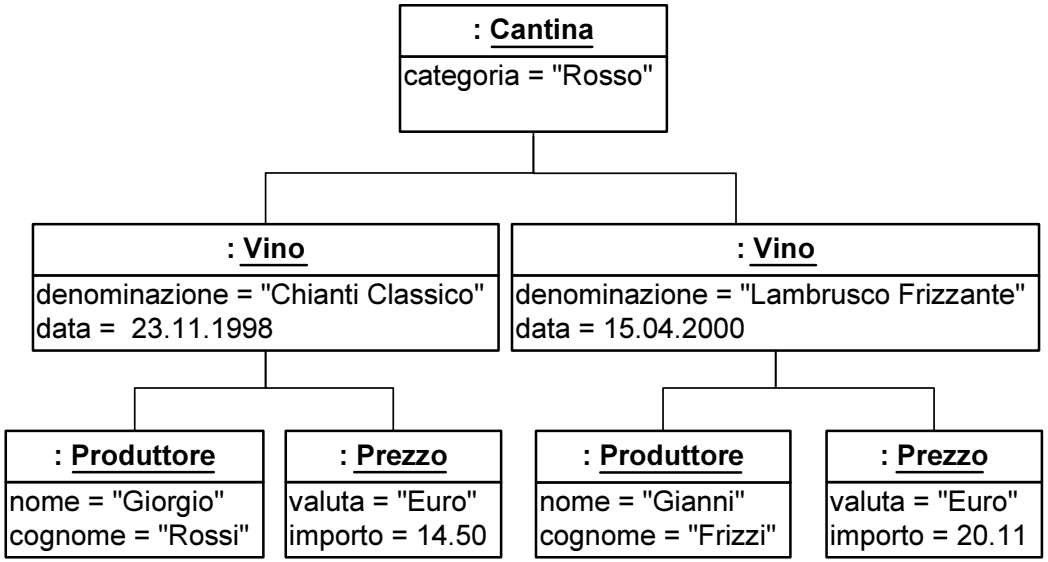


Fig.2 Diagramma degli oggetti (istanze)

- ✓ Una *Cantina* contiene diverse tipologie di *Vino* di una medesima categoria. Ciascun tipo di *Vino*, caratterizzato da una *denominazione* ed una *data* di produzione, si compone di un *Produttore* ed un *Prezzo*.

- ✓ Un *Produttore* è caratterizzato da *nome* e *cognome*, mentre il *Prezzo* dalla *valuta* e dall' *importo*.

- ✓ Vogliamo esprimere le entità di Fig.2 in un documento XML che ne mantenga la struttura gerarchica.
- ✓ Ci sono diversi modi di rappresentare gli oggetti di Fig.2 in *documenti delle istanze XML*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Formato n.1 -->
<Cantina>
  <categoria>Rosso</categoria>
  <Vino>
    <denominazione>Chianti Classico</denominazione>
    <data giorno = "23" mese = "11" anno = "1998"/>
    <Produttore formato = "nome cognome">Giorgio Rossi</Produttore>
    <Prezzo importo = "14.50 Euro"/>
  </Vino>
  <Vino>
    <denominazione>Lambrusco Frizzante</denominazione>
    <data giorno = "15" mese = "04" anno = "2000"/>
    <Produttore formato = "nome cognome">Gianni Frizzi</Produttore>
    <Prezzo importo = "20.11 Euro"/>
  </Vino>
</Cantina>
```

`<? Processing instruction?>` direttiva per l'applicazione

`formato` é un *attributo* dell' *elemento* **`Produttore`**

Fig. 3 – Un possibile documento XML delle istanze

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- file cantina.xml, Formato n.2 -->
<Cantina categoria = "Rosso"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation='cantina.xsd'>
  <Vino>
    <denominazione>Chianti Classico</denominazione>
    <data>1998-11-23</data>
    <Produttore>
     <nome>Giorgio</nome>
     <cognome>Rossi</cognome>
    </Produttore>
    <Prezzo valuta = "Euro" >14.50</Prezzo>
  </Vino>
  <Vino>
    <denominazione>Lambrusco Frizzante</denominazione>
    <data>2000-04-15</data>
    <Produttore>
     <nome>Gianni</nome>
     <cognome>Frizzi</cognome>
    </Produttore>
    <Prezzo valuta = "Euro" >20.11</Prezzo>
  </Vino>
</Cantina>
```

<!--commento-->

Progettazione di una struttura dati XML: attributo o elemento ?

a) nuovo elemento, se il dato è strutturato su linee multiple, o cambia spesso

b) nuovo attributo se il dato è una stringa semplice o numero (e non cambiano spesso), o appartiene a un set di possibilità predefinite.

Fig. 4 – Un altro possibile documento XML delle istanze

✓ Nello scambio di documenti XML, è necessario avere struttura e tipi di dato ben definiti, ossia che i documenti delle istanze siano conformi al medesimo schema. Lo schema è un documento XML, che definisce una classe di documenti XML, i cui

elementi ed attributi appartengono al namespace (ambito di visibilità) identificato da una URI predefinita.

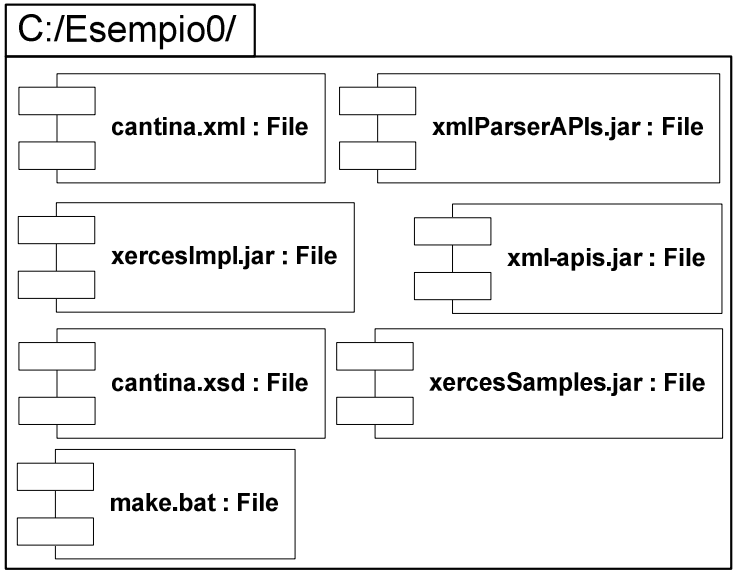
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- file cantina.xsd, Schema per il Formato n.2,  -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
 <xs:element name="Cantina">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Vino" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="categoria" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="Vino">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="denominazione" type="xs:string"/>
    <xs:element name="data"            type="xs:date"/>
    <xs:element ref ="Produttore"/>
    <xs:element ref ="Prezzo"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Produttore">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="nome"       type="xs:string"/>
    <xs:element name="cognome"    type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Prezzo">
  <xs:complexType>
   <xs:simpleContent>
    <xs:extension base="xs:decimal">
     <xs:attribute name="valuta" type="xs:string" use="required"/>
    </xs:extension>
   </xs:simpleContent>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Universal Resource Identifier (URI) è una stringa che identifica simbolicamente una risorsa nel web.
A differenza di una URL (Uniform Resource Locator), digitando una URI nel browser può non esserci alcuna risorsa.

Fig. 5 Un documento della classe (XML Schema) per il Formato n.2

Elaborazione di documenti XML

- ✓ Per validare il documento `cantina.xml` sullo schema `cantina.xsd` occorre un validating XML parser che supporti gli Schema, come il parser open source Xerces (progetto Apache XML). Scritto in Java, il package (http://xml.apache.org/dist/xerces-j/Xerces-J-bin.2.5.0.zip) include un programma a linea di comando chiamato `dom.Writer` (http://xml.apache.org/ xerces-j/domwriter.html).




```
rem make.bat
set CLASSPATH=
java -classpath xmlParserAPIs.jar;
     xercesImpl.jar;xercesSamples.jar;
     dom.Writer -v -s cantina.xml
```

Fig.6 File necessari e relativa composizione

- ✓ Se il documento è valido, DOMWriter farà semplicemente echo sullo schermo, altrimenti segnalerà gli errori (**`dom.Writer -h`** per l'help)

- ✓ *Document Object Model* (DOM) è un modello ad oggetti, definito dal W3C, che consente di manipolare un documento XML costruendo una struttura ad albero i cui nodi sono gli elementi, mediante una Interfaccia di Programmazione delle Applicazioni (API) uguale per tutti i linguaggi di programmazione.

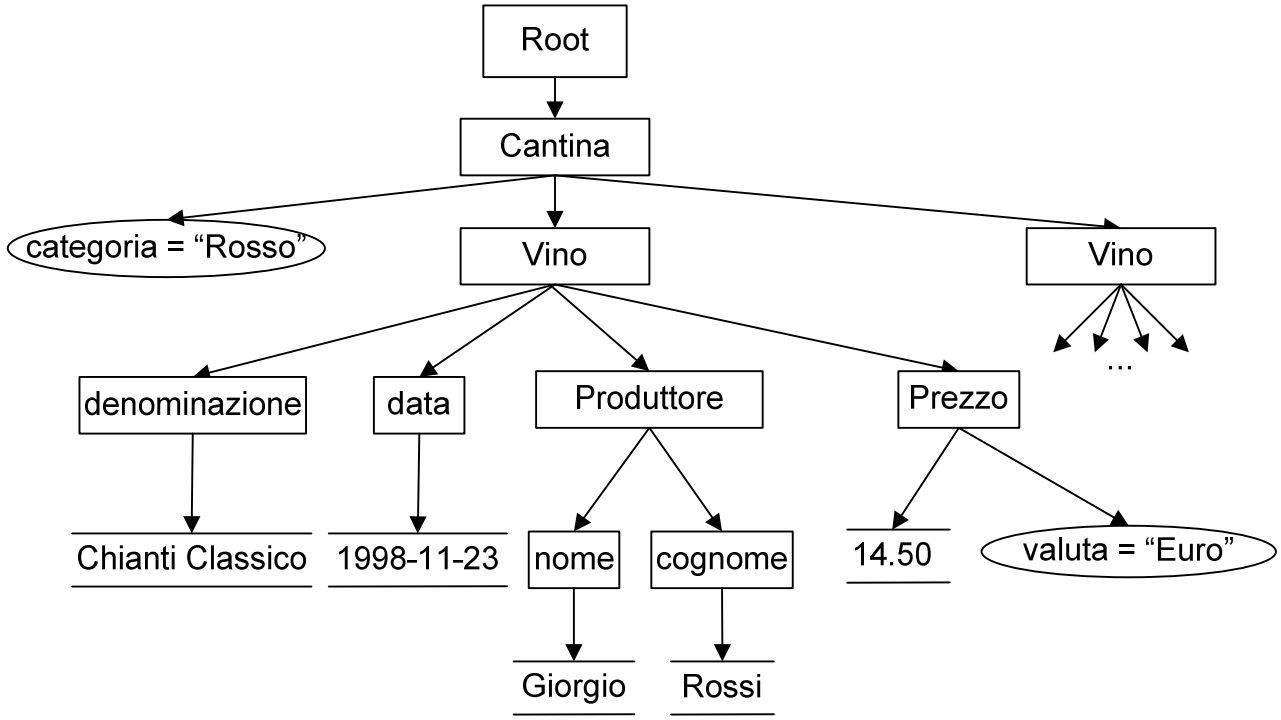


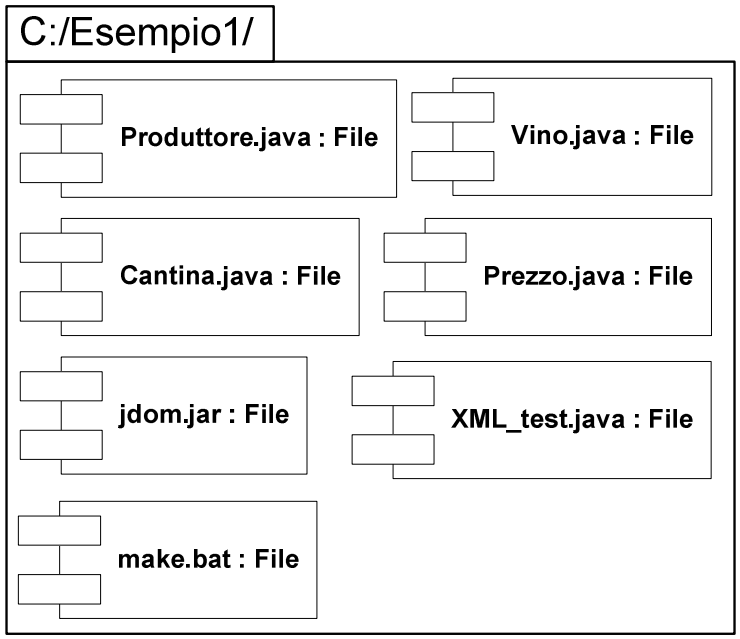
Fig.7 Struttura di parte del documento di Fig.4

- ✓ *Simple API for XML* (SAX) è un altro modello, basato su un paradigma ad eventi, che non costruisce un modello dell’intero documento ma consente un accesso in sola lettura e la definizione di un handler di eventi per segnalare le categorie di interesse da riscontrare nel documento.

- ✓ Es. un documento tipo Word Processor (molto strutturato e dimensioni contenute) si presta ad essere manipolato in DOM, mentre per dati di un'agenda (notevoli dimensioni e struttura poco annidata) si predilige un approccio SAX.

- ✓ Nella piattaforma *Java* i package *javax.xml*, *org.w3c.dom*, *org.xml.sax* forniscono le API generiche per entrambi i modelli.

- ✓ Un'altra interfaccia API, chiamata JDOM, rappresenta un livello più alto del modello DOM del W3C, sviluppato appositamente per gestire documenti XML sfruttando tutte le funzionalità di java che semplificano la programmazione, interfacciandosi anche alle principali implementazioni SAX e DOM (Sun, IBM, Oracle, Xerces, Crimson,...).
- ✓ Una semplice applicazione Java che adopera JDOM si compone come in Fig.8 e si esegue con il comando `make` da shell di Windows. Il package **`jdom.jar`** è disponibile su *www.jdom.org*.




```
rem make.bat
set CLASSPATH=
javac -classpath jdom.jar *.java
java -classpath jdom.jar; XML_test
```

Fig.8 File necessari e relativa composizione

```
// Prezzo.java
import org.jdom.Element;

public class Prezzo {

    private String  valuta = "Euro";
    private double  importo;

    public Prezzo(double importo) {
        this.importo = importo;
    }

    public Element getElement() {
        Element e = new Element("prezzo");
        e.setAttribute("valuta",valuta);
        e.addContent(Double.toString(importo));
        return e;
    }
}

// Produttore.java
import org.jdom.Element;
```

```
public class Produttore {

    private String nome;
    private String cognome;

    public Produttore(String nome, String cognome) {
        this.nome = nome;
        this.cognome = cognome;
    }

    public Element getElement() {
        Element e = new Element("Produttore");
        e.addContent(new Element("nome").addContent(nome));
        e.addContent(new Element("cognome").addContent(cognome));
        return e;
    }
}

// Vino.java
import java.util.Date;
import org.jdom.Element;
import java.text.SimpleDateFormat;

public class Vino {

    private String      denominazione;
    private Date        data;
    private Produttore  produttore;
    private Prezzo      prezzo;

    public Vino(String denominazione, Date data, Produttore produttore, Prezzo prezzo) {
        this.denominazione    = denominazione;
        this.data                   = data;
        this.produttore       = produttore;
        this.prezzo           = prezzo;
    }

    public Element getElement() {
        Element e = new Element("Vino");
        e.addContent(new Element("denominazione").addContent(denominazione));
        e.addContent(new Element("data").addContent(new SimpleDateFormat("yyyy-MM-dd").format(data)));
        e.addContent(produttore.getElement());
        e.addContent(prezzo.getElement());
        return e;
    }
```

```
}

// Cantina.java
import org.jdom.Element;

public class Cantina {

    private String  categoria;
    private Vino[]      vini;

    public Cantina(String categoria, Vino[] vini) {
        this.categoria = categoria;
        this.vini = vini;
    }

    public Element getElement() {
        Element e = new Element("Cantina");
        e.setAttribute("categoria",categoria);
        for (int i=0; i<vini.length; i++)
            e.addContent(vini[i].getElement());
        return e;
    }
}

// XML_test.java;
import java.util.*;
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;

public class XML_test {
    public static void main(String[] args) throws Exception {
        Vino[] vini = { new Vino(   "Chianti Classico",
                                    new GregorianCalendar(1998,10,23).getTime(),
                                    new Produttore("Giorgio","Rossi"),
                                    new Prezzo(14.50)
                                    ),
                        new Vino(   "Lambrusco Frizzante",
                                    new GregorianCalendar(2000,03,15).getTime(),
                                    new Produttore("Gianni","Frizzi"),
                                    new Prezzo(20.11)
                                )
                    };
        Cantina cantina = new Cantina("Rosso", vini);
        Document doc1 = new Document(cantina.getElement());
```

```
        XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
        outputter.output(doc1, new FileOutputStream("./cantina.xml"));

        SAXBuilder builder = new SAXBuilder();
        Document doc2 = builder.build("./cantina.xml");
        System.out.println(outputter.outputString(doc2));

        Element root = doc2.getRootElement();
        Attribute attribute = (Attribute) root.getAttributes().get(0);
        System.out.println("La " + attribute.getName() + " e` " + attribute.getValue());

        List children = root.getChild("Vino").getChildren();
        Element child = (Element) children.get(3);
        System.out.println("Il " + child.getName() + " e` " + child.getText());
        attribute = (Attribute) child.getAttributes().get(0);
        System.out.println("La " + attribute.getName() + " e` in " + attribute.getValue());
    }
}
```

- ✓ Come risultato della esecuzione, viene generato e visualizzato il file `cantina.xml` dopodiché vengono visualizzate alcune informazioni sulla struttura.

```
D:\Esempio1> make

<?xml version="1.0" encoding="UTF-8"?>
<Cantina categoria="Rosso">
  <Vino>
    <denominazione>Chianti Classico</denominazione>
    <data>1998-11-23</data>
    <Produttore>
      <nome>Giorgio</nome>
      <cognome>Rossi</cognome>
    </Produttore>
    <prezzo valuta="Euro">14.5</prezzo>
  </Vino>
  <Vino>
    <denominazione>Lambrusco Frizzante</denominazione>
    <data>2000-04-15</data>
    <Produttore>
      <nome>Gianni</nome>
      <cognome>Frizzi</cognome>
    </Produttore>
```

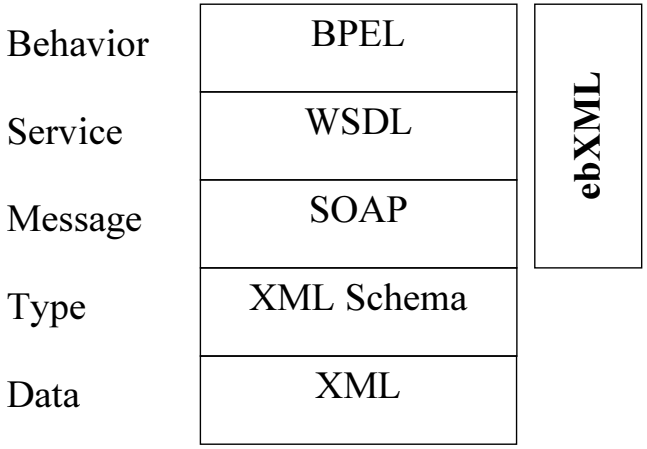
```
    <prezzo valuta="Euro">20.11</prezzo>
  </Vino>
</Cantina>


La categoria e` Rosso
Il prezzo e` 14.5
La valuta e` in Euro
```

ebXML e WS

- ✓ Tecnicamente parlando, due organizzazioni che intendono cooperare in rete hanno bisogno di accordarsi su come invocare i rispettivi servizi/processi di business e come scambiare dati. Entrambi hanno bisogno di avere a comune protocolli, formati e contenuti dei messaggi.
- ✓ Le *business collaboration* si possono racchiudere in tre categorie:
  - *Business information services*: le organizzazioni condividono informazioni, es. le quotazioni dei titoli o le news.
  - *Business integration services*: una organizzazione fornisce servizi integrativi ai clienti es. sistemi di prenotazione, di controllo del credito.
  - *Business transaction services*: due organizzazioni si impegnano in una operazione di business mutuamente vincolante, con obblighi chiaramente definiti, es. acquisto di prodotti, contrattazione di servizi di trasporto. Questa tipologia di servizi è spesso legata a processi di business.

- ✓ Si dice che le organizzazioni cooperano con “legame debole” se nessuna di esse può esercitare controllo sull’altra (es. un acquirente non deve poter bloccare le risorse di un venditore).
- ✓ Per trovare i servizi occorrono dei registri pubblici. Infine occorre stabilire le conseguenze in caso di condizioni di malfunzionamento (es. problemi di comunicazione).
- ✓ *ebXML* si colloca su un livello parallelo rispetto allo stack di tecnologie standard dei *Web Services (WS)*. Entrambi forniscono interoperabilità tecnica attraverso un protocollo non proprietario, ma sono complementari perchè si occupano di diverse categorie di business collaboration.
- ✓ I *WS* sono tipicamente appropriati per *e-business non collaborativo* ossia *business information services* e *business integration services*, mentre i *business transaction service* sono adeguatamente supportati da *ebXML*.



Web Service Standards

SOAP, Simple Object Access Protocol, insieme minimale di convenzioni per lo scambio di dati interpiattaforma, codificati in XML e trasferiti mediante HTTP.

WSDL, Web Service Definition Language, una grammatica XML per descrivere un servizio web come una collezione di punti di accesso in grado di scambiare messaggi secondo un paradigma a *procedure* o a *documenti*.

BPEL, Business Process Execution Language, linguaggio XML-based per definire una composizione di servizi nella forma di coreografie di WS, ossia di un'aggregazione di WS secondo regole di interazione prestabilite. Questo standard consente di formulare i WS come business process

- ✓ L'uso di un WS non richiede un accordo tra richiedente e fornitore, mentre le *business collaboration* previste da ebXML prevedono un *Collaboration Protocol Agreement (CPA)* che definisce i servizi messi a disposizione dalle due controparti (una sorta di interfaccia) sulle quali deve trovarsi un accordo prima di impegnarsi nelle transazioni.

	Web Services	ebXML
Type	Request/response	Collaboration
Communication	RPC-style synchronous communication between tightly coupled services, Document-style asynchronous communication between loosely coupled services	Synchronous, asynchronous communication
Business Service Interface description	WSDL	CPP, CPA (WSDL within CPP, CPA under research)
Protocol and Formats	SOAP, XML	ebXML Message Service (over SOAP), XML, BPSS (as "business" protocol)
Content Standards	None	Recommended Standards (e.g. OAGI BODs)
How to find business partners	UDDI Registry	ebXML Registry (UDDI Registry may point to an ebXML Registry or Registry objects (e.g. CPA))

UDDI, Universal Description, Discovery and Integration, metodo standardizzato, XML-based, per pubblicare e ritrovare informazioni sui WS

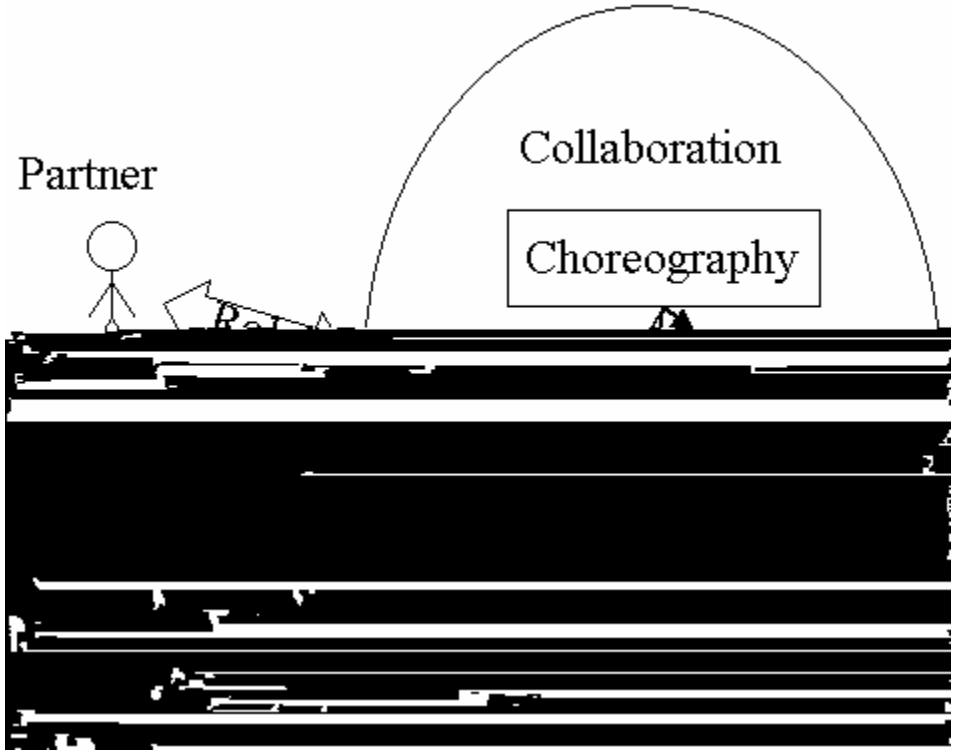
OAGI (Open Applications Group, Inc.) ha sviluppato un vasto insieme di messaggi di business (messaggi scambiati fra le parti, in ebXML) e scenari di integrazione per enterprise application e B2B, senza specificare una architettura di implementazione (implementation framework). ebXML fornisce uno standard per trasportare OAGI **BODs** (Business Object Documents).

- ✓ A differenza dell'interfaccia nel WSDL, un cambio dell'interfaccia nel CPA non influenza lo scambio tecnico dei messaggi, ma invalida solo il CPA. Inoltre, in ebXML anche i problemi a livello di business sono presi in considerazione, nel *Business Process Specification Schema (BPSS)*. Ad esempio, se una parte non risponde entro un predefinito periodo di tempo, il BPSS ritorna allo stato precedente.
- ✓ Negli scenari B2B, ebXML trova consenso per amministrare *enterprise-spanning business transaction services* nel contesto di business collaborativo; invece WS trovano posto in *intra-enterprise integration of back-end systems*.

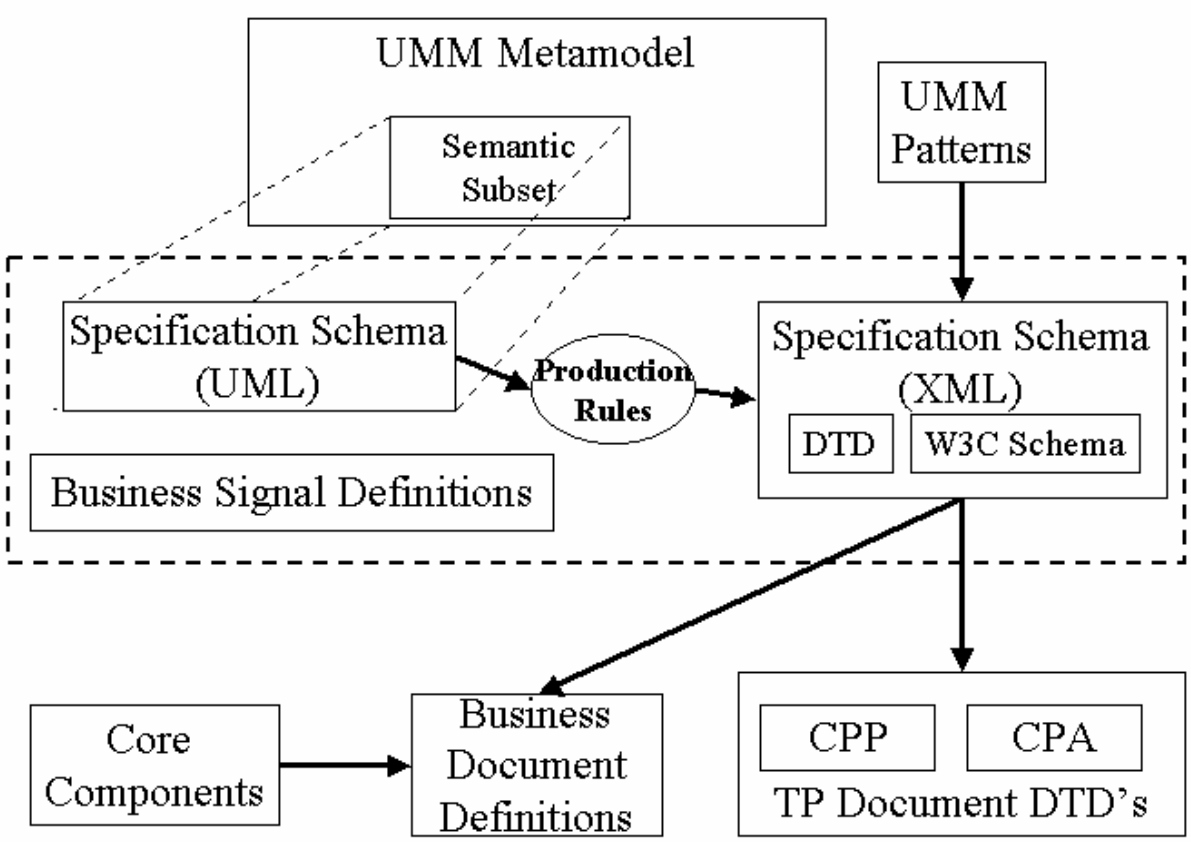
- ✓ In termini di standardizzazione, ebXML è in fase più avanzata di WS. Il primo fornisce una credibile collaborazione di business ed un'automazione di processi inter-enterprise, il secondo fornisce SOAP, WSDL ed UDDI.
- ✓ Secondo gli analisti la tecnologia WS non si diffonderà prima del 2005, quando tutti gli standard relativi avranno raggiunto un pari livello di stabilità. Alla luce dell'esistenza di ebXML, appare prematuro aggiungere ai WS una semantica per le business collaboration.

Entità fondamentali di ebXML

- ✓ *Business process*, la definizione di un *business process model* riguarda i "run time aspects" ossia l'ordine con cui vengono inviati i messaggi in una *business collaboration*, senza considerare la elaborazione dei dati, e definisce i *ruoli* dei partecipanti, cui corrispondono transazioni in un ordinamento stabilito dalla *business transaction coreograhy.* Una *business transaction* consiste in una fase atomica di comunicazione (con un meccanismo di *rollback* in caso di fallimento) in cui avviene lo scambio di documenti.

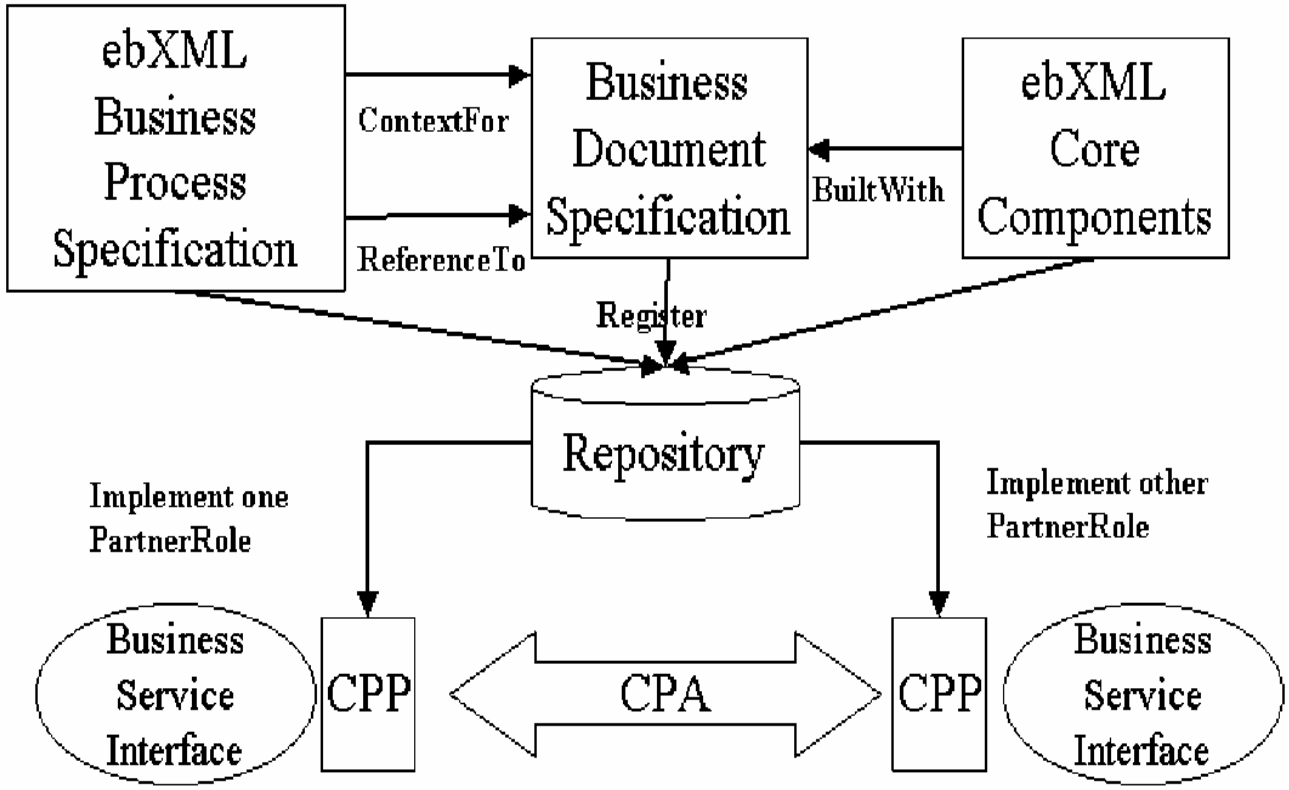


- ✓ **BPSS**, *Business Process Specification Schema* definisce un *business process model* attraverso (i) un diagramma delle classi UML, (ii) uno schema XML, (iii) delle regole di produzione da UML ad XML, (iv) le *Business Signal Definitions* (documenti a livello applicativo, distinti dal protocollo di trasporto di basso livello, che segnalano lo stato corrente delle transazioni)



- ✓ **UMM**, *UN/CEFACT Modeling Methodology*, metodologia per i processi di business e la modellazione consistente dell'informazione. UMM Meta Model è una descrizione di semantiche di business che permette a partner commerciali di inquadrare i dettagli per uno specifico scenario di business.
- ✓ I Business Document sono composti da Business Information Documents riusabili. A basso livello, i Business Information Objects sono composti da **Core Components** riusabili.

- ✓ **CPP**, *Collaboration Protocol Profile*, definisce le potenzialità sia tecnologiche (protocolli di messaggistica e di comunicazione supportati) sia commerciali (quali Business Collaboration supporta) ed i modi in cui una parte può impegnarsi in business elettronico con altre parti.
- ✓ **CPA**, *Collaboration Protocol Agreement*, definisce l'accordo tra le parti, ed è creato mediante elaborazioni e negoziazioni derivanti dall'incrocio di due CPP. Ad esempio: un CPA include solamente quegli elementi che sono comuni o compatibili fra le due parti.



- ***Business Service Interface***: descrive il modo in cui una società è in grado di eseguire le transazioni necessarie nel suo processo di business. Inoltre include le tipologie di messaggi supportati e i protocolli sopra cui questi viaggiano.
- ***Registry****/Repository*: è un contenitore pubblico per modelli di processo, vocabolari e profili dei partner. Può essere implementato con un server centrale.

Esempi

```
<BusinessTransaction name="Create Order">
   <RequestingBusinessActivity name=""
      isNonRepudiationRequired="true"
      timeToAcknowledgeReceipt="P2D"
      timeToAcknowledgeAcceptance="P3D">
      <DocumentEnvelope BusinessDocument="Purchase Order"/>
   </RequestingBusinessActivity>
   <RespondingBusinessActivity name=""
      isNonRepudiationRequired="true"
      timeToAcknowledgeReceipt="P5D">
      <DocumentEnvelope isPositiveResponse="true"
         BusinessDocument="PO Acknowledgement"/>
   </RespondingBusinessActivity>
</BusinessTransaction>
```

- *Esempio di Business Transaction*, in questo esempio ci sono due flussi di documenti e tre segnali di business. Poiché viene richiesto il "non ripudio" la business activity

dovrà conservare i business document nella forma originale in modo che l'informazione di ambedue le parti non possa essere disconosciuta. La richiesta richiede sia la ricevuta che l'accettazione, la risposta solo il riconoscimento di accettazione.

- ✓ "P#D" è uno schema W3C adottato per lo standard ISO8601 (per la rappresentazione della data ed ora in un contesto globale) e significa Period = # Days (a partire dall'invio della richiesta)
- ✓ Una business transaction consiste di una Requesting Business Activity, una Responding Business Activity, 1-2 flussi di documenti tra queste, eventualmente associati ad 1-2 Business Signals di riconoscimento dei flussi.
- ✓ Il segnale di failure viene inviato se ad esempio scade uno dei timeout.

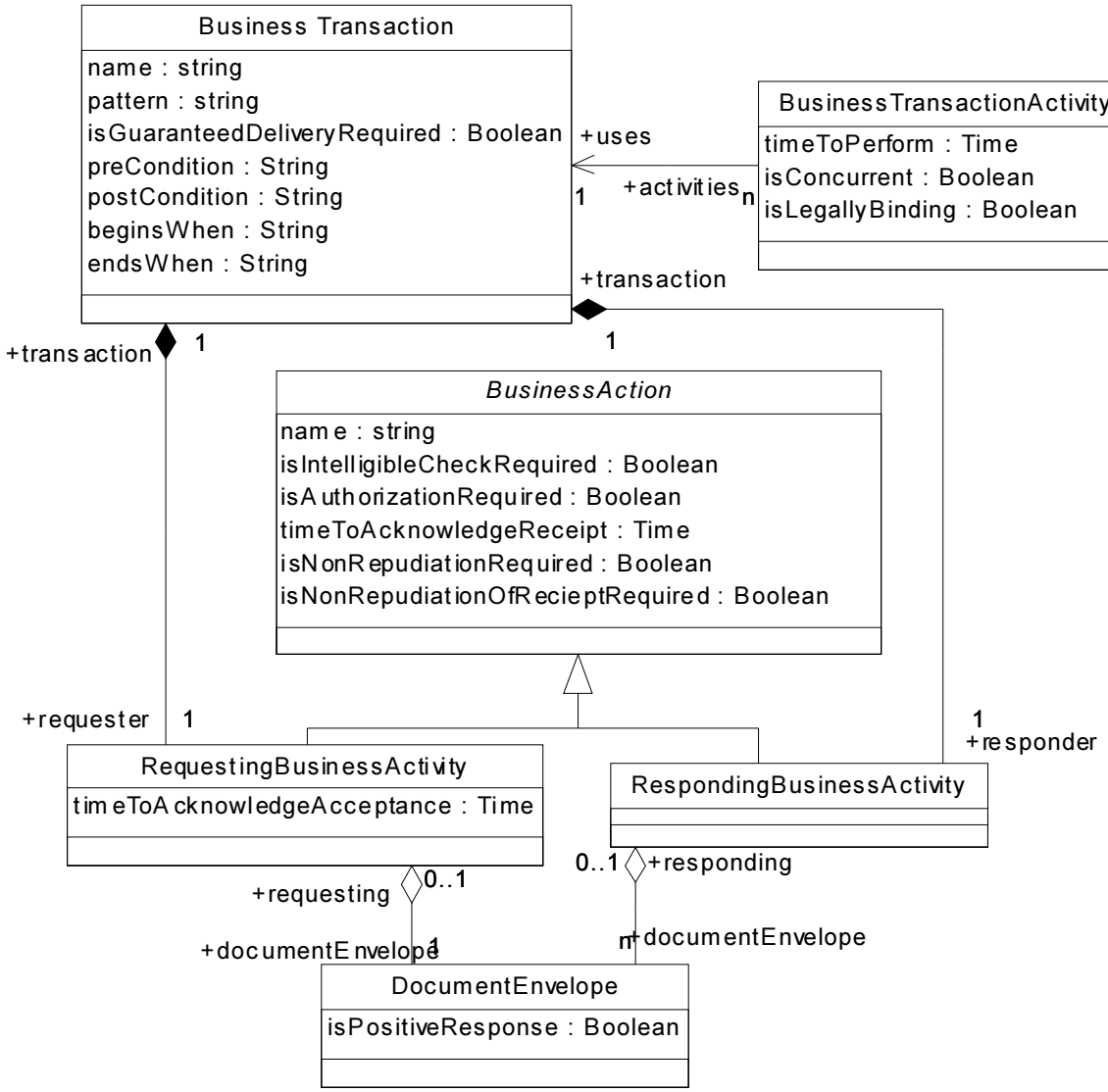
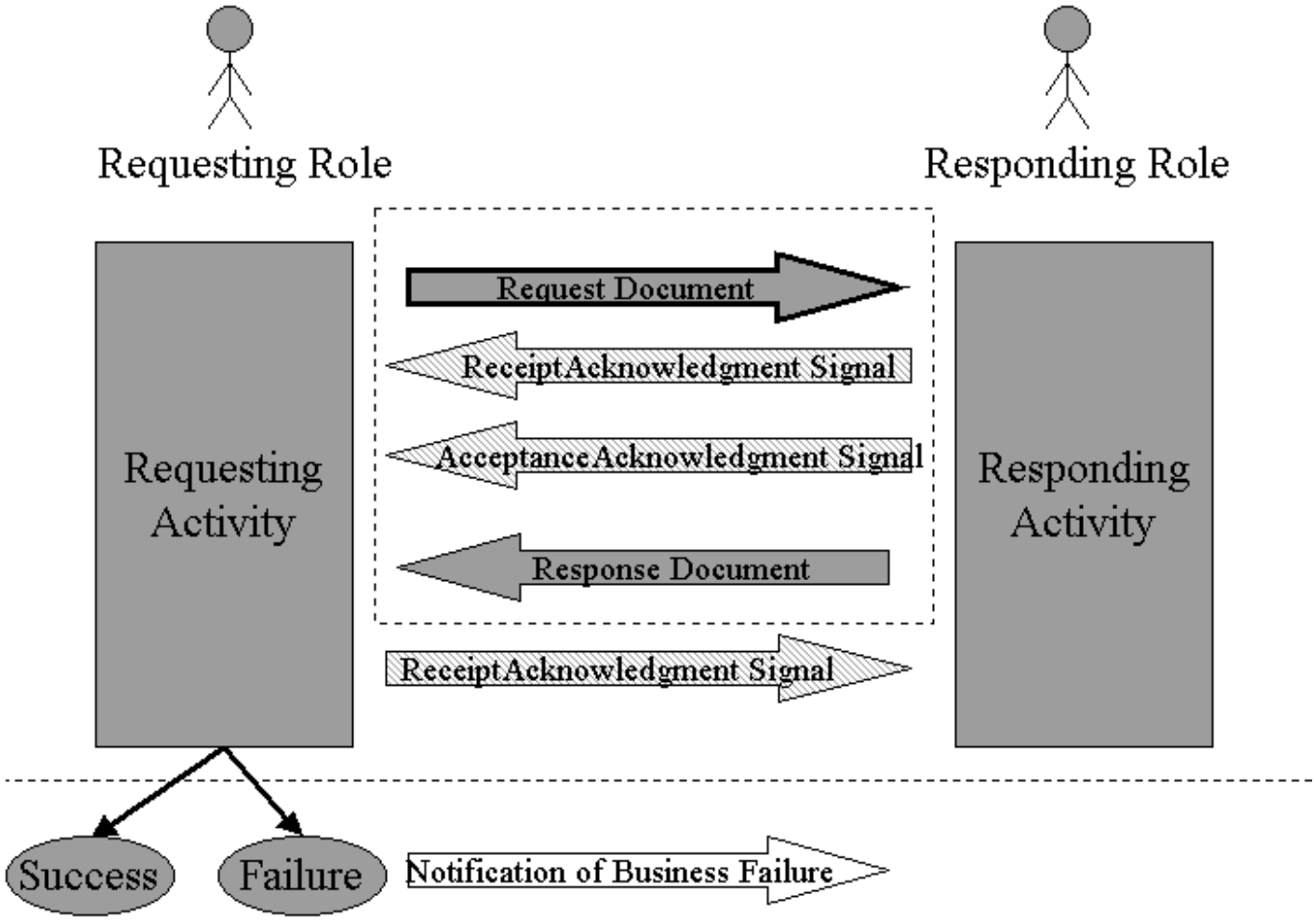


Diagramma UML di una *Business Transaction*



Possibile flusso di documenti e relativa sequenza

- ✓ *Esempio di Flusso di documenti*, sono modellati indirettamente come Involucri di Documenti (Document Envelope) inviati da una parte e ricevuti dall'altra, associati con una richiesta di attività si business ed una corrispondente risposta.

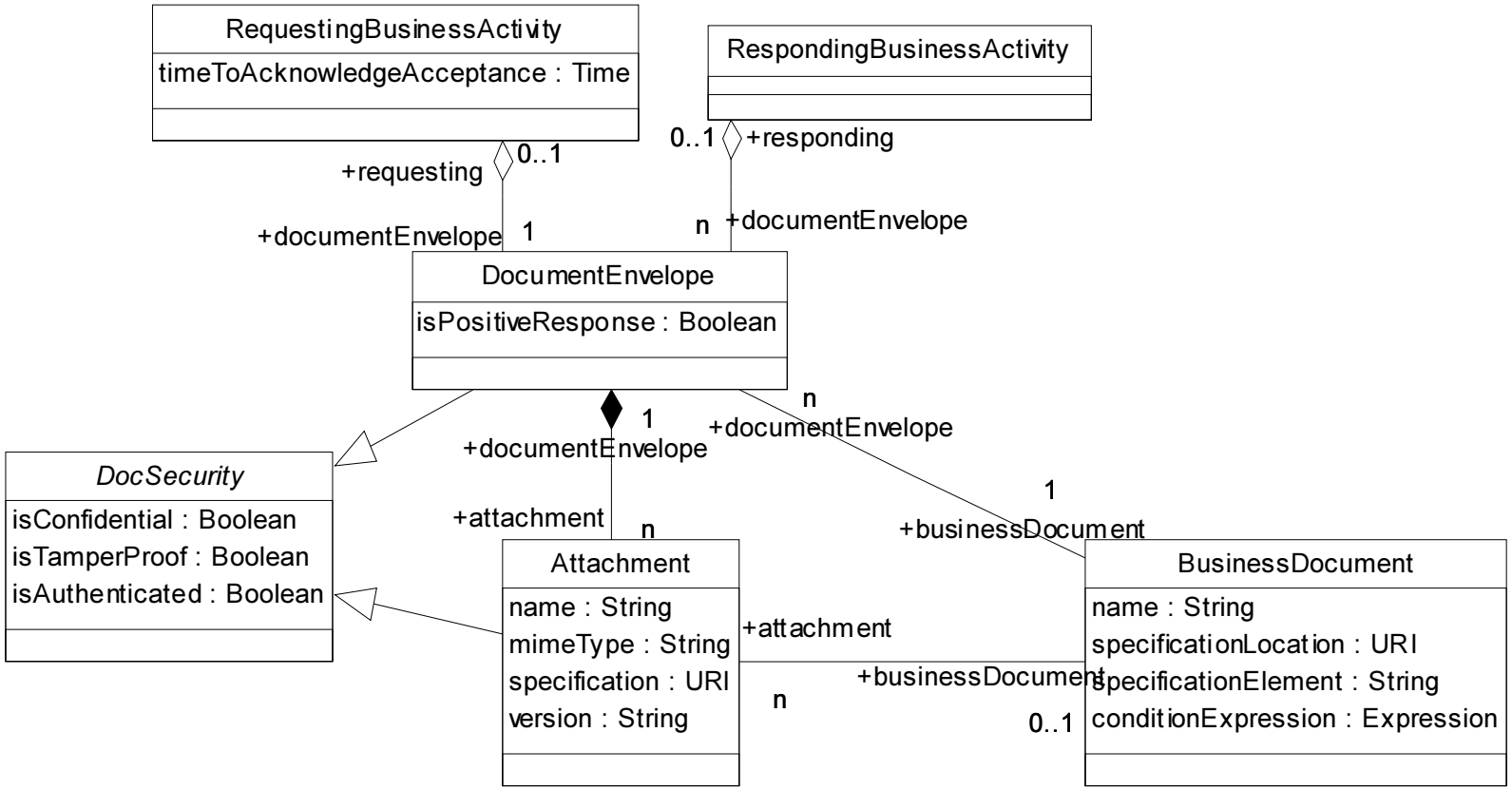


Diagramma UML di un *Document Flow*

- ✓ C'è sempre un solo Document Envelope associato ad una Requesting Activity, questo può avere degli attachment, tutti legati al Business Document primario.

- ✓ In questo esempio la transazione ha una richiesta e due possibili risposte, successo o fallimento. La richiesta ha un allegato. Tutti i business document sono qualificati con lo schema name.

```
<BusinessDocument name="Purchase Order"         specificationLocation="someplace"/>
<BusinessDocument name="PO Acknowledgement"     specificationLocation="someplace"/>
<BusinessDocument name="PO Rejection"           specificationLocation="someplace"/>
<BusinessDocument name="Delivery Instructions" specificationLocation="someplace"/>
<BusinessTransaction name="Create Order">
   <RequestingBusinessActivity name=""
      <DocumentEnvelope isPositiveResponse="true"
         BusinessDocument="ebXML1.0/PO Acknowledgement">
         <Attachment   name="DeliveryNotes"
                       mimeType="XML"
                       BusinessDocument="ebXML1.0/Delivery Instructions"
                       specification=""
                       isConfidential="true"
                       isTamperProof="true"
                       isAuthenticated="true">
         </Attachment>
      </DocumentEnvelope>
   </RequestingBusinessActivity>
   <RespondingBusinessActivity name=""
      <DocumentEnvelope BusinessDocument="ebXML1.0/PO Acknowledgement"/>
      <DocumentEnvelope isPositiveResponse="false"
         BusinessDocument="ebXML1.0/PO Rejection"/>
   </RespondingBusinessActivity>
</BusinessTransaction>
```

- ✓ *Esempio di Collaborazione Binaria*, definisce un protocollo di interazione tra due ruoli autorizzati, coreografata da un insieme di stati di questi ruoli.

timeToPerform	periodo di tempo dall'inizio della prima attività entro il quale l'intera collaborazione deve concludersi
preCondition	descrizione dello stato esterno a questa collaborazione, richiesto prima che essa inizi
postCondition	descrizione dello stato che non esiste prima ma esisterà dopo che questa collaborazione verrà eseguita
beginsWhen / endsWhen	descrizione di un evento esterno alla collaborazione, che normalmente causa l'inizio/fine della medesima.
pattern	riferimento opzionale ad un pattern sul quale questa collaborazione si basa
role	una collaborazione binaria consiste di due ruoli autorizzati, uno di iniziazione ed uno di risposta
states	stati della coll.binaria, statico o stato azione
usedBy	una coll. Binaria può essere usata dentro un'altra c.binaria in una attività di collaborazione
transitions	La transizione tra attività

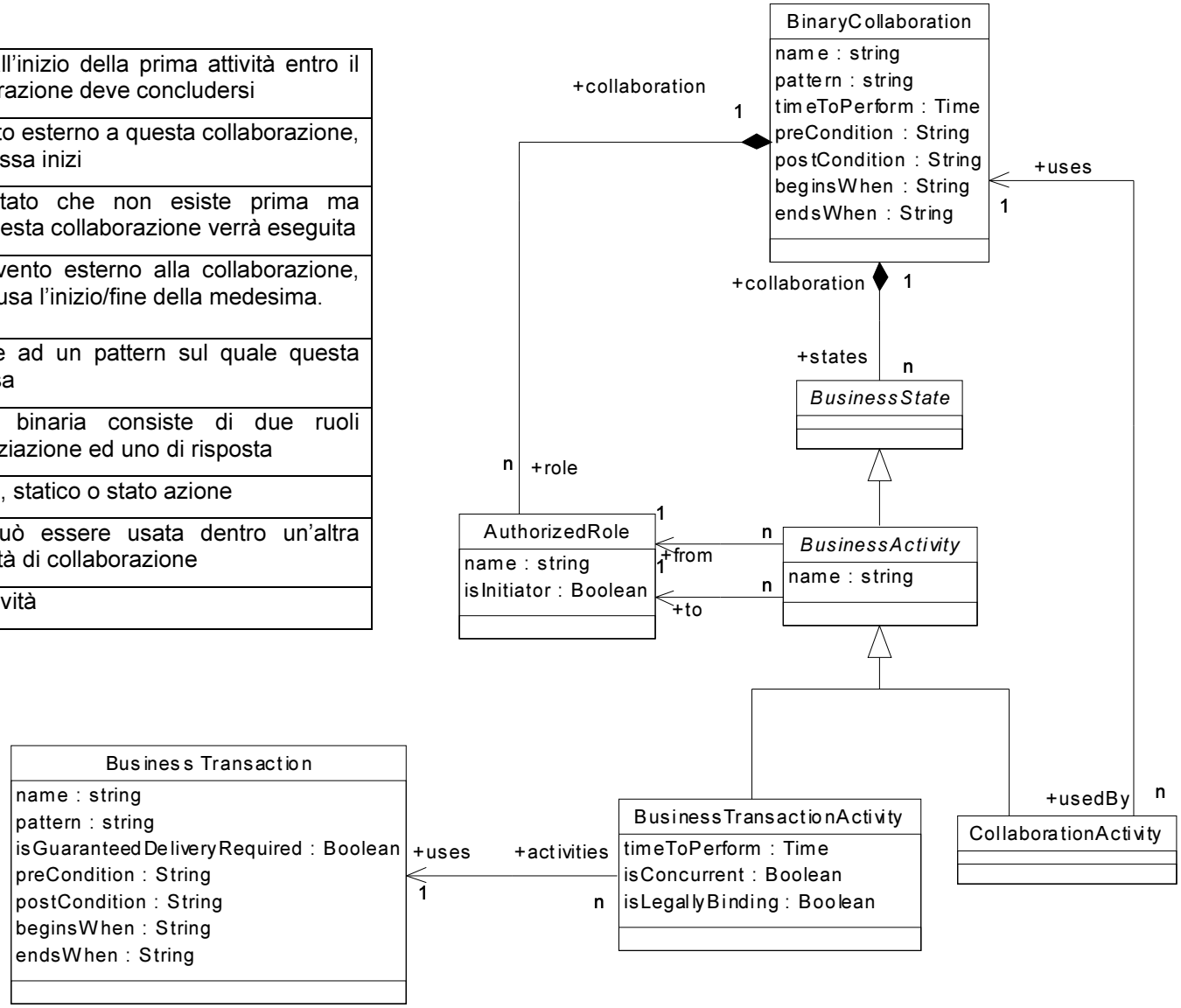
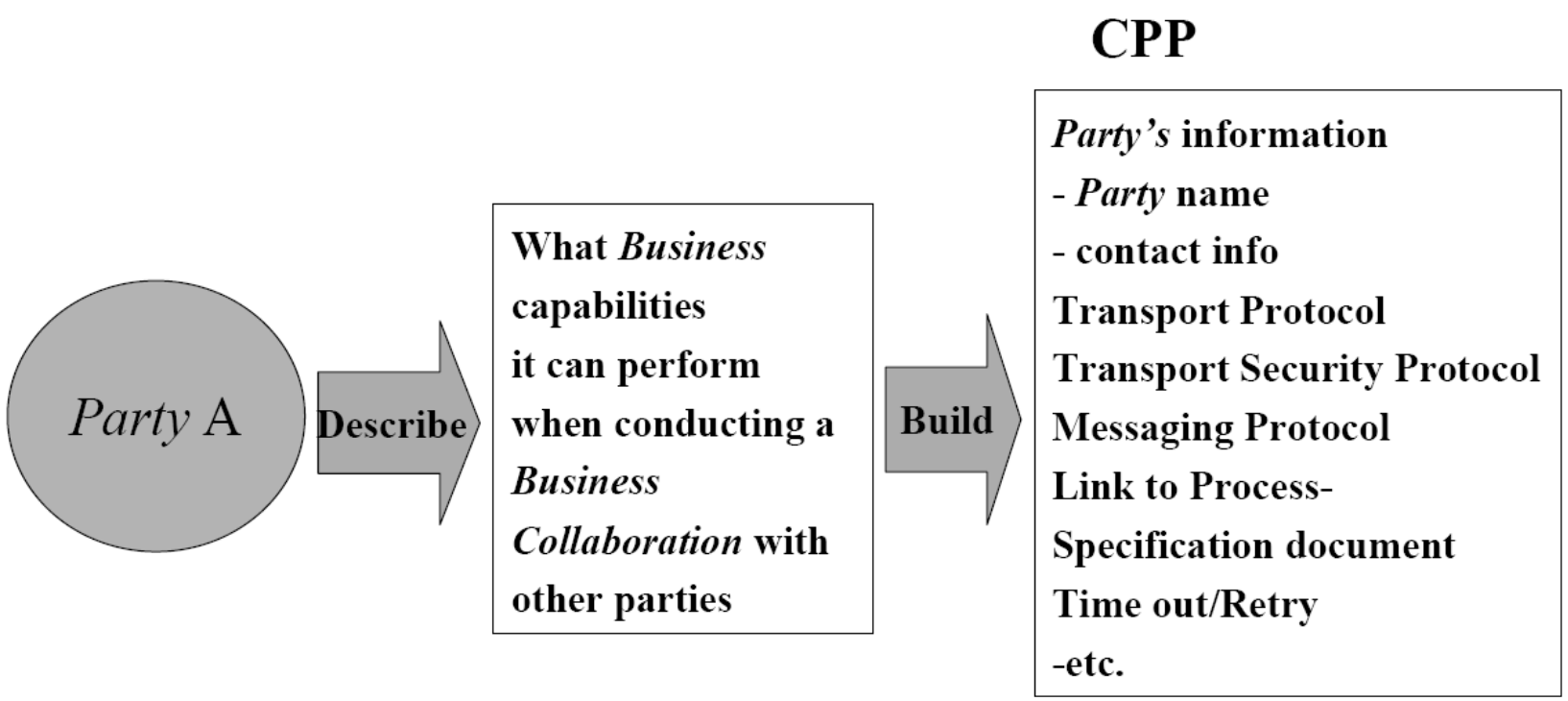


Diagramma UML di una collaborazione binaria

```
<BinaryCollaboration name="Product Fulfillment" 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"
      isLegallyBinding="true"/>
   <BusinessTransactionActivity
      name="Notify shipment"
      businessTransaction="Notify of advance shipment"
      fromAuthorizedRole="buyer"
      toAuthorizedRole="seller"/>
</BinaryCollaboration>
```

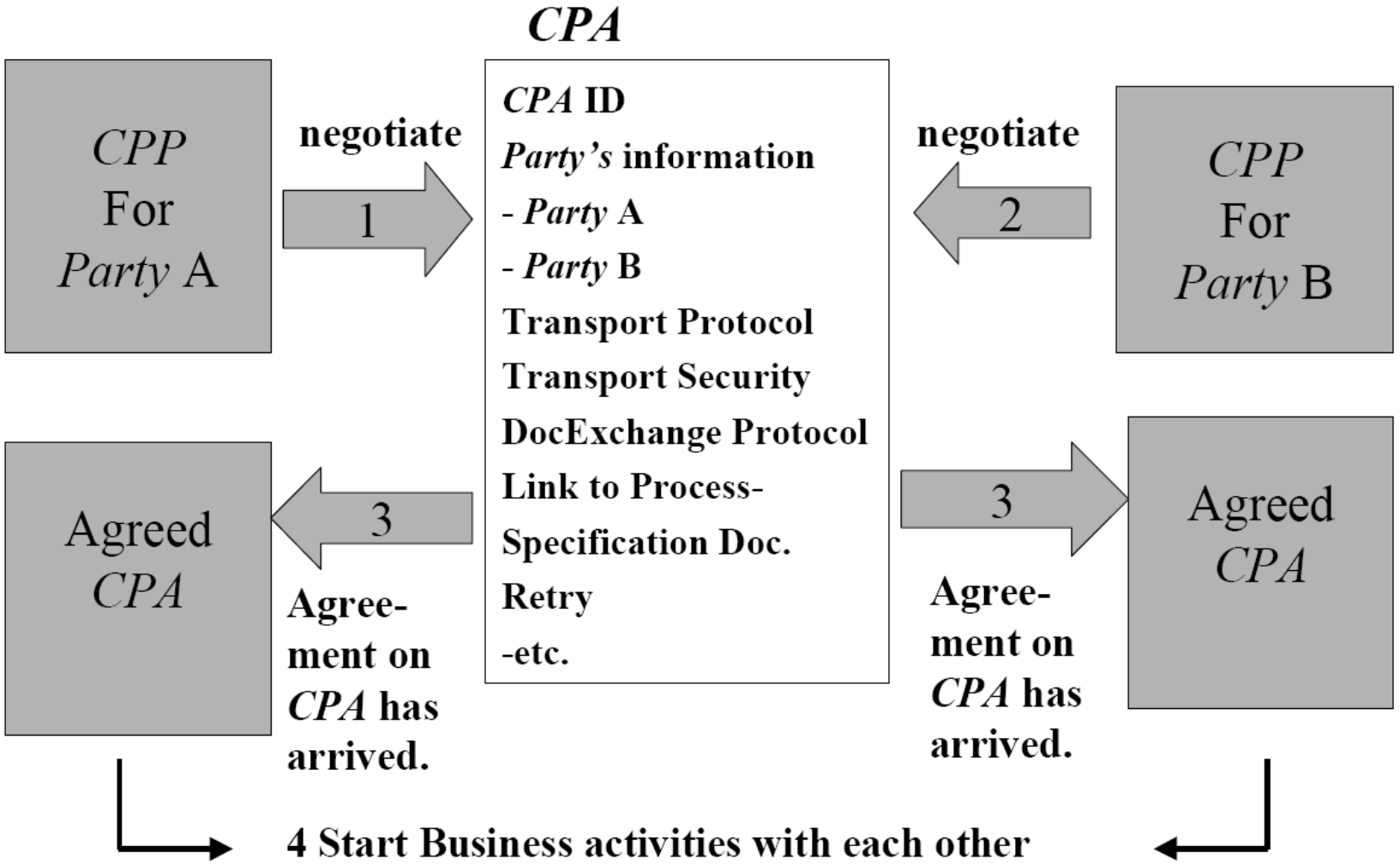
- ✓ *Esempio di Collaboration Protocol Profile*, i CPP possono essere immagazzinati in un apposito repository pubblico quale l'ebXML Registry. Tramite un processo di ricerca ed individuazione, codificato nelle specifiche del repository, una parte commerciale può trovare adeguati Business Partner.



CPP

- ✓ La parte A classifica le informazioni da inserire nel registry per il processo di discovery, costruisce un CPP con queste informazioni e le inserisce in un ebXML Registry.
- ✓ Un CPP è composto da quattro livelli. *Process Specification Layer* (le business transactions che le due parti possono chiedersi e le regole di transizione che determinano l'ordine delle richieste), *Delivery Channels* (le caratteristiche dei messaggi di ricezione e trasmissione), *Document-Exchange Layer* (specifica il

trattamento dei documenti di business es. encryption, digital signature, messaggistica affidabile), *Transport Layer* (protocollo di trasporto per mandare i messaggi nella rete e definire gli indirizzi degli endpoint)



CPA

- ✓ Le due parti usano i loro CPP per costuire congiuntamente un unico CPA calcolando l'intersezione tra i rispettivi CPP. Il risultato definisce come essi possono comportarsi nell'effettuare le loro Business Collaboration.
- ✓ Struttura di un CPP.

```
<tp:CollaborationProtocolProfile
   xmlns:tp="http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd
                       http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   tp:cppid="uri:companyA-cpp"
   tp:version="2_0b">
  <tp:PartyInfo>           ... </tp:PartyInfo>    <!-- one or more -->
  <tp:SimplePart id="..."> ... </tp:SimplePart>   <!-- one or more -->
  <tp:Packaging  id="..."> ... </tp:Packaging>    <!-- one or more -->
  <tp:Signature>           ... </tp:Signature>    <!-- zero or one -->
  <tp:Comment> text            </tp:Comment>      <!-- zero or more -->
</tp:CollaborationProtocolProfile>
```

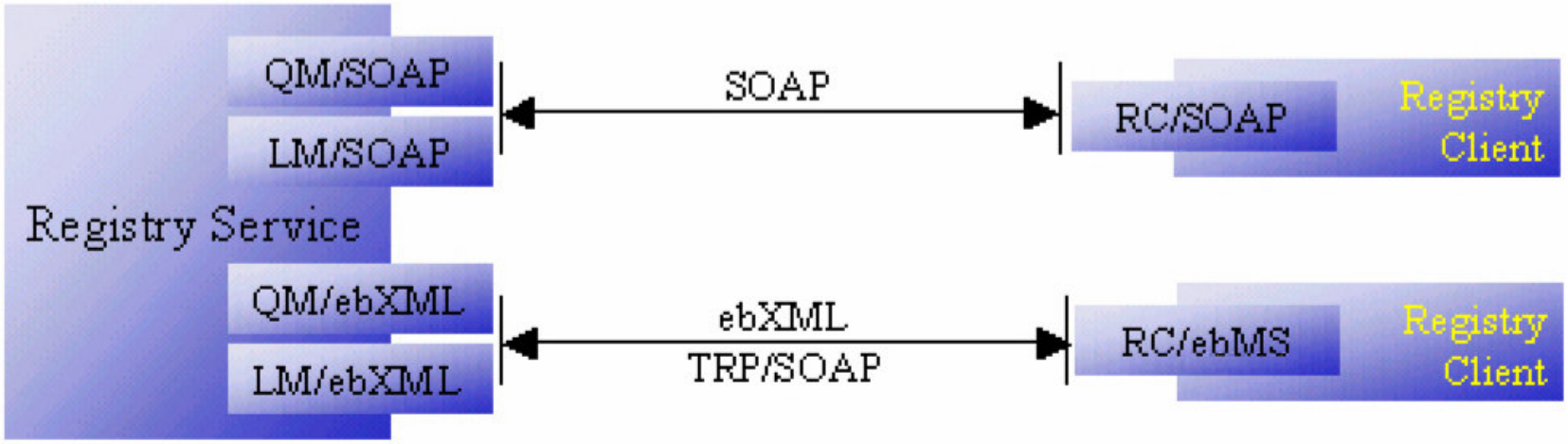
- ✓ *tp* (CPP/CPA namespace), *ds* (XML Digital Signature namespace), *xlink* (Xlink namespace, *Xlink* è un vocabolario XML standardizzato che può essere aggiunto a elementi di documenti istanza XML per descrivere collegamenti estesi, di qualsiasi complessità, fra risorse nel Web)
- ✓ Ciascun *PartyInfo* identifica l'organizzazione e contiene informazioni quali l'id (es. EAN/UCC) i ruoli che può assumere, link a documenti aggiuntivi, certificati,

dettagli relativi alla sicurezza, al protocollo di trasporto. *SimplePart* fornisce una lista di elementi costitutivi identificati dal corrispondente tipo MIME (Multi-purpose Internet Mail Extension, metodo per trasferire ogni tipo di file tramite la posta elettronica) *Packaging* indica informazioni sul Message Header ed il payload dei messaggi.*Signature* abilita il CPA ad essere firmato usando le specifiche XMLDSIG (XML Digital SIGnature specification).

- ✓ *CPA* come detto, è composto da due CPP, pertanto molti elementi sono a comune con questo. Ecco la struttura di alto livello.

```
<CollaborationProtocolAgreement
   xmlns:tp="http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   tp:cpaid="YoursAndMyCPA"
   tp:version="2.0a">
   <tp:Status tp:value="proposed"/>
   <tp:Start>1988-04-07T18:39:09</Start>
   <tp:End>1990-04-07T18:40:00</End>
   <!-- ConversationConstraints MAY appear 0 or 1 time -->
   <tp:ConversationConstraints
      tp:invocationLimit="100"
      tp:concurrentConversations="4"/>
   <tp:PartyInfo> ... </tp:PartyInfo>
   <tp:PartyInfo> ... </tp:PartyInfo>
   <tp:SimplePart tp:id="...">...  </tp:SimplePart>  <!-- one or more -->
   <tp:Packaging  tp:id="...">...  </tp:Packaging>   <!-- one or more -->
   <tp:Signature>              ...  </tp:Signature>   <!-- zero or one time -->
   <tp:Comment xml:lang="en-GB">any text</Comment>   <!-- zero or more -->
</tp:CollaborationProtocolAgreement>
```

essere implementato in un ebXML Registry sotto forma di uno schema di database relazionale, di uno schema di database ad oggetti, oppure di un altro schema fisico, come un insieme di interfacce e classi all'interno di una Registry Implementation.



✓ Fig. Implementazione concreta in cui il Registry Service supporta collegamenti con diversi protocoli (SOAP ed ebMS)

Core Component

✓ Elementi semantici usati per costruire la struttura semantica di un business document, raggruppati in categorie di *contesto*. Es. la struttura semantica di un *purchase order* avrà elementi semantici quali *item*, *quantity*, *price*, …

✓ Quando il business process context è *Purchasing* ed il geopolitical context è *EU* (European Union) allora il core component *tax.amount* è interpretato come *VAT.amount* (Value Added Tax = IVA).

✓ (Estratto dal Core Component Dictionary)

Category Type	**Basic**
Core Component Type	**amount. type**

CCT (Core Componente Type), Basic, o Aggregate

. . .

Name	tax. amount	***definition***	
UID	000149	The amount of tax.	
Datatype	n/a		
Core Component Type	amount. type		
Core component re-used	n/a		non applicabile
Synonyms		***remarks***	
Naming Convention			
object class	tax		
property term representation type	amount*		
	amount		

. . .

Category Type	**CCT**
Core Component Type	**n/a**

Name	amount. type	***definition***
UID	000105	A number of monetary units specified in a currency where the unit of currency is explicit or implied.
UID		
Datatype	n/a	
Core Component Type	n/a	
Core component re-used	n/a	
Synonyms		***remarks***

Naming Convention

object class	amount
property term	type
representation type	

✓ (Altro estratto)

- ✓ Inoltre viene definito un modo per creare nuovi vocabolari di business e per ristrutturare quelli già esistenti. Si cerca di garantire rappresentazioni delle informazioni che siano allo stesso tempo leggibili dall'utente e processabili da sistemi automatici.
- ✓ La standardizzazione viene portata avanti secondo un approccio "syntax-neutral", indipendente dalla sintassi. I Core Component sono descritti in UML e conservati nell'ebXML registry. Tale rappresentazione si può convertire in qualsiasi sintassi: XML Schema, XML DTD, …
- ✓ Usare i Core Component come parte fondamentale del framework ebXML aiuta a garantire che due diversi partner commerciali che usano differenti sintassi, utilizzino la semantica di business allo stesso modo, a condizione che le due sintassi siano basate sugli stessi Core Component.

Funzionamento di ebXML

- ✓ Riepilogo dei concetti principali:
  i. meccanismo standard per descrivere processi di business ed i corrispondenti modelli dell'informazione
  ii. meccanismo per registrare e conservare tali entità

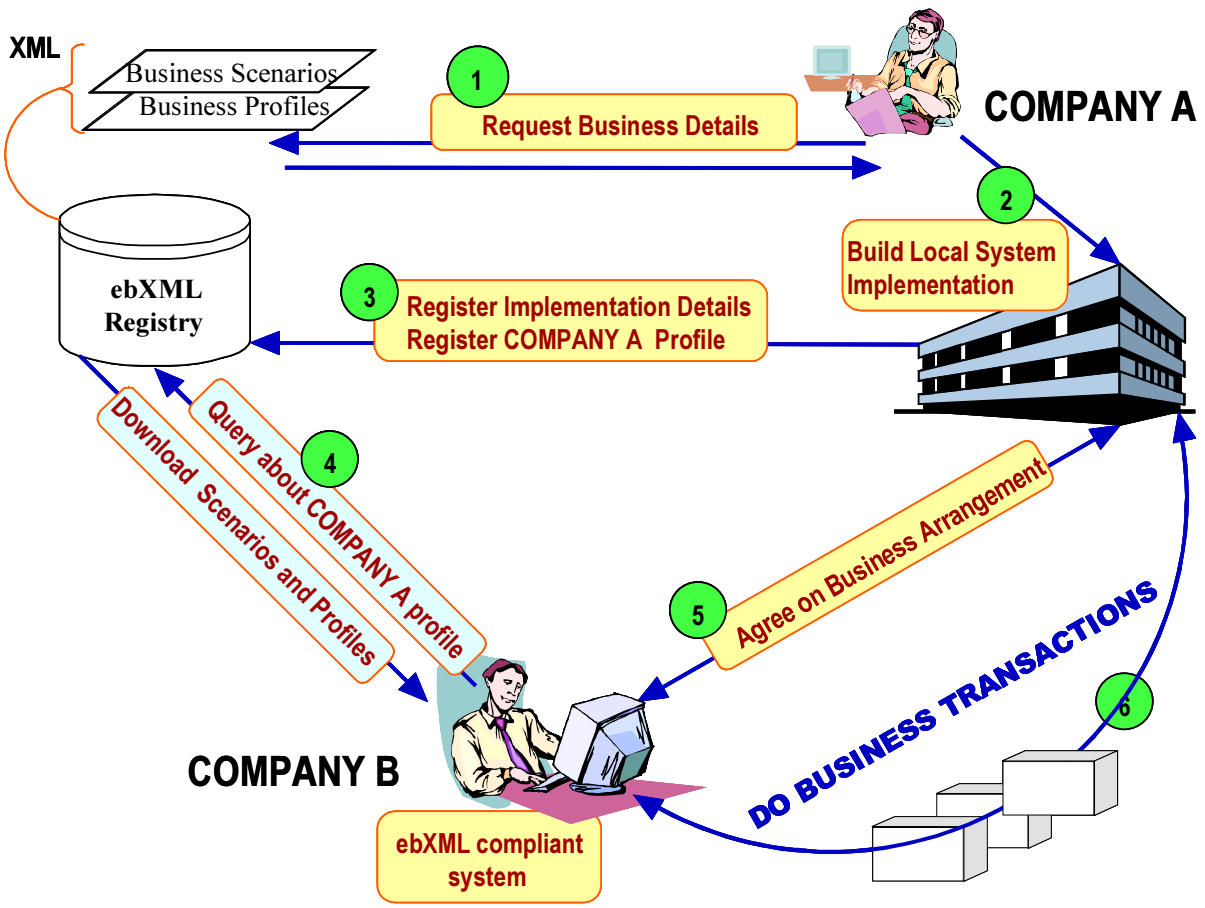
iii. definizione di ciascun partecipante mediante a) i Business Process (BP) che supporta, b) le Business Service Interfaces (BSI) che offre in supporto dei BP, c) i Business Messages scambiati tra le BSI, 4) configurazione tecnica di protocolli di trasporto, sicurezza e codifica.

iv. meccanismo per conservare tali entità

v. meccanismo per descrivere l'esecuzione di disposizioni mutuamente accordate (CPA)

vi. una piattaforma di messaggistica di business standardizzata (ebMS)

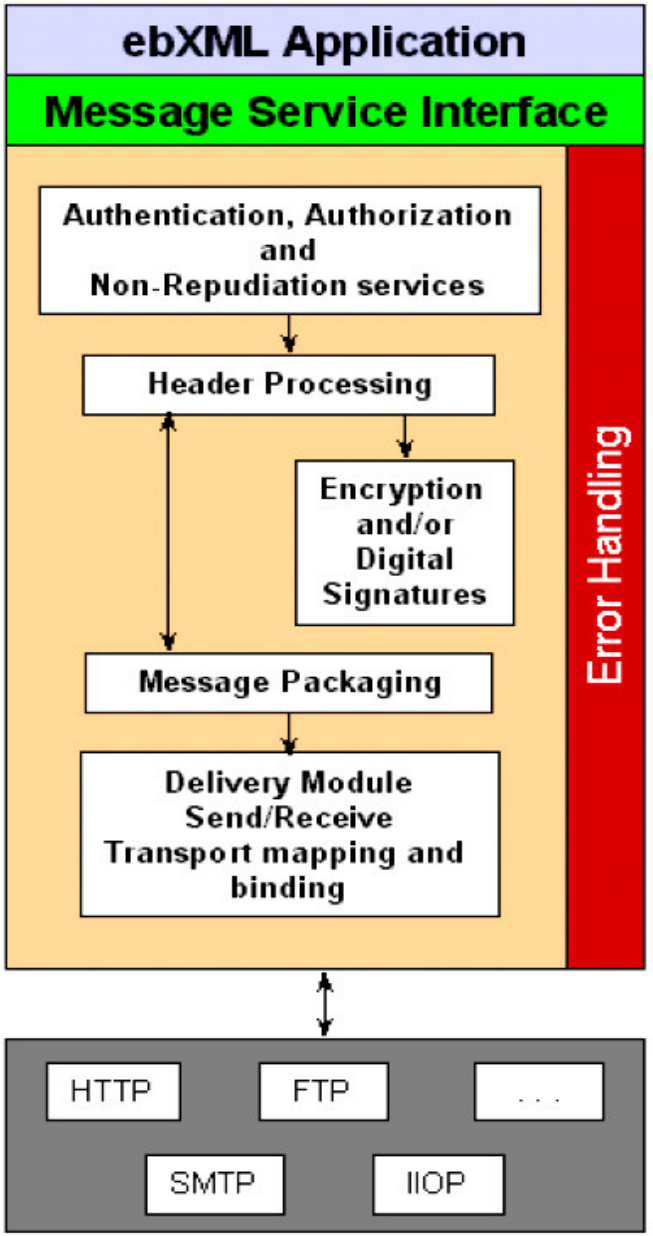
vii. meccanismo di configurazione dei servizi di messagistica in accordo con i vincoli definiti nell'accordo di business.


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

3) La società A invia le informazioni di *Business Profile* al *Registry*. Il Business Profile inviato al Registro descrive capacità e vincoli della società così come tutti gli scenari di business che essa supporta, ossia versione XML dei BP.
4) La società B scopre gli scenari di business supportati dalla Società A nel *Registry*.
5) La società B invia una richiesta alla società A segnalando l'intenzione di impegnarsi in una transazione commerciale appoggiandosi sull'architettura ebXML. La società B si dota di un'applicazione ebXML-compliant ed invia uno schema del business proposto direttamente all'Interfaccia del software ebXML-compliant della società A. Tale schema delinea i processi di business, ed altri specifici requisiti per lo scambio delle informazioni necessarie per il buon fine della transazione, eventuali piani di emergenza ed i requisiti relativi alla sicurezza. La società A accetta la proposta di business.
6) Le società A e B possono ora lanciarsi nell'eBusiness usando ebXML.

ebXML Messaging Service (ebMS)

- ✓ L'ebXML Message Service può essere concettualmente suddiviso in tre parti (i) una Message Service Interface astratta, (ii) funzioni fornite dal Message Service Handler – MSH, (iii) il mapping verso il servizio di trasporto sottostante.
- ✓ *Header Processing*, la creazione dell'ebXML Header Message avviene in base agli input dell'applicazione, passati attraverso la Message Service Interface, e del CPA che decide la struttura del messaggio, e poi informazioni quali firma digitale, identificatori,etc.
- ✓ *Message Packaging*, l'avvolgimento finale di un ebXML message (header+payload) nel messaggio SOAP with attachment container.
- ✓ *Message Service Interface* fornisce delle funzioni :
  - *Send*: spedisce un messaggio ebXML, i valori per i parametri sono ricavati dal Message Header.
  - *Receive*: ricezione dei messaggi
  - *Notify*: fornisce notifica di eventi attesi e inattesi.
  - *Inquire*: fornisce un metodo per conoscere lo stato di un particolare scambio di messaggi.



- ✓ SOAP with Attachment (SwA) è una combinazione del protocollo SOAP con il formato MIME che consente di includere in un messaggio SOAP qualsiasi tipo di dati, in analogia al modello usato per

```
<SOAP:Envelope
  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">
    <eb:MessageHeader ...>    ...   </eb:MessageHeader>
  </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">
    <eb:Manifest eb:version="2.0"> ... </eb:Manifest>
  </SOAP:Body>
</SOAP:Envelope>
```

✓ Esempio di MessageHeader e Manifest in uno scenario di creazione nuovo ordine:

```
<eb:MessageHeader SOAP:mustUnderstand="1" eb:version="2.0">
  <eb:From>
    <eb:PartyId>urn:duns:123456789</eb:PartyId>
    <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
  </eb:From>
  <eb:To>
    <eb:PartyId>urn:duns:912345678</eb:PartyId>
    <eb:Role>http://rosettanet.org/roles/Seller</eb:Role>
  </eb:To>
  <eb:CPAId>http://esempio.com/cpas/nostrocpaconvoi.xml</eb:CPAId>
  <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
  <eb:Service>urn:services:SupplierOrderProcessing</eb:Service>
  <eb:Action>NewOrder</eb:Action>
  <eb:MessageData>
    <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
    <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
  </eb:MessageData>
</eb:MessageHeader>
```

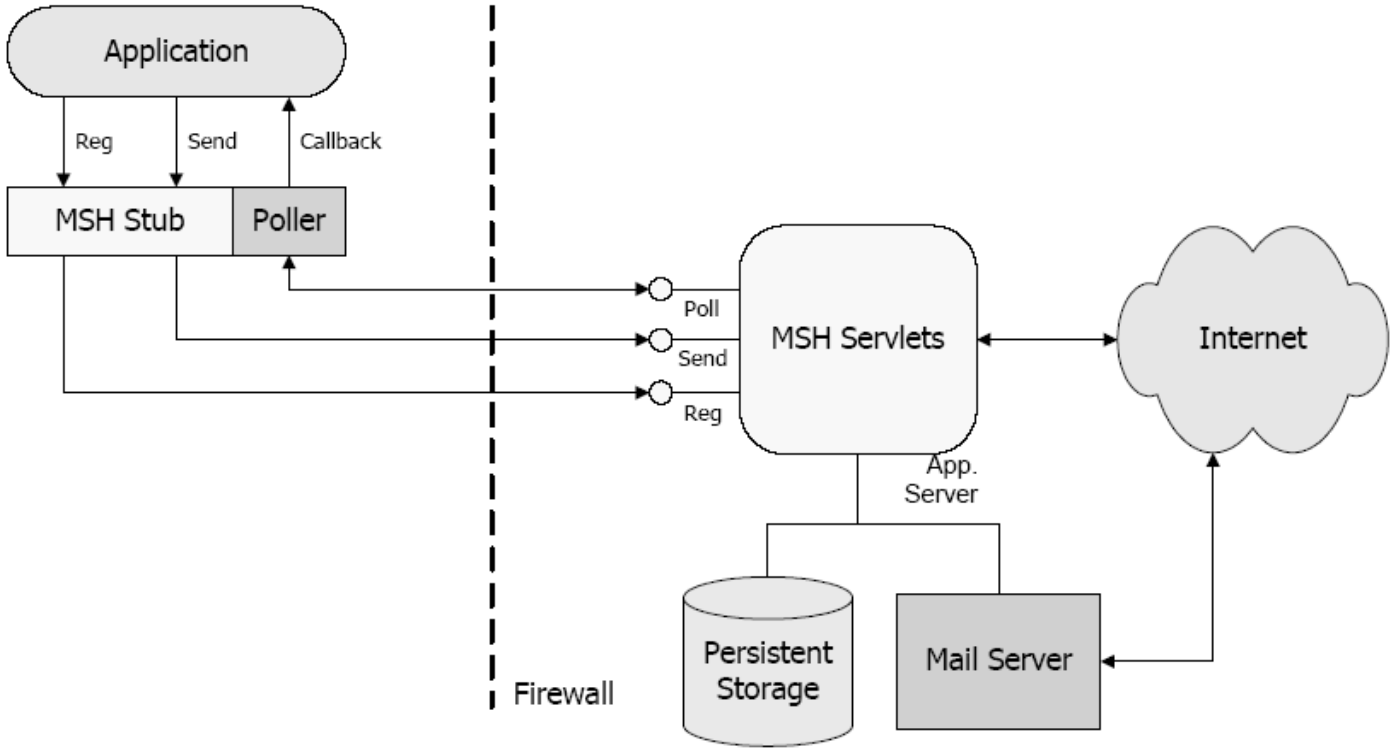
From/To: id del mittente/destinatario e ruolo assunto nella collaborazione.
CPAId: identificatore del CPA, spesso una URI.
ConversationId: identifica l'insieme dei messaggi di una conversazione tra due partner. È inizializzato dalla parte che inizia il dialogo, e verrà mantenuto per tutta la durata degli scambi.
Service: indica il servizio richiesto nel messaggio.
Action: identifica una particolare azione all'interno di un servizio. Ad esempio la creazione di un nuovo ordine.
MessageData: contiene tutte le informazioni necessarie all'identificazione unica di un messaggio, in particolare un id generato casualmente e un timestamp legato all'istante di creazione dell'header. Questo campo può contenere inoltre l'id del messaggio precedente, collegato a quello attuale ed il tempo di vita del messaggio.

```
<SOAP:Body>
  <eb:Manifest eb:version="2.0">
    <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
                  xlink:role="XLinkRole" xlink:type="simple">
      <eb:Schema eb:location="http://regrep.org/po.xsd" eb:version="2.0"/>
      <eb:Description xml:lang="en-US">Purchase Order 1</eb:Description>
    </eb:Reference>
  </eb:Manifest>
</SOAP:Body>
```

Manifest identifica, attraverso uno o più elementi *Reference*, i documenti allegati al messaggio o risorse esterne accessibili attraverso un URL. Ogni *Reference* a sua volta contiene, nel caso di allegati di tipo XML, un collegamento ad uno schema che ne definisce la struttura ed una descrizione del suo contenuto.

Trasmissione di documenti secondo ebXML

- ✓ *Hermes Message Service Handler* (Hermes MSH) è un sistema di messaggistica ebXML sviluppato dal Center for ECommerce dell' Università di Hong Kong nell'ambito del progetto *Phoenix*, che ha lo scopo di realizzare un'infrastruttura per il commercio elettronico basata sull'utilizzo dello standard ebXML.



- ✓ L'architettura si divide in due parti: una MSH Servlet ed un MSH Stub. Il motivo principale di questa soluzione è legata alla presenza di firewall. Generalmente gli

MSH endpoint sono connessi direttamente ad Internet all'esterno del firewall aziendale. Invece i destinatari dei messaggi ebXML sono applicazioni all'interno del firewall per ovvi motivi di sicurezza.

- ✓ Le funzionalità principali di MSH sono quindi implementate nella MSH Servlet, e l'applicazione può gestirle istanziando un MSH Stub e registrandosi mediante una opportuna chiamata di metodo di tale istanza.
- ✓ Le informazioni per la registrazione sono prelevate dal Collaboration Protocol Agreement (CPA) e sono informazioni necessarie per stabilire l'endpoint (indirizzo del partner) a cui inviare il messaggio e per indirizzare i pacchetti in ingresso al giusto destinatario.
- ✓ Una servlet MSH può infatti supportare numerosi client ma ognuno deve registrarsi con un diverso contesto affinché i messaggi siano recapitati correttamente al giusto destinatario.
- ✓ MSH Stub converte i parameri di registrazione in un messaggio SOAP ed invia tale chiamata alla MSH Servlet via HTTP.
- ✓ I messaggi ebXML generati dal client sono inviati allo stesso modo. MSH Stub in tal caso, non solo provvede al packaging del messaggio, ma anche a qualche piccola elaborazione (es. aggiungere firma digitale e criptare) e ad inoltrarlo all'MSH Servlet
- ✓ L'MSH Servlet provvede ad ulteriori elaborazioni (es. la gestione di messaggistica affidabile) ed ad inoltrarlo al destinatario.

- ✓ In ricezione esistono quattro modalità di funzionamento. La più semplice, prevede che MSH Servlet elabori il messaggio e ne individui l'applicazione client di destinazione sulla base delle informazioni di registrazione. MSH stub possiede un polling thread che periodicamente controlla se vi sono messaggi e li invia all'applicazione mediante un metodo callback.
- ✓ Per sviluppare un client ebXML in Java, la classe relativa deve implementare l'interfaccia `MessageListener` e definire i metodi `getClientURL()` (che in modalità base ritorna semplicemente `null`) ed `onMessage()` che viene invocata dal meccanismo di callback quando arriva un messaggio, pertanto contiene tutte le elaborazioni necessarie.
- ✓ La registrazione avviene istanziando un oggetto `Request(…)` in cui viene passato un riferimento all'applicazione (per il meccanismo di callback), il tipo di protocollo scelto, l'end-point dell'applicazione, ed il contesto (composto dal *CPAId*, *ConversationId*, il tipo di servizio richiesto e la particolare azione all'interno di esso)
- ✓ Questi parametri vengono prelevati dal *cpa*, pertanto occorre costruire una classe CPAParser con tutti i metodi per prelevare le informazioni suddette.
- ✓ L'invio avviene istanziando un oggetto EbxmlMessage, che dopo essere stato opportunamento "confezionato" viene spedito nel caso più semplice mediante il metodo `send()`.
- ✓ I messaggi ricevuti da MSH vengono mantenuti in delle apposite cartelle (Repository), mentre gli id degli stessi vengono inseriti in un database mysql insieme

ad altre informazioni prelevate dall'intestazione. Nel file di configurazione msh_client_property.xml è possibile indicare il tempo di attesa, espresso in secondi, fra un polling e il successivo.

- ✓ Le operazioni di ricezione sono invocate all'interno di `onMessage()` e riguardano l'estrazione del payload, la costruzione di un oggetto Document e quindi l'elaborazione del contenuto.

Semplice Client ebXML

a) Installazione e configurazione dei componenti di supporto

A meno che non specificato, scegliere le opzioni di installazione di default.

1. Installare J2SE v 1.4.2_05 SDK o superiore (http://java.sun.com/j2se/1.4.2/download.html) supponiamo per default su `C:\j2sdk1.4.2_05\`
2. Inserire nella variabile di ambiente PATH il percorso della cartella bin di J2SDK.
   `C:\j2sdk1.4.2_05\bin`
3. Assicurarsi che esista la variabile ambiente JAVA_HOME settata al percorso di installazione di j2sdk
   `JAVA_HOME = C:\j2sdk1.4.2_05`
4. Installare Tomcat (**full install**) versione 5.0.27 o superiore (http://jakarta.apache.org/site/binindex.cgi). Per brevità supponiamo su `C:\tomcat5\`
5. Scaricare Hermes Message Service Handler (MSH) implementation, release 0.9.3.0 o superiore, (**all in one .zip package**)(http://www.freebxml.org/mshdownload.htm). Ecco i file che serviranno:

`build\msh.jar`	Java API
`doc\`	javadoc, guida di sviluppo ed installazione
`conf\msh_client_properties.xml` `conf\msh_properties.xml`	vedere punto 6
`dist\hermes_...zip\msh.war`	vedere punto 10
`lib\`	Insieme di componenti API in formato jar, vedere fig.7

6. Copiare i file di configurazione `msh_client_properties.xml` e `msh_properties.xml`, nella home dell'utente (es. `C:\Documents and Settings\nomeutente`).
7. Modificare i suddetti file come di seguito:

ELEMENT	**VALUE**
msh.properties.xml/MSH/Log/LogPath	C:/tomcat5/webapps/msh/
msh.properties.xml/MSH/Config/URL	http://localhost:8080/msh/
msh.properties.xml/MSH/Config/ AuthenticationFile	C:/tomcat5/msh_passwd
msh.properties.xml/MSH/Config/ ToUrlResolver	eliminare l'intero elemento
msh.properties.xml/MSH/Mail	eliminare l'intero elemento
msh.properties.xml/MSH/DigitalSignature	<DigitalSignature> <TrustedAnchor> <KeyStore> <Path>C:/tomcat5</Path> <File>keystore</File> <Password>ebxmlms</Password> </KeyStore> </TrustedAnchor> <AckSign> <KeyStore> <Path>C:/tomcat5</Path> <File>keystore</File> <Algorithm>dsa-sha1</Algorithm> <Alias>ebxmlms</Alias> <Password>ebxmlms</Password> </KeyStore> </AckSign> </DigitalSignature>
msh.properties.xml/MSH/Persistent/Database	<Database> <Driver>org.gjt.mm.mysql.Driver</Driver> <User>msh</User> <Password>msh</Password> <URL> jdbc:mysql://localhost:3306/msh </URL> <TransactionIsolationLevel> READ_COMMITTED </TransactionIsolationLevel> <InitialConnections> 1 </InitialConnections> <MaximumConnections> 2

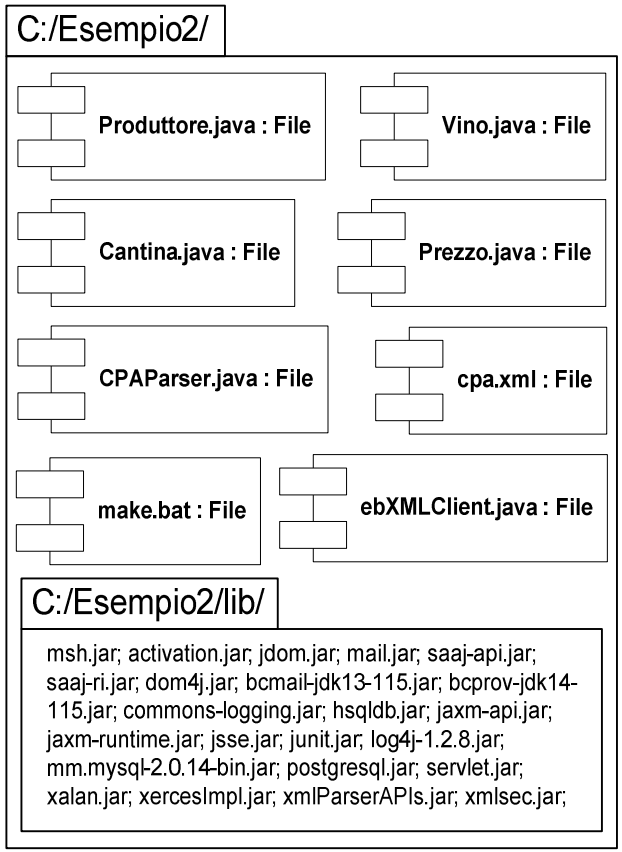
	</MaximumConnections> <MaximumWait>20000</MaximumWait> <MaximumIdle>60000</MaximumIdle> </Database>
msh.properties.xml/MSH/Persistent/ MessageRepository	C:/tomcat5/webapps/msh/messageRepository
msh.properties.xml/MSH/Persistent/ BackupFile	C:/tomcat5/MSHBackup.zip
msh.properties.xml/MSH/Persistent/ ArchiveDirectory	C:/tomcat5/msh_archive
msh.properties.xml/MSH/MessageListener/ TrustedRepository	C:/tomcat5/webapps/msh/trustedRepository1; C:/tomcat5/webapps/msh/trustedRepository2
msh.properties.xml/MSH/MessageListener/ ObjectStore	C:/tomcat5/webapps/msh/objectStore
msh_client_properties.xml/Request/ Log/LogPath	log
msh_client_properties.xml/Request/ Config/URL	http://localhost:8080/msh/
msh_client_properties.xml/Request/ Config/UserName	msh
msh_client_properties.xml/Request/ Config/Password	msh
msh_client_properties.xml/Request/ MessageListener/MessageRepository	C:/tomcat5/webapps/msh/client

8. Generare il file di autenticazione del client, `msh_passwd`, portandosi in un percorso dove ci sia il package `msh.jar`, e digitando `java -cp msh.jar hk.hku.cecid.phoenix.common.util.PassTool add`
   (Digitare `Pasword file: msh_passwd`, `User: msh`, `Password: msh`)
9. Copiare il file `msh_passwd` nella cartella `C:/tomcat5`
10. Copiare il file `msh.war`, nella cartella `C:/tomcat5/webapps`. I file *.war* (Web Application aRchive) sono un modo standard per la distribuzione di componenti servlet sugli application server. Il core di MSH e` una servlet contenuta in tale file. Analogamente al file *.jar* per i componenti *.class*, un file *.war* si può creare con l'applicativo *jar* e sfogliare con *winzip*.
11. MSH ha bisogno di un database per archiviare lo stato della comunicazione e fornire servizi di supporto (recupero, affidabilità, tracciabilità dei messaggi). Installare MySQL database server & standard clients, v. 4.0.18 o superiore, supponiamo per default su `c:/mysql` (Il software è scaricabile da http://dev.mysql.com/downloads/, è utile installare anche l'interfaccia grafica, MySQL Graphical clients ad. es. Control Center).
12. Avviare il server `C:/mysql/winmysqladmin.exe` (Si consiglia di porre un collegamento a questa applicazione nella cartella "esecuzione automatica" di windows)
13. Creare un database `msh` nel seguente modo. Collocarsi nel percorso `c:/mysql/bin` , digitare `mysql` (entrando di default come `root`) ed al prompt di mysql digitare `CREATE DATABASE msh;`
14. Creare un utente `msh` con password `msh` nel seguente modo. Creare un file di testo `myscript.sql` dal seguente contenuto e salvarlo nella cartella `c:/mysql/bin` :

```
USE MYSQL;
INSERT into user
( Host, User, Password, Select_priv, Insert_priv, Update_priv,
  Delete_priv, Alter_priv, Grant_priv ) Values
('localhost', 'msh', password('msh'), 'Y', 'Y', 'Y', 'Y', 'Y', 'Y' );
FLUSH PRIVILEGES;
GRANT CREATE,DROP,SELECT,INSERT,UPDATE,DELETE,ALTER ON msh.* TO msh@localhost
IDENTIFIED BY 'msh';
```

15. Dal prompt mysql avviato al punto 13 digitare `\. myscript.sql`. Se non sono visualizzati errori, è posibile uscire da mysql (mediante `\q`) ed entrare come utente `msh` nel seguente modo: `mysql -u msh -p`, quindi digitare la password `msh`, infine provare a digitare `USE msh;` per controllare che esista il database msh e sia accesibile dall'utente msh. In alternativa, i punti 14-15 possono essere eseguiti con l'interfaccia grafica.
16. Avviare e terminare Tomcat. Questa operazione permette l'istallazione automatica della servlet nella cartella `C:/tomcat5/webapps/msh`. Eseguire `C:/tomcat5/bin/startup.bat` per avviare Tomcat (controllare che nella consolle non appaiano messaggi di errore) e `C:/tomcat5/bin/shutdown.bat` per terminare.
17. Rimuovere il file `msh.war` ed il file `webapps\msh\WEB-INF\lib\servlet.jar` poiché causa conflitto con l'omonimo file di Tomcat.
18. Riavviare Tomcat, controllare che la console non segnali errori ed asicurarsi che la servlet msh sia attiva accedendo mediante un browser alla URL http://localhost/msh/. Eventuali errori possono essere indagati esaminando il file `C:/tomcat5/webapps/msh/msh.log`.
19. Modificare la riga 301 del file CPA.xml, contenuto inserendo l'end-point del destinatario, ossia l'indirizzo del server msh che serve il coordinatore, (es. http://localhost:8080/msh/).

b) Sviluppo ed esecuzione




```
rem make.bat
set CLASSPATH=
javac -classpath lib\msh.jar;... *.java
java  -classpath lib\msh.jar;...; ebXMLClient
```

I riferimenti classpath di cui sopra contengono l'elenco di file .jar contenuti nella cartella lib.

La cartella `lib` contiene `msh.jar` e tutti i file `.jar` della cartella lib del pacchetto msh.

Fig.9File necessari e relativa composizione

```
// ebXMLClient.java;

import java.util.*;
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;

import java.net.*;
import javax.activation.*;
import javax.xml.soap.*;
import hk.hku.cecid.phoenix.message.handler.*;
import hk.hku.cecid.phoenix.message.packaging.*;
```

```
public class ebXMLClient implements MessageListener {

    private CPAParser   cpa;
    private String            conversationId;
    private Request           mshReq;

    public ebXMLClient() {
      cpa = new CPAParser("AziendaT","cpa.xml");      //preleva il cpa
        conversationId = new String ("TEST");
        ApplicationContext ac = new ApplicationContext(cpa.getCpaId(),
                                                                    conversationId,
                                                                    cpa.getService(),
                                                                    cpa.getAction()
                                                                   );
        try {
         mshReq = new Request(ac,
                                    new URL(cpa.getEndPoint()),
                                    this,
                                    cpa.getProtocol()
                                   );
        } catch (Exception e)  {}
    }

    public URL getClientUrl() {
        return null;
    }

    public void onMessage(EbxmlMessage msg) {
      System.out.println("messaggio ricevuto");

        Iterator pl = msg.getPayloadContainers();
        if (pl.hasNext()){
                  PayloadContainer tmp = (PayloadContainer)pl.next();
            AttachmentPart att = tmp.getAttachmentPart();

            String contentId = tmp.getContentId();
            String msgID = msg.getMessageId();
            try {
                DataHandler dataH= tmp.getDataHandler();
                int numByte = att.getSize();
                InputStream content = dataH.getInputStream();
                SAXBuilder builder = new SAXBuilder();
                Document doc = builder.build(content);

                XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
                System.out.println(outputter.outputString(doc));
```

```
                outputter.output(doc, new FileOutputStream("./msg/cantina.xml"));
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

  public void invia(Cantina cantina) throws Exception {
    Document doc1 = new Document(cantina.getElement());

      System.out.print("Invio del messaggio...");

      XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
      String str = outputter.outputString(doc1);
        byte [] docByte = str.getBytes();

      AttachmentDataSource ads = new AttachmentDataSource(docByte, "text/xml");

      DataHandler dataHandler = new DataHandler(ads);

      EbxmlMessage message = new EbxmlMessage();

      MessageHeader header = message.addMessageHeader();
      header.addFromPartyId(cpa.getPartyName(), "Test");
      header.addToPartyId(cpa.getDestName(), "Test");
      header.setCpaId(cpa.getCpaId());
      header.setConversationId(conversationId);
      header.setService(cpa.getService());
      header.setAction(cpa.getAction());
      header.setTimestamp(Utility.toUTCString(new Date()));

        String messageId = Utility.generateMessageId(new Date(), message);

      header.setMessageId(messageId);

      message.addPayloadContainer(dataHandler, "contentId", "description");

      message.saveChanges();
      mshReq.send(message);
  }

  public static void main(String[] args) throws Exception {
      Vino[] vini = { new Vino(   "Chianti Classico",
                                  new GregorianCalendar(1998,10,23).getTime(),
                                  new Produttore("Giorgio","Rossi"),
```

```
                                        new Prezzo(14.50)
                                    ),
                            new Vino(   "Lambrusco Frizzante",
                                                new GregorianCalendar(2000,03,15).getTime(),
                                    new Produttore("Gianni","Frizzi"),
                                    new Prezzo(20.11)
                                )
                        };
            ebXMLClient s = new ebXMLClient();
            s.invia(new Cantina("Rosso", vini));


           ////////////////////////////////////////////////////////////////////////////

    }
}
// CPAParser.java

import java.io.*;
import java.util.*;

import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;

public class CPAParser {
    /* Classe SAX Driver di default */
    private static final String DEFAULT_SAX_DRIVER_CLASS = "org.apache.xerces.parsers.SAXParser";
    private static final String DEFAULT_NAMESPACE = "http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-
2_0.xsd";
    private static final String DEFAULT_ACTION = "Test";//"Send Trace Message";
    private SAXBuilder builder;
    /* Variabili per la creazione del contesto */
    String cpaid;
    String service;
    String action;
    String protocol;
    String endpoint;
    String partyName;
    String destName;
    Namespace eb;

    public CPAParser(){
        /* inizializzazione variabili */
        cpaid = new String();
        service = new String();
```

```
        action = new String(DEFAULT_ACTION);
        protocol = new String();
        endpoint = new String();
        partyName = new String();
        destName = new String();
    }

    public CPAParser(String name){
        /* inizializzazione variabili */
        cpaid = new String();
        service = new String();
        action = new String(DEFAULT_ACTION);
        protocol = new String();
        endpoint = new String();
        partyName = new String(name);
        destName = new String();
    }

    public CPAParser(String name, String cpa){
        /* inizializzazione variabili */
        cpaid = new String();
        service = new String();
        action = new String(DEFAULT_ACTION);
        protocol = new String();
        endpoint = new String();
        partyName = new String(name);
        destName = new String();

        /* scansione del documento contenente il CPA
           per il prelievo delle informazioni necessarie. */
        builder = new SAXBuilder(DEFAULT_SAX_DRIVER_CLASS);
        eb = Namespace.getNamespace(DEFAULT_NAMESPACE);

        Element node;
        try{
            Document doc = builder.build(new File(cpa));
            node = doc.getRootElement();
            cpaid = node.getAttributeValue("cpaid",eb);
            this.scan(node);
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**
```

```
     * Questo metodo deve essere chiamato nel caso in cui non sia stato usato il
     *  costruttore CPAParser(String name, String cpa) per prelevare le informazioni
     *  dal CPA.
     *  Il metodo ritorna un booleano:
     *  - true  : se tutto è andato a buon fine.
     *  - false : se qualche stringa non è stata trovata o se variabile partyName
     *            non è stata settata. Questa variabile va settata dal costruttore
     *            o dalla funzione SetPartyName(String name).
     */
    public boolean ScanCPA(String cpa){
        if(partyName.length() == 0)
            return false;
        builder = new SAXBuilder(DEFAULT_SAX_DRIVER_CLASS);
        eb = Namespace.getNamespace(DEFAULT_NAMESPACE);
        Element node;
        try{
            Document doc = builder.build(new File(cpa));
            node = doc.getRootElement();
            cpaid = node.getAttributeValue("cpaid",eb);
            this.scan(node);
        } catch(Exception e) {
            e.printStackTrace();
        }

        if((protocol.length() == 0) || (cpaid.length() == 0) || (service.length() == 0) || (endpoint.length() == 0))
            return false;
        return true;
    }

    private boolean scan(Element node){
            if(partyName.length() == 0)
                return false;
            for(Iterator i=node.getChildren().iterator(); i.hasNext();){
                Element x = (Element)i.next();
                this.scan(x);
                if("Service".equals(x.getName()))

if(partyName.equals(((Element)(x.getParent().getParent().getParent())).getAttributeValue("partyName",eb)))
                        service = x.getText();
                    else
                        destName =
((Element)(x.getParent().getParent().getParent())).getAttributeValue("partyName",eb);
                if("TransportProtocol".equals(x.getName()) &&
"TransportSender".equals(((Element)x.getParent()).getName()))
                    protocol = x.getText();
                if("Endpoint".equals(x.getName())){
```

```
                endpoint = x.getAttributeValue("uri",eb);
                if(!endpoint.endsWith("/"))
                    endpoint += "/";
            }
        }
        return true;
    }

    public void setNameSpace(String name){
        eb = Namespace.getNamespace(name);
    }

    public String getNameSpace(String name){
        return eb.toString();
    }

    public String getCpaId(){
        return cpaid;
    }

    public String getService(){
        return service;
    }

    public String getAction(){
        return action;
    }

    public String getProtocol(){
        return protocol;
    }

    public String getEndPoint(){
        return endpoint;
    }

    public void setPartyName(String name){
        partyName = name;
    }

    public String getPartyName(){
        return partyName;
    }

    public String getDestName(){
        return partyName;
```

```
    }
}
```

c) cpa.xml

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

<!-- Nell'elemento di root è necessario definire i namespace e l'id del CPA -->
<CollaborationProtocolAgreement
    xmlns:tp="http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    tp:cpaid="CPA_2004"
    tp:version="2.0a">

<!-- Lo status può assumere tre valori
       - proposed = il corrente CPA è una proposta ma deve ancora essere concordato
       - agreed = il CPA è stato accordato ma non ancora firmato
       - signed = il CPA è stato firmato in data <Start> ed è valido fino alla data <End>
-->
<tp:Status tp:value="signed"/>
<tp:Start>2004-04-04T18:39:09</tp:Start>
<tp:End>2006-04-04T18:40:00</tp:End>

<!-- ConversationConstraints può apparire al più una volta.
     Permette di definire il numero massimo di conversazioni dopo delle quali è
     necessaria una rinegoziazione del CPA (invocationLimit) e il numero massimo
     di conversazioni contemporanee (concurrentConversations)
-->
<tp:ConversationConstraints
    tp:invocationLimit="100"
    tp:concurrentConversations="4"/>

<!-- Nel CPA vanno definiti due campi PartyInfo, uno per ogniuna delle due parti
     che hanno stipulato questo accordo.
     E' possibile definire un nome e gli id del Channel e del Package di default definiti
     più avanti nel codice.
-->
<tp:PartyInfo
    tp:partyName="AziendaT"
    tp:defaultMshChannelId="TraceChannel"
    tp:defaultMshPackageId="TracePackaging">
<!-- E' necessario definire l'id dell'azienda che può essere in vari formati.
```

```
     Nel nostro caso si è usata la codifica EAN/UCC.
-->
    <tp:PartyId tp:type="urn:oasis:names:tc:ebxml-cppa:partyid-type:ean_ucc">
        3482579
    </tp:PartyId>
<!-- <tp:PartyId>urn:icann:example.com</tp:PartyId> -->

<!-- E' possibile definire un link ad un documento esterno che fornisce informazioni
     più dettagliate riguardo all'azienda.
-->
    <tp:PartyRef
        xlink:type="simple"
        xlink:href="http://www.aziendaT.com/info.xml"/>

<!-- Bisogna specificare il ruolo assunto dall'azienda in questione all'interno della
     collaborazione e i documenti che è autorizzato ad inviare ed a ricevere.
-->
    <tp:CollaborationRole>
<!-- In ProcessSpecification va inserito il collegamento ad un file XML che definisce
     le specifiche del Business Process. Nel Nostro esempio non è utilizzato e per il
     suo uso si rimanda alla documentazione fornita da OASIS e da UNCEFACT (www.ebxml.org)
-->
        <tp:ProcessSpecification
            tp:version="2.0a"
            tp:name="SendTraceability"
            xlink:type="simple"
            xlink:href="http://www.aziendaT.com/processes/TraceBPS.xml"/>

<!-- Nel nostro caso il ruolo definito prende il nome di TraceActor.
     I dettagli di questo ruolo vanno specificati nel Business Process ma non è necessario
     nel nostro prototipo.
-->
        <tp:Role
            tp:name="TraceActor"
            xlink:type="simple"
            xlink:href="http://www.aziendaT.com/processes/TraceBPS.xml#TraceActorId"/>

<!-- Il ServiceBinding definisce il traffico di messaggi che possono essere inviati o ricevuti
     attraverso il DeliveryChannel scelto, la cui definizione è più avanti nel codice.
-->
        <tp:ServiceBinding>
<!-- Il campo Service è molto importante nel nostro prototipo, infatti esso fornisce le
     informazioni di routing per l'ebXML Message Header. Serve per indirizzare i messaggi
     ricevuti al corretto entry point dell'applicazione.
     Il Service viene definito in dettaglio nel BP.
-->
```

```
            <tp:Service tp:type="anyURI">urn:TraceService</tp:Service>

<!-- E' possibile definire i documenti che è possibile inviare. Nel nostro caso,
     l'attore responsabile può inviare un unico documento che contiene le necessarie
     informazioni per la tracciabilità della filiera.
     TraceDocumentPackage è l'id del campo Packaging definito più sotto che definisce
     alcune informazioni sui contenuti del messaggio SOAP.
     E' possibile avere più di un campo CanSend.
-->
            <tp:CanSend>
                <tp:ThisPartyActionBinding
                    tp:id="AziendaT_STD"
                    tp:action="Send Trace Document Action"
                    tp:packageId="TraceDocumentPackage">

<!-- E' possibile definire le necessarie informazioni per l'implementazione dei
     protocolli di sicurezza. Per informazioni più dettagliate su questo campo si
     rimanda alla documentazione fornita nel sito www.ebxml.org.
     Nel nostro prototipo le informazioni sulla sicurezza sono inserite manualmente
     nel codice sorgente e non vengono prelevate da questo CPA, pertanto questo campo
     non viene da noi utilizzato.
-->
                    <tp:BusinessTransactionCharacteristics
                        tp:isNonRepudiationRequired="true"
                        tp:isNonRepudiationReceiptRequired="true"
                        tp:isConfidential="transient"
                        tp:isAuthenticated="persistent"
                        tp:isTamperProof="persistent"
                        tp:isAuthorizationRequired="true"
                        tp:timeToAcknowledgeReceipt="PT2H"
                        tp:timeToPerform="P1D"/>

<!-- Queste informazioni collegano questo documento con la o le collaborazioni binarie
     definite nel documento di specifica del BP.
-->
                    <tp:ActionContext
                        tp:binaryCollaboration="Send Trace Document"
                        tp:businessTransactionActivity="Send Trace Document"
                        tp:requestOrResponseAction="Send Trace Document Action"/>

<!-- ChanneId contiene l'id del canale di trasmissione utilizzato per l'invio del
     documento. Questo campo è definito più in basso.
-->
                    <tp:ChannelId>TraceChannel</tp:ChannelId>
                </tp:ThisPartyActionBinding>
            </tp:CanSend>
```

```
        </tp:ServiceBinding>
    </tp:CollaborationRole>

<!-- In questo campo è necessario definire le informazioni sul cetificato che andranno
     inserite nei messaggi SOAP.
-->
    <tp:Certificate tp:certId="AziendaTSigningCert">
        <ds:KeyInfo>
            A6VCsfd9TNN1208cjus078V0NgjVB0DF3v56J5JM0YBFgR80CWH40VjJH2UR05Kl7GRV
            j12gh3896RVG5h60jetrN3Y509Rj7Ek305Bj92RJTV893RHG4127HF9GV5456HY524C0
            dsf87G79E7W1Tlj5khF2HMOT82f5896kjhJYGFUY4476SDFrgfdD564FD767ff8H87H9
            0j998FD5Dw423w4D5g79G8H09j9j09ki9H7Tf5e6S43s45XrybHnluH9786T657EDUG7
            iTu66FHYhm67Tvc45e6BDEf75gh87J8kgfb7FV55fG8HM98h9G76f565V6D5n7j8H7bg
            6uiI00OImn4Cyhni7J86MGF6H7n75RF8REc7698B09GBumk8K8989G65fr7yt9PkjLo9
        </ds:KeyInfo>
    </tp:Certificate>

<!-- Il campo DeliveryChannel collega un elemento Transport ad un elemento DocExchange
     e definisce le caratteristictiche di comunicazione
-->
    <tp:DeliveryChannel
        tp:channelId="TraceChannel"
        tp:transportId="TraceTransport"
        tp:docExchangeId="TraceDocExchange">

<!-- Questo campo descrive alcune informazioni riguardanti i messaggi di Acknowledgement.
     Anche per questo ri rimanda alla documentazione di ebxml.
     Nel nostro prototipo non vengono trattate.
-->
        <tp:MessagingCharacteristics
            tp:syncReplyMode="mshSignalsOnly"
            tp:ackRequested="always"
            tp:ackSignatureRequested="always"
            tp:duplicateElimination="always"
            tp:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"/>
    </tp:DeliveryChannel>

<!-- Il campo Tranport definisce i protocolli di trasporto e di sicurezza utilizzati. -->
    <tp:Transport tp:transportId="TraceTransport">

<!-- Il campo TransportSender, se presente, definisce la capacita di inviare messaggi
     attraverso il protocollo specificato.
-->
        <tp:TransportSender>
            <tp:TransportProtocol tp:version="1.1">HTTP</tp:TransportProtocol>
            <tp:TransportClientSecurity>
```

```
                <tp:TransportSecurityProtocol tp:version="3.0">
                    SSL
                </tp:TransportSecurityProtocol>
                <tp:ClientCertificateRef tp:certId="AziendaTSigningCert"/>
            </tp:TransportClientSecurity>
        </tp:TransportSender>
    </tp:Transport>

<!-- Il campo DocExchange fornisce informazioni utili per lo scambio dei messaggi e le
     proprietà del Message Service.
-->
    <tp:DocExchange tp:docExchangeId="TraceDocExchange">
        <tp:ebXMLSenderBinding tp:version="2.0">

<!-- In questo campo è possibile definire il numero di tentativi per l'invio di un
     messaggio e l'intervallo tra un tentativo e l'altro (PT2H = Periodo di 2 ore)
-->
            <tp:ReliableMessaging>
                <tp:Retries>5</tp:Retries>
                <tp:RetryInterval>PT2H</tp:RetryInterval>
                <tp:MessageOrderSemantics>Guaranteed</tp:MessageOrderSemantics>
            </tp:ReliableMessaging>

<!-- Qui è possibile inserire informazioni sui protocolli per la firma digitale. -->
            <tp:SenderNonRepudiation>
                <tp:NonRepudiationProtocol>
                    http://www.w3.org/2000/09/xmldsig#
                </tp:NonRepudiationProtocol>
                <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:SigningCertificateRef tp:certId="aziendaTSigningCert"/>
            </tp:SenderNonRepudiation>

<!-- Qui è possibile inserire informazioni sui protocolli per il crittaggio dei messaggi. -->
            <tp:SenderDigitalEnvelope>
                <tp:DigitalEnvelopeProtocol tp:version="2.0">
                    S/MIME
                </tp:DigitalEnvelopeProtocol>
                <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
                <tp:EncryptionSecurityDetailsRef
                    tp:securityId="aziendaTSigningCert"/>
            </tp:SenderDigitalEnvelope>
```

```
        </tp:ebXMLSenderBinding>
    </tp:DocExchange>
</tp:PartyInfo>

<tp:PartyInfo
    tp:partyName="Coordinator"
    tp:defaultMshChannelId="TraceChannel"
    tp:defaultMshPackageId="TracePackaging">
    <tp:PartyId tp:type="urn:oasis:names:tc:ebxml-cppa:partyid-type:ean_ucc">
        3047264
    </tp:PartyId>
    <tp:PartyRef
        xlink:type="simple"
        xlink:href="http://www.coordinatore.com/info.xml"/>
    <tp:CollaborationRole>
        <tp:ProcessSpecification
            tp:version="2.0a"
            tp:name="ReceiveTraceability"
            xlink:type="simple"
            xlink:href="http://www.coordinatore.com/processes/TraceBPS.xml"/>
        <tp:Role
            tp:name="Coordinator"
            xlink:type="simple"
            xlink:href="http://www.coordinatore.com/processes/TraceBPS.xml#CoordinatorId"/>
        <tp:ServiceBinding>
            <tp:Service tp:type="anyURI">urn:TraceService</tp:Service>

<!-- Questo campo è simile al CanSend utilizzato nel PartyId dell'attore responsabile
     ma serve per definire i messaggi che è possibile ricevere.
-->
            <tp:CanReceive>
                <tp:ThisPartyActionBinding
                    tp:id="Coordinaotr_RTD"
                    tp:action="Receive Trace Document Action"
                    tp:packageId="TraceDocumentPackage">
                    <tp:BusinessTransactionCharacteristics
                        tp:isNonRepudiationRequired="true"
                        tp:isNonRepudiationReceiptRequired="true"
                        tp:isConfidential="transient"
                        tp:isAuthenticated="persistent"
                        tp:isTamperProof="persistent"
                        tp:isAuthorizationRequired="true"
                        tp:timeToAcknowledgeReceipt="PT2H"
                        tp:timeToPerform="P1D"/>
                    <tp:ActionContext
                        tp:binaryCollaboration="Receive Trace Document"
```

```
                    tp:businessTransactionActivity="Receive Trace Document"
                    tp:requestOrResponseAction="Receive Trace Document Action"/>
                <tp:ChannelId>TraceChannel</tp:ChannelId>
            </tp:ThisPartyActionBinding>
        </tp:CanReceive>
    </tp:ServiceBinding>
</tp:CollaborationRole>
<tp:Certificate tp:certId="CoordinatorSigningCert">
    <ds:KeyInfo>
        A6VCsfd9TNN1208cjus078V0NgjVB0DF3v56J5JM0YBFgR80CWH40VjJH2UR05Kl7GRV
        j12gh3896RVG5h60jetrN3Y509Rj7Ek305Bj92RJTV893RHG4127HF9GV5456HY524C0
        dsf87G79E7W1Tlj5khF2HMOT82f5896kjhJYGFUY4476SDFrgfdD564FD767ff8H87H9
        0j998FD5Dw423w4D5g79G8H09j9j09ki9H7Tf5e6S43s45XrybHnluH9786T657EDUG7
        iTu66FHYhm67Tvc45e6BDEf75gh87J8kgfb7FV55fG8HM98h9G76f565V6D5n7j8H7bg
        6uiI00OImn4Cyhni7J86MGF6H7n75RF8REc7698B09GBumk8K8989G65fr7yt9PkjLo9
    </ds:KeyInfo>
</tp:Certificate>
<tp:DeliveryChannel
    tp:channelId="TraceChannel"
    tp:transportId="TraceTransport"
    tp:docExchangeId="TraceDocExchange">
    <tp:MessagingCharacteristics
        tp:syncReplyMode="mshSignalsOnly"
        tp:ackRequested="always"
        tp:ackSignatureRequested="always"
        tp:duplicateElimination="always"
        tp:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"/>
</tp:DeliveryChannel>
<tp:Transport tp:transportId="TraceTransport">
    <tp:TransportReceiver>
        <tp:TransportProtocol tp:version="1.1">HTTP</tp:TransportProtocol>
        <tp:Endpoint
            tp:uri="http://localhost:8080/msh"
            tp:type="allPurpose"/>
        <tp:TransportServerSecurity>
            <tp:TransportSecurityProtocol tp:version="3.0">
                SSL
            </tp:TransportSecurityProtocol>
            <tp:ServerCertificateRef tp:certId="CoordinatorSigningCert"/>
        </tp:TransportServerSecurity>
    </tp:TransportReceiver>
</tp:Transport>
<tp:DocExchange tp:docExchangeId="TraceDocExchange">
    <tp:ebXMLReceiverBinding tp:version="2.0">
        <tp:ReliableMessaging>
            <tp:Retries>5</tp:Retries>
```

```
                <tp:RetryInterval>PT2H</tp:RetryInterval>
                <tp:MessageOrderSemantics>Guaranteed</tp:MessageOrderSemantics>
            </tp:ReliableMessaging>
            <tp:ReceiverNonRepudiation>
                <tp:NonRepudiationProtocol>
                    http://www.w3.org/2000/09/xmldsig#
                </tp:NonRepudiationProtocol>
                <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:SigningSecurityDetailsRef tp:certId="aziendaTSigningCert"/>
            </tp:ReceiverNonRepudiation>
            <tp:ReceiverDigitalEnvelope>
                <tp:DigitalEnvelopeProtocol tp:version="2.0">
                    S/MIME
                </tp:DigitalEnvelopeProtocol>
                <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
                <tp:EncryptionCertificateRef
                    tp:certId="CoordinatorSigningCert"/>
            </tp:ReceiverDigitalEnvelope>
        </tp:ebXMLReceiverBinding>
    </tp:DocExchange>
</tp:PartyInfo>

<!-- E' possibile definire i Content Type che costituiscono i messaggi -->
<tp:SimplePart tp:id="SP1" tp:mimetype="text/xml"/>
<tp:SimplePart tp:id="SP2" tp:mimetype="*/xml"/>

<!-- Il campo Packaging evidenzia le varie componenti del messaggio SOAP da inviare -->
<tp:Packaging tp:id="TracePackaging">
    <tp:ProcessingCapabilities tp:generate="true" tp:parse="true"/>

<!-- Il campo CompositeList indica come combinare i SimplePart -->
    <tp:CompositeList>

<!-- Encapsulation contiene un campo crittato che, nel nostro caso, è il messaggio
     di tracciabilità che è di tipo SP1 ed un altro messaggio che viene urilizzato per
     le informazioni sulla qualità sempre di tipo SP1 (text/xml)
-->
        <tp:Encapsulation
            tp:id="EP1"
            tp:mimetype="text/xml"
            tp:mimeparameters="smime-type=&quot;enveloped-data&quot;">
```

```
            <Constituent tp:idref="SP1"/>
            <Constituent tp:idref="SP1"/>
        </tp:Encapsulation>
    </tp:CompositeList>
</tp:Packaging>
</CollaborationProtocolAgreement>
```