Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

eRIC v3.2

ebXML Registry by ICAR CNR

A. Messina, P. Storniolo

Rapporto Tecnico N.:
RT-ICAR-PA-13-03 **Dicembre 2013**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sede di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

eRIC v3.2

ebXML Registry by ICAR CNR

A. Messina[1], P. Storniolo[1]

Rapporto Tecnico N.:
RT-ICAR-PA-13-03 **Dicembre 2013**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo, Viale delle Scienze edificio 11, 90128 Palermo.

I rapporti tecnici dell'ICAR-CNR sono pubblicati dall'Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche. Tali rapporti, approntati sotto l'esclusiva responsabilità scientifica degli autori, descrivono attività di ricerca del personale e dei collaboratori dell'ICAR, in alcuni casi in un formato preliminare prima della pubblicazione definitiva in altra sede.

Sommario

1 Prefazione

1.1 Introduzione

Il modello di *e-business* ha introdotto l'idea del controllo automatico di tutti i processi aziendali al fine di monitorare ogni tipo di attività grazie alla infrastuttura pervasiva fornita dalla rete Internet. Molte organizzazioni stanno rivedendo i propri processi di business in base ai futuri sviluppi e alle potenzialità offerte da Internet.

Il modello di e-business ha portato alla formulazione di un metalinguaggio (*eXtensible Markup language, XML*) per la definizione di grammatiche sui diversi contesti di business e l'introduzione di framework tecnologici quali "Web Services (WS) e-business using XML (ebXML)" che consentono di esprimere gli aspetti semantici ed operativi del business mascherando il substrato tecnologico ed adoperando il web come supporto.

Il sistema informativo di un'azienda dipende dalla organizzazione aziendale. La descrizione accurata dei processi aziendali è alla base per la realizzazione dei sistemi informatici. Un sistema informatico, per essere modellato dai processi aziendali deve essere innanzitutto modulare, estendibile e caratterizzato da altissimo livello di integrazione e di interoperabilità.

I processi aziendali sono generalmente scomponibili in attività più elementari, per cui occorre seguire dei pattern di tipo gestionale a cui riferirsi per analizzarli ed identificarli. Il progettista del sistema informatico traduce le attività, modellate come servizi a livello di business, in sottosistemi interagenti secondo pattern di tipo tecnologico.

Dal punto di vista degli utenti, le organizzazioni nella filiera necessitano di un metodo standard per definire e registrare processi di business, scambiare messaggi di business, condurre relazioni di commercio, comunicare dati in termini comuni. In tale direzione, ebXML offre un insieme di standard per i processi di business, componenti di business riusabili, protocolli di accordo, messaggistica, registri ed archivi.

Dal punto di vista dei progettisti, costruire un sistema di tracciabilità coinvolge tutti gli stadi di produzione, trasformazione e distribuzione. Occorre necessariamente uno stile architetturale che garantisca l'interoperabilità e la portabilità dei componenti software in rete. In questa prospettiva, SOA rappresenta un paradigma che assicura una cooperazione effettiva tra agenti software eterogenei.

L'ebXML è in grado di poter gestire un'ampia comunità di partecipanti, amministrare risorse centrali per consentire ai partner di unirsi rapidamente in comunità ed integrare le loro proprie applicazioni in una rete condivisa di servizi.

Constatata l'assenza in rete di una implementazione open-source aggiornata, si è ritenuto opportuno mettere a punto una implementazione che mettesse a disposizione della comunità di sviluppatori una versione del registro secondo le specifiche OASIS ebXML Registry 3.0 e che supportasse, per quanto concerne la gestione delle politiche di sicurezza, le modalità definite da SAML 2.0 (Security Assertion Markup Language).

2 Architettura di ebXML

2.1 Introduzione

ebXML (Electronic Business Extensible Markup Language) è un progetto internazionale ideato da UN/CEFACT (United Nation Center for Trade Facilitation and Electronic Business) e da OASIS (Organization for the Advancement of Structured Information Standard).

L'obiettivo del progetto è quello di fornire un framework che permetta al linguaggio XML di essere utilizzato in modo uniforme e consistente nello scambio di messaggi tra applicativi e utenti in un contesto di commercio elettronico.

Il vantaggio fornito da ebXML è apprezzabile anche per aziende di piccola e media grandezza e che abbiano un livello di informatizzazione anche minimo; la possibilità che viene fornita da ebXML è uno scambio di messaggi basati su XML che favorisca il commercio elettronico della ditta con altre strutture (della stessa società o di ditte diverse) geograficamente localizzate in luoghi diversi.

Questo vantaggio è garantito da un vocabolario di termini comuni che permette una comunicazione commerciale con messaggistica di tipo standard.

Una delle caratteristiche di ebXML è quella di fornire contatti d'affari *ad hoc*.

Vediamo un esempio per chiarificare il concetto.

Supponiamo di comperare ortaggi da un supermarket A. Dopo diverso tempo abbiamo sviluppato una relazione d'affari con il supermarket, basato sui prodotti forniti e sul processo intrapreso per comprarli, come l'interazione con i commessi e il pagamento tramite carta di credito dei beni. Possiamo definire questa relazione *ad hoc*.

Supponiamo ora che entri nel mercato un nuovo supermarket B, che venda lo stesso prodotto a prezzi inferiori. Sarebbe nel nostro interesse intraprendere degli affari con questo nuovo supermarket, e possiamo farlo perché ci aspettiamo di non incontrare ostacoli: i commessi si comporteranno come nel primo supermercato (parlano la nostra stessa lingua) e possiamo pagare la merce con la carta di credito.

Il commercio elettronico, in realtà, ha costi di infrastrutture che influiscono sul prezzo totale dell'affare. Durante una transazione elettronica, per esempio, i partecipanti devono affrontare i costi del software, del trasporto dati, così come decidere le politiche di interazione e sicurezza.

In pratica, se un nuovo fornitore offrisse beni a minor prezzo, potrebbe accadere di dover considerare i costi del servizio di acquisto come elemento discriminante. Ad un minor costo del prodotto potrebbe infatti corrispondere un maggior costo o complicazione del processo di acquisto.

Tornando all'esempio precedente, se il supermercato B avesse solo commessi stranieri, che parlassero esclusivamente una lingua diversa dalla nostra e non venissero accettati pagamenti con carta di credito ma solo con contanti e di piccolo taglio, è facile comprendere che il risparmio ottenuto sul bene acquistato andrebbe perso nell'adeguamento alle condizioni di vendita imposte.

Uno dei valori basilari di ebXML è la possibilità di ubiquità da una prospettiva tecnologica. E' costruito su *XML*, *SOAP*, *HTTP* e *SMTP*, tutti standard aperti e senza barriere. In teoria, mettere al centro l'ubiquità della tecnologia, dovrebbe permettere al commercio elettronico un approccio *ad hoc* al concetto di libero mercato che abbiamo visto nell'esempio del supermercato.

2.2 Elementi di ebXML

Gli elementi che compongono la struttura di ebXML sono molteplici:

- *Registry*: un server centrale che contiene una varietà di dati necessari per il funzionamento di ebXML. Tra le varie informazioni che il *Registry* mette a disposizione di XML troviamo: *Business Process & Information Meta Models*, *Core Library*, *Collaboration Protocol Profiles*, e *Business Library*. Il Registry in ebXML è necessario per individuare un partner adatto e per ricavare informazioni riguardo agli elementi necessari per rapportarsi con quel partner.

- *Business Processes*: attività che un'impresa può intraprendere (e per la quale sta cercando uno o più partner). Un Business Process è formalmente descritto da uno Schema di Specifiche (Business Process Specification Schema: uno Schema XML e un DTD), ma potrebbe essere modellato anche in UML (Unified Modeling Language).

- *Collaboration Protocol Profile (CPP)*: profilo archiviato in un *Registry* da un'impresa che vorrebbe intraprendere delle transizioni ebXML. Il CPP specifica alcuni Business Process dell'impresa e anche alcune Business Service Interface supportate.

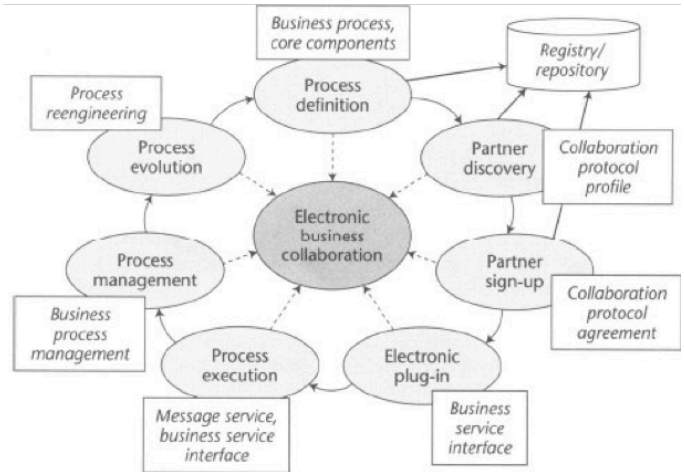
- *Business Service Interface*: i modi in cui un'impresa è in grado di portare avanti le transazioni necessarie nei suoi Business Process. La Business Service Interface include inoltre i tipi di Business Message supportati e i protocolli sui quali questi messaggi dovranno viaggiare.

- *Business Message*: le informazioni correnti comunicate come parte della transazioni d'affari. Un messaggio contenente diversi strati. Al livello più esterno deve essere utilizzato in protocollo di comunicazione (tipo HTTP o SMTP). SOAP viene raccomandato come envelope (busta, strato esterno) per un messaggio "payload". Gli altri strati possono concernere criptazione o autenticazione.

- *Core Library*: un insieme di "parti" standard che possono essere utilizzate in elementi ebXML più grandi. Per esempio i Core Process possono essere referenziati dai Business Process. Alla Core Library contribuisce l'iniziativa ebXML stessa, mentre agli elementi più grandi possono contribuire specifiche imprese o industrie.

- *Collaboration Protocol Agreement (CPA)*: in pratica, un contratto tra due o più imprese che può essere derivato automaticamente dai CPP delle rispettive compagnie. Per esempio se un CPP afferma "Io *posso* fare X", un CPA afferma "Noi *faremo* X insieme".

- *Simple Object Access Protocol (SOAP)*: un protocollo di W3C per lo scambio di informazioni in un ambiente distribuito sottoscritto dall'iniziativa ebXML. La parte di SOAP interessante per ebXML è la funzione di envelope che definisce un framework per la descrizione di cosa contiene il messaggio e come processarlo.



2.3 Funzionamento di ebXML

Per ottenere un approccio ad hoc, come visto precedentemente, ebXML fornisce un framework completo per le interazioni d'affari, le cui informazioni vengono inviate come un insieme di specifiche indipendenti dal partecipante all'affare. Il framework cerca di soddisfare i seguenti punti:

- Descrivere il processo e le interfacce specifiche
- Condividere il processo con altri partner
- Conoscere quali processi può supportare il partner
- Descrivere i messaggi per una particolare transizione
- Descrivere la politica di sicurezza e la configurazione tecnica da usare

Molte di queste informazioni possono essere salvate in un registro condiviso che ha lo scopo di rendere centralizzati gli accordi dell'affare e i processi, l'*ebXML registry*.

Accordarsi per una nuova relazione d'affari significa accedere al registry di ebXML che, solitamente, viene gestita dalla fase di funzionamento corrente.

L'architettura di base prevede tre fasi:

1. fase di implementazione
2. fase di ricerca
3. fase di esecuzione

Per ogni fase sono definiti processi e meccanismi di sicurezza propri. In generale si possono immaginare le prime due fasi come un'ipotetica stretta di mano tra i due contraenti l'affare, mentre la terza come la reale attuazione dello scambio commerciale.

2.3.1 Fase di implementazione

Inizia con la decisione dei partner di svolgere una transizione d'affari tramite il framework ebXML. Il partner analizza i suoi processi e li pubblica nel *registry* preoccupandosi di renderli compatibili con la generalizzazione fornita da ebXML.

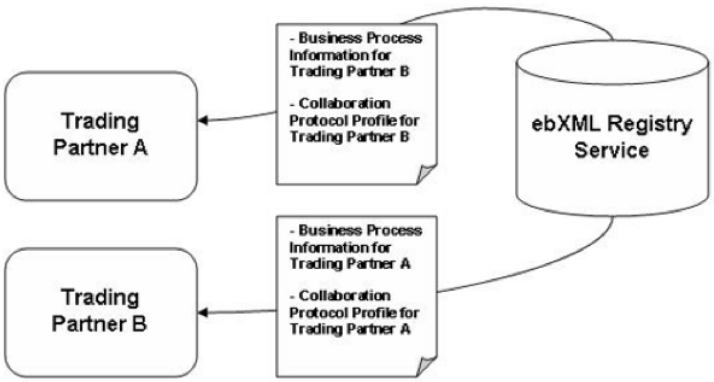
In questa fase deve essere prodotta un'implementazione, o dalle specifiche del nucleo di ebXML stesso o da terzi produttori.

Come risultato si ottiene un framework funzionante contenente un insieme di processi e interfacce. Il *Collaboration Protocol Profile* (CPP) viene creato in questo momento.

2.3.2 Fase di ricerca

In questa fase i partner consultano il *registry* per vedere (scoprire) i processi e le interfacce pubblicate dall'altro partner. Tipicamente i CPP di un partner specifico vengono scambiati in questa fase.

Il CPP descrive i dettagli dei processi, compresi quelli riguardanti la sicurezza, il trasporto e l'affidabilità.

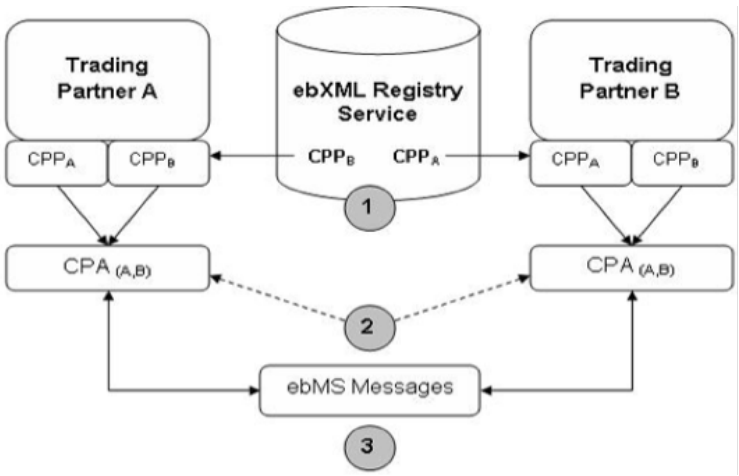


2.3.3 Fase di esecuzione

Gli accessi al registry sono terminati nelle fasi precedenti, per cui in questa avvengono esclusivamente gli scambi di messaggi tra i contraenti.

Le istanze di CPP pubblicate da ciascuno vengono ristrette per formare un Accordo sul *Collaboration Protocol Agreement* (CPA). Questo è un particolare accordo legato alle richieste formulate dai singoli CPP. P

rima di iniziare lo scambio dei messaggi con ebMS, solitamente i due contraenti si preoccupano di verificare la consistenza del CPA creato: quando si verifica da entrambi i lati la consistenza, comincia lo scambio di messaggi.



Facendo riferimento alla figura precedente possiamo riassumere la fase di runtime in 3 passi:

- Ogni partner è responsabile di ottenere i documenti CPP necessari per accordarsi con gli altri partner. Queste informazioni vengono ottenute tramite l'accesso ad un Registry.

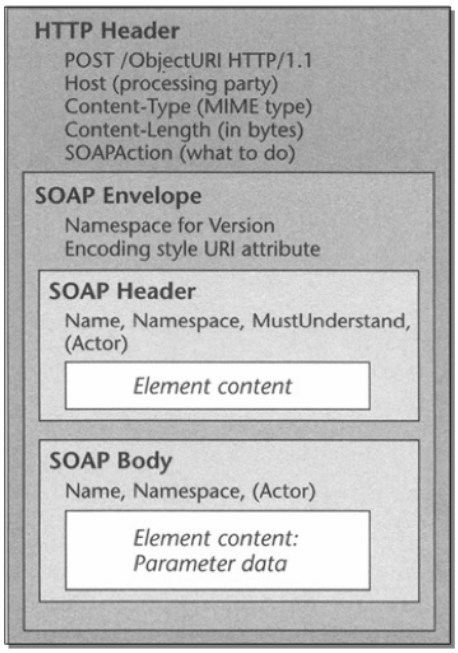
- Con i CPP ottenuti vengono creati i CPA, che esplicitano la gamma di scelte offerte nei CPP.

- Sotto il controllo del CPA, i partner cominciano le transazioni tramite ebMS.

2.4 Formato dei messaggi ebXML

Il servizio incaricato della spedizione dei messaggi (ebMS) definisce una serie di elementi che verranno inseriti all'interno dell'intestazione (Header) o del corpo (Body) che soddisfano le specifiche SOAP per rendere il messaggio compatibile con le specifiche definite da ebXML.

Per ottenere ciò si dovranno incapsulare questi elementi in un multipart MIME che permetterà di inserire degli allegati nello stesso messaggio SOAP:



dove:

- **HTTP Header**: Rappresenta il contenitore di tutti il messaggio e specifica i protocolli di trasferimento

- **SOAP Envelope**: Rappresenta il contenitore del messaggio

- **SOAP Header**: È un elemento opzionale che contiene informazioni globali sul messaggio; ad esempio, nell'header potrebbe essere specificata la lingua di riferimento del messaggio, la data dell'invio, ecc.

- **SOAP Body**: Rappresenta il contenuto vero e proprio del messaggio.

I principali obiettivi di ebXML a livello di messaging layer sono:

- Facilitare l'interscambio di messaggi di business all'interno di un framework XML

- Essere neutrale rispetto al protocollo usato

- Essere neutrale rispetto al contenuto del messaggio
- Essere sicuro
- Poter recapitare i messaggi in maniera affidabile

Basandosi su questi obiettivi, ebXML ha prodotto la *ebXML Message Service Specification* che definisce uno standard per il trasporto sicuro ed affidabile dei messaggi.

I messaggi sono caratterizzati dai seguenti punti:

- Il loro formato è scritto in XML (è adottato il formato "SOAP with Attachments") .
- Non viene imposto alcun protocollo di trasporto da utilizzare.
- I Messaggi di Business, identificati dal corpo (payload) dei messaggi ebXML, non sono necessariamente espressi in XML. I messaggi XML-based sono trasportati dall'ebMS.

2.5 Registry e Repository

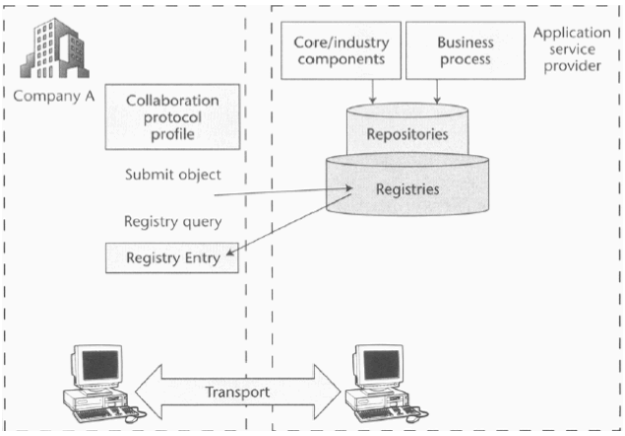
Il *Registry* fornisce una locazione stabile in cui rendere persistenti le informazioni ivi depositate da un apposito organismo. Tali informazioni vengono usate per facilitare partnership e transazioni Business to Business (B2B) basate su ebXML.

Il materiale sottoposto può consistere di schemi e documenti XML, descrizioni di processi, *ebXML Core Component*, descrizioni di contesto, modelli UML, informazioni varie circa le parti commerciali e persino componenti software.

Nel *Registry* vengono memorizzate le informazioni degli oggetti che sono depositati nel *Repository*. I due insieme possono essere considerati un database. Visti separatamente, il *Repository* contiene i modelli dei processi, i *core components*, i messaggi predefiniti e tutte quelle informazioni utili per permettere alle parti di scambiarsi informazioni in via elettronica.

Il *Registry* serve a collegare il *Repository* al mondo esterno; contiene quindi il software necessario per far in modo che i partner possano accedere alle informazioni presenti nel *Repository*.

Gli oggetti presenti nel *Repository* possono essere creati, aggiornati, cancellati solo con richieste fatte attraverso il *Registry*.



2.6 Core components

Per garantire l'interoperabilità fra diverse organizzazioni è necessario definire e creare una semantica di business comune (*Common Business Semantics*). ebXML ha quindi tentato di descrivere e specificare un nuovo approccio per risolvere il già noto problema dell'interoperabilità delle informazioni nel panorama e-business.

Tradizionalmente, gli standard per lo scambio di dati di business si sono sempre focalizzati su definizioni statiche dei messaggi, le quali non hanno però apportato un sufficiente grado di interoperabilità e flessibilità. Si è reso quindi necessario un nuovo e più flessibile metodo di standardizzazione.

Viene presentata una metodologia per sviluppare un insieme comune di componenti semantici elementari che rappresentano i tipi generali di dati di business correntemente in uso.

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.

L'approccio scelto è flessibile in quanto la standardizzazione viene portata avanti secondo un approccio "syntax-neutral", indipendente dalla sintassi.

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*.

Ciò permette un mapping semplice e pulito fra diversi tipi di definizioni di messaggi attraverso sintassi, confini industriali e geografici.

2.7 Collaboration Protocol Profile (CPP)

Lo scambio di informazioni fra due controparti richiede che ciascuna di esse sia a conoscenza delle collaborazioni binarie o multiparte (*Business Collaboration*) supportate dall'altra parte, quali il ruolo da questa ricoperto all'interno della collaborazione ed i dettagli tecnologici circa il modo in cui tale parte invia e riceve i messaggi. In alcuni casi, è necessario che le due controparti raggiungano un accordo su tali dettagli.

Per facilitare questo scambio di informazioni, ebXML descrive il *Collaboration Protocol Profile* (CPP) ed il *Collaboration Protocol Agreement* (CPA). Queste due tecnologie sono descritte in un'unica specifica.

Il CPP definisce le potenzialità ed i modi in cui una parte può impegnarsi in business elettronico con altre parti. Queste potenzialità sono sia tecnologiche (protocolli di messaggistica e di comunicazione supportati) sia commerciali (quali *Business Collaboration* supporta).

Una parte può descrivere se stessa in un unico CPP. Oppure, può creare differenti CPP che descrivono le *Business Collaboration* supportate, le sue operazioni in varie regioni del mondo, o le diverse sezioni della propria organizzazione.

Per facilitare la ricerca di possibili Business Partner, i CPP sono 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.

2.8 Collaboration Protocol Agreement (CPA)

Il *Collaboration Protocol Agreement* (CPA) definisce le potenzialità messe a disposizione dalle due controparti, sulle quali esse devono trovarsi in accordo prima di impegnarsi nelle transazioni descritte dal CPA.

Il CPA è indipendente dai particolari processi interni attuati da ciascuna parte. Questi processi interni vengono interfacciati verso l'esterno con le *Business Collaboration* descritte dal CPA e dal documento di Process-Specification (descritto nel BPSS).

In tal modo non vengono esposti alla controparte dettagli dei processi interni.

L'intento del CPA è di fornire una specifica ad alto livello che può essere facilmente comprensibile ed allo stesso tempo rigorosa e precisa da poter essere implementata in maniera automatica.

L'informazione di un CPA è usata per configurare i sistemi dei due partner commerciali per rendere possibile lo scambio di messaggi durante lo svolgimento delle *Business Collaboration* selezionate.

I CPA non sono documenti cartacei bensì elettronici, che possono essere processati direttamente dai sistemi delle controparti per preparare ed eseguire gli scambi di informazioni voluti. Eventuali termini e condizioni legali di un accordo commerciale sono al di là degli scopi di queste specifiche e quindi non sono inclusi né nel CPP né nel CPA.

Il CPA è creato mediante elaborazioni e negoziazioni derivanti dall'incrocio di due CPP. Ad esempio: un CPA può includere solamente quegli elementi che sono comuni o compatibili fra le due parti.

3 OASIS Regrep 3.0

3.1 Introduzione

Un Registro ebXML è un sistema informativo in grado di gestire in sicurezza qualunque tipo di contenuto digitale (*RepositoryItem*) e i metadati standardizzati che lo descrivono (*RepositoryObject*) e fornisce inoltre un set di servizi che consentono l'accesso e la condivisione dei contenuti e dei metadati tra entità organizzative in un ambiente federato.

Le specifiche OASIS *ebXML Registry Information Model* (RIM) definiscono i tipi di metadati ed i contenuti che possono essere memorizzati in un Registro ebXML, mentre le specifiche OASIS *ebXML Registry: Services and Protocols* (RS), definiscono i servizi erogati da un Registro ebXML ed i protocolli utilizzabili dai client del registro per interagire con tali servizi.

3.2 Registry Information Model (RIM)

L'*ebXML Registry Information Model* (RIM) definisce le classi e le loro relazioni utilizzate per rappresentare i metadati (*RegistryObject*).

In RIM la classe *RegistryObject* ed alcune altre classi sono derivate da una classe chiamata *Identifiable*, che permette di identificare gli oggetti mediante un attributo identificativo e consente l'estensibilità degli attributi prevedendo degli attributi dinamici denominati *Slot*.

Le sottoclassi di *RegistryObject* possono essere invece raggruppate nelle seguenti tipologie:

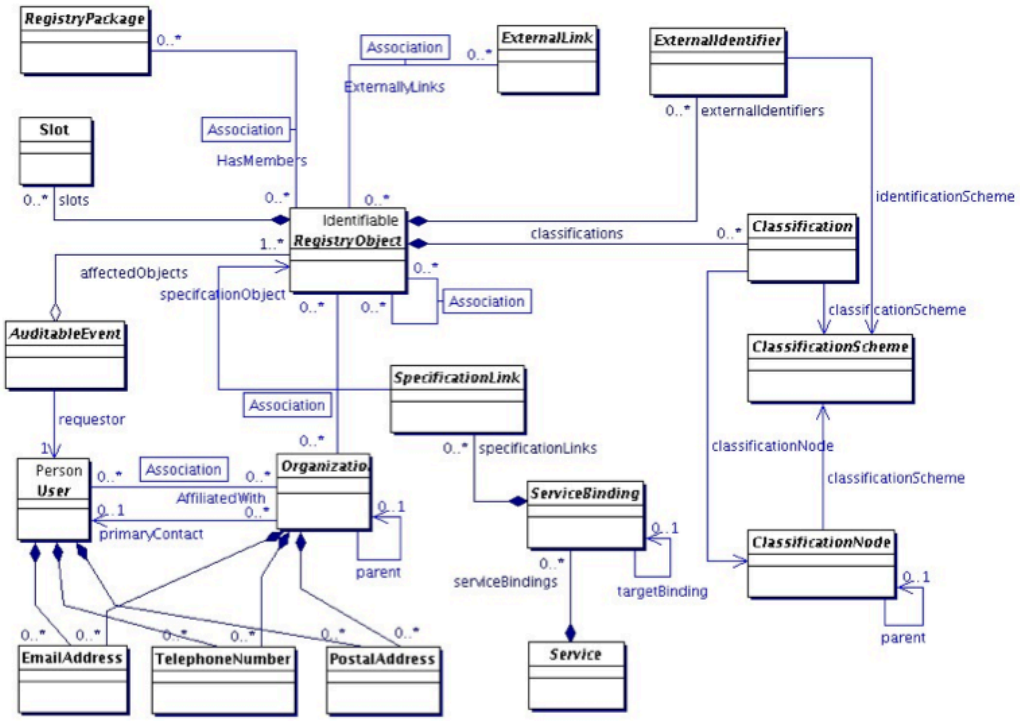
- *Core Information Model*: definisce le classi metadati nel modello di tipo core, incluse la classi base comuni;
- *Association Information Model*: definisce le classi che consentono alle istanze dei *RegistryObject* di essere associate le une alle altre;
- *Classification Information Model*: definisce le classi che consentono ai RegistryObject di essere classificati;
- *Provenance Information Model*: definisce la classi che consentono la descrizione della provenienza o la sorgente informativa di un *RegistryObject*;

- *Service Information Model*: definisce le classi che consentono la descrizione dei servizi;
- *Event Information Model*: definisce le classi che consentono la sottoscrizione agli eventi e le funzionalità di notifica come definite nell'*ebXML Registry: Services and Protocols* (RS).

3.2.1 Vista delle relazioni tra le classi

La figura seguente mostra una panoramica ad alto livello delle classe definite dal modello in termini delle loro relazioni di tipo "Has-A", come nei diagrammi di classe UML.

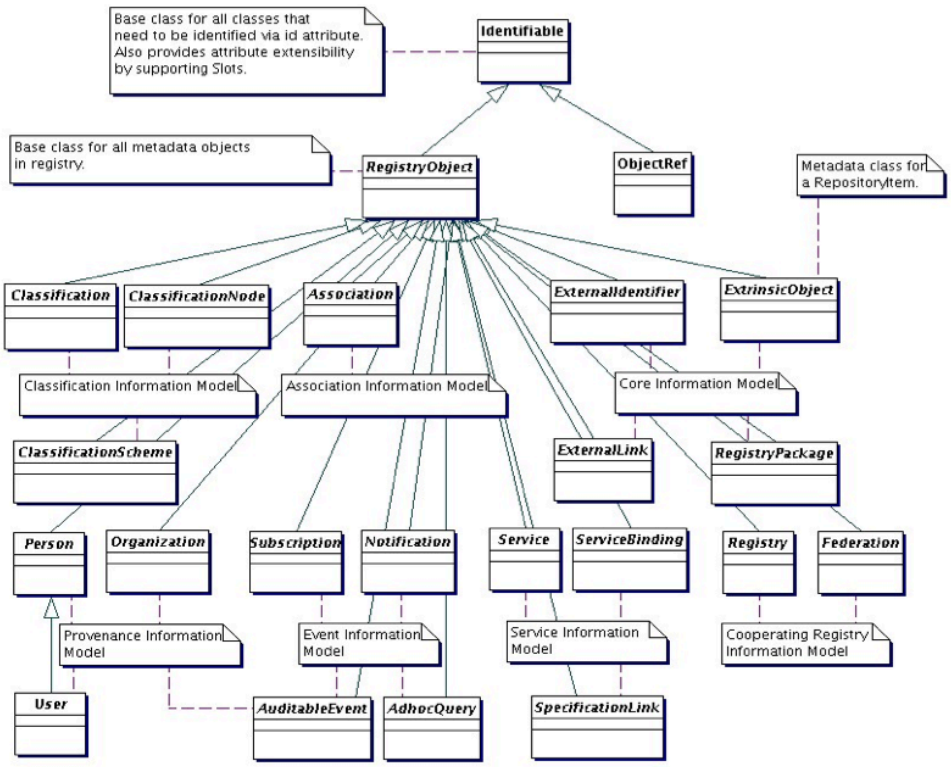
Non vengono mostrate le relazioni di tipo "His-A" (relazioni di ereditarietà) e nemmeno gli attributi delle classi.



3.2.2 Ereditarietà tra le classi

La figura seguente mostra le relazioni di ereditarietà tra le classi di tipo "Is-A".

Non vengono mostrati altri tipi di relazioni, come la “Has-A” vista precedentemente, e nemmeno gli attributi delle classi.



3.3 Registry Services (RS)

Un registro ebXML è accessibile mediante le seguenti interfacce di servizio:

- Un’interfaccia di tipo *LifeCycleManager*, che fornisce un insieme di operazioni per la gestione del ciclo di vita dei metadati e dei contenuti all’interno del registro. Tra queste troviamo le operazioni di pubblicazione, aggiornamento, approvazione e cancellazione dei metadati e dei contenuti;

- Un’interfaccia di tipo *QueryManager*, che fornisce un insieme di operazioni per la ricerca e la lettura dei metadati e dei contenuti dal registro.

Le specifiche prevedono le interazioni con il registro avvengano esclusivamente mediante i protocolli SOAP e HTTP:

- con SOAP si consente ai client l’accesso al registro utilizzando il protocollo *SOAP 1.1 with Attachments*;

- con HTTP si consente ai client di tipo web browser l’accesso al registro utilizzando il protocollo *HTTP 1.1*.

3.3.1 Autenticazione e Autorizzazione

Un registry client dovrebbe essere autenticato dal registro per determinare l'identità associata e tipicamente questa sarà l'identità di un utente del Registro.

Una volta determinata l'identità, il Registro dovrà effettuare i controlli di autorizzazione e controllo degli accessi prima di consentire l'elaborazione delle richieste del client.

4 OMAR v3.1

4.1 Introduzione

Il progetto OMAR (Object, Metadata and Artifacts Registry), rilasciato nel 2006 nella sua ultima sua release ufficiale v3.1, è stato sviluppato con l'obiettivo di fornire una implementazione di riferimento completa delle specifiche OASIS ebXML Registry, così come definite dall'OASIS ebXML Registry Technical Committee.

Tra le sue caratteristiche principali troviamo:

1. un server conforme alle specifiche OASIS ebXML Registry 3.0;
2. una API JAXR conforme alle specifiche JAXR;
3. un'applicazione Registry Browser di tipo Java UI;
4. un'applicazione Registry Browser di tipo Web UI;
5. diversi tools per la gestione dei contenuti e dei metadati presenti nel registro ebXML.

4.2 Funzionalità standard

OMAR implementa tutte le features mandatorie ed opzionali così come richiesto dalle specifiche OASIS ebXML Registry 3.0, ad eccezione del SAML 2 based Single Sign On (SSO). Di seguito si riportano le ultime e più importanti funzionalità implementate richieste dalle specifiche ebXML Registry.

4.2.1 Role Based Access Control

Il server OMAR supporta politiche di controllo degli accessi (ACP) di tipo custom, compresa quelle di tipo Role Based Access Control (RBAC), così come specificato

nel capitolo 9 “Access Control Information Model” delle specifiche ebXML Registry Information Model.

Questa funzionalità consente ai client di specificare delle regole personalizzate per proteggere l’accesso ai RegistryObject e RepositoryItem pubblicati. Ad esempio, consente di definire una policy per consentire l’accesso in lettura ad una risorsa solo da parte di un utente, gruppo di utenti o ruoli. Oppure, ne consente la modifica o la cancellazione solo ad altri utenti, gruppi o ruoli.

Un RegistryObject può essere associato ad una ACP mediante una associazione speciale di tipo *AccessControlPolicyFor*. Questa associazione ha un riferimento all’ExtrinsicObject che rappresenta l’ACP come valore del proprio sourceObject e un riferimento del RegistryObject come valore dell’attributo targetObject.

Se un RegistryObject non ha una ACP esplicitamente associata, allora esso sarà implicitamente associato alla ACP di default definita per il registro.

4.2.2 Interfaccia HTTP al Registro

Il server OMAR è dotato di un’interfaccia di accesso tramite protocollo http, così come indicato nella sezione 4 “HTTP Binding” delle specifiche ebXML Registry Service Specification.

Il registro offre le seguenti modalità per accedere via HTTP ai RegistryObject ed ai ReporitoryItem:

- **RPC Encoding URL**: consente l’accesso dei client tramite una URL basata sulla codifica di una Remote Procedure Call all’interfaccia del registro come richiesta HTTP.

- **Submitter defined URL**: consente l’accesso agli oggetti tramite URL definite dal submitter.

- **File Path Based URL**: consente l’accesso agli oggetti tramite una URL di default basata su un percorso derivato dalla membership dell’oggetto nell’ambito della gerarchia dei RegistryPackage.

4.2.3 RPC Encoding URL

Per accedere ad un RegistryObject identificato dall'id `urn:uuid:e3373a7b-4958-4e55-8820-d03a191fb76a` si può utilizzare direttamente nel browser web una URL del tipo:

http://localhost:8080/omar/registry/http/?interface=QueryManager&method=getRegistryObject&param-id=urn:uuid:e3373a7b-4958-4e55-8820-d03a191fb76a

Per accedere invece ad un RepositoryItem identificato dall'id `urn:uuid:e3373a7b-4958-4e55-8820-d03a191fb76a` si può utilizzare direttamente nel browser web una URL del tipo:

http://localhost:8080/omar/registry/http/?interface=QueryManager&method=getRepositoryItem&param-id=urn:uuid:e3373a7b-4958-4e55-8820-d03a191fb76a

4.2.4 Submitter defined URL

OMAR consente ad un utente di specificare una URL per ogni RegistryObject o RepositoryItem che pubblica. Tale URL potrà quindi essere utilizzata per accedere all'oggetto tramite l'interfaccia HTTP in alternativa alla URL con indicazione dei parametri per il servizio QueryManager. In tal modo un oggetto sarà accessibile mediante una URL più significativa e facile da ricordare per l'utente.

Per i RegistryObject tale funzionalità si traduce nella possibilità di utilizzare una URL del tipo:

http://localhost:8080/omar/registry/http/pictures/nikola/zeusDescription

in luogo della tipica

http://localhost:8080/omar/registry/http/?interface=QueryManager&method=getRegistryObject&param-id=urn:uuid:e3373a7b-4958-4e55-8820-d03a191fb76a

Per i RepositoryItem invece avremo URL del tipo:

http://localhost:8080/omar/registry/http/pictures/nikola/zeus

in luogo di

http://localhost:8080/omar/registry/http/?interface=QueryManager&method=getRepositoryItem&param-id=urn:uuid:e3373a7b-4958-4e55-8820-d03a191fb76a

Per utilizzare questa feature bisogna operare nel seguente modo:

- Per assegnare una URL ad un RegistryObject un client deve aggiungere uno slot di nome:

```
urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:locator
```

- Per assegnare una URL ad un RepositoryItem un client deve aggiungere uno slot di nome:

```
urn:oasis:names:tc:ebxml-
regrep:rim:RegistryObject:RegistryEntry:ExtrinsicObject:contentLocator
```

- Il valore dello Slot specifica la URL, indicata come URL relativa rispetto alla URL base del registro. Ad esempio:

  URL base del Registro: http://localhost:8080/omar/registry

  URL prefisso per l'interfaccia HTTP: http://localhost:8080/omar/registry/http

  URL relativa: `/pictures/nikola/zeus`

  URL di accesso: http://localhost:8080/omar/registry/http/pictures/nikola/zeus

Un oggetto può avere più URL diverse associate e ogni URL deve essere unica nell'ambito del registro.

4.3 Funzionalità extra

Oltre alle features mandatorie richieste dalle specifiche ebXML Registry 3.0, in OMAR 3.1 sono state implementate anche le seguenti ulteriori funzionalità:

- Registrazione automatica dell'utente
- Supporto API JAXR
- Registry Browser – Web UI
- Registry Browser – Java UI
- Esecuzione Client-Server embedded
- Generatore UUID
- Modalità "Do not commit"

4.3.1 Registrazione automatica dell'utente

Le specifiche non indicano quale possa essere la procedura di registrazione degli utenti del registro/repository.

Per tale ragione, nel Browser di Registro basato su Java è fornito un semplice wizard che aiuta ad automatizzare la registrazione utente.

Per registrare un utente si utilizza il seguente meccanismo:

1. Si controlla che il certificato conenuto nell'elemento di percorso `/Envelope/Header/Signature/KeyInfo/X509Data[0]/X509Certificate` di un messaggio di tipo *SubmitObjectRequest* sia già presente nel server keystore, il cui percorso è specificato dalla proprietà `omar.security.keystoreFile` in *omar.properties*

2. Se il certificato è già presente nel server keystore, allora l'utente è già registrato nel registro.

3. Se il certificato non è presente, si verifica che questo sia stato rilasciato da una certification autority fidata. Allo stato attuale, quando viene invocata la classe *SOAPSender* per inviare una richiesta firmata, a questa viene allegata l'intera chain del certificato. Ogni certificato nella chain è allegato come elemento del path `/Envelope/Header/Signature/KeyInfo/X509Data` e quando viene verificato il certificato, viene ricostruito il suo chain path.

4. Se la verifica fallisce, fallisce anche la registrazione dell'utente. Se la verifica ha successo, il server controlla che il campo *SubmitObjectRequest* contenga un solo oggetto di tipo user: quando si registra un nuovo utente l'identificativo dell'oggetto user deve essere un UUID reale, perché gli identificativi temporanei non vengono accettati. L'organizzazione di riferimento inoltre deve essere già presente all'interno del registro. La richiesta può contenere anche altri oggetti, di cui però soltanto uno di tipo user contenente le informazioni del nuovo utente.

5. Se le condizioni nel punto precedente sono verificate, il certificato viene installato nel server keystore, quindi la prossima volta che il client invia una richiesta al server, il server non inizierà di nuovo la procedura di registrazione utente. Inoltre, l'oggetto user viene memorizzato nel registro.

4.3.2 Supporto API JAXR

Il progetto freebXML registry è dotato di un provider JAXR (Java API for XML Registries) di livello 1. In tal modo si incoraggia la realizzazione delle applicazioni ebXML client utilizzando le API JAXR.

4.3.3 Esecuzione Client-Server embedded

Il provider JAXR supporta la possibilità di eseguire il processo server (il registro) embedded nella JVM del processo del provider JAXR stesso. Ciò torna utile quando il client deve garantire delle particolari condizioni di alte prestazioni.

Per attivare tale modalità, bisogna impostare a true la seguente proprietà del file conf/jaxr-ebxml.properties:

```
org.freebxml.omar.client.xml.registry.localCall=false
```

4.3.4 Generatore UUID

Il server OMAR fornisce un servizio generatore di UUID esposto mediante l'interfaccia HTTP. Una richiesta a tale servizio deve essere effettuata tramite una URL del tipo:

```
http://localhost:8080/omar/registry/http?interface=QueryManager&method=newUUID
```

A livello di API invece, il supporto per la generazione di UUID è fornito dal seguente metodo:

```
org.freebxml.omar.common.Utility.getInstance().createId();
```

4.3.5 Modalità "Do not commit"

Per poter effettuare test di interazione con il registro in modalità sicura e non distruttiva, è stata implementata una particolare modalità che consente di processare le richieste ma di non effettuare veramente le modifiche.

Questa modalità, denominata “Do not commit”, viene attivata aggiungendo alla richiesta uno slot speciale nel modo seguente:

```
<SubmitObjectsRequest ...>

  <rim:Slot name="urn:oasis:names:tc:ebxmlregrep::rs:RegistryRequest:doNotCommit">

    <rim:ValueList>

      <rim:Value>true</rim:Value>

    </rim:ValueList>

  </rim:Slot>

  …

  …

</SubmitObjectsRequest>
```

4.4 Installazione e Setup

I requisiti di sistema necessari per il corretto funzionamento di Omar sono:

- JDK 1.4.2/1.5
- JWSDP 1.6
- Tomcat 5.x
- Database relazionale con supporto alla sintassi SQL-92, come Derby, HSQLDB, PostgreSQL 7 e Oracle 9.

La procedura di installazione consigliata prevede il download dei sorgenti di Omar mediante il servizio CVS:

```
cd /opt/
export CVSROOT=:pserver:anonymous@ebxmlrr.cvs.sourceforge.net:/cvsroot/ebxmlrr
cvs login
cvs -z3 co -P omar
```

Impostate le variabili d’ambiente relative a Java e a Tomcat, si crea il file

`/opt/omar/local.build.propertes`

all’interno del quale vengono indicate le direttive personalizzate per il processo di build.

Nell’esempio seguente si assume che Tomcat sia installato all’interno della directory `/opt/tomcat`, che le librerie JWSDP siano disponibili all’interno della directory

`/opt/jwsdp-1.6` e che venga utilizzato come DBMS di backend PostgreSQL installato sulla stessa macchina:

```
omar.name=omar
dist.version=3.1
omar.home=/opt/${omar.name}/${dist.version}
omar.home.template=/opt/${omar.name}/${dist.version}
compile.target=1.5
catalina.home=/opt/tomcat
database=postgresql
dbDialect=net.sf.hibernate.dialect.PostgreSQLDialect
dbLargeBinaryType=binary
dbLargeBinaryTypePropLength=length\="2147483647"
dbName=${omar.name}
dbParamsFile=
dbUsername=omar
dbPassword=omar
dbTransactionIsolation=TRANSACTION_READ_COMMITTED
dbURL=jdbc:postgresql://localhost:5432/${omar.name}
jdbcClassName=org.postgresql.Driver
jdbcDriver=postgresql.jar
dbDeploymentJars=${jdbcDriver}
dbShutdownURL=${dbURL}
jwsdp.home=/opt/jwsdp-1.6
war.includes.webstart=true
appserver-jndi-context.filename=jndi-context-5.5.x.xml
```

Inoltre, bisogna modificare il file `/opt/omar/conf/web.xml.template` aggiungendo tra i tag `<web-app></web-app>` la seguente sezione:

```
<listener>
    <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
</listener>
```

A questo punto si può effettuare la compilazione di OMAR, creare il certificato RSA del server, fare il deploy dell'applicazione e popolare il database:

```
cd /opt/omar
./build.sh compile
./build.sh genKeys
./build.sh stop.tomcat
./build.sh deploy
./build.sh cleandb
./build.sh createDemoDB
./build.sh start.tomcat
./build.sh createTestUser
```

5 eRIC v3.2

5.1 Introduzione

Come si è visto nel capitolo precedente, una possibile implementazione di registro ebXML è costituita dal prodotto open-source denominato OMAR (Objects, Metadata and Artifacts Registry) nella sua ultima versione disponibile 3.1, rilasciata nel 2006.

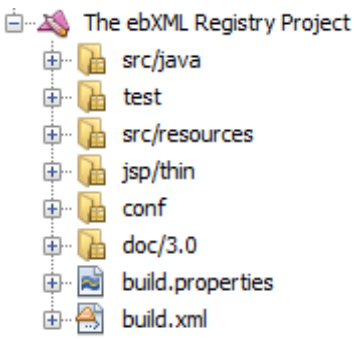
OMAR implementa le features mandatorie ed opzionali così come richiesto dalle specifiche OASIS ebXML Registry 3.0, ad eccezione del SAML 2 based Single Sign On (SSO).

Pur essendo perfettamente funzionante e conforme alle specifiche di riferimento, dal 2006 il progetto OMAR, nato come progetto monolitico NetBeans 4.x di tipo free-form basato su ANT, non è stato più ulteriormente sviluppato né supportato.

Poiché si è ritenuto essenziale per gli utilizzi attuali e futuri un adeguamento tecnologico della implementazione di OMAR e per soddisfare pienamente anche i requirements relativi alla sicurezza, si è ritenuto opportuno effettuare un vero e proprio fork del progetto originale rilasciando una nuova implementazione pienamente conforme alle specifiche OASIS ebXML Registry 3.0: *eRIC* (ebXML Registry by ICAR CNR).

5.2 Implementazione di OMAR v3.1

OMAR nasce come progetto monolitico NetBeans 4.x di tipo free-form basato su ANT:

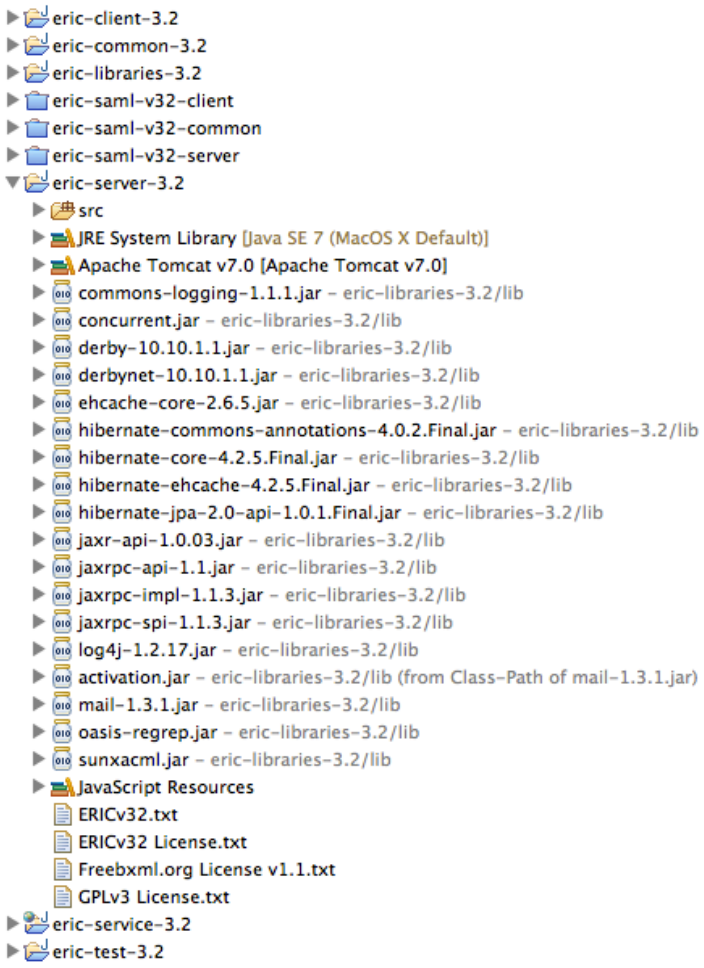


ed è caratterizzato dall'utilizzo delle seguenti tecnologie ormai obsolete e difficilmente integrabili in infrastrutture e servizi moderni:

- Java 1.4 a 32 bit
- JAXB (Java Architecture for XML Binding) 1.x
- JWSDP (Java Web Services Developer Pack) 1.x
- Hibernate 2.x
- Apache Tomcat 5.5

5.3 Implementazione di eRIC v3.2

Il progetto eRIC, nella sua attuale versione di riferimento 3.2, nasce invece come progetto modulare sviluppato con Eclipse Juno 2013:



ed è caratterizzato dalle seguenti features e tecnologie:

- Java 7 a 64 bit;
- JAXB (Java Architecture for XML Binding) 2.x;
- JWSDP (Java Web Services Developer Pack) 2.x;
- Apache Web Services Security (WSS4J);
- Hibernate 4.x;
- Supporto ai database PostgreSQL 9.x / MySQL 5.x / Oracle 11gR2 / Derby 10.x;
- Apache Tomcat 7.x;
- Supporto SAML 2 (mancante in OMAR).

Per avere un'idea della complessità del progetto, basti pensare che questo è composto da più di 1300 files per un totale di più di 140000 linee di codice, come meglio esplicitato nella tabella seguente:

Project	Code Lines	Packages	Files
eric-client-3.2	51495	39	558
eric-common-3.2	12466	37	132
eric-libraries-3.2	0	0	46
eric-saml-client-3.2	204	8	8
eric-saml-common-3.2	1517	9	9
eric-saml-server-3.2	1277	10	10
eric-server-3.2	49640	47	214
eric-service-3.2	7546	0	157
eric-test-3.2	13987	49	108
oasis-regrep-3.0	4391	14	105
	142523	213	1347

5.4 Sviluppo di eRIC

Come accennato nel paragrafo precedente, eRIC è stato sviluppato seguendo un approccio modulare, separando il codice su più progetti Eclipse e impostando, in fase di configurazione delle proprietà di ogni singolo progetto, le relazioni di interdipendenza tra un progetto e gli altri.

Il salto "generazionale" operato sulle tecnologie utilizzate e soprattutto sulla versione del linguaggio Java ha reso necessaria un'attività di verifica puntuale su ogni singola linea del codice originario al fine di risolvere i circa 900 errori e 8700 warnings che immediatamente si sono presentati in fase di impostazione iniziale dei progetti.

5.4.1 Generic types

Per ottenere un codice più robusto e maggiori controlli già a livello di compilazione, è stato evitato, ove possibile, l'utilizzo dei *raw types*, introducendo al loro posto i *generic types*, disponibili a partire dalla versione 5.0 di Java, che consentono di parametrizzare in modo puntuale classi ed interfacce e forniscono un strumento di controllo statico molto espressivo ed estremamente sofisticato.

Ad esempio, il metodo *getObjectRefsInRegistryObject* presente in Omar nella classe *org.freebxml.omar.common.BindingUtility* e definito come:

```
public Set getObjectRefsInRegistryObject
   (RegistryObjectType ro, Map idMap, Set processedObjects, int depth)
   throws JAXRException {

}
```

in eRIC è contenuto nella classe *it.cnr.icar.eric.common.BindingUtility* ed è diventato:

```
public Set<ReferenceInfo> getObjectRefsInRegistryObject
   (RegistryObjectType ro, Map<?, ?> idMap,
    Set<RegistryObjectType> processedObjects, int depth)
   throws JAXRException {

}
```

O ancora, nella classe *org.freebxml.omar.server.common.ServerRequestContext* di Omar alcuni attributi sono definiti come:

```
…

public class ServerRequestContext extends CommonRequestContext {

    …
    private Map topLevelObjectsMap = new HashMap();
    private Set newSubmittedObjectIds = null;
    private Map newROVersionMap = new HashMap();
    private Map newRIVersionMap = new HashMap();
    private Set referencedInfos = null;
    private SortedSet checkedRefs = new TreeSet();
    private SortedMap fetchedOwners = new TreeMap();
    …

}
```

Nella corrispondente classe *it.cnr.icar.server.common.ServerRequestContext* di eRIC, invece, diventano:

```
…

public class ServerRequestContext extends CommonRequestContext {

    …
    private Map<String, RegistryObjectType> topLevelObjectsMap = new HashMap<String,
RegistryObjectType>();
    private Set<String> newSubmittedObjectIds = null;
    private  Map<RegistryObjectType,  RegistryObjectType>  newROVersionMap  =  new
HashMap<RegistryObjectType, RegistryObjectType>();
    private   Map<RepositoryItem,   RepositoryItem>   newRIVersionMap   =   new
HashMap<RepositoryItem, RepositoryItem>();
    private  HashMap<Object,  Object>  composedObjectsMap  =  new  HashMap<Object,
Object>();
    private Set<Object> referencedInfos = null;
    private SortedSet<String> checkedRefs = new TreeSet<String>();
    private SortedMap<String, String> fetchedOwners = new TreeMap<String, String>();
    …

}
```

5.4.2 Marshalling/Unmarshalling con JAXB 2.x

L'utilizzo di JAXB 2.x ha comportato la reimplementazione di tutti i metodi in cui si richiedono le operazioni di marshalling e unmarshalling dei tipi complessi definiti in OASIS ebXML 3.0, necessarie per la trasmissione degli stessi nell'ambito delle comunicazione tramite webservices.

Effettuare l'unmarshalling di un documento XML significa creare un albero di oggetti che rappresenta il contenuto e l'organizzazione del documento. Gli oggetti creati sono istanze delle classi prodotte dal binding compiler. Per effettuare un'operazione di unmarshalling occorre istanziare un oggetto *JAXBContext*, un oggetto *Unmarshaller* e invocare il metodo per effettuare l'unmarshalling.

Il *JAXBContext* fornisce l'entry point per le JAXB API. Occorre specificare il contesto, che rappresenta una lista di uno o più nomi di package che contengono le interfacce generate da un binding compiler. Se gli schemi utilizzati sono ben strutturati, è sufficiente passare il nome del package contentente la classe contenitore del documento, ossia l'entry point.

L'oggetto *Unmarshaller* controlla il processo di unmarshalling, fornendo un metodo che invocato permette di effettuare l'unmarshalling del documento XML passato sotto forma di File. Ciò che viene restituito è un oggetto, oggetto che può essere convertito nella classe contenitore con una operazione di casting. Questo aspetto è importante perché consente di creare una classe generica utilizzabile per l'unmarshalling di tutti i documenti XML.

Il marshalling è l'operazione opposta dell'unmarshalling. Partendo da un albero di oggetti JAXB crea il corrispondente documento XML. Più in generale, ciò che consente di fare è serializzare oggetti Java in XML. Detto altrimenti, l'oggetto viene convertito su un supporto di memorizzazione lineare, che può essere un file o un'area di memoria, e/o per trasmetterlo su una connessione di rete.

Un esempio di utilizzo del marshalling avviene appunto nei webservices: gli oggetti passati come argomenti delle operazioni remote vengono serializzati per poter essere trasmessi sulla connessione di rete.

Come già accennato, si è reso necessario reimplementare i metodi in cui sono richieste le operazioni di marshalling e unmarshalling.

Ad esempio, il metodo *cloneRegistryObject* presente in Omar nella classe *org.freebxml.omar.common.BindingUtility* e implementato nel modo seguente:

```
public RegistryObjectType cloneRegistryObject(RegistryObjectType ro)
       throws JAXRException {
    RegistryObjectType roNew = null;
    try {
        StringWriter sw = new StringWriter();
        rimFac.createMarshaller().marshal(ro, sw);
        roNew = (RegistryObjectType)rimFac.createUnmarshaller().unmarshal(
                new StreamSource( new StringReader( sw.toString())) );
    } catch (javax.xml.bind.JAXBException e) {
        throw new JAXRException(e);
    }
   return roNew;
}
```

in eRIC è contenuto nella classe *it.cnr.icar.eric.common.BindingUtility* ed è diventato:

```
public RegistryObjectType cloneRegistryObject(RegistryObjectType ebROType)
       throws JAXRException {

    RegistryObjectType ebRegistryObjectTypeNew = null;
    try {
        StringWriter sw = new StringWriter();
        // wrap ComplexType as Element
        JAXBElement<RegistryObjectType> ebRegistryObject =
                                        rimFac.createRegistryObject(ebROType);
        Marshaller marshaller = jaxbContext.createMarshaller();
        marshaller.marshal(ebRegistryObject, sw);
        Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();

        @SuppressWarnings("unchecked")
        JAXBElement<RegistryObjectType> ebRegistryObjectNew =
            (JAXBElement<RegistryObjectType>) unmarshaller
                    .unmarshal(new StreamSource(new StringReader(sw.toString())));

        // take ComplexType from Element
        ebRegistryObjectTypeNew = ebRegistryObjectNew.getValue();

    } catch (javax.xml.bind.JAXBException e) {
        throw new JAXRException(e);
    }
    return ebRegistryObjectTypeNew;
}
```

5.4.3 Apache Web Services Security for Java

Oltre agli interventi sul codice già presentati nei sottoparagrafi precedenti e riguardanti l'utilizzo dei *generic types* e le operazioni di marshalling/unmarshalling in ambiente JAXB 2.x, un'ulteriore importante attività ha riguardato la sostituzione dei meccanismi di autenticazione e di sicurezza nell'invocazione dei webservice mediante SOAP.

Nel progetto OMAR la sicurezza dello scambio di messaggi SOAP viene garantita dall'utilizzo del framework *XWS-Security*, contenuto nel Java Web Service Developer Pack 1.x.

Corentemente con i più moderni standard sulla sicurezza dei webservices, obbedienti alle specifiche OASIS Web Services Security (WS-Security), in eRIC si è scelto di utilizzare *Apache WSS4J* (Web Services Security for Java), che fornisce la piena implementazione dei seguenti standard WS-Security:

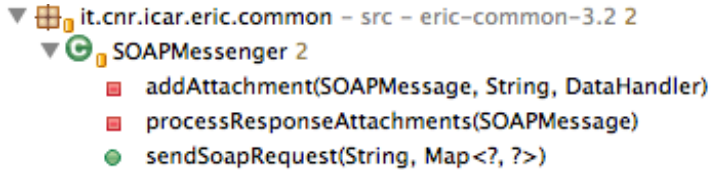
- SOAP Message Security 1.1
- Username Token Profile 1.1
- X.509 Certificate Token Profile 1.1
- SAML Token Profile 1.1
- Kerberos Token Profile 1.1
- SOAP Messages with Attachments Profile 1.1
- Basic Security Profile 1.1

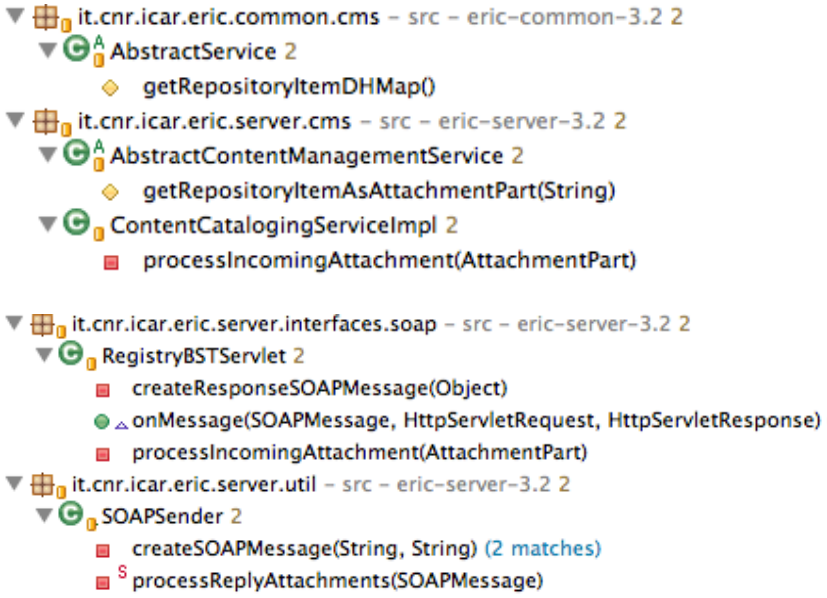
Sono state quindi introdotte sotto *it.cnr.icar.eric.common.security.wss4j* le seguenti classi:

- *WSS4JSecurityUtilBase*
- *WSS4JSecurityUtilBST*
- *WSS4JSignatureBST*

che sostituiscono quanto contenuto in *org.freebxml.omar.common.security.xwssec* e *org.freebxml.omar.common.security.xwssec20FCS*.

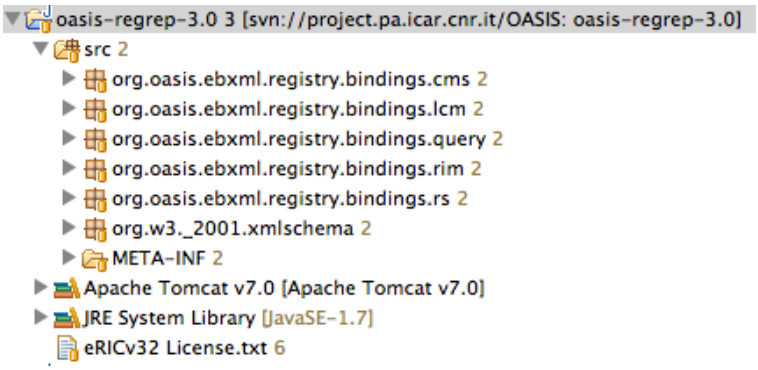
Di conseguenza è stato anche modificato il codice che faceva riferimento a XWS-Security introducendo le opportune invocazioni alle nuove classi nei seguenti metodi:





5.4.4 oasis-regrep-3.0

Contiene i nuovi bindings JAXB 2.x per tutti le classi di oggetti definite nelle specifiche OASIS ebXML 3.0:



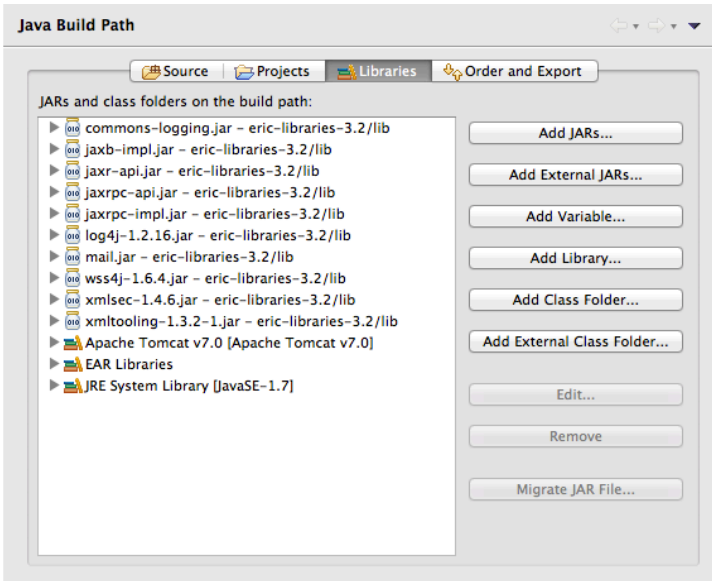
Va sottolineato che già in OMAR è presente compilata una libreria simile contenuta nel file *oasis-regrep.jar*, ma in questa viene anche staticamente forzato l'utilizzo di JAXB 1.x mediante la definizione dell'attributo `javax.xml.bind.context.factory` all'interno dei file *jaxb.properties*:

```
javax.xml.bind.context.factory=com.sun.xml.bind.ContextFactory_1_0_1
```

5.4.5 eric-libraries

Funge da repository unico per tutti i file di libreria utilizzati dai vari moduli. I file *.jar* sono contenuti nella directory *lib* del progetto ed i riferimenti a essi vengono impostati mediante opportuna configurazione del *Java Build Path* dei vari progetti.

Ad esempio, per *eric-common* si ha:



5.4.6 eric-common, eric-client ed eric-server

Costituiscono gli equivalenti diretti dei tre rami in cui è suddiviso il progetto OMAR: *omar-common*, *omar-client* e *omar-server*.

La struttura dei package è rimasta immutata, ma è stato effettuato a monte il refactoring dei loro nomi:

```
org.freebxml.omar.common  ⇒  it.cnr.icar.eric.common

org.freebxml.omar.client  ⇒  it.cnr.icar.eric.client

org.freebxml.omar.server  ⇒  it.cnr.icar.eric.server
```

Anche su eRIC sono stati implementati i tools di amministrazione ad alto livello del registro ebxml già disponibili in OMAR, come il *Web Registry Browser*, il *Java Registry Browser* e gli *Admin Tools*.

5.4.7 eric-saml-common, eric-saml-client ed eric-saml-server

In questi progetti è stato implementato il supporto al protocollo di autenticazione SAML 2, mancante in OMAR.

SAML è uno standard OASIS nato nel 2002 la cui attuale versione (v2.0) risale al 2005.

Questo protocollo definisce due attori principali:

- i *Service Provider*, ovvero le applicazioni web che desiderano utilizzare il protocollo per l'autenticazione dei propri utenti,

- gli *Identity Provider*, ovvero le applicazioni web che gestiscono la fase di autenticazione per conto dei *Service Provider* ad essi integrati.

La comunicazione tra questi attori avviene tramite messaggi XML, che vengono scambiati attraverso semplici GET e/o POST HTTP governate dal browser dell'utente.

In particolare sono stati definiti due costrutti principali:

- <*AuthnRequest*>: utilizzati dai Service Provider per inviare richieste di autenticazione agli Identity Provider, e

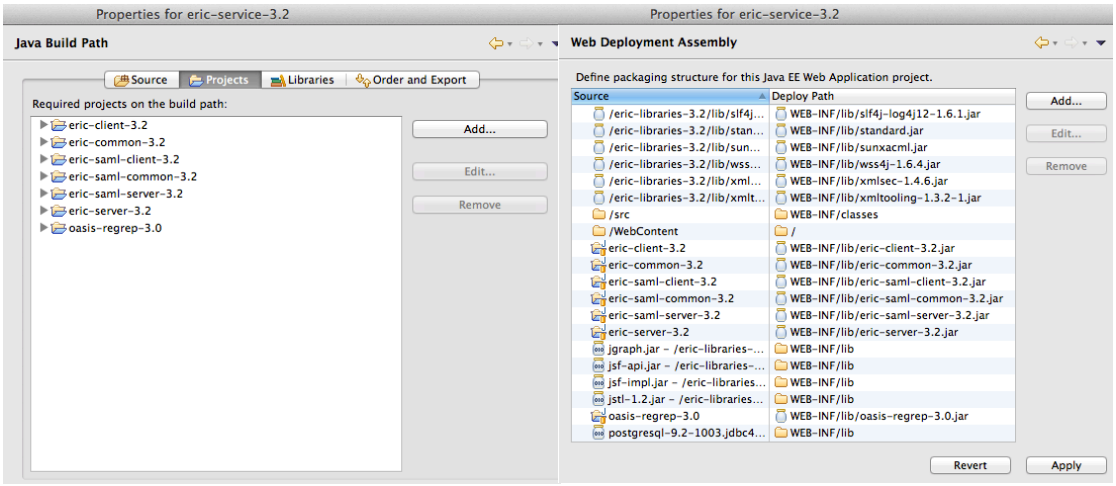
- <*Response*>: utilizzati dagli Identity Provider per comunicare l'esito dell'autenticazione ai Service Provider.

Ovviamente delegare la fase di autenticazione ad un IdP (Identity Provider) esterno, potenzialmente gestito da persone differenti dai gestori dei SP (Service Provider), significa far circolare in rete messaggi contenenti informazioni personali (o persino sensibili) che potrebbero essere raccolte o addirittura modificate da malintenzionati, per questo SAML permette la cifratura delle informazioni personali e la firma digitale dei messaggi <*AuthnRequest*> e <*Response*> scambiati, oltre a consigliare vivamente l'utilizzo di connessioni sicure (HTTPS).

5.4.8 eric-service

Poiché eRIC, come d'altra parte OMAR, è un'applicazione che viene eseguita all'interno di un container come Tomcat, si è ritenuto opportuno creare un ulteriore progetto Eclipse di tipo Java EE Web Application in modo da semplificare le operazioni di build, deploy e debug.

Nel *Java Build Path* sono state rese esplicite le dipendenze dagli altri progetti mentre nel *Web Deployment Assembly* è stata definita la struttura di packaging della web application:

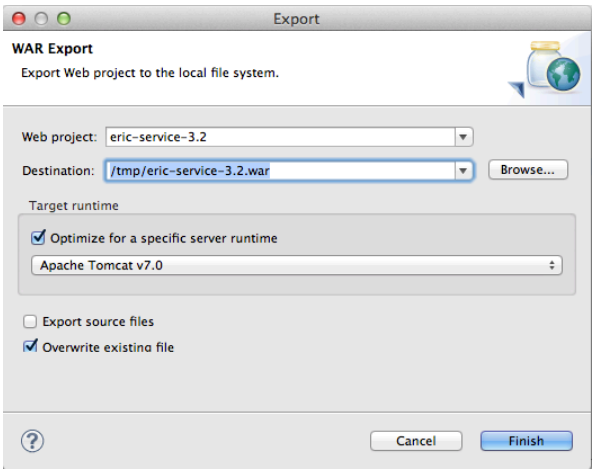


La definizione del deployment descriptor *web.xml* è stata fatta direttamente all'interno di Eclipse:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">

  <display-name>eric-service-3.2</display-name>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <resource-ref>
    <description>DB Connection for server</description>
    <res-ref-name>jdbc/eric-registry</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>

  <servlet>
    <servlet-name>ebxmlrr-http-receiver-servlet</servlet-name>
    <servlet-class>it.cnr.icar.eric.server.interfaces.rest.RestServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>ebxmlrr-soap-receiver-servlet</servlet-name>
    <servlet-
class>it.cnr.icar.eric.server.interfaces.soap.RegistryBSTServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet>
       <servlet-name>ebxmlrr-thin-receiver-servlet</servlet-name>
       <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
       <load-on-startup>1</load-on-startup>
```

```
  </servlet>

  <servlet-mapping>
    <servlet-name>ebxmlrr-soap-receiver-servlet</servlet-name>
    <url-pattern>/registry/soap</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>ebxmlrr-http-receiver-servlet</servlet-name>
    <url-pattern>/registry/http/*</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
       <servlet-name>ebxmlrr-thin-receiver-servlet</servlet-name>
       <url-pattern>/registry/thin/*</url-pattern>
  </servlet-mapping>

  <jsp-config>
       <taglib>
              <taglib-uri>/components</taglib-uri>
              <taglib-location>/WEB-INF/components.tld</taglib-location>
       </taglib>
  </jsp-config>

</web-app>
```

La costruzione di un package *.war* pronto per il deploy in un container è immediatamente ottenibile utilizzando la funzionalità di *Export to WAR* interna di Eclipse:



Nella root del progetto è stato inoltre realizzato lo script ANT *setup.xml*, nel quale sono stati definiti i target utili per le operazioni di setup preliminari, sulla base di quanto specificato nel file di proprietà *setup.properties*.

5.4.9 eric-test

Già in OMAR erano presenti delle classi per il test delle funzionalità interne mediante il tool JUnit. Alla luce delle importanti modifiche introdotte in eRIC si è ritenuto che

opportuno, se non necessario, avere la possibilità di effettuare dei test funzionali quanto più approfonditi sulle varie componenti.

L'ambiente di test necessita della definizione di due file di proprietà:

- *eric.properties*
- *jaxr-ebxml.properties*

In questa sede si sono utilizzate le seguenti impostazioni:

eric.properties

```
eric.registry.baseurl=http://localhost:8080/eric/registry

eric.interfaces.soap.signedResponse=true

eric.persistence.rdb.databaseURL=jdbc:postgresql://localhost:5432/xregistry
eric.persistence.rdb.databaseDriver=org.postgresql.Driver
eric.persistence.rdb.databaseUser=xregistry
eric.persistence.rdb.databaseUserPassword=xregistry
eric.persistence.rdb.useConnectionPooling=false
eric.persistence.rdb.transactionIsolation=TRANSACTION_READ_COMMITTED
eric.persistence.rdb.largeBinaryType=binary
eric.persistence.rdb.adhocQueryQueryLength=4096
eric.persistence.rdb.pool.debug=false
eric.persistence.rdb.pool.initialSize=20
eric.persistence.rdb.pool.maxSize=30
eric.persistence.rdb.pool.connectionTimeOut=180
eric.persistence.rdb.IdentifiableDAO.identifiableExistsBatchCount=100
eric.persistence.rdb.ClassificationNodeDAO.CodeCannotBeNull=true
eric.persistence.rdb.ExternalLinkDAO.checkURLs=false
eric.persistence.rdb.ServiceBindingDAO.checkURLs=false

it.cnr.icar.eric.server.query.fetchChildObjects=true

eric.repository.home=/opt/eric/3.2/data/repository
eric.repository.quota=10

eric.security.keystorePassword=ebxmlrr
eric.security.keystoreType=JKS
eric.security.keystoreFile=/opt/eric/3.2/data/security/keystore.jks
eric.security.userCacheSize=50

eric.server.cache.initCacheOnServerInit=true
eric.server.cache.primeCacheEvent=never
eric.server.cache.primeCacheDelay=60000
eric.server.cache.ClassificationSchemeCache.depth=4
eric.server.cache.ClassificationSchemeCache.depth.urn.oasis.names.tc.ebxml-
regrep.classificationScheme.ObjectType=-1
eric.server.cache.ClassificationSchemeCache.depth.urn.oasis.names.tc.ebxml-
regrep.classificationScheme.AssociationType=-1
eric.server.cache.ClassificationSchemeCache.depth.urn.freebxml.registry.demo.schemes.
iso-ch.3166.1999=1
eric.server.cache.ClassificationSchemeCache.depth.urn.uuid.fa278afc-d0fc-4c4a-abf0-
9f27292ca387=1
eric.server.cache.ClassificationSchemeCache.depth.urn.freebxml.registry.demo.schemes.
HL7=1
eric.server.cache.ClassificationSchemeCache.depth.urn.uuid.8b078ee1-3c91-465c-9872-
9b47c43d41f7=1

eric.security.validateCertificates=false
eric.security.trustAnchors.keystoreFile=/j2sdk1.4.1/jre/lib/security/cacerts
eric.security.trustAnchors.keystorePassword=changeit
eric.security.trustAnchors.keystoreType=jks

eric.server.persistence.PersistenceManagerFactory.persistenceManagerClass=it.cnr.icar
.eric.server.persistence.rdb.SQLPersistenceManagerImpl
eric.server.repository.RepositoryManagerFactory.repositoryManagerClass=it.cnr.icar.er
ic.server.repository.hibernate.HibernateRepositoryManager
eric.server.event.EventManagerFactory.eventManagerClass=it.cnr.icar.eric.server.event
.EventManager
```

```
eric.persistence.rdb.skipAssociationConfirmation=false
eric.persistence.rdb.skipReferenceCheckOnRemove=true

eric.security.authorization.defaultACP=urn:oasis:names:tc:ebxml-regrep:acp:defaultACP
eric.security.authorization.assumeCanonicalObjectsUseDefaultACP=true
eric.security.authorization.customAccessControlPoliciesEnabled=false
eric.security.authorization.enableOverride.permitAllRead=true
eric.security.authorization.registryAdministrators=urn:freebxml:registry:predefinedus
ers:registryoperator
eric.security.authorization.customAttributeFinderModules=it.cnr.icar.eric.server.secu
rity.authorization.RegistryAttributeFinderModule
eric.security.authorization.customPolicyFinderModules=it.cnr.icar.eric.server.securit
y.authorization.RegistryPolicyFinderModule
eric.security.authorization.customFunctions=it.cnr.icar.eric.server.security.authoriz
ation.ClassificationNodeCompare,it.cnr.icar.eric.server.security.authorization.Associ
ationExistsFunction

eric.server.event.defaultNotificationFormatter=urn:freebxml:registry:xslt:notificatio
nToHTML.xsl
eric.server.event.EmailNotifier.smtp.debug=false
eric.server.event.EmailNotifier.smtp.from=eric@localhost
eric.server.event.EmailNotifier.smtp.host=localhost
eric.server.event.EmailNotifier.smtp.user=
eric.server.event.EmailNotifier.smtp.password=
eric.server.event.EmailNotifier.smtp.port=
eric.server.event.EmailNotifier.smtp.auth=none
eric.server.event.EmailNotifierTest.recipient=root@localhost

eric.server.lcm.VersionManager.versionableClassList=

it.cnr.icar.eric.server.lcm.bypassCMS=false
it.cnr.icar.eric.server.query.bypassCMS=true

eric.server.cms.classMap.urn\:oasis\:names\:tc\:ebxml-
regrep\:Service\:CanonicalXMLFilteringService=it.cnr.icar.eric.server.cms.CanonicalXM
LFilteringService
eric.server.cms.classMap.urn\:oasis\:names\:tc\:ebxml-
regrep\:Service\:CanonicalXMLCatalogingService=it.cnr.icar.eric.server.cms.CanonicalX
MLCatalogingService
eric.server.cms.classMap.urn\:oasis\:names\:tc\:ebxml-
regrep\:Service\:CanonicalXMLValidationService=it.cnr.icar.eric.server.cms.CanonicalX
MLValidationService
eric.server.cms.classMap.urn\:oasis\:names\:tc\:ebxml-
regrep\:profiles\:ws\:wsdl\:cataloging\:Service\:default=it.cnr.icar.eric.server.prof
ile.ws.wsdl.cataloger.WSDLCataloger

eric.server.cms.classMap.urn\:oasis\:names\:tc\:ebxml-
regrep\:ContentManagementService\:ContentValidationService=it.cnr.icar.eric.server.cm
s.ContentValidationServiceImpl
eric.server.cms.classMap.urn\:oasis\:names\:tc\:ebxml-
regrep\:ContentManagementService\:ContentCatalogingService=it.cnr.icar.eric.server.cm
s.ContentCatalogingServiceImpl

eric.server.query.plugin.urn\:oasis\:names\:tc\:ebxml-
regrep\:query\:FindObjectByIdAndType=it.cnr.icar.eric.server.query.FindByIdQueryPlugi
n
eric.server.query.plugin.urn\:oasis\:names\:tc\:ebxml-
regrep\:query\:GetClassificationSchemesById=it.cnr.icar.eric.server.query.GetSchemesB
yIdQueryPlugin
eric.server.query.plugin.urn\:oasis\:names\:tc\:ebxml-
regrep\:query\:ArbitraryQuery=it.cnr.icar.eric.server.query.ArbitraryQueryQueryPlugin
eric.server.query.filter.plugin.urn\:freebxml\:registry\:query\:filter\:CompressConte
nt=it.cnr.icar.eric.server.query.CompressContentQueryFilterPlugin

it.cnr.icar.eric.server.query.sql.SQLQueryProcessor.bypassSQLParser=false

eric.server.query.plugin.urn\:oasis\:names\:tc\:ebxml-
regrep\:query\:GetClassificationNodeByPath=it.cnr.icar.eric.server.query.GetClassific
ationNodeByPathQueryPlugin
```

jaxr-ebxml.properties

```
jaxr-ebxml.home=/opt/eric/3.2/data

jaxr-ebxml.security.providerappname=jaxr-ebxml-provider-3.2
jaxr-ebxml.security.storetype=JKS
jaxr-ebxml.security.keystore=security/keystore.jks
jaxr-ebxml.security.storepass=ebxmlrr
jaxr-ebxml.security.createSecureSession=false
jaxr-ebxml.security.guestPrincipalName=RegistryGuest

jaxr-ebxml.security.test.alias=urn:freebxml:registry:predefinedusers:registryoperator
jaxr-
ebxml.security.test.keypass=urn:freebxml:registry:predefinedusers:registryoperator

it.cnr.icar.eric.client.xml.registry.localCall=false

jaxr-ebxml.soap.url=http://localhost:8080/eric/registry/soap
jaxr-ebxml.http.url=http://localhost:8080/eric/registry/http

eric.client.admin.AdminShellFactory.adminShellClass=it.cnr.icar.eric.client.admin.Sim
pleAdminShell

jaxr-
ebxml.extensionclass.extrinsicobject.urn\:freebxml\:registry\:sample\:profile\:cpp\:o
bjectType\:cppa\:CPP=it.cnr.icar.eric.client.xml.registry.example.CPP,CPP
jaxr-ebxml.thinbrowser.jsp-
extensions=urn:freebxml:registry:sample:profile:cpp:objectType:cppa:CPP

eric.client.thinbrowser.supportedlocales=en_US|fi|fr_CA|pt_BR
eric.client.thinbrowser.defaultlocale=en_US
eric.client.thinbrowser.numSearchResults=25
eric.client.thinbrowser.cssFile=ebxml.css
eric.client.thinbrowser.logoFile=images/freebxmlLogo.jpg
```

5.5 Installazione e setup

E' consigliabile installare eRIC mediante l'utilizzo dello script ANT *setup.xml* contenuto nella directory radice del progetto *eric-service*.

L'esecuzione dello script senza alcun parametro provoca la visualizzazione delle operazioni disponibili:

```
Buildfile: /Users/mssntn/Documents/eric-workspace/eric-service-3.2/setup.xml
usage:
     [echo] eRIC Setup instructions
     [echo] ----------------------
     [echo]  While there are many more targets, the following hi-level target are typically used:
     [echo]
     [echo]    genKeys             --> Builds KeyStore and loads it with keys for pre-defined Users
     [echo]    createMinDB         --> Create the minimal database.
     [echo]    createDemoDB        --> Create the demo database.
     [echo]    cleandb             --> Cleans all the tables at the database server
BUILD SUCCESSFUL
Total time: 455 milliseconds
```

Come si può vedere, sono stati definiti quattro target principali, presentati in ordine di esecuzione:

- ***genKeys***: inizializza il keystore e crea le chiavi per gli utenti predefiniti che possono utilizzare il registro ebxml;

- ***createMinDB***: crea lo schema e le tabelle necessarie sul database di backend e lo popola con i dati necessari al funzionamento del registro;

- ***createDemoDB***: carica dei dati di esempio sul registro, come varie tassonomie, associazioni, ecc. Questi dati sono necessari per la corretta esecuzione dei test-case JUnit definiti nel progetto *eric-test*;

- ***cleandb***: svuota il database di qualunque dato, mantenendo la definizione di tabelle ed indici.

La corretta esecuzione dei target dipende dalla preventiva configurazione del file di proprietà *setup.properties*, realizzato sulla falsariga dell'originario *build.properties* di OMAR, ma reso di gran lunga più semplice e di immediata impostazione.

Una modalità alternativa per l'installazione di eRIC consiste nell'effettuare il deploy su un container come Tomcat del file *.war* relativo al progetto *eric-service*, come documentato precedentemente.

In questo caso è necessario popolare manualmente il database di appoggio, mediante ad esempio un'operazione di "*import db*".

Qualunque sia il metodo di installazione scelto, bisogna sempre aver cura di verificare e modificare ove necessario le impostazioni contenute nei file di proprietà dei progetti.

5.5.1 Esempio di setup.properties

```
eric.name=eric
dist.version=3.2

eric.home=/opt/${eric.name}/${dist.version}
eric.container.url=http://localhost:8080
eric.registry.baseurl=${eric.container.url}/${eric.name}/registry

## PostgreSQL Database
database=postgresql
dbDialect=net.sf.hibernate.dialect.PostgreSQLDialect
dbLargeBinaryType=binary
dbLargeBinaryTypePropLength=
dbName=xregistry
dbParamsFile=
dbPassword=xregistry
dbTransactionIsolation=TRANSACTION_READ_COMMITTED
dbURL=jdbc:postgresql://localhost:5432/xregistry
dbUsername=xregistry
jdbcClassName=org.postgresql.Driver
jdbcDriver=postgresql-9.2-1003.jdbc4.jar
dbDeploymentJars=${jdbcDriver}
dbShutdownURL=${dbURL}

localCall=true
primeCacheEvent=onCacheInit
apacheLog=org.apache.commons.logging.impl.NoOpLog
deployWithSecurityConstraints=false
```

```
ant.home=/usr/local/apache-ant-1.9.2
libs.ant.jar=${ant.home}/lib/ant.jar
libs.ant-junit.jar=${ant.home}/lib/ant-junit.jar
libs.ant-launcher.jar=${ant.home}/lib/ant-launcher.jar
```

5.6 Configurazione “interna”

Essendo derivato da OMAR, anche per eRIC è necessario tenere conto della configurazione dei file di proprietà e dell’ordine con cui questi vengono caricati ed elaborati.

Il progetto consta, tra gli altri, di tre rami principali di codice: eric-common, eric-server ed eric-client. Poiché alcune delle classi contenute in eric-common hanno anch’esse bisogno di opportune impostazioni e allo stesso tempo queste classi non possono sapere a priori se saranno utilizzate lato server o lato client, anche per eric-common è necessario definire dei file di proprietà.

Ciò implica che ciascuno dei tre rami di codice utilizza un diverso set di proprietà tramite una corrispondente classe:

- `it.cnr.icar.eric.common.CommonProperties` per *eric-common*
- `it.cnr.icar.eric.server.common.RegistryProperties` per *eric-server*
- `it.cnr.icar.eric.client.xml.registry.util.ProviderProperties` per *eric-client*

Queste classi implementano un pattern di tipo *singleton* ed estendono la classe `it.cnr.icar.common.AbstractProperties`, che contiene dei metodi comuni per il loading delle proprietà.

5.6.1 Sequenza di loading delle proprietà

Allo stato attuale ciascuna delle classi per la gestione delle proprietà segue la sequenza qui presentata:

1. Caricamento delle proprietà dal classpath utilizzando il “fully qualified name” della risorsa:
    - `it/cnr/icar/eric/common/eric-common.properties` per *eric-common*
    - `it/cnr/icar/eric/server/eric.properties` per *eric-server*
    - `it/cnr/icar/eric/client/jaxr-ebxml.properties` per *eric.client*

Queste sono le proprietà di default, vengono inserite nei .jar delle classi e non dovrebbero essere modificate dall'utente.

2. Caricamento dalla root del classpath:

    - `eric-common.properties` per *eric-common*
    - `eric.properties` per *eric-server*
    - `jaxr-ebxml.properties` per *eric-client*

    Questo è lo step consigliato per la modifica delle proprietà nel caso in cui si desiderino più installazioni del registro nella stessa macchina

3. Definizione della sola proprietà *eric.home* da `System.properties`, se definita, in modo che possa essere utilizzata dallo step successivo.

4. Caricamento condizionale dal file system, eseguito soltanto quando il caricamento dalla root del classpath fallisce. Utilizza la proprietà *eric.home* definita nello step precedente:

    - `${eric.home}/eric-common.properties` per *eric-common*
    - `${eric.home}/eric.properties` per *eric-server*
    - `${eric.home}/jaxr-ebmxl.properties` per *eric-client*

5. Caricamento da `System.properties`, definite ad esempio utilizzando lo switch a linea di comando `-DpropName=propValue` e dove il nome della proprietà inizia per

    - `eric.` per *eric-common*
    - `eric.` per *eric-server*
    - `eric.` o `jaxr-ebxml.` per *eric-client*.

Alla fine della fase di loading delle proprietà viene effettuata un'operazione di sostituzione di variabili. Le variabili consentite sono:

- `${user.home}`
- `${eric.home}`
- `${jaxr-ebmxl.home}`

6 Bibliografia

Java Community Process, "*JSR 93: Java*TM *API for XML Registries 1.0 (JAXR)*", https://jcp.org/en/jsr/detail?id=93

Java Community Process, "JSR 222: JavaTM Architecture for XML Binding (JAXB) 2.0", https://jcp.org/en/jsr/detail?id=222

OASIS ebXML Registry Reference Implementation Project, "*The freebXML Registry version 3.1 (OMAR)*", http://ebxmlrr.sourceforge.net/index.html

OASIS ebXML Registry Technical Committee, "*OASIS ebXML Registry 3.0 specifications*", http://docs.oasis-open.org/regrep/v3.0/regrep-3.0-os.zip

OASIS Security Services Technical Committee, "SAML v2.0 Specifications", http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip

Oracle Inc., "Java Web Services Developer Pack (JWSDP) 2.0", http://www.oracle.com/technetwork/java/webservicespack-jsp-140788.html

Apache Software Foundation, "Apache WSS4J – Web Services Security for Java", https://ws.apache.org/wss4j/