Università di Roma "La Sapienza"
Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

Tesina di Seminari di Ingegneria del software
Prof. Giuseppe De Giacomo

Analisi,Studio e Sperimentazione di Tecnologie Web Service

Supervisore:
Ing.Massimo Mecella

A cura di:
Alessandro Gabrielli

Anno accademico 2007-08

Indice

Introduzione

L'integrazione tra sistemi e piattaforme eterogenee in ambito distribuito è senza dubbio uno dei problemi più complicati da risolvere.
Un'altra problematica alla quale si sta cercando di dare delle risposte tecnologiche efficienti è il riutilizzo delle componenti software. La necessità di sviluppare applicazioni distribuite ha ben presto evidenziato le difficoltà di tecnolgie iniziali quali COM,DCOM CORBA che erano sufficienti a garantire il riutilizzo delle componenti applicative,ma che hanno ben presto mostrato evidenti limiti nella distribuzione della soluzione sulla rete. Dai problemi legati all'utilizzo di queste tecnologie è nata l'esigenza di definire un nuovo standard indipendente dalla piattaforma per descrivere le funzionalità offerte da una componente,ed in contemporanea,di un protocollo di dialogo tra chiamante e componente applicativa che fosse indipendente dal trasporto,semanticamente completo e sicuro. Sono così comparsi i web service,che realizzano attraverso il protocollo SOAP il dialogo con le componenti e con WSDL (Web Service Description Language) la descrizione dell'interfaccia della componente.

Un Web Service,sostanzialmente,è un componente software (che esegue uno specifico compito)che può essere pubblicato,localizzato e consumato attraverso il Web. Esso è indipendente dalla piattaforma e può quindi essere esposto su differenti sistemi e comunicare con ogni altro;inoltre,esso è indipendente dal linguaggio di programmazione,essendo possibile svilupparlo in Java,Visual Basic,C++,C, etc. La tecnologia dei Web Services non è una tecnologia proprietaria ed i dettagli implementativi sono nascosti da una interfaccia in formato XML. Le tecnologie abilitanti alla base dei Web Services sono,infatti,tutte tecnologie aperte e/o standard *de-facto* (XML, SOAP, WSDL e UDDI).

Seguendo le linee di questa evoluzione,oggi nelle architetture applicative si ragiona in termini di componenti che offrono servizi applicativi,sia verso l'interfaccia utente sia verso altre applicazioni e componenti.
Questo è il concetto che sta alla base di SOA (Service Oriented Architecture) e COM,DCOM,CORBA e Web Services rappresentano le tecnologie per realizzarla.

Il principio che ispira una Service-Oriented Architecture è semplice:gli sviluppatori costruiscono diversi servizi invece di una grande applicazione monolitica e tali servizi devono risultare installabili e riusabili per supportare applicazioni e processi diversi. I benefici immediati di questo approccio sono evidenti:aumentare il riuso della funzionalità del software e guadagnare in flessibilità,poiché gli sviluppatori possono modificare e ottimizzare l'implementazione di un servizio senza influire sui client di quel servizio. Il requisito centrale di SOA risiede nel disaccoppiare l'interfaccia del servizio dall' implementazione. Questa separazione comporta un vantaggio in termini generali e non solo da un punto di vista programmatico: lo sviluppo del sistema è semplificato poichè tutti i suoi sottosistemi (legacy o moderni, interni o esterni) possono essere esposti e consumati come servizi.

Con SOA diventa quindi obsoleto il concetto di *applicazione* mentre diventa fondamentale quello di *servizio* inteso come una funzionalità di business realizzata tramite componenti che rispettano un'interfaccia standard.

I servizi possono essere quindi realizzati attraverso differenti tecnologie,implementati in diversi linguaggi di programmazione e distribuiti su piattaforme eterogenee. Con SOA viene definita l'architettura che li caratterizza ma accanto ad essa nasce l'esigenza di una rappresentazione che astragga dalla loro implementazione e mostri semplicemente il loro comportamento: behavior. Questo si può fare mediante una rappresentazione sotto forma di stati e transizioni:i transition system.

Un Transition System TS è quindi un modello relazionale astratto basato sulle nozioni primitive di *stato* e *transizione* rappresentato dalla tupla $T = < A, S, So, \delta, F>$ dove:

- A è l'insieme delle azioni
- S è l'insieme degli stati

– So $\subseteq$ S è l'insieme degli stati iniziali
– $\delta \subseteq S \times A \times S$ è l'insieme delle transizioni
- $F \subseteq S$ è l'insieme degli stati finali

Ogni servizio,sia un COM,DCOM,CORBA o Web Service,viene quindi rappresentato attraverso uno specifico TS ed implementa un'architetura di tipo SOA.

Lo scopo di questa tesina è quello di approfondire lo studio su una perticolare tipologia di servizi:i Web Service,analizzando funzionalità offerte ed approcci seguiti da differenti tipi di tecnologie nelle fasi di implementazione,compilazione,deploy ed esecuzione.

Nel primo capitolo parleremo del legame tra l'architettura SOA e i Web Services per passare poi ad analizzare gli standard caratteristici di quest'ultimi.
Nel secondo capitolo verrà presentato Axis 1.4,un engine SOAP open source realizzato da Apache che permette di creare servizi in modo semplice ed efficiente e rappresenta forse una delle tecnologie migliori per iniziare a costruire Web Services.
Nel terzo capito parleremo di Axis 2,che può essere visto come l'evoluzione di Axis 1.* essendo stato sviluppato sempre da Apache;verranno descritti i motivi che hanno portato alla creazione di una nuova architettura anziché continuare ad effettuare modifiche su quella di Axis per poter cooperare con i nuovi standard che si vanno via via definendo.
Nel quarto capitolo analizzeremo il legame tra gli Enterprise Java Bean e i Web Services discutendo le modalità attraverso le quali è possibile esporre un EJB come servizio;vedremo quindi come esporre una business logic implementata in un Session Bean e strettamente legata alla piattaforma J2EE come Web Service,discutendo i vantaggi derivanti da tale scelta progettuale tra i quali appunto,quello di disaccoppiare completamente la piattaforma sulla quale è implementato il servizio da quella utilizzata dai client che interagiscono con esso.
Nel quinto capitolo verranno presentati alcuni gli standard per la notifica di eventi nei Web Services e in particolare la specifica del WS-Notification;mostrerò quindi un possibile modo per implementare tale specifica attraverso l'utilizzo del progetto open source Apache Muse attraverso il quale è possibile implementare publisher e subscriber che comunicano rispettando appunto la specifica del WS-Notification. Nell'esempio che verrà presentato verranno descritti in dettaglio il producer,il consumer,i metodi utilizzati,le classi implementate i problemi riscontrati e ogni messaggio o evento che viene generato.

Una cosa importante da sottolineare è che nessuna delle tecnologie(Axis1.x,Axis2 e JBoss) utilizzate in questo contesto supporta l'implementazione di servizi statefull in grado di mantenere lo stato della conversazioni con i client. E' compito del programmatore implementare tali tipologie di servizi:si rappresenta inizialmente il servizio con un TS e successivamente si scrive il codice che concretamente implementa i metodi esposti. Tali tecnologie non contengono infatti alcun tool che a partire da un file xml che descrive il TS costruisce automaticamente la classe che implementa il servizio.

Capitolo 1 – Service-Oriented Architecture e Web Services: concetti base

Lo scopo di questo capitolo è quello di introdurre i concetti fondamentali dell'Architettura Orientata ai Servizi (SOA: Service-Oriented Architecture),per poi discutere quali siano i punti di contatto con la tecnologia dei Web Services.

1.1 SOA: Service-Oriented Architecture

Una Service-Oriented Architecture (SOA, Architettura Orientata ai Servizi) è un modello architetturale per la creazione di sistemi residenti su una rete che focalizza l'attenzione sul concetto di servizio. Un sistema costruito seguendo la filosofia SOA è costituito da applicazioni, chiamate servizi,ben definite ed indipendenti l'una dall'altra,che risiedono su più computer all'interno di una rete (ad esempio la rete interna di una azienda o una rete di connessione fra più aziende che collaborano: intracompany e intercompany network). Ogni servizio mette a disposizione una certa funzionalità e può utilizzare quelle che gli altri servizi hanno reso disponibili,realizzando,in questo modo,applicazioni di maggiore complessità. SOA è una forma particolare di Distributed System[1].

1.1.1 Caratteristiche di una SOA

L'astrazione delle SOA non è legata ad alcuna specifica tecnologia, ma semplicemente definisce alcune proprietà, orientate al riutilizzo e all'integrazione in un ambiente eterogeneo, che devono essere rispettate dai servizi che compongono il sistema [1,2]. In particolare un servizio dovrà:

• essere ricercabile e recuperabile dinamicamente.

Un servizio deve poter essere ricercato in base alla sua interfaccia e richiamato a tempo di esecuzione. La definizione del servizio in base alla sua interfaccia rende quest'ultima (e quindi l'interazione con altri servizi) indipendente dal modo in cui è stato realizzato il componente che lo implementa.

• essere autocontenuto e modulare.

Ogni servizio deve essere ben definito, completo ed indipendente dal contesto o dallo stato di altri servizi.

• essere definito da un'interfaccia ed indipendente dall'implementazione.

Deve cioè essere definito in termini di ciò che fa, astraendo dai metodi e dalle tecnologie utilizzate per implementarlo. Questo determina l'indipendenza del servizio non solo dal linguaggio di programmazione utilizzato per realizzare il componente che lo implementa ma anche dalla piattaforma e dal sistema operativo su cui è in esecuzione:non è necessario conoscere come un servizio è realizzato ma solo quali funzionalità rende disponibili.

• essere debolmente accoppiato con altri servizi (loosely coupled).

Un'architettura è debolmente accoppiata se le dipendenze fra le sue componenti sono in numero limitato. Questo rende il sistema flessibile e facilmente modificabile.

• essere reso disponibile sulla rete attraverso la pubblicazione della sua interfaccia (in un Service Directory o Service Registry) ed accessibile in modo trasparente rispetto alla sua allocazione. Essere disponibile sulla rete lo rende accessibile da quei componenti che ne richiedono l'utilizzo e l'accesso deve avvenire in maniera indipendente rispetto all'allocazione del servizio. La pubblicazione dell'interfaccia deve rendere noto anche le modalità di accesso al servizio.

• fornire un'interfaccia possibilmente a "grana grossa" (coarse-grained).

Deve mettere a disposizione un basso numero di operazioni, cioè poche funzionalità, in modo tale da non dover avere un programma di controllo complesso. Deve essere invece orientato ad un elevato livello di interazione con gli altri servizi attraverso lo scambio di messaggi. Per questo motivo e per il fatto che i servizi possono trovarsi su sistemi operativi e piattaforme diverse è necessario che i messaggi siano composti utilizzando un formato standard largamente riconosciuto(piattaforma indipendente). I dati che vengono trasmessi attraverso i messaggi possono essere costituiti sia dal risultato dell'elaborazione di un certo servizio sia da informazioni che più servizi si scambiano per coordinarsi fra loro.

• essere realizzato in modo tale da permetterne la composizione con altri.

Nell'architettura SOA le applicazioni sono il risultato della composizione di più servizi. E' per questo motivo che ogni servizio deve essere indipendente da qualsiasi altro,in modo tale da ottenere il massimo della riusabilità. La creazione di applicazioni o di servizi più complessi attraverso la composizione dei servizi di base viene definita Service Orchestration.
Queste sono dunque le caratteristiche di un sistema di tipo SOA, di cui adesso passiamo a descrivere il funzionamento.

1.1.2 Come funziona una SOA

Gli attori di un sistema SOA sono tre:
- Service Provider
- Service Consumer
- Service Registry.

Il Service Provider è un'entità che mette a disposizione un qualche servizio. Tale servizio, per poter essere trovato da altre entità che vogliono utilizzarlo, deve essere reso visibile sulla rete, in termine tecnico Pubblicato. A tal fine il Service Provider comunica al Service Registry le informazioni relative al servizio, perchè vengano memorizzate. Il Service Registry possiede quindi le informazioni, come URL e modalità di accesso, di tutti i servizi disponibili.
Nel momento in cui un Service Consumer dovrà utilizzare un servizio farà richiesta delle informazioni ad esso relative al Service Registry. Con queste informazioni il Service Consumer potrà comunicare direttamente con il Service Provider ed utilizzare il servizio. In figura sono riportate le interazioni fra le entità appena descritte. Tutte queste interazioni passano attraverso quella che in figura viene genericamente definita Rete di Comunicazione, la quale in un'implementazione reale di una SOA può essere costituita sia da Internet sia da una intranet.

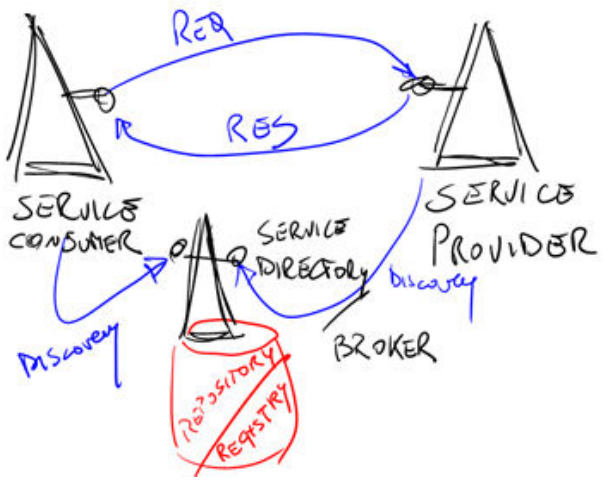


Figura 1:Esempio di Architettura SOA

SOA definisce,dunque,le caratteristiche che i componenti facenti parte di un sistema devono avere al fine di poter definire quest'ultimo un'architettura orientata ai servizi.
Dopo aver descritto cos'è l'architettura SOA ed il funzionamento di un sistema di questo tipo, vediamo adesso cosa sono i Web Services,quali tecnologie utilizzano ed il loro legame con SOA.

1.2 Web Services

I Web Services sono delle applicazioni web che cooperano fra loro,indipendentemente dalla piattaforma sulla quale si trovano,attraverso lo scambio di messaggi.
Ognuna di queste applicazioni viene chiamato Web Service (Servizio Web),o più semplicemente servizio,del quale il Web Services Architecture Working Group (del W3C) [1,3] dà la seguente definizione:
Un Web Service è un'applicazione software identificata da un URI (Uniform Resource Identifier), le cui interfacce pubbliche e collegamenti sono definiti e descritti come documenti XML, in un formato comprensibile alla macchina (specificatamente WSDL). La sua definizione può essere ricercata da altri agenti software situati su una rete,i quali possono interagire

direttamente con il Web Service, con le modalità specificate nella sua definizione, utilizzando messaggi basati su XML (SOAP), scambiati attraverso protocolli Internet (tipicamente HTTP).
Le tecnologie su cui si basano i Web Services, appena citate nella definizione, sono:
• XML, eXtensible Markup Language
• SOAP, Simple Object Access Protocol
• WSDL, Web Services Description Language
• UDDI, Universal Description, Discovery and Integration.

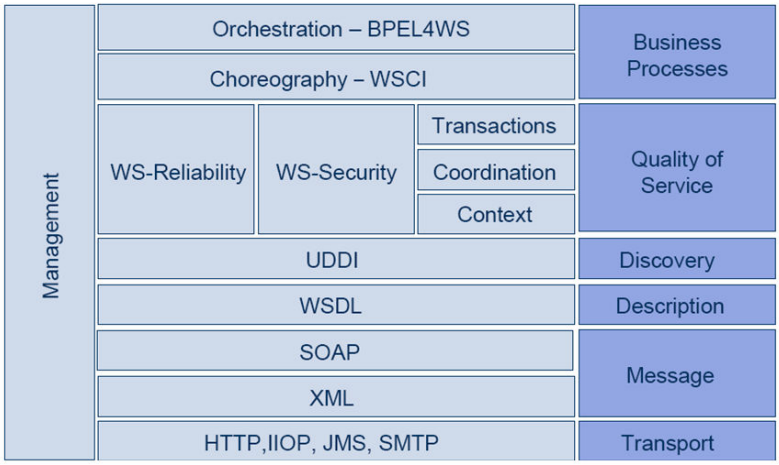


Figura 2:Standard dei Web Services

Attraverso l'utilizzo di questi ed altri standard i Web Services rendono possibile la comunicazione e la cooperazione, attraverso il web, di più applicazioni (servizi) che mettono a disposizione alcune funzionalità e, allo stesso tempo, utilizzano quelle rese disponibili da altre. Si può cioè ricercare e invocare servizi che possono essere composti per formare un'applicazione per l'utente finale, per abilitare transazioni di business o per creare nuovi Web Services.[4] Di queste tecnologie XML ha dato un contributo molto importante alla nascita dei Web Services. Linguaggio a marcatori (tag) derivato da SGML, Standard Generalization Markup Language, come ad esempio il più conosciuto HTML, HyperText Markup Language, l'XML è utilizzato per la memorizzazione di informazione in maniera strutturata. XML è un formato indipendente dalle varie piattaforme; ciò è dovuto, oltre che all'essere universalmente riconosciuto come standard, anche al fatto che tale tecnologia si basa sul formato testo e quindi un documento XML può essere letto chiaramente su qualsiasi sistema operativo. Questa indipendenza lo rende la soluzione ideale per lo scambio di informazioni attraverso il Web.

I vantaggi offerti dai Web Services sono:
• Indipendenza dalla piattaforma: i Web Services possono, infatti, comunicare fra loro anche se si trovano su piattaforme differenti.
• Indipendenza dall'implementazione del servizio: l'interfaccia che un Web Service presenta sulla rete è indipendente dal software che implementa tale servizio. In futuro tale implementazione potrà essere sostituita o migliorata senza che l'interfaccia subisca modifiche e quindi senza che dall'esterno (da parte di altri utenti o servizi sulla rete) si noti il cambiamento.
• Riuso dell'infrastruttura: per lo scambio di messaggi si utilizza SOAP che fa uso di HTTP, grazie al quale si ottiene anche il vantaggio di permettere ai messaggi SOAP di passare attraverso sistemi di filtraggio del traffico sulla rete,quali "Firewall".
• Riuso del software: è possibile riutilizzare software implementato precedentemente e renderlo disponibile attraverso la rete. Il concetto di Web Services implica quindi un modello di architettura ad oggetti distribuiti (oggetti intesi come applicazioni), che si trovano localizzati in punti diversi della rete e su piattaforme di tipo differente.

Il legame con l'architettura SOA sta nel fatto che,sfruttando al meglio tutte le caratteristiche della tecnologia dei Web Services,il sistema che si ottiene implementa proprio un'architettura orientata ai servizi. Ad oggi i Web Services rappresentano la soluzione migliore per la realizzazione di una SOA su larga scala,ovvero su Internet.

1.3 Web Services con Architettura SOA

La presenza del Service Registry (o anche Service Directory o Service Broker) è ciò che rende il sistema,nell'esempio di utilizzo dei Web Services visto precedentemente un'architettura Service-Oriented (SOA).
Per implementare il Service Registry i Web Services fanno uso di UDDI, Universal Description, Discovery and Integration. UDDI è un servizio di registro pubblico in cui le aziende possono registrare (pubblicare) e ricercare Web Services. Esso mantiene informazioni relative ai servizi come l'URL e le modalità di accesso. Anche UDDI è un Web Service, il quale mette a disposizione due operazioni:

- Publish, per la registrazione
- Inquiry, per la ricerca.

Si ottiene così quella che oggigiorno è da molti considerata la migliore soluzione per l'implementazione di un sistema con architettura ServiceOriented.
In figura 3 è riportata la schematizzazione del funzionamento di un sistema con architettura SOA, realizzato attraverso l'uso dei Web Services.

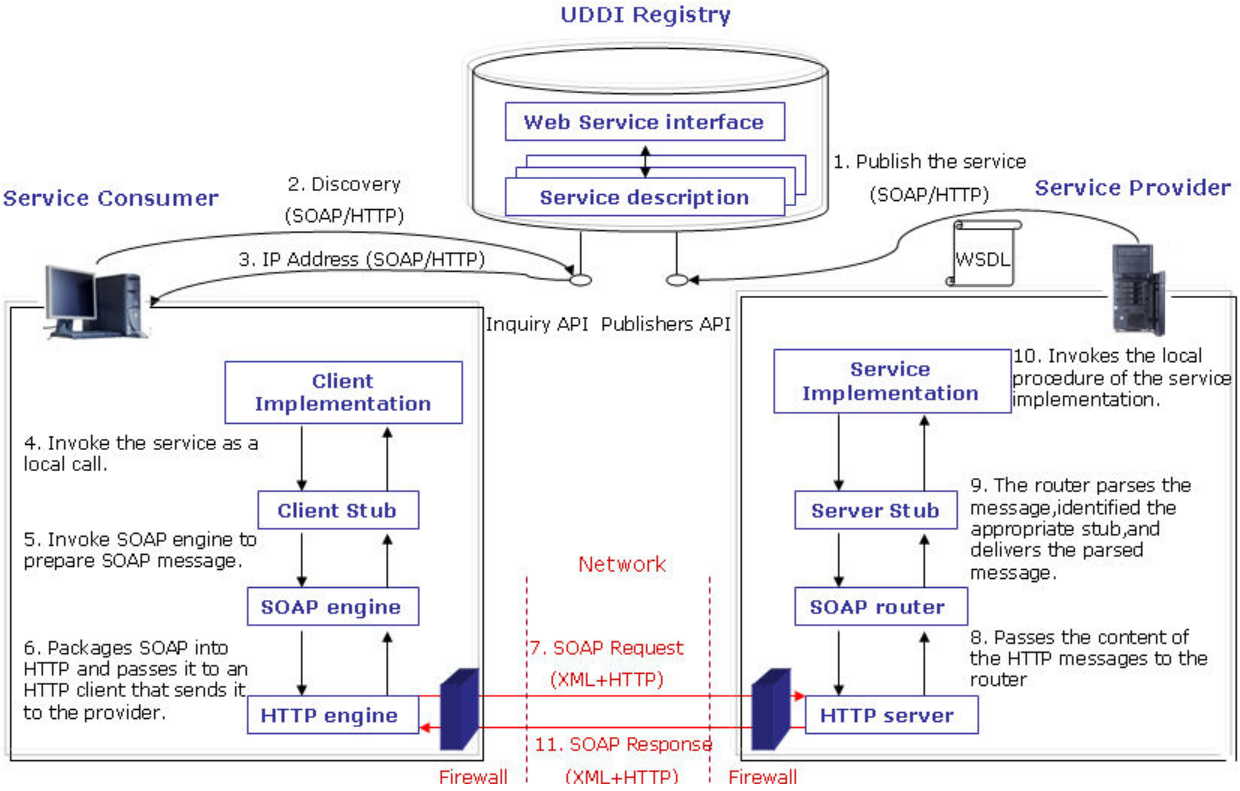


Figura 3:Architettura Service-Oriented realizzata con la tecnologia dei Web Services.

1.4 Standard e Tecnologie alla base dei Web Services

I Web Services si basano su tecnologie oggi riconosciute universalmente come standard: XML, WSDL, SOAP e UDDI. Vediamole quindi una per una partendo da quella che è un pò il comune denominatore di tutte le altre:XML.

1.4.1 XML

XML, eXtensible Markup Language, è un metalinguaggio nato nel 1998 e derivato da SGML (Standard Generalization Markup Language). Il termine "metalinguaggio" significa che esso è un linguaggio per mezzo del quale se ne possono creare altri. "Esteticamente" simile ad HTML, poichè anch'esso è basato su marcatori o "tag" (questo è un tag: <tag name>), ne differisce profondamente. Essi infatti non sono semplicemente due linguaggi diversi ma appartengono a categorie diverse; HTML è un linguaggio, in cui il comportamento di ogni marcatore è già stabilito, mentre XML è, come già anticipato, un metalinguaggio, che permette allo sviluppatore di definire marcatori personalizzati e di specificarne il ruolo.

Il motivo che ha portato alla creazione di XML è stata la necessità di avere documenti strutturati e flessibili che potessero essere utilizzati sulla rete per lo scambio di dati. I due linguaggi HTML e SGML, non erano adatti allo scopo, per due motivi opposti. HTML ha una struttura rigida dove i tag disponibili sono predefiniti e dal comportamento già stabilito. Al contrario SGML fornisce una struttura personalizzabile ma troppo ampia e complessa per giustificarne un utilizzo via web.
Inoltre XML ha un formato indipendente dalle varie piattaforme;ciò è dovuto, oltre che all'essere universalmente riconosciuto come standard,al fatto che tale tecnologia si basa sul formato testo e quindi un documento XML può essere letto chiaramente su qualsiasi sistema operativo. Il contenuto di un documento XML è costituito da marcatori e dati strutturati secondo un ordine logico determinato da una struttura ad albero. Il formato del documento è testuale e questo permette all'utente di accedervi direttamente in lettura.

Il compito di un documento XML è memorizzare i dati all'interno di una struttura gerarchica che rappresenti le relazioni esistenti fra di essi,senza curarsi minimamente della loro rappresentazione visuale. Tali dati possono poi essere visualizzati in molti modi differenti,a seconda del caso,come ad esempio una semplice pagina HTML. Abbiamo detto che ogni sviluppatore può creare i propri marcatori ma,affinchè questi siano interpretabili correttamente da chiunque voglia accedere ai dati contenuti nel documento,c'è bisogno di un meccanismo che definisca quali elementi sono presenti,la loro struttura e le relazioni fra di essi. Tecnologie create per assolvere questo compito sono Document Type Definition,XML-Schema e XML Namespace.

1.4.1.1 DTD: Document Type Definition
Una Document Type Definition è un file in cui è riportata la definizione di tutti gli elementi, e dei loro attributi, usati nel documento XML, specificando inoltre la correlazione tra di essi. Tale file permette ad un'applicazione di sapere se il documento XML che sta analizzando è corretto o no, dato che gli elementi, essendo stati creati dallo sviluppatore, risulterebbero privi di significato senza una loro definizione. Una DTD definisce quindi la struttura di un particolare tipo di documento XML, rispetto al quale si può valutare la conformità di una data istanza XML.
Le DTD, primo strumento di questo genere, presentano però delle limitazioni: possiedono una sintassi complessa (non sono scritte in XML), non permettono di specificare tipi di dato e sono difficilmente estendibili.

1.4.1.2 XML-Schema
Uno strumento, creato allo stesso scopo delle DTD,che superà le limitazioni di queste ultime è XML-Schema.
Un documento XML-schema definisce[5]:
- gli elementi e gli attributi che possono comparire in un documento,
- quali elementi sono elementi figlio,
- l'ordine ed il numero degli elementi figlio,
- se un elemento è vuoto o può includere testo,
- i tipi di dato per gli elementi e gli attributi,
- i valori di default ed i valori costanti per gli elementi e gli attributi.

Rispetto alle DTD, gli XML-Schema sono estensibili e scritti in XML, rendono possibile la definizione di tipi di dato e di vincoli, ammettono l'ereditarietà e supportano i namespace.

XML Namespace
XML Namespace è utilizzato per risolvere la possibile ambiguità fra elementi di documenti diversi. Gli elementi di un documento XML sono identificati da un nome che è unico all'interno del documento stesso,ma può accadere che un elemento appartenente ad un altro file abbia lo stesso nome. Questo fatto crea un problema di ambiguità quando ci si riferisce a questi elementi;tale ambiguità viene risolta con l'introduzione dei namespace. Un namespace identifica l'insieme di tutti gli elementi di un documento,semplicemente associando un prefisso ai loro nomi. In questo modo due elementi appartenenti a file diversi ed aventi lo stesso nome possono essere identificati univocamente grazie al fatto che essi appartengono a namespace differenti.

1.4.2 WSDL

WSDL, ovvero Web Services Description Language,è un linguaggio,basato su XML,usato per descrivere,in modo completo, un Web Service. Più precisamente un documento WSDL fornisce informazioni riguardanti l'interfaccia del Web Service in termini di:

- servizi offerti dal Web Service,
- URL ad essi associato,
- modi per l'invocazione,
- argomenti accettati in ingresso e modalità con cui debbono essere passati,
- formato dei risultati restituiti,
- formato dei messaggi.

In altri parole si può dire che un file WSDL fornisce la descrizione relativa ad un Web Service in termini di:

- cosa fa,
- come comunica,
- dove si trova.

Attraverso tale file si può quindi conoscere tutti i dettagli per poter invocare correttamente un servizio.

1.4.2.1 Struttura di un documento WSDL

Un documento WSDL è un file XML costituito da un insieme di definizioni. Il documento inizia sempre con un elemento radice chiamato definitions ed al suo interno utilizza i seguenti elementi principali nella definizione dei servizi:

- Types - definizione dei tipi dei dati utilizzati.
- Message - definizione dei messaggi che possono essere inviati e ricevuti.
- Port Type - insieme di servizi, Operation, offerti da un Web Service.
- Binding - informazioni sul protocollo ed il formato dei dati relativo ad un particolare Port Type.
- Service - insieme di endpoint relativi al servizio.

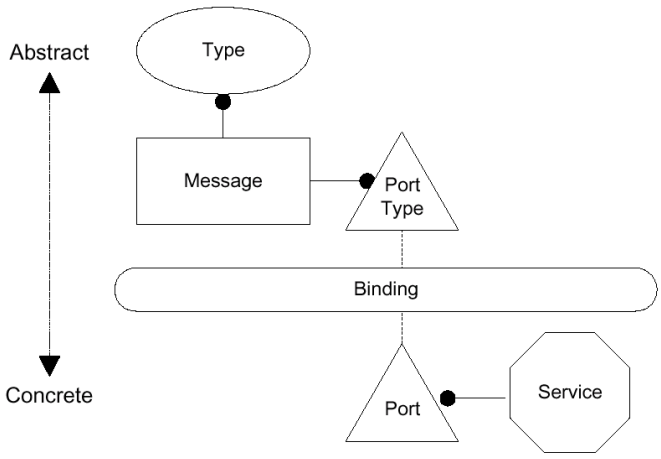


Figura 4:Elementi WSDL

Ciascuno di questi elementi identifica una distinta sezione del documento contenente informazioni relative ad uno specifico aspetto. Vediamo meglio come è strutturato un file WSDL analizzando un esempio e descrivendo i costrutti di cui è costituito.

Cosa fa un Web Service è descritto nelle sezioni delimitate dai tag:

- types
- message
- portType

Le caratteristiche di comunicazione sono invece descritte in:

- binding

Infine il punto di accesso ad un servizio è definito da:

- service

L'elemento types racchiude le definizioni dei tipi dei dati che sono coinvolti nello scambio dei messaggi. Come sistema standard per la tipizzazione,WSDL si basa su quello definito per gli

schemi XML (XSD, XML Schema Definition) ed allo stesso tempo è però possibile aggiungere anche altri tipi.
La sezione relativa ai messaggi definisce invece l'input e l'output dei servizi. Ogni elemento message racchiude le informazioni, come parametri e loro tipi, relative ad uno specifico messaggio. Si possono avere ad esempio due message: uno relativo al messaggio di input ed uno relativo al messaggio di output. Ogni elemento part identifica un parametro.
La sezione portType riporta dettagli relativi alle operazioni che un Web Service rende utilizzabili. Ogni operazione viene descritta facendo uso di un'ulteriore elemento chiamato operation. Al suo interno vi sono i messaggi di input e/o di output accettati dal servizio e l'ordine che è necessario seguire per il passaggio dei parametri. Messaggi di input ed output vengono identificati rispettivamente dagli elementi input e output e la presenza di solo uno dei due o di entrambi e, nel secondo caso, l'ordine in cui vengono riportati determinano la tipologia del servizio che, come vedremo fra poco, può avere diversa natura. L'elemento (opzionale) fault specifica il formato del messaggio per ogni errore che può essere restituito in output dall'operazione.
La sezione binding porta alla seconda parte di un documento WSDL,passando da cosa fa un Web Service a come comunica. Qui viene stabilito il legame di ogni servizio del Web Service al protocollo per lo scambio di messaggi,il quale può essere HTTP GET/POST,MIME o SOAP; quest'ultimo è il protocollo che viene utilizzato più frequentemente. In particolare si specifica il protocollo per ognuno dei messaggi di un servizio offerto dal Web Service. Inoltre di solito viene scelto HTTP(S) come protocollo di trasporto per i messaggi SOAP. Questa combinazione viene appunto definita "SOAP over HTTP(S)". Assumendo "SOAP over HTTP(S)" come protocollo di scambio messaggi e trasmissione,all'interno di binding avremmo prima di tutto la definizione dello stile del collegamento,che può essere rpc o document, e la specificazione di HTTP come protocollo di trasporto,facendo uso di un elemento relativo a SOAP chiamato wsdlsoap.
Per quanto riguarda lo stile del collegamento, la differenza fra "rpc" e "document", che influisce sulla struttura del corpo del messaggio SOAP (elemento <Body>), è la seguente:
• Document: il contenuto dell'elemento <Body> del messaggio SOAP è costituito da un documento.
• RPC: l'elemento <Body> contiene l'invocazione ad una procedura remota, una Remote Procedure Call (RPC) appunto. La struttura del corpo del messaggio SOAP (elemento <Body>) deve rispettare alcune regole riportate nelle specifiche di SOAP. Secondo queste regole l'elemento <Body> può ad esempio contenere solamente un elemento il cui nome è lo stesso dell'operazione,cioè la procedura remota, che viene invocata e tutti i parametri di tale operazione devono essere riportati come sottoelementi di questo elemento.
Vi sono poi tanti elementi operation quanti sono i servizi (operazioni) messi a disposizione. All'interno di ogni elemento operation si possono trovare gli elementi input,output e fault; input ed output riportano,facendo uso dell'elemento wsdlsoap:body, il tipo di encoding che può essere encoded o literal:nel caso encoded deve essere specificato anche l'attributo encodingStyle.
L'ultima sezione,identificata dall'elemento service,è relativa alla localizzazione del Web Service. Al suo interno si trova l'elenco di tutti i servizi messi a disposizione;ognuno di essi è definito da un'elemento port che riporta il collegamento utilizzato ed al suo interno ha un'ulteriore elemento wsdlsoap:address che indica l'indirizzo URL,chiamato anche endpoint,al quale può essere trovato il Web Service.

1.4.2.2 Tipologia di un servizio

Prima ho accennato,nella sezione relativa a portType ed ai messaggi di ingresso ed uscita delle operazioni,al fatto che un servizio può essere di natura diversa,cioè appartenere ad una tipologia piuttosto che ad un altra,in relazione alla presenza e all'ordine degli elementi input ed output. Le tipologie a cui un servizio pu`o appartenere sono quattro:
• One-way
• Request-response
• Solicit-response
• Notification
One-way. Nel caso One-way è presente il solo elemento input. Come intuibile dal nome,la comunicazione avviene in una sola direzione: viene inviato un messaggio da un client al

servizio,dopodichè il primo continua la sua computazione senza attendere una risposta dal secondo.
Request-response.In questo caso sono presenti entrambi gli elementi input ed output,in questo ordine. Il servizio riceve il messaggio Request dal client e,dopo aver eseguito l'elaborazione relativa alla richiesta,manda indietro un messaggio Response.
Solicit-response. Come nel caso precedente vi sono entrambi gli elementi ma in ordine inverso. E' il servizio ad iniziare la comunicazione inviando un messaggio al client ed attendendo poi una sua risposta.
Notification. Questo caso è l'opposto del One-way. Il servizio spedisce un messaggio al client senza attendere una sua risposta. E' quindi presente solo l'elemento output.

Ognuna di queste tipologie individua la natura del servizio che stiamo descrivendo. Ad esempio la tipologia Request-response è quella utilizzata nel modello di comunicazione RPC (Remote Procedure Call),mentre si può avere il caso One-way quando sia presente un servizio che memorizza dati ricevuti da più client, senza restituire un messaggio di risposta.

1.4.2.3 Utilizzo di un documento WSDL

Esistono alcuni strumenti,come il tool WSDL2Java (facente parte del framework Axis di cui parleremo più avanti),che prendono in input un documento di questo tipo per creare a runtime l'implementazione del client per accedere al servizio. Ma più semplicemente si può vedere il vantaggio apportato dall'uso dei documenti WSDL nel disaccoppiamento del servizio Web dal protocollo di trasporto e dai percorsi fisici,come gli endpoint. Si ottiene così un livello di indirezione,simile a quello che si ha fra i DNS e gli indirizzi internet,grazie al quale non è necessario configurare direttamente l'URL del servizio. In questo modo se vi saranno molti client che utilizzano il servizio,nel momento in cui esso cambia indirizzo,non si dovrà riconfigurarli tutti,ma semplicemente aggiornare tale informazione nel documento WSDL,poichè è da questo che i client otterranno l'endpoint[6].

1.4.4 SOAP

SOAP è un acronimo per Simple Object Access Protocoll,un protocollo leggero pensato per facilitare l'interoperabilità tra applicazioni e piattaforme eterogenee nell'era della programmazione distribuita su Internet. Alla base di SOAP sta un'idea tanto semplice quanto intelligente che si può riassumere in questo modo: non inventare nessuna nuova tecnologia, ma utilizzare al meglio quelle esistenti.

Con SOAP il web diviene qualcosa di più di un semplice scambio di documenti: infatti SOAP mette le applicazioni in comunicazione tra loro, e si candida per costruire il framework su cui costruire i web services. Inoltre SOAP è candidato a diventare il meccanismo che consente a tutti i sevizi di esporre le proprie caratteristiche e di comunicare "service to service" o "componente to service", sia tra piattaforme omogenee che, soprattutto tra piattaforme eterogenee. Il cambiamento apportato da SOAP è dovuto al fatto che oggi come oggi è possibile utilizzare una varietà di protocolli binari per l'invocazione di oggetti remoti. Questo significa che le applicazioni client sono create per comunicare con applicazioni server specifiche. Ma il più grande dei problemi consiste nel fatto che se si vuole eseguire i client al di là di un firewall, è necessario configurare quest'ultimo in modo apposito, sempre che sia possibile. E SOAP risolve brillantemente entrambi i problemi, consentendo uno sviluppo e soprattutto una distribuzione semplificata delle applicazioni.

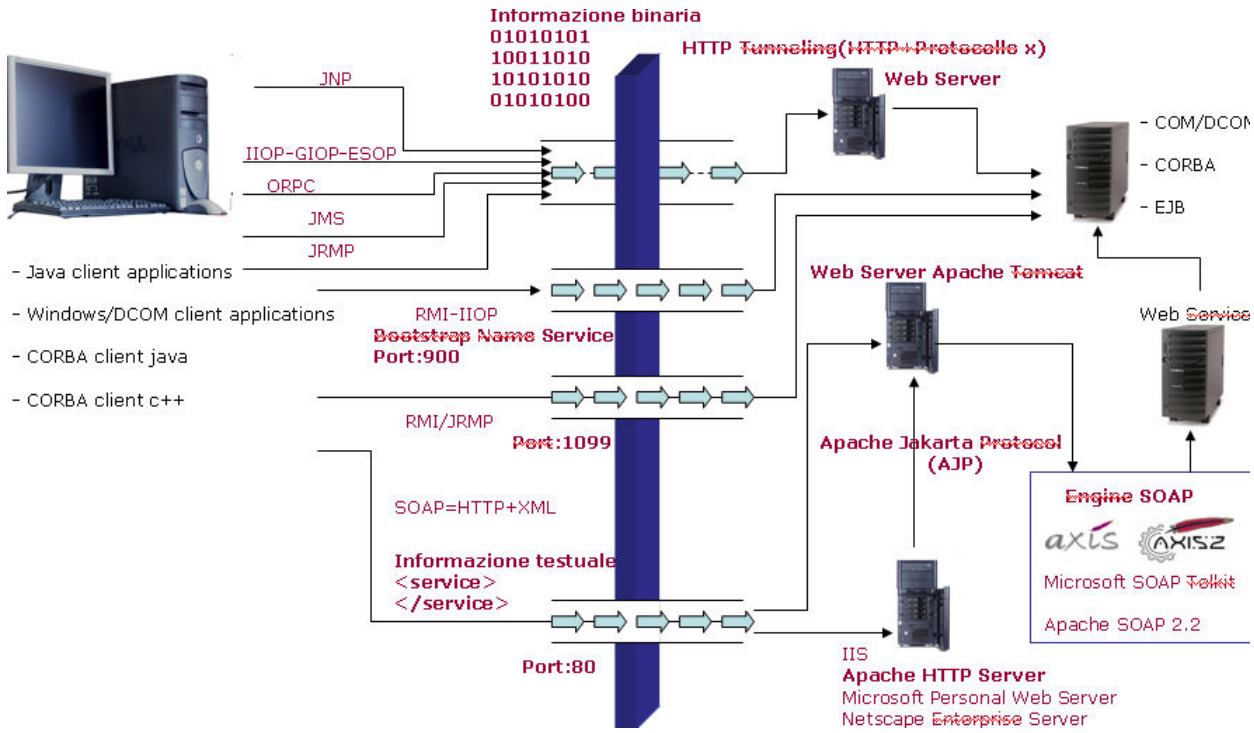


Figura 5:Utilizzo del protocollo SOAP in un contesto distribuito

1.4.4.1 Toolkit per implementare SOAP

I principali toolkit che implementano il protocollo SOAP:

1. Apache SOAP 2.2.
2. Microsoft SOAP Toolkit.
3. Apache Axis

Apache SOAP 2.2

- Conforme alla specifica SOAP 1.1 del W3C.
- Sul lato client estende Java con alcuni package.
- Sul lato server è semplice codice Java +deployment descriptor per la pubblicazione.

- Limitazioni:

  - Non supporta il linguaggio WSDL.
  - Presenta delle limitazioni rispetto alla specifica 1.1 (Attributo encodingStyle e mustUnderstand hanno valore predefinito, non supporta l'elemento root XML, non supporto l'attributo actor).

Microsoft SOAP Toolkit 2.0

Microsoft è stata una tra le prime aziende a muoversi nella direzione di SOAP, anche per permettere a tutti gli sviluppatori che realizzano soluzioni distribuite utilizzando COM e COM+, di orientarsi verso uno standard davvero universale quale è SOAP. Il toolkit di Microsoft è composto essenzialmente da alcune utilità che permettono di creare uno strato di comunicazione SOAP attorno a qualsiasi componente COM progettato ad hoc. In particolare il SOAP Toolkit è composto da:

- Alcune librerie ActiveX da utilizzare come oggetti per lo sviluppo di applicazioni client
- Una utilità, il WSDL Generator, che permette la generazione dei file WSDL e WSML
- I filtri ISAPI per poter gestire i file WSDL e WSML da Internet Information Server
- Altre utilità

La filosofia di base di Microsoft Toolkit è la seguente: qualsiasi componente COM può essere utilizzato come fornitore di servizi via SOAP,a condizione che per esso venga predisposta un'opportuna coppia di file WSDL e WSML che contengano la descrizione del servizio da esporre attraverso Internet Information Server. Sarà lo stesso Internet Information Server, attraverso il filtro ISAPI opportuno, a saper gestire i file WSDL attraverso i quali il client richiederà il servizio al server. Il client pertanto, attraverso SOAP,farà una request HTTP al file WSDL generato appositamente per un certo componente COM attraverso il WSDL Generator.
Il filtro ISAPI installato come plugin in Internet Information Server si occuperà di trasformare la chiamata al file WSDL in una reale richiesta di servizio al componente COM sul server ed effettuerà la response SOAP contenente il payload di ritorno.
Anche nel caso di Microsoft SOAP Toolkit,ha senso parlare di installazione lato client ed installazione lato server. L'installazione client ha come obiettivo la copia e la registrazione degli oggetti ActiveX necessari per fare in modo che un'applicazione client che voglia utilizzare servizi SOAP possa farlo, nel caso specifico si tratta di una serie di librerie DLL da utilizzare come riferimenti all'interno degli ambienti di sviluppo che si utilizzano.
L'installazione server invece,ha come obiettivo l'installazione e la configurazione dei filtri ISAPI da utilizzare per fare in modo che Internet Information Server possa ricevere request di file WSDL.

Le caratteristiche supportate da SOAP Toolkit 3.0 sono le seguenti:

- Specifiche del consorzio W3C relative a WSDL 1.1
- Specifiche del consorzio W3C relative a SOAP 1.1
- Specifiche del consorzio W3C relative a XML Schema Part 0 (Primer), Part 1 (Structure) e Part (Datatypes).
- Array di tipo semplice e complesso e array multidimensionali
- Type Complex
- Supporto per operazioni WSDL RPC-encoded e document-literal.
- SOAP Headers

Il toolkit permette di eseguire chiamate a servizi e pubblicare questi attraverso due livelli di API (Application Program Interface): "high-level" oppure "low". La scelta dipenderà dalle caratteristiche dei messaggi SOAP che si desidera inviare o dal livello di monitoraggio desiderato. Ovviamente le "high-level" API facilitano il lavoro dello sviluppatore il quale può ignorare diversi meccanismi interni come: connessioni, serializzazioni, deserializzazioni, ecc.

Fra le "high-level" API troviamo gli oggetti:
- SoapClient
- SoapServer

Fra le "low-level" API troviamo gli oggetti:
- SoapConnector
- SoapSerializer
- SoapReader

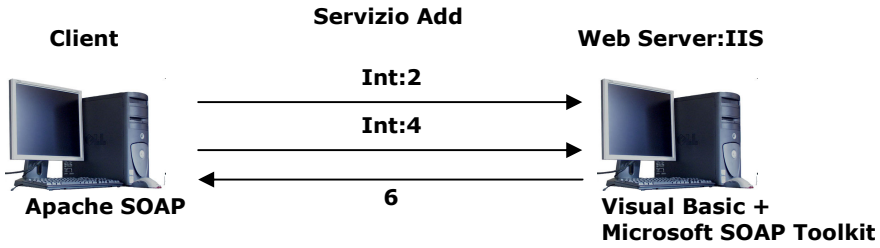


Figura 6:Incompatibilità tra toolkit SOAP

Apache SOAP per tutti i valori di risposta richiede sempre la loro tipologia. Ad esempio: **<**return xsi:type="xsd:int">6</return> è la risposta attesa, dove viene specificato esplicitamente che si tratta di un intero. Microsoft SOAP Toolkit non specifica la tipologia dei dati inviati. Ad esempio: <return>6</return> è la risposta inviata.

Problema: Apache SOAP e Microsoft SOAP Toolkit di base sono incompatibili. (Apache SOAP rigetta i dati inviati dal server).

È stato necessario introdurre un nuovo deserializzatore sul client Apache SOAP, in questo modo, il client accetta la risposta del server VB anche se non è specificato il tipo di dato inviato.

Apache Axis

- Rappresenta l'evoluzione di Apache SOAP 2.2, da cui eredita le caratteristiche di base. Elimina i difetti del predecessore: È perfettamente compatibile con Microsoft SOAP Toolkit.
- Supporta le specifiche WSDL 1.1.

Novità:

  - Supporto parziale delle specifiche SOAP 1.2.
  - Supporto per la pubblicazione automatica dei servizi (Java Web Service).
  - Generazione automatica (direttamente da un browser Internet) del documento WSDL per un servizio pubblicato.
  - Tool WSDL2Java e Java2WSDL.

1.4.4.2 Struttura dei messaggi SOAP

Abbiamo detto SOAP è basato su XML. In effetti,un messaggio SOAP non è altro che un documento XML che descrive una richiesta di elaborazione o il risultato di una elaborazione, dove le informazioni contenute nella richiesta e nella risposta vengono serializzate secondo un formato prestabilito, utilizzando XML come strumento che garantisce l'indipendenza dalla piattaforma.
SOAP consiste di tre parti: una busta, che definisce un framework, per la descrizione del contenuto del messaggio e della modalità di elaborazione (`SOAP envelope construct`), un insieme di regole di codifica per l'espressione di istanze di tipi di dati definiti dalle applicazioni (`SOAP encoding`)ed una convenzione per la rappresentazione di chiamate e risposte di una procedura remota(`SOAP RPC`).
SOAP necessita di un protocollo di trasporto: nella sua versione iniziale si appoggiava all' HTTP standard mentre nelle ultime versioni utilizza anche FTP e altri standard.

HTTP supporta diversi modi per la richiesta di informazioni mediante un'intestazione di richiesta; utilizza cioè dei metodi per descrivere le intenzioni del server oppure alcuni campi dell'intestazione per includere le coppie nome-valore dei dati di richiesta. Quando il server risponde genera un messaggio costituito da un'intestazione di risposta che include una riga di stato e i campi dell'intestazione per includere coppie nome-valore dei dati di risposta. Metodi e intestazioni forniscono un framework flessibile e semplice da capire.
Il protocollo SOAP è adatto a supportare un'architettura client-server:i dati richiesti ed elaborati fra client e server sono organizzati in messaggi SOAP e vengono trasportati attraverso il protocollo HTTP o un altro protocollo di trasporto. I messaggi SOAP sono fondamentalmente trasmissioni unidirezionali da un mittente ad un destinatario ma sono spesso combinati per implementare modelli del tipo richiesta/risposta.
Un vantaggio notevole offerto da SOAP consiste nell'imposizione di una struttura di dati per la richiesta e, quando disponibili, per la risposta.
Di seguito analizzeremo i messaggi SOAP generati quando un client si connette al servizio BancaDiRomaWS attraverso i quali mostrerò come viene rispettata la sintassi ed alcune caratteristiche dei messaggi SOAP, che si possono schematizzare come in figura:

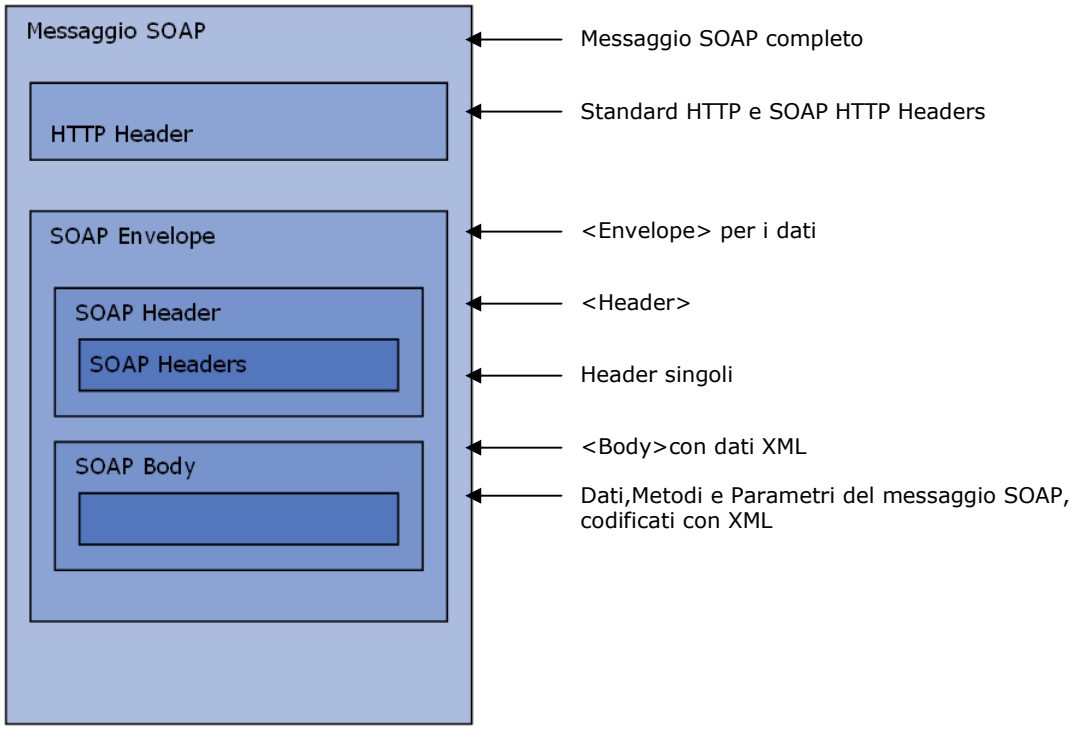


Figura 7:Struttura di un messaggio SOAP

1.4.4.3 Messaggi SOAP:la serializzazione dei dati

Un messaggio SOAP è costituito dai seguenti elementi:

- Envelope(obbligatorio):rappresenta il contenitore del messaggio e costituisce l'elemento root del documento XML;
- Header(opzionale):è un elemento che contiene informazioni globali sul messagio;
- Body(obbligatorio):rappresenta la richiesta di elaborazione o la risposta da una elaborazione;
- Fault(opzionale):se presente,fornisce informazioni sugli errori che si sono verificati durante l'elaborazione;è presente solo nei messaggi di risposta.

E' opportuno evidenziare che SOAP definsce soltanto la struttura dei messaggi non il loro contenuto. I tag per descrivere una richiesta di elaborazione o un risultato vengono definiti in uno schema specifico ed utilizzati all'interno della struttura SOAP sfruttando il meccanismo dei `namespace`. Prima di analizzare nel dettaglio la composizione del messaggio, ci sono alcuni aspetti nell'utilizzo che SOAP fa di XML che è opportuno evidenziare. Innanzitutto per descrivere i tipi di dati vengono utilizzati gli XML Schema. In secondo luogo SOAP ricorre ai namespace per evitare ambiguità nei messaggi, ed ogni elemento di un messaggio creato da un'applicazione può essere qualificato da un opportuno namespace. Comunque, il protocollo ne definisce due propri:

- `http://schema:xmlsoap.org/soap/envelope/`, per la envelope;
- `http://schema:xmlsoap.org/soap/encoding/`, per il meccanismo di serializzazione.

In un messaggio SOAP ogni elemento appartiene ad una delle seguenti categorie:

- elementi strutturali;
- elementi radice;
- elementi accessor;
- elementi indipendenti.

Gli elementi strutturali costituiscono la struttura del messaggio e sono `SOAP-ENV:Envelope`,`SOAP-ENV:Header` e `SOAP-ENV:Body`.

L'envelope è il primo elemento del documento XML e rappresenta l'intero messaggio.L'header è opzionale ma se presente deve fare seguito all'envelope e serve per poter estendere il messaggio. Il body,obbligatorio, è il contenitore dove sono inserite le informazioni che si devono inviare o ricevere e che tipicamente riguardano l'invocazione di metodi remoti o condizioni d'errore. Dei quattro elementoi che costituiscono un messaggio,quelli strutturati sono i soli che sono utilizzati per rappresentare istanze di tidi di dati.

Un elemento radice è un immediato discendente di uno dei due elementi strutturali, `SOAP-ENV:Body` o `SOAP-ENV:Header`. `SOAP-ENV:Body` può avere esattamente un solo elemento radice che costituisce la chiamata,la risposta o un errore. SOAP-ENV:Header invece può avere più elementi radici, uno per ogni estensione associata al messaggio. Gli `elementi accessor` sono usati per rappresentare dati, proprietà o campi di un oggetto. Ad ogni campo di un oggetto, infatti, corrisponde un elemento accessor, il cui nome del tag che lo rappresenta è lo stesso del nome del campo. Gli `elementi indipendenti` rappresentano istanze di tipi che sono referenziate, in modo indiretto, da altri elementi accessor.

1.4.4.4 SOAP e Remote Procedure Calls

Uno degli obiettivi di SOAP, oltre al trasferimento di dati,è l'incapsulamento all'interno dei messaggi delle chiamate a procedure remote (Remote Procedure Calls). Le informazioni necessarie ad invocare una SOAP RPC sono le seguenti:

- Indirizzo del nodo SOAP a cui è indirizzata la chiamata.
- Nome della procedura o metodo da invocare.
- Parametri con relativi valori da passare al metodo e parametri di output.
- Trasmissione (opzionale) della signature del metodo.
- Dati (opzionali) che possono essere stati trasportati come parte dei blocchi header.

SOAP RPC stabilisce come inserire e rappresentare queste informazione all'interno del corpo di un messaggio SOAP. Prima di tutto specifica come viene mappata l'interfaccia del metodo in strutture request/response,le quali vengono poi codificate come XML. La struttura relativa alla richiesta(request) viene chiamata con lo stesso nome del metodo che è invocato. Al suo interno sono contenuti tanti elementi quanti sono i parametri di input e input/output; tali elementi sono denominati con lo stesso nome dei parametri ai quali fanno riferimento ed appaiono nello stesso ordine che questi ultimi hanno nell'interfaccia del metodo. Allo stesso modo viene modellata la risposta del metodo (response) con la differenza che il nome associato alla struttura è per convenzione il nome di tale metodo seguito da "Response". All'interno di tale struttura si trova un elemento contenente il valore di ritorno del metodo e tanti elementi quanti sono i parametri di output e input/output.

RPC e il campo <Body> di SOAP

I metodi di chiamata e risposta RPC, sono entrambi posti nel campo Body di SOAP,mediante la seguente rappresentazione:

- l'invocazione al metodo è modellata come una struttura,
    - l'invocazione al metodo è vista come una singola struttura, contenente un ingresso per ogni parametro di input o di input/output. Questa struttura ha lo stesso nome e tipo del metodo,
        - ogni parametro è visto come un ingresso, con un nome e un tipo, corrispondenti rispettivamente al nome e al tipo del parametro. Questi compaiono nello stesso ordine della firma del metodo,
- la risposta del metodo è modellata come una struttura,
    - la risposta del metodo è vista come una singola struttura, contenente un ingresso per il valore di ritorno, e per ogni parametro di input o di input/output. Il primo ingresso è per il valore di ritorno, seguito dai parametri, posizionati nelle stesso ordine della firma del metodo,

- ogni parametro è visto come un ingresso, con un nome e un tipo, corrispondenti rispettivamente al nome e al tipo del parametro. Il nome dell'ingresso del valore di ritorno non è significativo. Allo stesso modo, il nome della struttura non è significativo. Per convenzione, si usa dare un nome alla struttura, dopo il nome del metodo, con l'aggiunta della stringa "Response".

• Il fallimento di un metodo, è codificato utilizzando l'elemento SOAP Fault.

Un'applicazione può elaborare delle richieste con dei parametri mancanti, ma può anche ritornare un errore. Dato che un risultato indica successo, e un errore indica fallimento, è un errore se la risposta del metodo contenere sia il risultato che l'errore.

RPC e il campo <Header> di SOAP

Informazioni addizionali relative alla codifica della richiesta del metodo, ma non facenti parte della formale firma del metodo, possono essere espresse nella codifica RPC. Se questo si verifica, devono essere espresse come sotto-elemento del campo Header di SOAP.

RPC e il campo <Fault> di SOAP

L'elemento Fault nel Body del messaggio SOAP, viene ancora una volta, utilizzato per la gestione degli errori per le RPC. Dalla specifica di SOAP 1.2, i principali codici di errore che il sottoelemento Faultcode può assumere, sono i seguenti:

• Server: il server non può gestire il messaggio, per esempio se non ha memoria sufficiente.
• DataEncodingUnknown: il server non comprende come sono stati codificati i dati all'interno del Body e/o dell'Header di richiesta (il valore dell'attributo encodingStyle).
• ProcedureNotPresent: il server non riesce a trovare la procedura specificata.
• BadArguments: il server non riesce ad analizzare i parametri oppure non c'è corrispondenza fra ciò che il server si aspetta e ciò che gli ha inviato il client.

1.4.5 UDDI

Abbiamo visto come i Web Services vengono descritti (WSDL, 1.4.2) ed in quale modo essi comunicano (SOAP, 1.4.3). Ma, a questo punto, una cosa che appare evidente è il fatto che due Web Service possono comunicare tra loro solo se l'uno conosce locazione e modalità di accesso dell'altro; ciò che manca è un sistema che renda possibile la ricerca di Web Service, secondo certi criteri come ad esempio la tipologia del servizio, sapere cioè quali mettono a disposizione una certa funzionalità, o l'appartenenza ad una data azienda. La tecnologia che si occupa di questo aspetto è UDDI. UDDI (Universal Description, Discovery and Integration), che è basato su XML ed utilizza SOAP per le comunicazioni da e verso l'esterno,definisce un meccanismo comune per pubblicare e trovare informazioni sui Web Services, in base alle loro descrizioni WSDL. Ciò che UDDI mette a disposizione è un registro nel quale si possa accedere, attraverso specifiche funzioni,per:
• pubblicare servizi che un'azienda rende disponibili,
• cercare aziende che mettono a disposizione un certo tipo di servizio,
• avere informazioni "Human Readable", cioè comprensibili all'utente, circa indirizzi, contatti o altro relativi ad una azienda.
• avere informazioni tecniche "Machine Readable", cioè interpretabili ed utilizzabili dalla macchina, relative ad un servizio in modo tale da potervisi connettere. Un registro UDDI è costituito in realtà da un database distribuito,cioè da molti registri distribuiti sulla rete,ognuno dei quali si trova sul server di una azienda che contribuisce allo sviluppo di questo archivio pubblico,e connessi fra loro. Il sistema mantiene una centralizzazione virtuale,cioè l'informazione che viene registrata su uno dei nodi (registri) del sistema viene propagata e resa disponibile su tutti gli altri tramite una loro sicronizzazione. Questo,oltre che ad alleggerire il carico di lavoro che un singolo nodo deve sopportare,contribuisce alla protezione del sistema contro possibili situazioni di failure del database,grazie alla ridondanza dei dati.

Per semplificità consideriamo UDDI come un unico grande registro. Esso può essere visto come le pagine gialle,nelle quali cerchiamo informazioni sulle aziende che attraverso esse ottengono maggiore visibilità e di conseguenza la possibilità di avere un più alto numero di clienti.

La registrazione di un servizio all'interno di un UDDI Registry è costituita da 3 parti:
• Yellow Pages
• White Pages
• Green Pages
Con le Yellow Pages (pagine gialle) le aziende ed i loro servizi vengono catalogati sotto differenti categorie e classificazioni. Nelle White Pages (pagine bianche) possiamo trovare informazioni come gli indirizzi di una azienda o contatti ad essa relativi. Infine vi sono le Green Pages (pagine verdi), dove sono contenute informazioni tecniche relative ai servizi,grazie alle quali questi ultimi possono essere invocati. Questa è la suddivizione dell'informazione dentro UDDI a livello concettuale ma vediamo come è realmente strutturato un UDDI Registry.

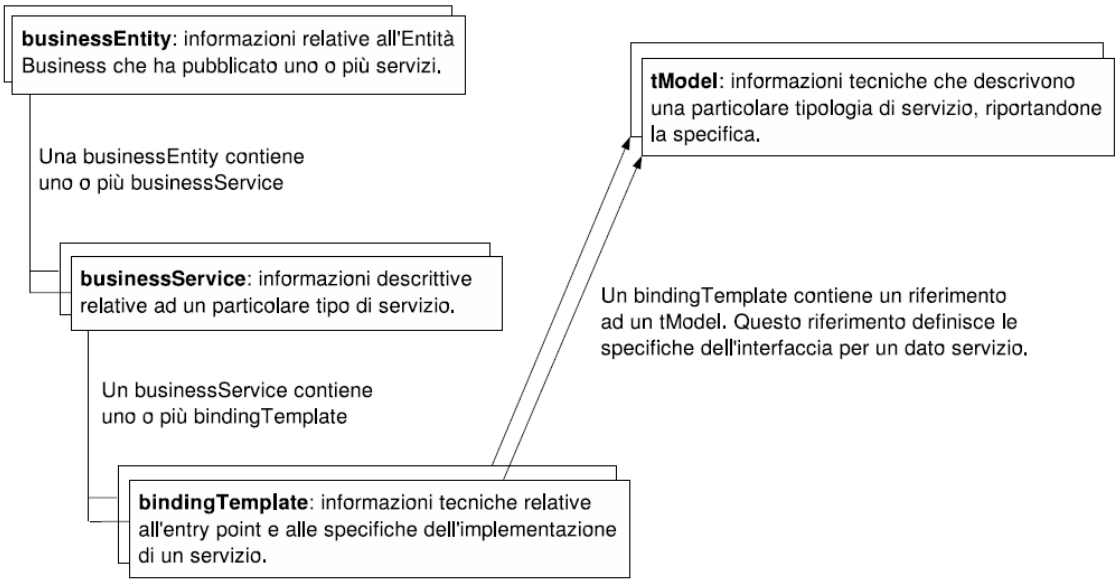


Figura 8:Struttura principale di un registro UDDI

1.4.5.1 Interazioni con UDDI

Le interazioni con un registro UDDI sono suddivisibili in due categorie: quelle finalizzate alla ricerca di dati e quelle il cui scopo è l'inserimento o la modifica degli stessi. Questa suddivizione è rappresentata da due API (Application Programming Interface) tramite le quali si accede ai contenuti di UDDI: Inquiry API e Publish API.

- Inquiry API è utilizzata per la ricerca di informazioni;
- Publish API è utilizzata invece per l'inserimento, l'aggiornamento e la cancellazione dei dati.

Per informazioni più dettagliate su UDDI fare riferimento a[8].

Capitolo 2 – Apache Axis

2.1 Axis

Invocare un Web Service significa inviargli un messaggio SOAP contenente una richiesta di esecuzione di un'operazione e i dati di input necessari. Quello che il Web service ritornerà sarà un altro messaggio SOAP, contenente il risultato dell'operazione richiesta. SOAP non definisce un binding ad uno specifico linguaggio di programmazione: è compito del progettista scegliere un linguaggio e definirne l'implementazione. In questo contesto usare Java come linguaggio di programmazione per sviluppare applicazioni SOAP richiede l'implementazione Java per SOAP del binding specifico. Allo stato attuale, ci sono molti venditori di applicazioni SOAP che hanno dato vita ad implementazioni di SOAP basate su Java per sviluppare da applicazioni Web a Web services. In generale, l'uso di Java per lo sviluppo di applicazioni SOAP rende scalabile e portabile le applicazioni che si vogliono costruire e possono, inoltre, interoperare con applicazioni eterogenee residenti su piattaforme diverse risolvendo problemi di incompatibilità. Inoltre, avere applicazioni basate su SOAP che adottano un'infrastruttura basata su J2EE, permette di ereditare le caratteristiche dei containers come transazioni, sicurezza, connettività a back-end. Una lunga lista di comunità open source e fornitori di piattaforme basate su Web services hanno realizzato implementazioni SOAP adottando la piattaforma Java e le API che mette a disposizione. Tra queste ha avuto un notevole successo l'implementazione di SOAP per Java fornita dall'Apache Group denominata Axis (Apache eXtensible Interaction System). Apache Axis rappresenta quindi un'implementazione del protocollo SOAP ("Simple Object Access Protocol") definito da W3C.

AXIS (Apache eXtendable Interction System) è essenzialmente un motore SOAP in grado di processare messaggi SOAP e permette di sviluppare client, server e gateway SOAP. In realtà AXIS non è propriamente un semplice engine SOAP, ma piuttosto un frame work per realizzare sistemi di integrazione basati su SOAP.
Caratteristiche fondamentali che sono alla base di AXIS sono:

- flessibilità: è possibile estendere l'engine per eseguire elaborazioni custom degli header dei messaggi. Axis fornisce funzionalità per facilitare l'inserimento di nuove estensioni nell'engine che consentono di personalizzare l'elaborazione e la gestione del sistema. Axis fornisce un deployment descriptor per descrivere le vari componenti come i servizi, gli oggetti handler, i serializzatori e i deserializzatori e così via. Il deployment descriptor (WSDD) è usato per eseguire il deployment di un Web service;
- componibilità: i gestori di richieste, chiamati Handler in terminologia Axis, possono essere riutilizzati;
- indipendenza dal meccanismo di trasporto dei messaggi: in questo modo è possibile utilizzare vari protocolli oltre ad HTTP come SMTP, FTP e messaging.
- velocità: il meccanismo di parsing utilizzato nell'implementazione di Apache SOAP era basato su DOM (Document Object Model), il quale impiega più tempo rispetto ad un parser basato su SAX (Simple API for XML). Infatti, Axis usa SAX, un parser basato sugli eventi che opera più a basso livello e permette di ottenere alte prestazioni;
- deployment orientato alle componenti: Axis presenta il concetto di handlers per implementare patterns comuni di elaborazione per le applicazioni. Un handler, un oggetto capace di processare un messaggio, è il responsabile delle elaborazioni specifiche associate al flusso di input, output e fault;
- stabilità: Axis definisce una serie di interfacce stabili, che facilitano la migrazione verso nuove versioni Axis;
- framework di trasporto: Axis, come anticipato, offre un framework di trasporto fornendo *senders* e *listeners* per SOAP sopra vari protocolli come SMTP, FTP, etc.

Axis è quindi di un'API di programmazione e deployment di Web services che permette di lavorare ad un livello di astrazione elevato, evitando così di dover maneggiare direttamente l'envelope SOAP. Con Axis è possibile, dunque, implementare Web services e anche sviluppare client di servizi di terzi. Per semplicità, negli esempi che tratterò viene esposto solo l'aspetto di funzionalità RPC e non quello relativo ai vari approcci di messaging supportati da Axis. Ciò è

comunque in linea con la maggior parte della casistica Web services che si basa su meccanismi request/response a dispetto del modello asincrono.

Axis permette lo sviluppo di WS sia in Java che in C++ ma in questo caso l'attenzione sarà focalizzata solo sul linguaggio di casa Sun, il quale implementa tra i tanti gli standard:

- JAX-RPC: SOAP con il classico RPC
- SAAJ: permette di manipolare il messaggio SOAP e gestire gli attachment
- tanti altri…

Axis implementa quindi il modello JAX-RPC e supporta anche SAAJ.
Non tutto nasce però da Apache, il prodotto iniziale è merito di IBM, che poi molto gentilmente decise di regalare al consorzio Apache tutto il codice. Viene alla luce quindi Apache SOAP e poi la sua evoluzione: Apache Axis.

Le caratteristiche più importanti del framework sono:

- implementazione SOAP 1.1/1.2;
- supporto JWS (Java Web Services): permette un facile e immediato deploy dei WS;
- supporto serializzazione/de-serializzazione;
- implementazione WSDL;
- utility WSDL2Java e Java2WSDL;
- Soap Monitor e TCP Monitor: due applicazioni scritte in Java che permettono di monitorare il traffico SOAP;
- possibilità di usare tre differenti metodi per l'invocazione dei WS: Dynamic Invocation Interface, Stub generato dal WSDL e Dynamic Proxy.

I componenti di Axis sono quindi: un engine per l'elaborazione dei messaggi SOAP, handler per la gestione dei messaggi, vari meccanismi di deploy, serializzazione/de-serializzazione, configurazione e di trasporto dei messaggi.

2.2 JAX-RPC

Una parte molto complessa dell'elaborazione distribuita consiste nella gestione delle chiamate remote. Si tratta di chiamare del codice residente su un altro server,passandogli eventualmente dei parametri, e riceverne la risposta,anch'essa con un'opportuna notifica. Attualmente esistono molti standard per la codifica e la trasmissione delle chiamate e delle risposte.

- CORBA,un frame work per la gestione di oggetti distribuiti,che utilizza l'internet inter-ORB Protocol (IIOP)
- DCOM, di microsoft,utilizza l'Object Remote Procedure Call (ORPC)
- .NET Remoting, per la piattaforma .NET di Microsoft,può utilizzare diversi protocolli,compreso SOAP stesso.
- Java, per la Java Remote Method Invocation (RMI) utilizza il Java Remote Method Protocol (JRMP)

Come detto nel capitolo precedente SOAP si propone come sostituto per tutti questi protocolli. Invece di utilizzare complicati bridge per tradurre un protocollo in un altro, quando due framework diversi devono comunicare tra loro,SOAP si propone come protocollo universale di trasmissione dei dati di RPC. Tuttavia SOAP è stato studiato per avere usi che si estendono oltre la semplice RPC. SOAP non si basa su tecnologie proprietarie e la sua applicazione è completamente libera.

Le tecnologie messe a disposizione dalla piattaforma Java sono le seguenti:

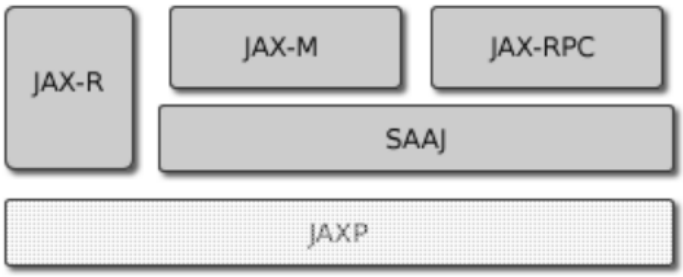


Figura 9:Tecnologie Java per SOAP

Queste tecnologie hanno il seguente scopo:

- JAXM (Java API for XML Messaging). Implementazione della messaggistica SOAP a basso livello; è simile a JMS (Java Message Service); fornisce una robusta infrastruttura di messaging per spedire e ricevere messaggi SOAP.
- SAAJ (SOAP with Attachments API for Java). Implementazione del modello informativo degli elementi di SOAP;è un'API che modella la struttura di un messaggio SOAP e anche qualche limitata message-delivery capability.
- JAX-RPC (Java API for XML Remote Procedure Call). Supporto client e server allo sviluppo rapido di servizi Web basati su SOAP (e non solo) e WSDL con un modello di sviluppo simile ad RMI.
- JAXR (Java API for XML Registries). Supporto a registri di servizi Web come UDDI.

Sia JAXM che JAX-RPC implementano SOAP,quindi si potrebbe pensare che l'utilizzo, o l'approfondimento, delle due sia indifferente. In realtà non è così: JAXM e JAX-RPC non sono una ripetizione della stessa tecnologia con nomi diversi.
Per prima cosa, entrambe utilizzano SAAJ come implementazione del modello informativo di SOAP, ma il loro scopo è differente.

- **JAXM** fornisce un controllo fine sul messaggio (tramite SAAJ), consentendo di manipolare ciascun elemento della richiesta ed accedere a qualsiasi elemento della risposta SOAP. Viene utilizzato in soluzioni di messaggistica, come ad esempio: l'invio asincrono di informazioni real-time, l'interfacciamento con servizi strutturati in modo particolare (non standard), collegamento a servizi che hanno una risposta dinamica (cambia la struttura al variare della richiesta o dei dati ritornati);
- **JAX-RPC** implementa invece un generico sistema di RPC su XML; la tecnologia non è vincolata a SOAP, ma aperta ad eventuali evoluzioni future. E' invece basata fortemente su WSDL, che utilizza per implementare la mappatura XML->Java e viceversa. Con JAX-RPC non è necessario codificare a basso livello, se si dispone di un WSDL, si possono generare le classi Java che implementano il client ed il server. La stessa cosa può avvenire a partire da una interfaccia che estende *java.rmi.Remote*, in modo similare al funzionamento di RMI e di *rmic*.

Le JAX-RPC sono delle api definite con lo scopo di supportare le Remote Procedure Call basate su XML nella piattaforma Java. Definiscono un mapping tra java e WSDL, sia per generare classi Java a partire da un WSDL,sia per generare web service endpoints a partire da classi e interfacce,varie api per la scrittura dei client, un mapping per i messaggi SOAP e soprattutto un framework estendibile composto di moduli che processano i messaggi SOAP e che sono chiamati handlers. Client e Server side handlers sono organizzati in una lista ordinata, una handler chain, e sono invocati in ordine ogni qual volta un messaggio SOAP viene spedito o ricevuto prima che esso raggiunga la sua destinazione e sia trasformato nella chiamata ad un metodo java. Gli handlers sono invocati passando loro come parametro un oggetto MessageContext che consente di accedere al messaggio SOAP, modificarlo o settare alcune proprietà a beneficio degli handler successivi. Ogni handler ha un ben determinato ciclo di vita, quando il JAX-RPC runtime system lo crea,prima chiama il suo metodo init, poi altri metodi che processano in qualche modo il messaggio a seconda dell'implementazione e infine lo

termina chiamando il metodo destroy. Il fatto che il framework sia estendibile, cioè che sia possibile creare nuovi handlers ed inserirli nella catena è molto importante poiché consente all'utente accesso al messaggio SOAP. Anche l'architettura di Axis è esattamente questa, si tratta fondamentalmente di un SOAP processor che processa i messaggi con una catena di handler e l'ultimo della catena, chiamato pivot handler,è incaricato di chiamare il metodo java, che corrisponde all'operazione wsdl desiderata, nella classe deployata come implementazione del servizio.

JAX-RPC è dunque più indicato per sviluppare applicazioni complesse, con molti servizi, e per integrare Web Service esistenti e completamente standard, quando non sia necessario entrare nel dettaglio tecnico della struttura dei messaggi SOAP di richiesta e risposta. Con JAX-RPC un servizio Web viene acceduto come se fosse un oggetto Java locale.

Axis implementa JAX-RPC ed è compilato nel JAR file axis.jar, che implementa appunto le API JAR-RPC dichiarate nei file JAR jaxrpc.jar e saaj.jar. Tutti i file,le librerie di cui necessita Axis sono contenute in un package,axis.war, che può essere copiato in un servlet container.

In Axis agli handler viene passato un oggetto MessageContext definito da Axis che aggiunge funzionalità rispetto a quello delle jax-rpc. L'oggetto MessageContext contiene tutte le informazioni rilevanti sul servizio al quale l'operazione correntemente invocata e processata dall'handler appartiene. E' infatti possibile oltre ad ottenere il messaggio SOAP di richiesta, (e quello di risposta se abbiamo un handler anche nella catena di ritorno),recuperare informazioni sul nome del servizio, il nome dell'operazione, una serie di parametri specificati dall'utente al momento della pubblicazione dell'handler e altro ancora.

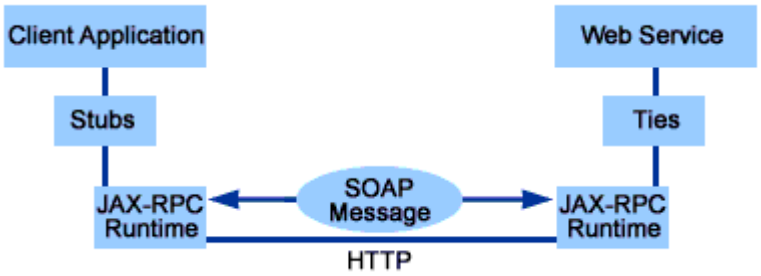


Figura 10:Posizionamento del supporto JAX-RPC nella comunicazione fra client e web service.

2.3 Architettura di Axis

Essendo un processore SOAP, Axis sostanzialmente elabora messaggi. In generale, in Axis, un messaggio SOAP viene elaborato passando per una serie di handler. L'oggetto Java che passa tra un handler ed un altro è il MessageContext che contiene tutte le informazioni necessarie all'elaborazione del messaggio stesso. In particolare il MessageContext contiene il messaggio di richiesta e di risposta ed un insieme di proprietà (oggetti) che i vari Handler possono passarsi. Più Handler possono essere raggruppati in insiemi ordinati per formare un *Chain* (catena di Handler). Un chain è una collezione di handler ma è esso stesso un handler infatti l'interfaccia org.apache.axis.Chain estende l'interfaccia org.apache.axis.Handler integrandola con alcuni metodi per aggiungere e rimuovere gli handler della catena. Ogni handler appartenente al chain,ha la possibilità di leggere il messaggio e di modificarlo prima di passarlo all'handler successivo della catena. Alcuni chain hanno la peculiarità di avere un pivot-point. Un pivot-point è un Handler in cui cessa di essere processata una richiesta ed inizia ad essere processata una risposta. I chain con pivot-point sono anche chiamati targeted chain. Gli Handler possono essere divisi in tre tipologie: *trasporto*, *globali* e *service-specific*. Gli *handler di trasporto*, se presenti, possono svolgere funzioni di compressione dati, autenticazione e si differenziano proprio in base al protocollo di trasporto. Il terzo tipo di handler, chiamato *service-specific*, è utile quando si vuole che l'handler venga eseguito solo con un determinato servizio allo scopo di implementare funzionalità avanzate come crittografia, autenticazione o altro. Le specifiche sul come e sull'ordine di invocazione degli handler service-specific,vengono

definite nella fase di deploy. Quando arriva un messaggio SOAP sul server, esso viene prima letto dal TransportListener che lo inserisce in un oggetto Message (org.apache.axis.Message), successivamente l'oggetto Message viene inserito in un MessageContext (org.apache.axis.MessageContext). Il MessageContext, viene caricato, oltre che del messaggio, anche di altre informazioni tra cui il parametro SOAP Action presente nell'header http ed il parametro transportName che contiene il tipo di protocollo di trasporto usato (es. HTTP). Il MessageContext a questo punto viene passato alla catena di handler di trasporto, che contiene un request chain,un response chain o entrambi. Se la catena di trasporto è configurata,il MessageContext viene passato come parametro al metodo invoke() dell'handler. Dopo essere passato per la catena di handler di trasporto, il MessageContext passa per quella degli handler globali(se definiti). Alla fine il MessageContext, arriva al targeted-chain service-specific che procederà ad inviare il MessageContext ad un provider. Il provider, anche esso un handler, si occupa di gestire la logica di codifica e di background del servizio. Per esempio nel caso in cui si stia usando lo stile RCP per la codifica dei messaggi SOAP,la classe incaricata come provider è org.apache.axis.providers.java.RPCProvider la quale si occupa di chiamare le operazioni del servizio vero e proprio.

Il percorso lato client è speculare rispetto a quello del server con la differenza che non è presente un provider in quanto non vi alcuna fornitura di servizio. Il sender si occupa di effettuare le operazioni relative al protocollo di trasporto (handshake, connessione ecc) e di inviare e ricevere i messaggi. Anche lato client è possibile specificare delle chain il cui funzionamento è identico a quello visto sul server.

Axis gioca un ruolo fondamentale sia sul lato client del Web service (colui che utilizza il servizio) che sul lato server (colui che fornisce il servizio). Verranno analizzati entrambi gli aspetti architetturali mettendo in rilievo le componenti che formano il nucleo di Axis e della comunicazione.

2.3.1 Handlers e Message Path in Axis

Molto semplicemente, Axis è tutto ciò che riguarda l'elaborazione dei messaggi. Durante l'elaborazione dei messaggi vengono eseguiti una serie di *Handlers* invocate in un ordine che varia a secondo della configurazione del deployment e della posizione dell'*Engine* (client o server). L'elaborazione di un servizio SOAP è fatta passando un *context message* ad ogni componente. Un *context message* è una struttura che contiene un *request message*, un *response message* e una lista di proprietà (attributi dell'header SOAP). Prima di analizzare il funzionamento dell'architettura definiamo quelli che sono i concetti chiave di Axis:

- Le *Handler* sono utili ai fini del processamento dei messaggi; un *handler* potrebbe spedire una richiesta e ricevere una risposta oppure processare una richiesta e produrre una risposta. Un *Handler* può appartenere a tre famiglie: *transport-specific*, *service-specific* o *global*.
- Le *Handlers* di ciascuna famiglia sono combinate in Chains**.** Quindi la sequenza complessiva di *Handlers* comprende tre tipi di Chains: *transport*, *global* e *service*.

Come detto Axis interviene in due modi: quando viene invocato come server e quando viene invocato come client.

2.3.1.1 Message path server-side

Il cammino di un messaggio lato server è mostrato nella seguente figura:

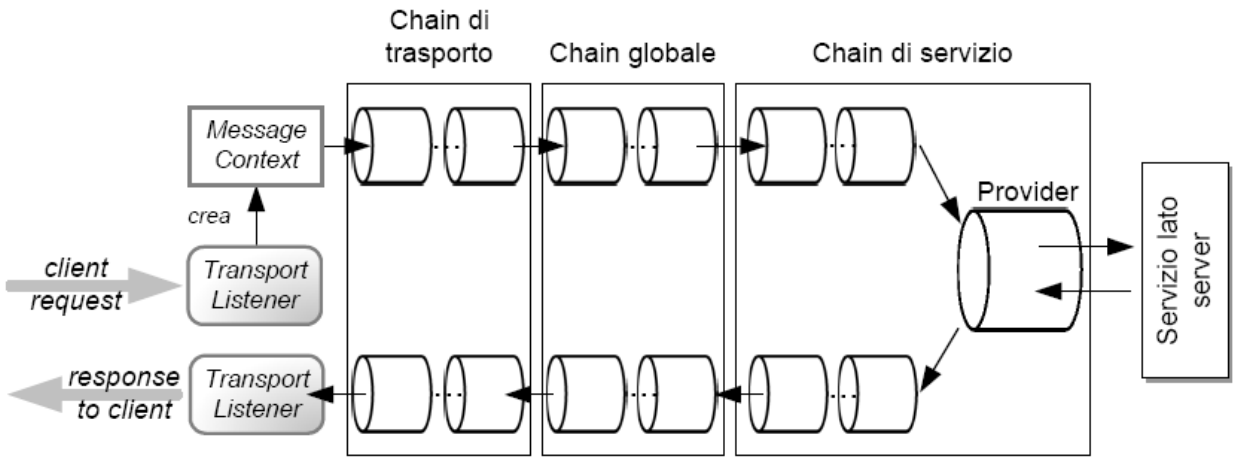


Figura 11:Il cammino del *Message* sul Server. I cilindri piccoli rappresentano le *handlers*, quelli grandi, invece, rappresentano le *chains.*

Un *request message* è spedito(con una certa specifica di protocollo) al nodo di elaborazione del messaggio del *transport listener.* In questo caso assumo che il Listener sia una servlet http. Il *transport listener* converte il messaggio in entrata in un oggetto *Message* (org.apache.axis.Message)e lo incapsula in un oggetto *MessageContext*. Successivamente il *transport listener* carica varie proprietà specificate come attributi di SOAP (Es: l'action header di SOAP) e imposta il tipo di trasporto come proprietà sul *MessageContext* – in questo caso SOAPAction HTTP header sarà settato con la proprietà "http.SOAPAction". Il *Transport Listener* setta quindi il valore del transportName nel MessageContext , in questo caso "http".
Il tipo di trasporto, come evidenziato, può essere HTTP, SMTP o FTP. Una volta che il *MessageContext* è inizializzato viene passato all'*AxisEngine*. Dopo che l'*AxisEngine* ha individuato il tipo di trasporto, viene invocata la *request Chain* del *Transport*, se questa esiste, passandogli il *MessageContext*. L'*Engine*, a questo punto, localizza una g*lobal request Chain* e invoca gli *Handlers* specificati; dopodiché alcuni Handlers impostano il campo *serviceHandler* del *MessageContext* (questo viene tipicamente effettuato nel trasporto http dal "URLMapper" Handler, che definisce un URL tipo "http://localhost/axis/services/AdminService" to the "AdminService" service). Il campo serviceHandler indicherà l'handler da invocare per eseguire il servizio richiesto (Es: RPC). I Services in Axis sono tipicamente istanze della "SOAPService" class (org.apache.axis.handlers.soap.SOAPService) che, oltre a contenere una *request Chain* e una *response Chain*, contengono anche un *Provider* e un *Handler* responsabile dell'implementazione della logica del back end del servizio. Nel caso di richieste RPC il *Provider* è la classe org.apache.axis.providers.java.RPCProvider; il provider è semplicemente un altro Handler che,quando invocato, tenta di chiamare l'oggetto java la cui classe è definita nel parametro "className" specificato nel file di *deploy*. Il provider utilizza la convenzione SOAP RPC per determinare il metodo da chiamare,e verificare che i tipi definiti nel file XML combacino con i tipi dei parametri richiesti dal metodo richiamato.

2.3.1.2 Message path client-side

Il cammino di un messaggio lato client,come visualizzato in figura, è simile al cammino lato server ma l'ordine dell'invocazione delle *Chain* è inversa.

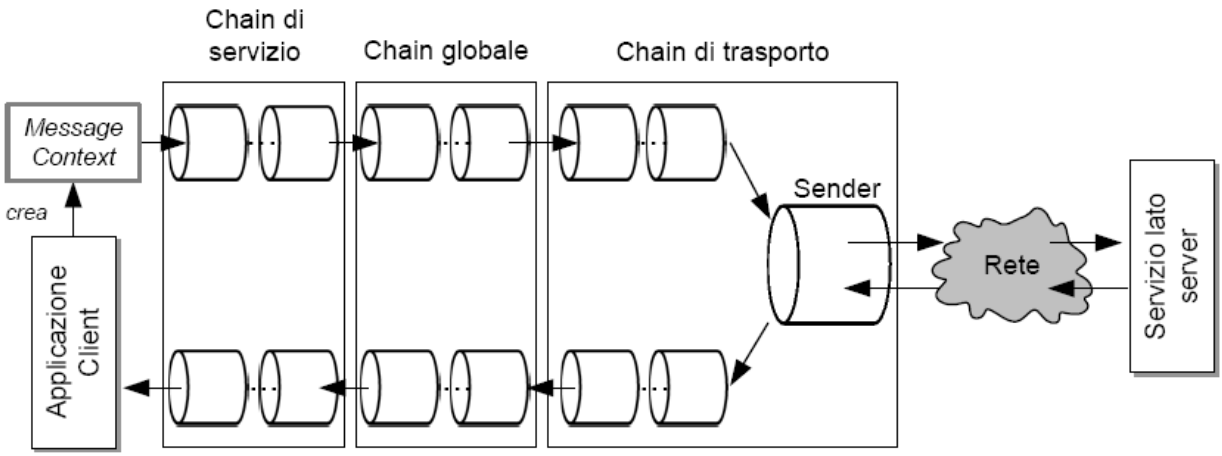


Figura 12:Il cammino del Message sul Client. I cilindri piccoli rappresentano le handlers, quelli grandi, invece, rappresentano le chains.

Nello strato *Service* non compare il *provider* poiché il servizio è fornito da un nodo remoto, ma vi sono comunque la *request Chain* e la *response Chain* che processano le *service-specific* della richiesta e della risposta. Successivamente viene invocato il *Global request Chain* e infine il *Transport*. Nel *Transport* gioca un ruolo fondamentale l'handler *Transport Sender* che invia la richiesta SOAP al Server Axis e ottiene eventualmente una risposta. La risposta viene incapsulata nel campo *responseMessage* del *MessageContext*. Il *MessageContext* viene propagato attraverso le *response Chains* del *transport*, del *global* e del *service*.

2.3.1.3 Schema riassuntivo del message path

Transport Listener

1. Un messaggio arriva (in modo specifico al transport protocol) al Transport Listener [*HTTP servlet*];
2. Il Transport Listener formatta i dati ricevuti in un oggetto Message e lo inserisce in un MessageContext;
3. Il Transport Listener setta altre informazioni nel MessageContext quali il trasportName (*HTTP*)
4. Passa il MessageContext all'engine di Axis.

Axis Engine

1. Cerca il transport attraverso il transportName. Il trasport contiene un request Chain, un response Chain o entrambi. Un Chain e' una sequenza di Handler invocati a turno;
2. Se esiste un trasport request Chain viene passato il MessageContext al suo metodo invoke;
3. Successivamente viene invocata il global request Chain
   - Durante i passi 2 e 3 qualche Handler ha settato il campo serviceHandler del MessageContext (*URLMapper Handler*)
4. Il servizio deve avere un provider, che e' un Handler responsabile dell'implementazione della logica back end del servizio
   - Nelle request RPC il provider e' org.apache.axis.providers.java.RPCProvider, che chiama una classe Java determinata attraverso il parametro className in fase di deploy.

TrasportSender

- Il suo compito e' quello di eseguire le operazioni specifiche per il protocollo di trasporto per inviare e ricevere I messaggi verso e da il server SOAP di destinazione;
- La Response, se esiste, e' inserita nel campo responseMessage del MessageContext;

- Il MessageContext viene infine propagato all'indietro attraverso i Chain Transport, Global e Service.

2.4 Installare Axis

Per iniziare bisogna scaricare il file d'installazione dal sito ufficiale[15]. All'interno di questo file, oltre alla documentazione, esempi, etc., è presente la cartella axis (sotto la cartella webapps) che contiene tutte le risorse necessarie al funzionamento dell'engine. Questa cartella va inserita nel Servlet Engine di cui si dispone. In questo contesto utilizzerò la versione 5.5.17 di Apache Tomcat da [16] e la versione 1.4 di Axis.
Tomcat di default resta in ascolto sulla porta 8080. Nel caso in cui tale porta sia utilizzata da una qualche altra applicazione nel sistema bisogna modificarla aprendo il file server.xml contenuto nella directory */web/tomcat/conf* e sostituendo 8080 con la porta desiderata,come mostrato di seguito:

```
<ConnectorclassName="org.apache.tomcat.service.PoolTcpConnector">
<Parametername="handler"
value="org.apache.tomcat.service.http.HttpConnectionHandler"/>
<Parametername="port" value="8081"/> </Connector>
```

In questo caso Tomcat si metterà in ascolto sulla porta 8081 e la cartella di Axis va semplicemente copiata sotto la webapps dello stesso. In questo modo ho dotato il server di un Web service engine (di fatto una struttura Servlet) in grado di esaudire richieste provenienti da qualsiasi client WS. Se l'installazione è andata a buon fine e il Web Server Tomcat è stato avviato, è possibile connettersi ad esso tramite un browser. In locale basta digitare `http://localhost:8081` e apparirà la pagina inizale di Tomcat.

Successivamente bisogna aggiungere le seguenti variabili d'ambiente:

```
AXIS_HOME: C:\Programmi\axis-1_4
AXIS_LIB: %AXIS_HOME%\lib
AXISCLASSPATH: %AXIS_LIB%\axis.jar;
                          %AXIS_LIB%\commons-discovery-0.2.jar;
                          %AXIS_LIB%\commons-logging1.0.4.jar;
                          %AXIS_LIB%\jaxrpc.jar;
                          %AXIS_LIB%\saaj.jar;
                          %AXIS_LIB%\log4j-1.2.8.jar;
                          %AXIS_LIB%\xml-apis.jar;
                          %AXIS_LIB%\xercesImpl.jar;
                          %AXIS_LIB%;
```

Aggiungere alla variabile d'ambiente `CLASSPATH` la stringa `%AXISCLASSPATH%;`

La variabile `$AXISCLASSPATH` rappresenta i riferimenti alle librerie jar che contengono l'implementazione degli standard sui WS come SAAJ e JAX-RPCl. L'ultima libreria include il parser XML Xerces;nella distribuzione di Axis Java include anche il parser Crimson ma Apache consiglia Xerces: entrambi funzionano adeguatamente quindi la scelta è puramente personale e la mia è ricaduta su Xerces.

La verifica della corretta integrazione tra Axis e Tomcat si ottiene digitando l'indirizzo(localhost se in locale oppure l'indirizzo ip della machina sulla quale sono installati Tomcat ed Axis) `http://localhost:8081/axis`: dovrebbe compare la semplice home page testuale di Axis da cui è possibile effettuare alcune operazioni. L'Home Page che verrà visualizzata è la seguente:

Apache-AXIS

Language: [en] [ja]

Hello! Welcome to Apache-Axis.

What do you want to do today?

- Validation - Validate the local installation's configuration
  see below if this does not work.
- List - View the list of deployed Web services
- Call - Call a local endpoint that list's the caller's http headers (or see its WSDL).
- Visit - Visit the Apache-Axis Home Page
- Administer Axis - [disabled by default for security reasons]
- SOAPMonitor - [disabled by default for security reasons]

To enable the disabled features, uncomment the appropriate declarations in WEB-INF/web.xml in the webapplication and restart it.

Validating Axis

If the "happyaxis" validation page displays an exception instead of a status page, the likely cause is that you have multiple XML parsers in your classpath. Clean up your classpath by eliminating extraneous parsers.

If you have problems getting Axis to work, consult the Axis Wiki and then try the Axis user mailing list.

La cosa importante da verificare è che siano state caricate tutte le librerie necessarie al corretto funzionamento di Axis seguendo il link `Validation` presente nell'Home Page. Seguendo tale link verrà mostrata una pagina simile alla seguente, nella quale non dovranno esserci messaggi di errore o di warning.

Axis Happiness Page

Examining webapp configuration

Locazione delle librerie correttamente installate

Language: [en] [ja]

Needed Components

- Found SAAJ API (javax.xml.soap.SOAPMessage) at an unknown location
- Found JAX-RPC API (javax.xml.rpc.Service) at C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\lib\jaxrpc.jar
- Found Apache-Axis (org.apache.axis.transport.http.AxisServlet) at C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\lib\axis.jar
- Found Jakarta-Commons Discovery (org.apache.commons.discovery.Resource) at C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\lib\commons-discovery-0.2.jar
- Found Jakarta-Commons Logging (org.apache.commons.logging.Log) at C:\Programmi\Apache Software Foundation\Tomcat 5.5\bin\commons-logging-api.jar
- Found Log4j (org.apache.log4j.Layout) at C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\lib\log4j-1.2.8.jar
- Found IBM's WSDL4Java (com.ibm.wsdl.factory.WSDLFactoryImpl) at C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\lib\wsdl4j-1.5.1.jar
- Found JAXP implementation (javax.xml.parsers.SAXParserFactory) at an unknown location
- Found Activation API (javax.activation.DataHandler) at an unknown location

Optional Components

Componenti aggiuntivi installati

- Found Mail API (javax.mail.internet.MimeMessage) at C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\lib\mail.jar
- Found XML Security API (org.apache.xml.security.Init) at C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\lib\xmlsec-1.3.0.jar
- Found Java Secure Socket Extension (javax.net.ssl.SSLSocketFactory) at an unknown location

The core axis libraries are present.The optional components are present.

Note: Even if everything this page probes for is present, there is no guarantee your web service will work, because there are many configuration options that we do not check for. These tests are *necessary* but not *sufficient*

Examining Application Server

Servlet version	2.4
XML Parser	org.apache.xerces.jaxp.SAXParserImpl
XML ParserLocation	C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\lib\xercesImpl.jar

Parser utilizzato

Locazione del Parser

Per conoscere i servizi attualmente pubblicati nel Web Server si deve cliccare il link `List-View the list of deployed Web Services` dalla pagina iniziale.

Cliccando sul link `wsdl` è possible visualizzare il file wsdl di ogni servizio e verificare che questo è effettivamente deployato ed in esecuzione.

Da notare che gli 'instant' JWS Web Services che Axis supporta non sono mostrati in questa lista.

2.5 Deployment di Web Service su Axis

Il deployment è un'operazione che deve essere effettuata su un web service affinchè questo possa essere utilizzato. Attraverso l'operazione di deployment si notifica ad Axis la presenza di un nuovo servizio specificando il suo nome e la sua locazione.
In Axis ci sono due modi per effettuare il deployment di un servizio: [13, 12]
• deployment dinamico attraverso l'uso di file .jws,
• deployment statico attraverso l'uso di deployment descriptor.
Nel caso di un web service semplice si può utilizzare il primo tipo di deployment. Dopo aver creato il file java che implementa il web service è sufficiente sostituire l'estensione .java di tale file con l'estensione .jws e copiarlo nella directory $CATALINA HOME/webapps/axis/ o in una sua subdirectory. A questo punto il web service è immediatamente disponibile all'indirizzo http://localhost:8081/axis/nomeFile.jws?method=nomeMetodo poichè Axis tratta i file .jws in modo simile a quello in cui Tomcat tratta una JSP (Java Server Page). Ciò che Axis fa è localizzare il file,compilarlo e posizionare attorno ad esso un wrapper,cioè una sorta di filtro, che converte le chiamate SOAP dirette verso il servizio in invocazioni di metodi java. E' anche possibile vedere il WSDL del servizio all'indirizzo http://localhost:8081/axis/nomeFile.jws?wsdl
Via browser apparirà una pagina bianca della quale è necessario visualizzare il sorgente per vedere il WSDL. Questa soluzione rappresenta un meccanismo di deployment facile ed immediato,ma che può essere utilizzata solo nei casi di web service molto semplici. Inotre tale soluzione presenta alcuni inconvenienti come ad esempio il fatto di non poter specificare quali metodi devono essere esposti e quali invece non devono esserlo. Altro svantaggio è quello di non poter definire un mapping dei tipi SOAP/Java personalizzato.
Nel caso invece di web service più complessi o nel caso in cui le caratteristiche della soluzione appena vista non siano adeguate per la nostra applicazione,vi è il metodo di deployment standard di Axis. Questo metodo utilizza dei particolari file chiamatiWeb Services Deployment Descriptor (WSDD) all'interno dei quali sono inserite le informazioni relative al servizio di cui vogliamo fare il deployment. I file .wsdd permettono di specificare molte più cose (vedi [14]) riguardo all'esposizione in rete del servizio. Vedremeo nel dettaglio le caratteristiche di un file .wsdd successivamente nel paragrafo 2.7.2 quando analizzaremo il file descrittore di un servizio concretamente realizzato.
Una volta creato il file .wsdd, che potremmo ad esempio chiamare deploy.wsdd,dobbiamo per prima cosa copiare il package contenente le classi che realizzano il web service nella directory

"$CATALINA_HOME/webapps/axis/WEB-INF/classes" e,successivamente,da tale posizione eseguire l'operazione di deployment utilizzando un apposito tool di Axis da linea di comando(vedi paragrafo 2.7.2):

```
java org.apache.axis.client.AdminClient PackagePath/deploy.wsdd
```

2.6 Gestione delle sessioni

Axis fornisce le seguenti modalità per gestire le sessioni durante la comunicazione tra il sevizio e i client che lo invocano:
1) HTTP Cookie
2) SOAP headers

Indipendentemente dal meccanismo scelto bisogna specificare nel file .wsdd di deploy che servizio gestirà le sessioni in modalità session.
Tale informazione va inserita nel parametro scope:

```
<service name="service_name">
<parameter name="scope"
value = "[application | request | session]" />
</service>
```

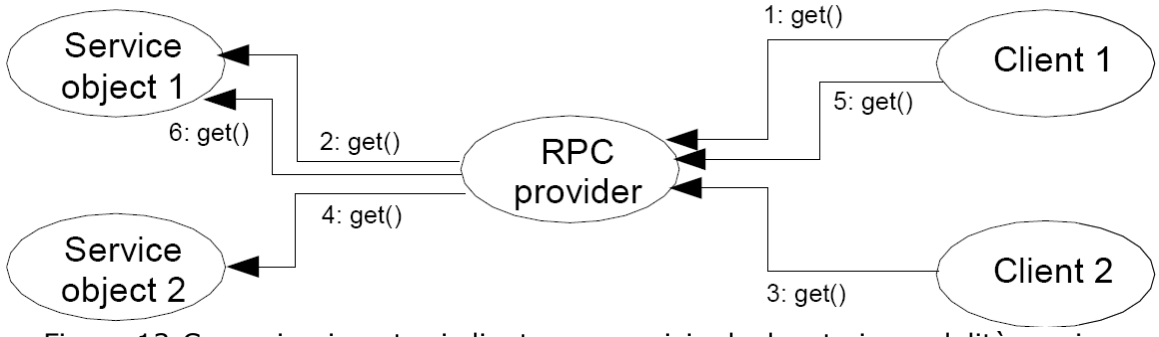


Figura 13:Comunicazione tra i client e un servizio deployato in modalità session

Sono disponibili i seguenti scope:

- request: i servizi vengono istanziati ad ogni richiesta di una istanza di servizio.. Se un client invoca il servizio n volte vengono create n istanze del servizio. Se si utilizza questo value nel paramentro scope il servizio implementato sarà di tipo stateless.
- session: i servizi vengono istanziati una sola volta per utente. Questo significa che possono essere servizi statefull: possono mantenere alcune informazioni nell'istanza perchè per ogni utente verranno utilizzate istanze diverse,ma per lo stesso utente verrà utilizzata sempre la stessa istanza.
- application: i servizi vengono istanziati una sola volta per ogni applicazione all'interno della Java Virtual Machine (in un cluster di N nodi, i servizi saranno istanziati N volte). Anche questi servizi possono essere statefull,ma non devono mantenere informazioni relative agli utenti poichè di ciascun servizio ne esiste solo un'istanza condivisa da tutti(sono equivalenti a un Singleton). Tutti i metodi esposti dal servizio devono essere thread-safe:il servizio dovrà gestire più richieste in parallelo. Ogni stato del servizio sarà condiviso da tutte le invocazioni.

HTTP Cookie

In questa modalità la gestione delle sessioni per mantenere lo stato della conversazione è affidata al protocollo utilizzato:nel caso di HTTP vengono utilizzati i cookie.
Quando un client contatta per la prima volta il servizio questo associa gli associa un identificativo univoco,un cookie,che il client includerà nei successivi messaggi richiesta. Se un messaggio di richiesta non contiene un cookie allora o il client non vuole mantenere lo stato della conversazione oppure è la prima volta che contatta il servizio. In entrambi i casi il servizio risponderà inserendo nel messaggio di risposta un nuovo cookie:è compito del client

inserire nei successivi messaggi di richiesta il cookie ricevuto dal servizio. Vengono quindi generate una nuova istanza del servizio e un nuovo identificativo per ogni messaggio di richiesta che non contiene un cookie;per ogni messaggio contenente un cookie il servizio riconosce a quale client è associato e non viene creata una nuova istanza del servzio. Un esempio dell'utilizzo dei cookie è il seguente:

1.Il client invoca un metodo del servizio;
2.Il servizio genera un cookie,lo associa al client e lo include nel messaggio di risposta;

Response 1
```
Cookie: JSESSIONID=24562F7A98121217AF4B88BA6B0285F0
```

3.Il client invoca un altro metodo del servizio includendo nel messaggio il cookie che gli è stato assegnato;

```
Response 1
Cookie: JSESSIONID=24562F7A98121217AF4B88BA6B0285F0
```

4.Viene attivata l'istanza del servizio relativa al client associato al cookie ricevuto:

- non viene generata una nuova istanza del servizio,
- non viene generato un nuovo cookie,
- il messaggio di risposta non contiene il cookie

Le sessioni sono mantenute dal servlet framework.
Se si utilizza questa modalità di gestione delle sessioni bisogna utilizzare setMaintainSession(true) client side.

SOAP Headers

Per utilzzare questo approccio bisogna effettuare il deploy dell'handler SimpleSessionHandler[46]:utilizza il SOAP header per gestire le sessioni.

Server-side:
Aggiungere il SimpleSessionHandler sia al request che al response flow nel file .wsdd di deploy del servizio:

```
 <deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

<handler name="session"
type="java:org.apache.axis.handlers.SimpleSessionHandler"/>
<service name="Sessions" provider="java:RPC" style="wrapped">
<namespace>my_namespace</namespace>
<requestFlow>
<handler type="session"/>
</requestFlow>
<responseFlow>
<handler type="session"/>
</responseFlow>
<parameter name="className" value="my_stateful_class"/>
<parameter name="allowedMethods" value="*"/>
</service>

</deployment>
```

- Ad ogni request è associato un session ID header. Se presente si aggiorna il SimpleSession altrimenti viene creata una nuova sessione. In entrambi i casi viene inserita la sessione nel MessageContext e il relativo ID nel SESSION_ID property.
- La response viene costruita inserindo nell'header SOAP l'ID della sessione.

Client-side:

- Scrivere un file client-config.wsdd nel quale inserire il SimpleSessionHandler sia nel request che nel response flow:

```
<globalConfiguration>
 <requestFlow>
  <handler type="session"/>
 </requestFlow>
 <responseFlow>
  <handler type="session"/>
 </responseFlow>
</globalConfiguration>
<parameter name="scope" value="session"/>
<handler name="session"
type="java:org.apache.axis.handlers.SimpleSessionHandler"/>
```

- bisogna poi inserire tale file nella directory dell'applicazione client;
- `org.apache.axis.utils.Admin client client-config.wsdd`
- Configurare setManageSession a `true`
- Se si utilizza lo stub modificarlo in questo modo:

```
public class HelloSoapBindingStub extends
org.apache.axis.client.Stub implements hello.HelloWorld {
private SimpleSessionHandler sessionHandler = null;
public HelloSoapBindingStub(javax.xml.rpc.Service service) {
sessionHandler = new SimpleSessionHandler();
...
}
protected org.apache.axis.client.Call createCall() {
...
_call.setClientHandlers(sessionHandler, sessionHandler);
return _call;
...
}
```

- Si verifica se nel messaggio di risposta è contenuto un session ID headers. Se presente viene inserito nelle opzioni dell'AxisClient;questo perché ogni oggetto Call è associato ad un singolo AxisClient.
- Quando viene generata una request si verifica se un ID option è presente nell'AxisClient associato con il MessageContext:in qesto caso si inserisce il session ID header con il relativo ID.

Il SimpleSessions viene scandito periodicamente ogni *readPeriodicity*:ogni volta che viene invocato l'handler si verifica se questo readPeriodicity è scaduto;in questo caso si scorre la lista delle sessioni attive e vengono rimosse quelle che non sono state utilizzate per un tempo maggiore di un *timeout*.

I messaggi SOAP avranno un aspetto di questo tipo:

```
<soapenv:Header>
<ns1:sessionID soapenv:actor="" soapenv:mustUnderstand="0" xsi:type="xsd:long"
xmlns:ns1="http://xml.apache.org/axis/session">
-1919645576528915916
</ns1:sessionID>
</soapenv:Header>
```

2.7 Strumenti per il monitoraggio delle comunicazioni

Axis mette a disposizione dello sviluppatore due utili strumenti per il monitoraggio di connessioni e comunicazioni, quali TCPMonitor e SOAPMonitor.

2.7.1 TcpMonitor

TCPMonitor è utilizzato per monitorare il flusso di dati su una connessione TCP. Viene posizionato fra un client ed un server, dove acquisisce i dati, inviati in una connessione stabilita con esso dal client, li visualizza nella sua interfaccia grafica ed infine li inoltra al server. Allo stesso modo, i dati inviati come risposta dal server verranno visualizzati nell'interfaccia di TCPMonitor prima di essere inoltrati al client. Per utilizzare TCPMonitor è necessario digitare da linea di comando la seguente istruzione:

```
java org.apache.axis.utils.tcpmon [listenPort targetHost targetPort]
```

Appare quindi la seguente interfaccia per l'inserimento dei parametri:

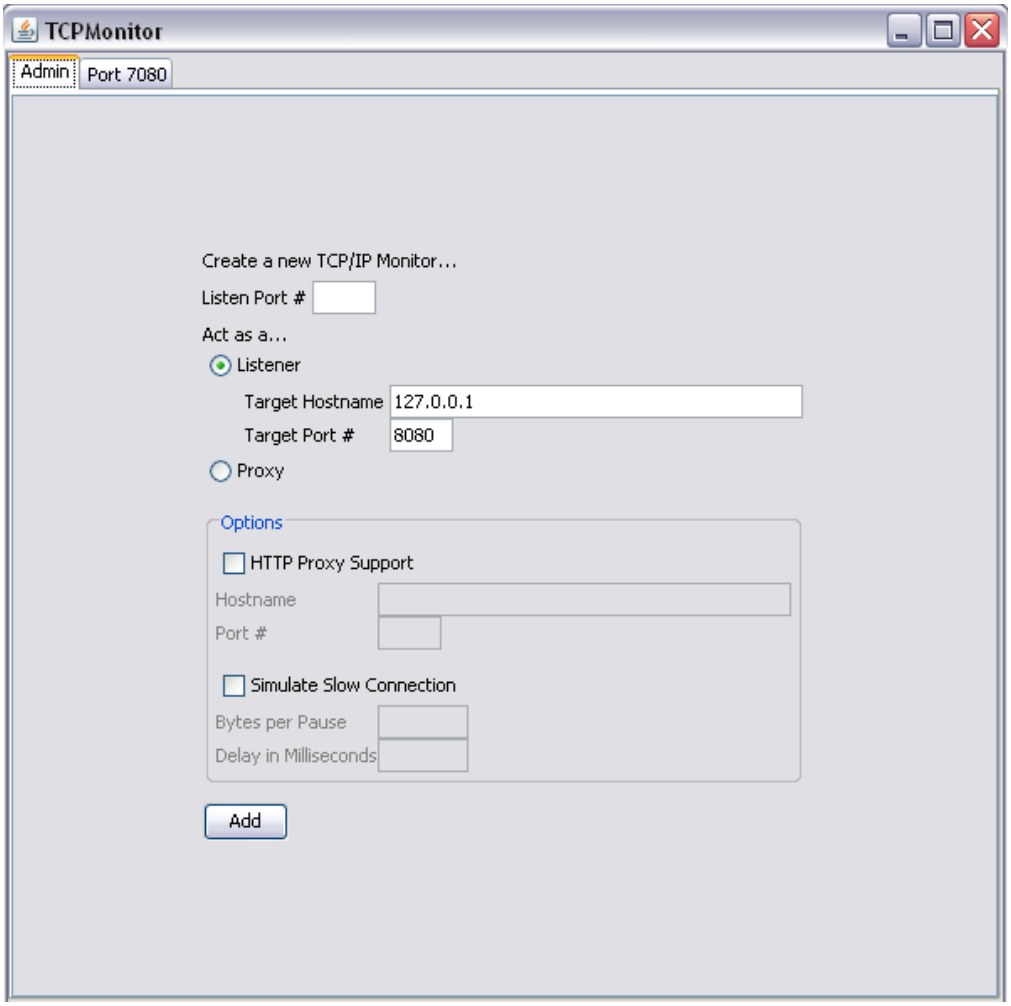


Figura 14:Intercaccia di TcpMonitor

Viene caricata l'interfaccia grafica di TCPMonitor, visualizzando la pagina principale (figura precedente) quale può essere creato un nuovo TCP/IP Monitor, specificando alcuni dati, come ad esempio la porta sulla quale attendere connessioni in entrata (Listen Port) e l'Hostname (Target Hostname) e la porta (Target Port) sui quali inoltrare tali connessioni. La crazione di un nuovo TCP/IP Monitor farà apparire una nuova finestra nell'interfaccia grafica.
In essa, ogni volta che verrà effettuata una connessione alla "Listen Port", saranno visualizzati i messaggi di richiesta verso il server e quelli di risposta da esso, nei riquadri ad essi relativi.

Nella parte alta del pannello sono riportate tutte le connessioni effettuate, che possono essere selezionate per visualizzare i relativi messaggi di richiesta e risposta.

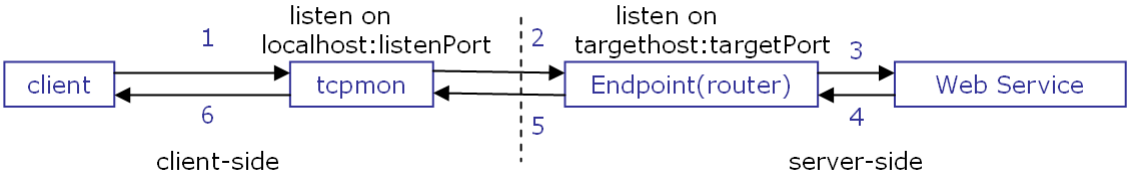


Figura 15:Funzionamento di TcpMonitor

Utilizzando il tool come Listener si devono impostare:

- Listen Port #: numero di porta su cui TCPMon si pone in ascolto per le richieste
- Target Hostname: host su cui inoltrare le richieste
- Target Port #: il numero di porta utilizzato dall'host target

2.7.2 SOAPMonitor

SOAPMonitor è uno strumento molto utile per poichè consente di vedere i messaggi SOAP utilizzati per invocare i web service. Per poter utilizzare questa utility,al fine di monitorare i messaggi SOAP ricevuti e restituiti da un web service, bisogna prima seguire alcuni passi di preparazione.
Per prima cosa è necessario compilare l'applet che implementa SOAP Monitor eseguendo da linea di comando, dalla directory "webapps/axis" di Tomcat, la seguente istruzione:

```
javac -classpath WEB-INF/lib/axis.jar SOAPMonitorApplet.java
```

OSS.: se axis.jar è nel classpath non è necessario specificare l'opzione -classpath ma è sufficiente eseguire:

```
javac SOAPMonitorApplet.java
```

Dopo aver compilato l'applet, dobbiamo effettuare il deployment creando un file di deployment per SOAPMonitor come il seguente:

deploy-SOAPMonitor.wsdd

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<handler name="soapmonitor"
type="java:org.apache.axis.handlers.SOAPMonitorHandler">
<parameter name="wsdlURL"
value="/axis/SOAPMonitorService-impl.wsdl"/>
<parameter name="namespace"
value="http://tempuri.org/wsdl/2001/12/SOAPMonitorService-impl.wsdl"/>
<parameter name="serviceName" value="SOAPMonitorService"/>
<parameter name="portName" value="Demo"/>
</handler>
<service name="SOAPMonitorService" provider="java:RPC">
<parameter name="allowedMethods" value="publishMessage"/>
<parameter name="className"
value="org.apache.axis.monitor.SOAPMonitorService"/>
<parameter name="scope" value="Application"/>
</service>
</deployment>
```

chiamato ad esempio deploy-SOAPMonitor.wsdd, ed eseguendo il comando:

```
java org.apache.axis.client.AdminClient deploy-monitor.wsdd
```

Una cosa importante da notare è il valore `Application` passato al paramentro `scope` nel file .wsdd:in questo modo nel sistema sarà presente un'unica istanza del servizio SoapMonitor per ogni invocazione;il servizio rappresenta quindi un Singleton.

Dopo aver effettuato tali operazioni,SOAPMonitor è abilitato e viene inserito nella lista dei servizi disponibili alla pagina `http://localhost:8080/axis/servlet/AxisServlet`, raggiungibile dalla prima pagina di Axis tramite il link `View`.

A questo punto bisogna specificare i servizi per i quali vogliamo monitorare i messaggi di richiesta e di risposta. Per fare questo si deve effettuare modificare il file di deployment di tali web service, inserendo,immediatamente dopo al tag di apertura dell'elemento <service>, le definizioni dei due elementi requestFlow e responseFlow, come riportato nel seguente codice:

```
...
<service name="BancaDiRomaWS" provider="java:RPC">
<requestFlow>
<handler type="soapmonitor"/>
</requestFlow>
<responseFlow>
<handler type="soapmonitor"/>
</responseFlow>
...
```

Con questi due elementi viene specificato soapmonitor come gestore del flusso dei messaggi di richiesta e del flusso dei messaggi di risposta. Affinchè queste modifiche abbiano effetto dobbiamo effettuare il deployment del servizio ma non prima di averne eseguito l'undeployment nel caso in cui tale servizio fosse già attivo su Axis. A questo punto è possibile eseguire il servizio SOAPMonitor, caricando da browser l'URL `http://localhost:8080/axis/SOAPMonitor`, ed ogni messaggio ricevuto ed inviato dai web service che vengono monitorati apparirà nei relativi box dell'interfaccia di SOAPMonitor:

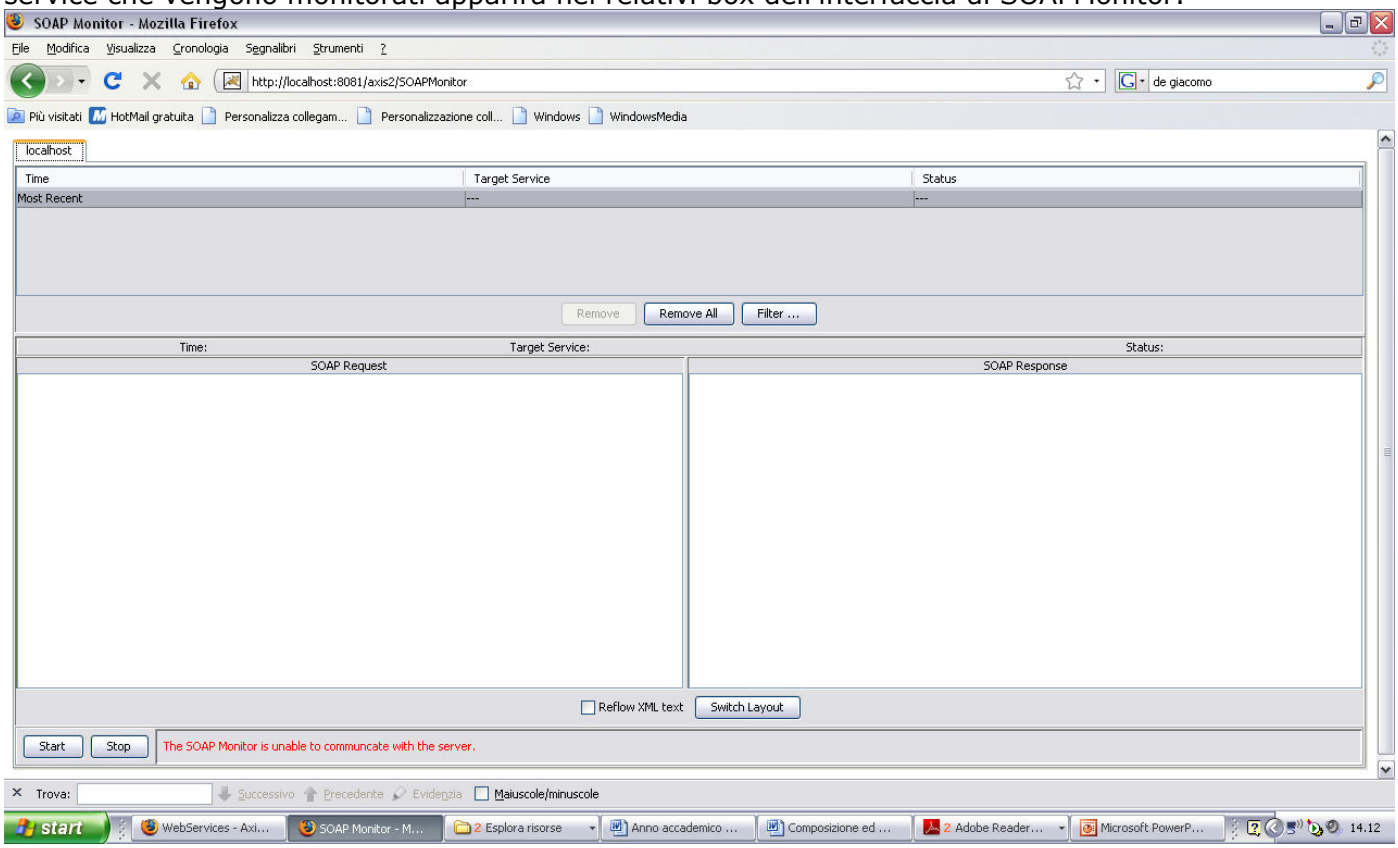


Figura 16:Intercaccia di SOAPMonitor

2.8 Esempio: BancadiRomaWS

Per comprendere meglio quanto discusso nei paragrafi precedenti seguiamo ora passo dopo passo le fasi di progettazione,sviluppo ed implementazione di un servizio,dalla sua interfaccia,al deployment,fino ad un possibile client che ne invochi i metodi esposti.

2.8.1 Scenario

La Banca di Roma ha deciso di esporre come Web Services alcuni servizi che offre alla sua clientela,come ad esempio il calcolo del tasso d'interesse associato ad un determinato prestito e la possibilità in un momento successivo da parte del cliente di richiedere il prestito stesso. BancaDiRomaWS è quindi un Web Services progettato appositamente per la Banca di Roma e i metodi,che corrispondono ai servizi che la banca intende offrire,sono:

- getLoanQuote() – per il calcolo del tasso d'interesse;
- getLoan() – per richiedere un prestito.

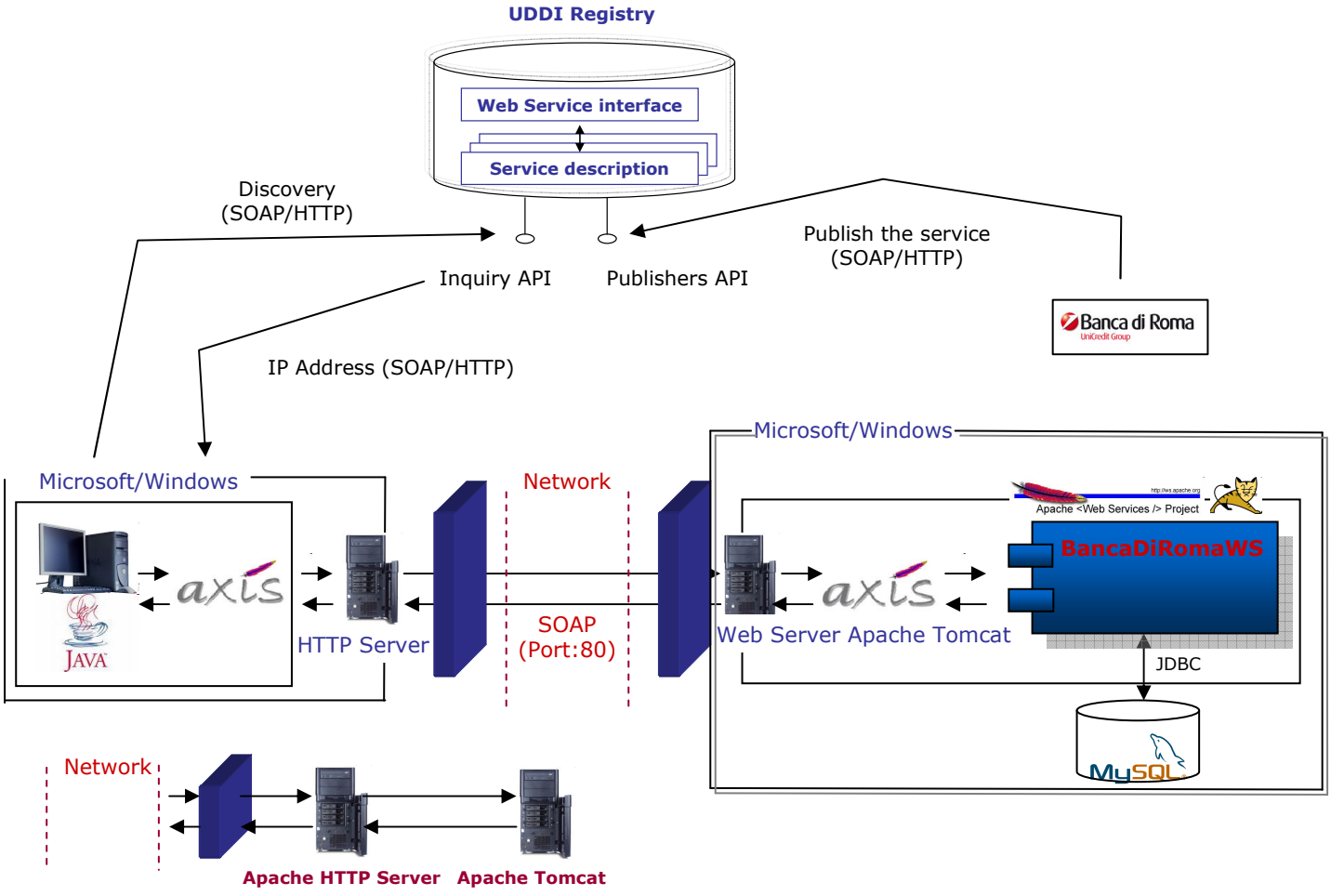


Figura 17:Schema generale del servizio

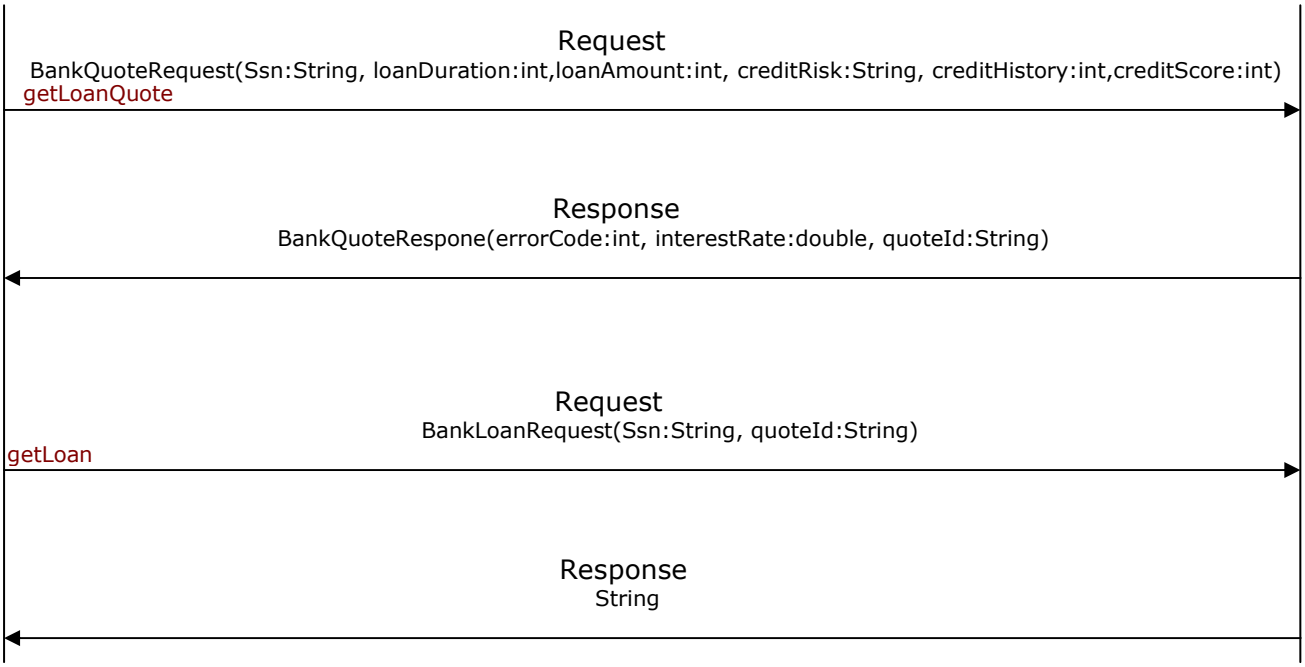


Figura 18:Scenario stile UML Sequence Diagram del servizio

2.8.2 Server side

1. Costruire l'interfaccia del servizio

BancaDiRomaWS.java

```
package banca_di__roma;
import banca_di_roma.To.*;

public interface BancaDiRomaWS {

    public BankQuoteResponse getLoanQuote (BankQuoteRequest in0);

public String getLoan(BankLoanRequest in0);
}
```

Dall'interfaccia è possibile osservare che i ComplextType del servizio,che rappresentano i value object di input e di output dei metodi esposti sono:

- BankQuoteRequest
- BankQuoteResponse
- BankLoanRequest

Axis gestisce i Web services che utilizzano tipi semplici mappando questi ultimi in maniera automatica negli appositi tipi SOAP/WSDL, in base alle definizioni di schema XML, XSD (Xml, Schema, Definition). A questo punto occorre soffermarci un momento sul fatto che qualsiasi informazione viaggi sul canale SOAP sia in realtà un documento XML. In questo formato, dunque, deve essere convertito tutto ciò che stia per percorrere il cammino client/server e viceversa. Per quanto concerne gli oggetti necessari come parametri di una invocazione RPC e il corrispettivo valore di ritorno, essi devono appunto essere rappresentati all'interno di una struttura XML. Axis esegue questa operazione,ovvero è in grado di:

- serializzare un oggetto Java in un blocco XML contenente le medesime informazioni,
- deserializzare un blocco XML rappresentante un dato nel suo omologo oggetto Java.

E' facile intuire, dunque, con quale ordine vengano eseguite queste operazioni.
La serializzazione viene effettuata sugli oggetti che il client invia al service engine come parametri richiesti dal metodo. Il Web service engine deserializza tali informazioni XML istanziando gli appositi oggetti Java e valorizzandone i campi. A seguito dell'esecuzione della logica Web service, il valore da restituire subisce anch'esso una serializzazione prima di essere spedito al client sul "canale" SOAP. Infine Il client deserializza il blocco XML restituito rendendo possibile un utilizzo locale dell'oggetto associato. Questa procedura di [de]serializzazione viene eseguita per tutti i tipi di dato, da quelli semplici che Axis gestisce in maniera trasparente, a quelli costituiti da aggregati di tipi semplici, che Axis gestisce in base ad alcune indicazioni che devono essere fornite durante la fase di deployment.

La definizione dei ComplexType permette di poter serializzare/deserializzare il risultato fornito dal server o la richiesta inviata dal client, in xml, in modo tale da poter essere letto o scritto all'occorrenza. Più semplicemente, quando un web service invia un risultato, se questi è di un qualche tipo primitivo, viene serializzato in un tag xml che lo contiene. Se il risultato è di tipo più complesso dovrà anch'esso essere serializzato in una qualche forma tale per cui sarà poi possibile la de-serializzazione client-side.

Il problema è che non si possono inviare oggetti per i quali non esista un `AXIS Serializer` registrato. AXIS fornisce il Bean Serializer per la serializzazione dei Java Bean. Si devono quindi costruire e registrare altri Serializer per poter trasmettere oggetti arbitrari.

Si delineano quindi due possibili strategie di soluzione:

a) si descrivono i dati in XML Schema e si implementa per ognuno di essi uno specifico serializer e deserializer;
b) si rendono le classi Java che costituiscono tali tipi, qualora non lo fossero, dei JAX-RPC Value Type e si ricorre quindi alla generazione automatica.

La scelta dipende chiaramente dal contesto e possono essere prese in considerazione anche soluzioni ibride.
L'opzione a) necessita lo sviluppo di serializzatori e deserializzatori che implementino le interfacce `org.apache.axis.encoding.Serializer` e `org.apache.axis.encoding.Deserializer`, comportando quindi uno sforzo di rilievo che implica la conoscenza della logica di funzionamento dei serializzatori e deserializzatori di Axis, nonché il trattamento di aspetti di basso livello riguardanti rispettivamente costruzione e parsing di messaggi XML. Va comunque rilevato che Axis fornisce la classe implementativa `org.apache.axis.encoding.SerializerImpl` dalla quale si può ereditare sgravandosi buona parte degli oneri di sviluppo.

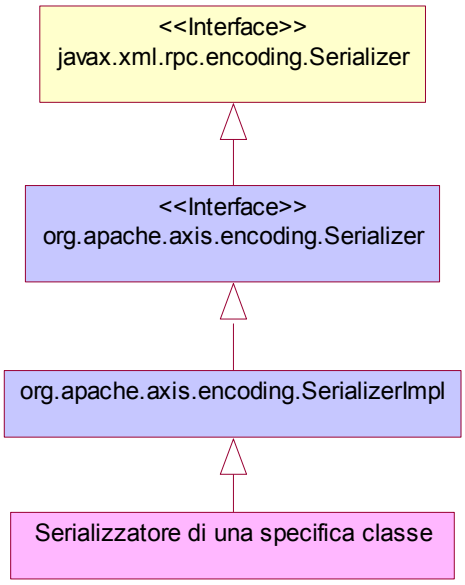


Figura 19:Gerarchia di classi e interfacce per i serializzatori di Axis.

La gerarchia di classi e interfacce è riportata in figura dove si può notare che l'interfaccia del serializzatore di Apache Axis estende un'interfaccia definita dalle specifiche JAX-RPC. Tuttavia quest'ultima è decisamente minimale, per cui l'interfaccia da implementare può dirsi interamente specifica per Axis. Quindi deve essere considerato che il futuro utilizzo di altre piattaforme per lo sviluppo di web services diverse da Apache Axis, anche se basate sulle stesse specifiche JAX-RPC,implica molto probabilmente l'implementazione di nuovi serializzatori e deserializzatori,costituendo un costo aggiuntivo per la transizione al nuovo sistema.
La soluzione b) appare quindi indubbiamente preferibile,anche se,in contesti inerenti componenti legacy, essa deve confrontarsi con il codice presente. Se il codice è già stato strutturato secondo le specifiche JavaBean, o JAX-RPC Value Type, ci si trova nella situazione più favorevole e non è richiesto alcuno sforzo; viceversa si deve provvedere a rendere il codice compatibile con dette specifiche. A questo scopo si possono rendere necessarie una o più delle seguenti azioni:

1. realizzare, per ciascuna classe da esporre, una classe *BeanInfo* che descriva quali sono i campi da esporre;
2. modificare i codici sorgenti;
3. creare delle nuove classi di interfaccia che utilizzino quelle originali e provvedano alle dovute conversioni.

Il punto numero 1) rappresenta la soluzione da applicare in tutti quei casi in cui la classe da esporre soddisfi le specifiche JavaBean ma contenga, per esempio, dei campi che non devono essere esportati; oppure sono presenti metodi del tipo getXxx() e/o setXxx() che però non fanno riferimento ad alcun campo xxx della classe. La semplice scrittura di una classe BeanInfo che descriva una classe siffatta consente di superare questo genere di difformità. Da notare che molti ambienti di sviluppo forniscono dei tool automatici per generare le classi BeanInfo.
Questo tipo di soluzione può essere ad esempio adottato per tradurre una classe che è strutturata come JavaBean, ma possiede un metodo che provoca la generazione di un campo nel WSDL, nonostante tale campo non sia effettivamente presente nel codice della classe.
Per tutte le altri classi invece, risultano praticabili solo le soluzioni 2) e 3). In generale la soluzione 2) non è sempre possibile poiché si potrebbe non disporre affatto dei codici sorgenti. Nella maggior parti dei casi si trattava di una soluzione praticabile e quindi preferibile, dal momento che è sicuramente più efficiente. Infatti l'alternativa consiste nel creare un'insieme di classi che ricalchino quelle originali e che costituiscano dei JAX-RPC Value Type, in modo da potere essere automaticamente convertite in XML e viceversa dai BeanSerializer e BeanDeserializer. Questa soluzione implica quindi che avvenga una seconda conversione fra questo nuovo insieme di classi e quelle originali, con evidente consumo di risorse computazionali. Si noti inoltre che nello sviluppo del nuovo insieme di classi l'utilizzo dei meccanismi di ereditarietà non è sempre una soluzione percorribile. Lo è nei casi in cui si tratta per esempio di aggiungere un metodo getXxx() o setXxx() per consentire l'accesso in lettura/scrittura ad una variabile *protected* delle superclasse. Tuttavia questo non è possibile se tali variabili sono *private*, oppure se la superclasse non è dotata di costruttore di default. Infatti, conseguentemente, la classe che da essa eredita, non può avere a sua volta un costruttore di default, come invece è richiesto affinché il BeanDeserializer possa ricostruire l'oggetto a partire dallo stream XML che ne costituisce la sua serializzazione

In questo contesto le classi che rappresentano i ComplexType sono state implementate secondo la specifica JavaBean per poterle serializzare automaticamente (metodi get/set per tutti i parametri).
Ogni ComplexType viene visto come un oggetto di business del servizio e la classe java che lo rappresenta ha le seguenti caratteristiche:

- Deve essere serializzabile per permette il trasporto su http attraverso i messaggi SOAP;
- Contiene una lista di attributi;
- Per ogni attributo i metodi set e get.

BankQuoteRequest.java

```
package banca_di_roma.To;
import java.io.*;
public class BankQuoteRequest  implements Serializable {
    private int creditHistory; //da quanto tempo si percepisce un creditScore
    private String creditRisk; //rischio:High o Low associato ad un prestito
    private int creditScore;  //guadagno netto mensile
    private int loanAmount; //ammontare del prestito
    private int loanDuration; //durata del prestito
    private String ssn;
    public BankQuoteRequest(){}

    public int getCreditHistory() { return creditHistory; }
    public void setCreditHistory(int creditHistory) {this.creditHistory =
creditHistory;}
    public String getCreditRisk() {return creditRisk;}
    public void setCreditRisk(String creditRisk) {this.creditRisk = creditRisk;
}
    public int getCreditScore() {return creditScore; }
    public void setCreditScore(int creditScore) {this.creditScore = creditScore;
}
    public int getLoanAmount() {return loanAmount; }
    public void setLoanAmount(int loanAmount) {this.loanAmount = loanAmount; }
    public int getLoanDuration() {return loanDuration; }
    public void setLoanDuration(int loanDuration) {this.loanDuration =
loanDuration; }
    public String getSsn() { return ssn;}
    public void setSsn(String ssn) {this.ssn = ssn; }
   }
```

BankQuoteResponse.java

```
package banca_di_roma.To;
public class BankQuoteResponse  implements java.io.Serializable {
    private int errorCode;  // :0/1 per verificare se il prestito può essere
concesso
    private double interestRate;  //tasso d'interesse
    private java.lang.String quoteId; //identificativo di un prestito
    public BankQuoteResponse() {}
    public int getErrorCode() {return errorCode;}
    public void setErrorCode(int errorCode) {this.errorCode = errorCode;}
    public double getInterestRate() {return interestRate;}
    public void setInterestRate(double interestRate) {this.interestRate =
interestRate;
    }
    public java.lang.String getQuoteId() {return quoteId;}
    public void setQuoteId(java.lang.String quoteId) {this.quoteId = quoteId;}
    }
```

BankLoanRequest.java

```
package banca_di_roma.To;
import java.io.Serializable;
public class BankLoanRequest implements Serializable {
    private   String quoteId; //identificativo inviato dalla banca per una
precedente richiesta di prestito
    private  String ssn;//Security Social Number:identificativo di un cliente
    public BankLoanRequest() {}

    public String getQuoteId() {return quoteId;}
    public void setQuoteId(String quoteId) {this.quoteId = quoteId;}
    public String getSsn() {return ssn;}
```

```
    public void setSsn(String ssn) {this.ssn = ssn;
}
```

Si osservi che le tre classi devono implementare l'interfaccia `java.io.Serializable` (senza metodi). Questa infatti è l'unica condizione che occorre rispettare per fare in modo che gli oggetti vengano serializzati,cioè rappresentati attraverso una sequenza di byte.

A questo punto non rimane che compilare le classi appena descritte:

```
C:\> cd banca_di_roma
C:\banca_di_roma>javac To/*.java  //Compilo le classi che rappresentano i
CompletType
C:\banca_di_roma>cd..
C:\> javac banca_di_roma/*.java //Compilo l'intrfaccia del servizio
```

2. Creare il file WSDL a partire dall'interfaccia appena creata.

Creare un documento WSDL a partire da una classe java equivale ad eseguire una mappatura Java su WSDL. Le regole di mappatura Java su WSDL vengono utilizzate dal comando `Java2WSDL` per l'elaborazione ascendente. Nell'elaborazione ascendente si utilizza un'implementazione del servizio Java esistente(Interfaccia del servizio:BancaDiRomaWS.java) per creare un file WSDL che definisce il servizio Web.

La tabella seguente mostra la mappatura da una struttura Java su una struttura WSDL e XML correlata.

Struttura Java	**Struttura WSDL e XML**
SEI (Service endpoint interface)	wsdl:portType
Metodo	wsdl:operation
Parametri	wsdl:input, wsdl:message, wsdl:part
Return	wsdl:output, wsdl:message, wsdl:part
Tipi primitivi	Tipi semplici xsd e soapenc
Bean Java	xsd:complexType
Proprietà del bean Java	xsd:elements nidificati di xsd:complexType

```
Java org.apache.axis.wsdl.Java2WSDL –o BancaDiRomaWS.wsdl
–l"http://localhost:8081/axis/services/BancaDiRomaWS"
banca_di_roma.BancaDiRomaWS
--style RPC
--use ENCODED
```

Dove:

- -o nome del documento WSDL da ottenere in output;
- -l url del servizio;
- nome della classe che contiene l'interfaccia del servizio;
- `--style RPC`;
- `--use ENCODED`

`--style RPC`: i parametri, o un valore restituito, vengono posizionati automaticamente in un elemento il cui padre è l'elemento `Body` del messaggio SOAP. Parametri e valori restituiti appaiono automaticamente senza qualifica dello spazio dei nomi. Questo schema SOAP è

descritto nella Sezione 7 della specifica SOAP 1.1. Per una richiesta SOAP, l'elemento sotto il `Body` SOAP viene chiamato come un elemento dell'`operazione` WSDL che corrisponde al metodo del servizio Web. Ogni elemento all'interno di quell'elemento rappresenta un parametro ed è chiamato come il rispettivo parametro. Per una risposta SOAP, il nome dell'elemento sotto il Body SOAP corrisponde al nome dell'operazione, con l'aggiunta di `Response`. Il nome dell'elemento di sotto, che rappresenta il valore restituito, corrisponde al nome dell'operazione ma con suffisso `Return`.

`--use ENCODED`: i dati vengono formattati secondo uno schema descritto nella sezione 5 della specifica SOAP 1.1[9]a codifica SOAP utilizza un sottoinsieme dello schema XML per l'associazione tra documenti XML e i dati che rappresentano e utilizza anche riferimenti per elementi che sono visualizzati più volte in un documento.

Vediamo ora come si presenta il documento appena generato e come questo rispetti quanto definito precedentemente nel paragrafo 1.4.2. Come ogni file .xml inizia con un'istruzione riguardante la versione di XML e la codifica di caratteri utilizzata. Successivamente inizia il vero e proprio documento WSDL che inizia sempre con l'elemento `definitions`. I namespace importati nel tag di apertura dipendono dalla tecnologia utilizzata per realizzare il servizio e in questo caso sono:

-SOAP(xmlns:apachesoap):protocollo;
-XML Schema(xmlns:xsd):definizione tipi;
-SOAP Encoding(xmlns:soapenc):definizione tipi;
-WSDL(xmlns:wsdl):namespace per gli elementi WSDL;
-Servizio(xmlns:tns1,targetNamespace,xmlns:impl,xmlns:intf):namespaces del servizio;

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://banca_di_roma"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://banca_di_roma"
  xmlns:intf="http://banca_di_roma"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns1="http://To.banca_di_roma"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

La codifica utilizzata è quindi quella standard SOAP/XML Schema:

```
-xmlns:apachesoap="http://xml.apache.org/xml-soap"
-xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
-xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

L'elemento `types` racchiude le definizioni dei tipi dei dati che sono coinvolti nello scambio dei messaggi. Come sistema standard per la tipizzazione,WSDL si basa su quello definito per gli schemi XML (XSD, XML Schema Definition) ed allo stesso tempo è possibile aggiungere anche altri tipi.

```
<wsdl:types>
```

Il contenuto dell'elemento `types` per questo servizio è un normale Schema XML:

```
 <schema targetNamespace="http://To.banca_di_roma"
xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
```

Lo schema definisce tre tipi globali che saranno utilizzati nel seguito del documento:

```
  <complexType name="BankQuoteRequest">
  <complexType name="BankQuoteResponse">
  <complexType name="BankLoanRequest">
 </schema>
</wsdl:types>
```

La sezione relativa ai messaggi definisce invece l'input e l'output del servizio. Ogni elemento `message` racchiude le informazioni, come parametri e loro tipi, relative ad uno specifico messaggio. Sono presenti due `message`: uno relativo al messaggio di input ed uno relativo al messaggio di output.

```
<wsdl:message name="getLoanResponse"> //Request message del metodo getLaon()
<wsdl:message name="getLoanRequest">//Response message del metodo getLaon()
<wsdl:message name="getLoanQuoteResponse">//Request message del metodo getLaonQuote()
<wsdl:message name="getLoanQuoteRequest">//Response message del metodo getLaonQuote()

<wsdl:portType name="BancaDiRomaWS">
<wsdl:binding name="BancaDiRomaWSSoapBinding" type="impl:BancaDiRomaWS">
<wsdl:service name="BancaDiRomaWSService">
</wsdl:definitions>
```

Dopo aver descritto gli elementi principali che compongono il documento analizziamo ora l'intero file WSDL.

Molti tipi Java vengono mappati direttamente su tipi XML standard. Ad esempio java.lang.String viene mappato su xsd:string o soapenc:string in base alla codifica utilizzata. Queste mappature sono descritte nella specifica JAX-RPC.
I tipi Java che non possono essere mappati direttamente sui tipi XML standard vengono generati nella sezione `wsdl:types`;i comlexType del servizio che corrispondono alle classi Java che rispettano ll modello bean Java, vengono mappate su un `xsd:complexType`.

```
<wsdl:types>
<schema targetNamespace="http://To.banca_di_roma"
  xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
```

Il complexType definiti nel servizio sono:
-BankQuoteRequest;
-BankQuoteResponse;
-BankLoanRequest.
Ognuno è rappresentato secondo lo Schema XML dall'elemento `complexType` al cui interno è inserito il nome del tipo complesso;all'interno di `compexType` è presente l'elemento `sequence` che contiene una sequenza di `element` che compongono il tipo di dato. Da notare che l'attributo nillable è presente solo in alcuni elementi:dipende da come sono stati definiti i tipi di dato degli elementi nelle classi java che definiscono i relativi complexType. L'attributo nillable sarà presente in tutti quegli elementi i cui tipi di dato sono stati definiti come oggetti appartenenti alle classi wrapper dei tipi semplici ed indica che un elemento può avere valore nullo:se il valore è nullo nell'istanza del messaggio SOAP viene inserito l'attributo `xsi:nil="true"`. Bisogna comunque dire che il WSDL generato automaticamente dal tool può essere considerato come un documento di defalut sul quale è possibile effettuare delle modifiche;va considerato quindi come un buon punto di partenza per creare il WSDL definitivo che rappresenterà l'interfaccia del servizio.

Sono quindi codificati con `SOAP Encoding` e con l'attributo nillable=true gli oggetti appartenenti alle classi wrapper:
-String--->soapenc:string;
-Integer--->soapenc:int;
-Short--->soapenc:short;
-Double--->soapenc:double;
-Character--->soapenc:char;
-Long--->soapenc:long;
-Byte--->soapenc:byte;
-Float--->soapenc:float;
-Boolean--->soapenc:boolean.

Mentre sono codificati con XML Schema Definition i tipi primitivi associati alle classi wrapper precedenti:

-int--->xsd:int;
-short--->xsd:short;
-double--->xsd:double;
-char--->xsd:char;
-long--->xsd:long;
-float--->xsd:float;
-boolean--->xsd:boolean;

Vediamo ora come vengono mappati i tre complexType all'interno del documento.

Il complexType BankQuoteRequest.java viene mappato su:

```
<complexType name="BankQuoteRequest">
<sequence>
<element name="creditHistory" type="xsd:int" />
```

L'elemento creditRisk è stato definito nella classe java Bean come un oggetto appartenente alla classe String e non come un tipo semplice;di conseguenza viene codificato con soapenc e viene inserito l'attributo nillable=true:

```
<element name="creditRisk" nillable="true" type="soapenc:string" />
<element name="creditScore" type="xsd:int" />
<element name="loanAmount" type="xsd:int" />
<element name="loanDuration" type="xsd:int" />
<element name="ssn" nillable="true" type="soapenc:string" />
  </sequence>
  </complexType>
```

Il complexType BankQuoteResponse.java viene mappato su:

```
<complexType name="BankQuoteResponse">
<sequence>
<element name="errorCode" type="xsd:int" />
<element name="interestRate" type="xsd:double" />
<element name="quoteId" nillable="true" type="soapenc:string" />
  </sequence>
  </complexType>
```

Il complexType BankLoanRequest.java viene mappato su:

```
<complexType name="BankLoanRequest">
<sequence>
<element name="quoteId" nillable="true" type="soapenc:string" />
<element name="ssn" nillable="true" type="soapenc:string" />
  </sequence>
  </complexType>
  </schema>
  </wsdl:types>
<wsdl:message name="getLoanResponse">
<wsdl:part name="getLoanReturn" type="soapenc:string" />
  </wsdl:message>
<wsdl:message name="getLoanRequest">
<wsdl:part name="in0" type="tns1:BankLoanRequest" />
  </wsdl:message>
<wsdl:message name="getLoanQuoteResponse">
<wsdl:part name="getLoanQuoteReturn" type="tns1:BankQuoteResponse" />
  </wsdl:message>
<wsdl:message name="getLoanQuoteRequest">
<wsdl:part name="in0" type="tns1:BankQuoteRequest" />
  </wsdl:message>
```

Nel portType è possibile vedere come le operazioni definite del WSDL e che corrispondono ai metodi del servizio getLoanQuote e getLoan sono del tipo Request-Response:il client spedisce

un messaggio di richiesta e riceve una risposta. Gli elementi `input` e `output` specificano il messaggio(definito in precedenza) associato rispettivamente all'operazione di request e response.

```
<wsdl:portType name="BancaDiRomaWS">
 <wsdl:operation name="getLoanQuote" parameterOrder="in0">
  <wsdl:input message="impl:getLoanQuoteRequest" name="getLoanQuoteRequest" />
  <wsdl:output message="impl:getLoanQuoteResponse" name="getLoanQuoteResponse" />
 </wsdl:operation>
<wsdl:operation name="getLoan" parameterOrder="in0">
  <wsdl:input message="impl:getLoanRequest" name="getLoanRequest" />
  <wsdl:output message="impl:getLoanResponse" name="getLoanResponse" />
 </wsdl:operation>
</wsdl:portType>
```

All'interno dell'elemento `binding` è contenuta prima di tutto la definizione dello stile del collegamento,che in questo caso è di tipo RPC, e la specificazione di HTTP come protocollo di trasporto, facendo uso di un elemento relativo a SOAP chiamato wsdlsoap.
Sono presenti poi tanti elementi `operation` quanti sono i metodi (operazioni) messi a disposizione.

```
-<wsdl:operation name="getLoanQuote">
<wsdl:operation name="getLoan">
```

All'interno di ogni elemento `operation` si trovano gli elementi `input` ed `output`:

```
-<wsdl:input name="getLoanQuoteRequest">
-<wsdl:output name="getLoanQuoteResponse">
-<wsdl:input name="getLoanRequest">
-<wsdl:output name="getLoanResponse">
```

Gli elementi di input/output sono gli stessi elementi input e/o ouptut del `portType` associato,senza specificare il messaggio. L'elemento `wsdlsoap:body`,nidificato negli elementi `input` e `output`,definisce la composizione del Body del messaggio SOAP. Viene indicato il namespace di appartenenza degli elementi usati(`namespace="http://banca_di_roma"`) e viene specificato che le parti del messaggio andranno codificate(`use="encoded"`) usando la codifica standard SOAP/XML Schema(`encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`).

```
<wsdl:binding name="BancaDiRomaWSSoapBinding" type="impl:BancaDiRomaWS">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="getLoanQuote">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getLoanQuoteRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://banca_di_roma" use="encoded" />
  </wsdl:input>
<wsdl:output name="getLoanQuoteResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://banca_di_roma" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getLoan">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getLoanRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://banca_di_roma" use="encoded" />
  </wsdl:input>
<wsdl:output name="getLoanResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://banca_di_roma" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
```

L'ultima sezione, identificata dall'elemento `service`,è relativa alla localizzazione del Web Service. Al suo interno si trova l'unico servizio creato(**`BancaDiRomaWSService`**); il servizio è definito da un'elemento `port` che riporta il collegamento utilizzato ed al suo interno ha un'ulteriore elemento `wsdlsoap:address` che indica l'indirizzo URL, chiamato anche endpoint, al quale può essere trovato il Web Service.

```
<wsdl:service name="BancaDiRomaWSService">
<wsdl:port binding="impl:BancaDiRomaWSSoapBinding" name="BancaDiRomaWS">
<wsdlsoap:address location="http://localhost:8081/axis/services/BancaDiRomaWS" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

3. Generare lo skeleton del servizio

A partire dal documento WSDL è ora possibile utilizzare un altro tool, WSD2Java, che permette di generare:

- le classi proxy da usare lato client;
- lo skeleton che fa da bridge fra il runtime di Axis e l'implementazione vera e propria del servizio.

Partendo dal documento WSDL precedente è possibile generare le classi da usare lato client nel modo seguente (nel classpath devono essere inclusi axis.jar, commons-logging.jar, jaxrpc.jar, wsdl4j.jar e il parser XML):

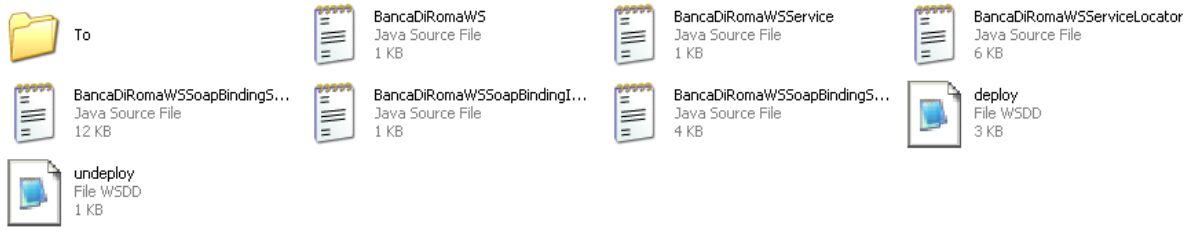
```
java org.apache.axis.wsdl.WSDL2Java --skeletonDeploy true BancaDiRomaWS.wsdl
```

La seguente tabella mostra la struttura Java su XML.

Struttura WSDL e XML	**Struttura Java**
xsd:complexType	Classe bean Java, classe eccezione Java o matrice Java
xsd:element/xsd:attribute nidificato	Proprietà del bean Java
xsd:simpleType (enumerazione)	Classe di enumerazione JAX-RPC
wsdl:message La firma del parametro del metodo è determinata, generalmente, da wsdl:message.	Firma del metodo SEI (Service endpoint interface)
wsdl:portType	SEI (Service endpoint interface)
wsdl:operation	Metodo SEI (Service endpoint interface)
wsdl:binding	Stub
wsdl:service	Interfaccia del servizio
wsdl:port	Metodi del programma di accesso alla porta nell'interfaccia del servizio

Per il documento precedente verranno generate quindi le seguenti classi:

- **`BancaDiRomaWS.java`**: Rappresenta l'interfaccia del servizio ed estende java.rmi.Remote; è ottenuta dal portType ed in terminologia Axis viene denominata SDI, Service Definition Interface.
- **`BancaDiRomaWSService.java`**: Questa interfaccia viene derivata dal servizio definito nel documento WSDL.
- **`BancaDiRomaWSServiceLocator.java`**: La classe locator implementa l'interfaccia del servizio e permette di ottenere un'istanza dello stub.
- **`BancaDiRomaWSSoapBindingStub.java`**: Questa classe implementa l'interfaccia SDI, BancaDiRomaWS in questo caso, e contiene il codice che converte le invocazioni di metodi utilizzando la API di Axis. In questo modo vengono nascosti allo sviluppatore tutti i dettagli della chiamata SOAP.
- **`BancaDiRomaWSSoapBindingImpl.java`**: Implementazione del Web Service. Questa classe è un template,i metodi generati sono vuoti;bisogna scrivere il codice che il servizio deve concretamente eseguire.
- **`BancaDiRomaWSSoapBindingSkeleton.java`**: Rappresenta lo Skeleton lato server che si pone fra l'engine Axis e l'implementazione del servizio.
- **`deploy.wsdd`**: Descrittore del deployment del Web Service;
- **`undeploy.wsdd`**: Descrittore del undeployment del Web Service;



Clausola WSDL **Classi Java generate**

Mappatura dei tipi XML definiti nella sezione wsdl:types:

Per i complexType: → Classe java:

`BankQuoteRequest`	`BankQuoteRequest.java`
`BankQuoteResponse`	`BankQuoteResponse.java`
`BankLoanRequest`	`BankLoanRequest.java`

Mappatura della struttura wsdl:portType

Per il portType: → Interfaccia java:

`BancaDiRomaWS`	`BancaDiRomaWS.java`

Mappatura di wsdl:binding

Per il binding: → Classe Stub,Skeleton e Template:

`BancaDiRomaWSSOAPBinding`	`BancaDiRomaWSSOAPBindingStub.java`
	`BancaDiRomaWSSOAPBindingSkeleton.java`
	`BancaDiRomaWSSOAPBindingImpl.java`

Mappatura di wsdl:service

Per il servizio: → Implementazione del servizio:

`BancaDiRomaWSService`	`BancaDiRomaWSService.java(Interfaccia)`
	`BancaDiRomaWSServiceLocator.java(Implementazione)`
	`deploy.wsdd e undeploy.wsdd`

All'interno dell'elemento `deployment`, che riporta i namespace a cui si fa riferimento, si trova l'elemento `service` che specifica le informazioni relative al servizio di cui vogliamo fare il deployment. Nel tag `service` sono presenti gli attributi `name,provider,style ed use` con i quali vengono specificati il nome del servizio e lo stile dell'applicazione, in questo esempio RPC. All'interno dell'elemento `service` sono presenti una serie di elementi `parameter` che contengono informazioni sul servizio e di `typeMapping` generati per il mapping dei complexType.

deploy.wsdd

```
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
```

L'attributo name all'interno del tag `service` rappresenta il nome che si vuole assegnare al servizio e che servirà ad identificarlo univocamente mentre l'attributo `provider java:RPC` indica ad Axis di pubblicare il servizio secondo un meccansmo RPC:

```
  <service name="BancaDiRomaWS" provider="java:RPC" style="rpc" use="encoded">
      Specific ail nome
      <parameter name="wsdlTargetNamespace" value="http://banca_di_roma"/>
      <parameter name="wsdlServiceElement" value="BancaDiRomaWSService"/>
      <parameter name="schemaUnqualified" value="http://To.banca_di_roma"/>
      <parameter name="wsdlServicePort" value="BancaDiRomaWS"/>
```

Specifica il nome della classe che implement ail servizio e il package in cui si trova:

```
      <parameter name="className"
value="banca_di_roma.BancaDiRomaWSSoapBindingSkeleton"/>
      <parameter name="wsdlPortType" value="BancaDiRomaWS"/>
      <parameter name="typeMappingVersion" value="1.2"/>
```

Specifica i metodi che possono essere esposti,in questo caso tutti I metodi della classe sono utilizzabili dall'esterno:

```
      <parameter name="allowedMethods" value="*"/>
```

Una cosa importante da notare è che questo generato automaticamente dal tool descritto in precedenza non contiene il seguente parametro:

```
      <parameter name="scope" value="Request|Session|Application"/>
```

che definisce il ciclo di vita della classe che implementa il servizio. Se non viene indicata l'opzione –d Request|Session|Application quando si esegue il tool WSDL2Java il file deploy.wsdd che viene generato non contiene il parametro precedente;questo equivale ad inserire manualmente il parametro nel file con il valore Request nell'attributo value. In questo modo quindi,se non si specifica l'opzione –d Request nel tool e se tale parametro viene inserito manualmente a posteriori,il servizio creato sarà di tipo Stateless e la relativa classe verrà istanziata, utilizzata e distrutta ad ogni richiesta.

Per ogni complexType del servizio è stato generato un `element typeMapping`;gli attributi evidenziati specificano le classi serielizer e deserializer di default fornite da Axis per la serializzazione dei JavaBean. Le classi che rappresentano i complexType sono state appositamente implementate rispettando lo standard dei Bean java per poter poi utilizzare il relativo Bean Serializer messo a disposizione da Axis. In questo modo il compito di gestire la serializzazione e deserializzazione dei complexType è completamente a carico di Axis,l'unica cosa che ho dovuto fare è attenermi alla specifica java Bean. Nel caso avessi implementato una classe che rappresenta un complexType senza rispettare la specifica Bean non avrei potuto utilizzare il Bean Serializer di Axis e avrei pertanto dovuto costruire sia la classe per la relativa serializzazione che quella relativa alla deserializzazione inserendole sempre nel typeMapping al posto delle classi di default di Axis.

```
      <typeMapping
        xmlns:ns="http://To.banca_di_roma"
        qname="ns:BankQuoteRequest"
        type="java:banca_di_roma.To.BankQuoteRequest"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="http://To.banca_di_roma"
        qname="ns:BankLoanRequest"
        type="java:banca_di_roma.To.BankLoanRequest"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

```
      />
      <typeMapping
        xmlns:ns="http://To.banca_di_roma"
        qname="ns:BankQuoteResponse"
        type="java:banca_di_roma.To.BankQuoteResponse"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
  </service>
</deployment>
```

4. Modificare la classe BancaDiRomaWSSoapBindingImpl.java

Questa classe è un template,i metodi generati sono vuoti e bisogna scrivere l'implementazione concreta del servizio.

```
package banca_di_roma;

public class BancaDiRomaWSSoapBindingImpl implements
banca_di_roma.BancaDiRomaWS{

public banca_di_roma.To.BankQuoteResponse
getLoanQuote(banca_di_roma.To.BankQuoteRequest in0) throws
java.rmi.RemoteException {return null;}

public java.lang.String getLoan(banca_di_roma.To.BankLoanRequest in0) throws
java.rmi.RemoteException {return null;}

}
```

Bisogna quindi sostituire i return null con il corretto codice applicativo.

```
package banca_di_roma;
import banca_di_roma.To.*;
import java.util.*;
import java.util.Calendar;
import java.sql.*;
import java.util.*;
import javax.sql.*;
import javax.naming.*;

public class BancaDiRomaWSSoapBindingImpl implements
banca_di_roma.BancaDiRomaWS{
Connection connection;
private boolean connectionFree = true;

private Statement stmtMT=null;

public double PRIME_RATE=2.0;
public int TIME=10;

private String bankName;
private double ratePremium;
private int maxLoanTerm;


public BancaDiRomaWSSoapBindingImpl ()throws Exception{
bankName = "Banca di Roma";
```

```
ratePremium = 2.0;
maxLoanTerm = 48;
  try{
//Datasource:gestisco la connessione al database
```

Recupero l'initialContext per effettuare il JNDI lookup: `Context` è un oggetto appartenente alla classe `javax.naming.Context` mentre `InitialContext` fa parte di `javax.naming.InitialContext`. `InitialContext` è il contesto di partenza per eseguire le operazioni di associazione dei nomi, le quali sono relative ad un contesto che implementa l'interfaccia Context, fornendo la base per la risoluzione dei nomi.

```
      InitialContext initialContext = new InitialContext ();
Ottengo il context
      Context envContext = (Context) initialContext.lookup ("java:comp/env");
Recupero il datasource
      DataSource dataSource = (DataSource) envContext.lookup ("jdbc/ardeadb");
Ottengo la connessione dal datasource
            this.connection = dataSource.getConnection ();
      }catch (Exception e){throw new Exception ("Couldn't open connection to
Public database: " + e.getMessage ());
        }
    }
//Gestione delle connessioni al database
protected synchronized Connection getConnection (){
        while (this.connectionFree == false){
            try{wait ();
                }catch (InterruptedException e){}
            }
        this.connectionFree = false;
        notify ();
        return this.connection;
}
    protected synchronized void releaseConnection (){
        while (this.connectionFree == true) {
            try{wait ();
             }catch (InterruptedException e){}
                    }
        this.connectionFree = true;
        notify ();
    }
public void close (){
        try{this.connection.close ();
               }catch (SQLException e) {System.out.println (e.getMessage ());}
            }


//Metodo getLoanQuote
    public banca_di_roma.To.BankQuoteResponse
getLoanQuote(banca_di_roma.To.BankQuoteRequest in0) throws
java.rmi.RemoteException {
    String stato="False";
    BankQuoteResponse reply = new BankQuoteResponse();
    if(in0.getLoanDuration()<=maxLoanTerm && in0.getCreditScore()>800 &&
(in0.getCreditRisk()).equals("Low") ){//prima conditione
       reply.setInterestRate(this.PRIME_RATE + ratePremium +
(in0.getLoanDuration()/12)/10 +
        (new Random()).nextInt(10)/10);
       reply.setErrorCode(new Integer(0));
       reply.setQuoteId(bankName + (new Random()).nextInt(100000) + 99999);
int timestamp=getTimestamp();
try{
this.getConnection();
```

```
Statement statement =
this.connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONC
UR_UPDATABLE);
statement.executeUpdate("INSERT INTO loantb(Ssn,Id,Timestamp,Rate,Stato)
VALUES('"+in0.getSsn()+"','"+reply.getQuoteId()+"','"+timestamp+"','"+reply.getI
nterestRate()+"','"+stato+"') ");

statement.close ();
      }catch (SQLException e){System.out.println("SQLException : " +
e.getMessage());System.exit(0);}
        this.releaseConnection ();
    }
    else if(in0.getLoanDuration()<=maxLoanTerm && in0.getCreditScore()>800 &&
(in0.getCreditRisk()).equals("High") ){//seconda condizione
       int decide = (int)(Math.random() * 1);
       if(decide==0){
       reply.setInterestRate(this.PRIME_RATE + ratePremium +
(in0.getLoanDuration()/12)/10 +
        (new Random()).nextInt(10)/10);
       reply.setErrorCode(new Integer(0));
       reply.setQuoteId(bankName + (new Random()).nextInt(100000) + 99999);
int timestamp=getTimestamp();
try{
this.getConnection ();
Statement statement =
this.connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONC
UR_UPDATABLE);
statement.executeUpdate("INSERT INTO loantb(Ssn,Id,Timestamp,Rate,Stato)
VALUES('"+in0.getSsn()+"','"+reply.getQuoteId()+"','"+timestamp+"','"+reply.getI
nterestRate()+"','"+stato+"') ");
statement.close ();
      }catch (SQLException e){System.out.println("SQLException : " +
e.getMessage());System.exit(0);}
        this.releaseConnection ();
    }
       else{
    reply.setInterestRate(0.0);
    reply.setErrorCode(new Integer(1));
    reply.setQuoteId(bankName + (new Random()).nextInt(100000) + 99999);
     }
    }
    else{//terza condizione
    reply.setInterestRate(0.0);
    reply.setErrorCode(new Integer(1));
    reply.setQuoteId(bankName + (new Random()).nextInt(100000) + 99999);
    }
      return reply;
    }

//Metodo getLoan
    public java.lang.String getLoan(banca_di_roma.To.BankLoanRequest in0) throws
java.rmi.RemoteException {
        int timestampLoan=getTimestamp();
        String result=null;
        try
        {
            this.getConnection ();

PreparedStatement preparedStatement = this.connection.prepareStatement ("SELECT
* FROM loantb WHERE Ssn='"+in0.getSsn()+"' AND Id='"+in0.getQuoteId()+"'");
ResultSet rsltMT = preparedStatement.executeQuery ();

         if(!rsltMT.next())
```

```
          result= "Ssn: "+in0.getSsn()+"Nessun prestito richiesto";//Nessuna
richiesta effettuata
          else{
          if((rsltMT.getString(5)).equals("True"))
          result= "Ssn: "+in0.getSsn()+"Prestito già concesso";//Prestito già
erogato

          else if((timestampLoan-rsltMT.getInt(3))>TIME)//La richiesta del
prestito è scaduta

          result="Ssn: "+in0.getSsn()+"Richiesta di prestito scaduta";//Richiesta
scaduta...sono passati più di 10 giorni

          else{
stmtMT=this.connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultS
et.CONCUR_UPDATABLE);
stmtMT.executeUpdate("UPDATE loantb SET Stato='True' WHERE Ssn='"
+in0.getSsn()+"' AND Id='"+in0.getQuoteId()+"'");
stmtMT.close();
result ="Prestito concesso\n"+"Ssn: "+in0.getSsn()+"\n"+"InterestRate:
"+rsltMT.getDouble(4);
               }
          }

preparedStatement.close();

        }catch (SQLException e){System.out.println("SQLException : " +
e.getMessage());System.exit(0);}

this.releaseConnection ();
return result;
    }

//Metodo per ottenere la data corrente utilizzato per verificare se la richiesta
di un determinato prestito è scaduta.

public int getTimestamp(){

Calendar c = Calendar.getInstance();
int gi=c.get(Calendar.DAY_OF_MONTH);
int me=c.get(Calendar.MONTH)+1;
int an=c.get(Calendar.YEAR);

String gg=null;
if(gi==1) gg="01"; else if(gi==2) gg="02"; else if(gi==3) gg="03"; else
if(gi==4) gg="04"; else if(gi==5) gg="05"; else if(gi==6) gg="06"; else
if(gi==7) gg="07"; else if(gi==8) gg="08"; else if(gi==9) gg="09"; else
gg=""+gi+"";

String gio=null; if(me==1) gio="01"; else if(me==2) gio="02"; else if(me==3)
gio="03"; else if(me==4) gio="04"; else if(me==5) gio="05"; else if(me==6)
gio="06"; else if(me==7) gio="07"; else if(me==8) gio="08"; else if(me==9)
gio="09"; else gio=""+me+"";

String da=""+an+""+""+gio+""+""+gg;
int dat=Integer.parseInt(da);
return dat;
  }
}
```

Entriamo ora nel dettaglio del codice appena esposto.

Gestione delle connessioni al database

Per garantire connessioni sicure al database mysql ho utilizzato i metodi sincronizzati getConnection e releaseConnection.

```
protected synchronized Connection getConnection (){
        while (this.connectionFree == false){//rimango in attesa che si liberi
una connessione
            try{wait ();
                }catch (InterruptedException e){}
            }
        this.connectionFree = false;//appena una connessione viene rilasciata
imposto la variabile a false
        notify ();//ed eseguo una notify()a tutti gli altri thread in attesa
        return this.connection;//infine restituisco la connessione
}

    protected synchronized void releaseConnection (){
        while (this.connectionFree == true) { //rimango in attesa fino a che la
connessioni diventi libera
            try{wait ();
             }catch (InterruptedException e){}
                    }
        this.connectionFree = true; //libero la connessione
        notify ();//eseguo una notify() a tutti gli altri thread in attesa di
una connessione
    }
public void close (){
        try{this.connection.close ();
               }catch (SQLException e) {System.out.println (e.getMessage ());}
            }
```

Metogo getLoanQuote

Questo metodo ritorna un ComplexType che contiene il tasso d'interesse associato ad un determinato prestito.

BankQuoteResponse

- errorCode: 0/1
- interestRate: tasso d'interesse
- quoteId: identificativo di ogni richiesta di prestito

I parametri necessari al calcolo di questo tasso sono,contenuti nell'Object Value BankQuouteResponse,sono:

- creditHistory,
- creditRisk,
- creditScore,
- loanAmount,
- loanDuration,
- ssn.

```
String stato="False";
//Creo l'oggetto complesso BankQuoteResponse che conterrà la risposta della
banca.
BankQuoteResponse reply = new BankQuoteResponse();
//La richiesta di un prestivo viene accettata solo se:
   • la durata è minore o uguale alla massima durata di un prestito che la
     banca è disposta ad accettare,
   • il creditScore del cliente che richiede il prestito è >800
   • e il rischio di concedere un prestito ad un certo cliente è basso
if(in0.getLoanDuration()<=maxLoanTerm && in0.getCreditScore()>800 &&
(in0.getCreditRisk()).equals("Low") ){//prima conditione
 reply.setInterestRate(this.PRIME_RATE + ratePremium +
(in0.getLoanDuration()/12)/10 +
```

```
        (new Random()).nextInt(10)/10);
       reply.setErrorCode(new Integer(0));
       reply.setQuoteId(bankName + (new Random()).nextInt(100000) + 99999);
//utilizzo il metodo getTimestamp() per prendere il momento preciso in cui la
richiesta del prestito è stata accettata dalla banca
int timestamp=getTimestamp();
//memorizzo nel database tutte le informazioni relative alla richiesta di
prestito appena approvata
try{
this.getConnection();
Statement statement =
this.connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONC
UR_UPDATABLE);
```

Le informazioni da memorizzare sono:

- Ssn:identificativo del cliente
- Id:identificativo della richiesta di prestito generato dalla banca
- Timestamp:un cliente ha un certo tempo,determinato dalla variabile TIME, per decidere se confermare il prestito
- Rate:tasso d'interesse calcolato
- Stato:FALSE per default,indica che la richiesta di prestito è stata accettata dalla banca ma che il cliente ancora non lo ha confermato

```
statement.executeUpdate("INSERT INTO loantb(Ssn,Id,Timestamp,Rate,Stato)
VALUES('"+in0.getSsn()+"','"+reply.getQuoteId()+"','"+timestamp+"','"+reply.getI
nterestRate()+"','"+stato+"') ");

statement.close ();
      }catch (SQLException e){System.out.println("SQLException : " +
e.getMessage());System.exit(0);}
        this.releaseConnection ();
    }
//Nel caso in cui il rischio di concedere il prestito sia alto,per simulare la
decisione che potrebbe prendere un direttore di banca,genero randomicamente un
numero 0/1:se esce 0 il prestito non viene concesso in caso contrario si,e,come
nel caso precedente,viene costruito l'oggetto complesso BankQuoteResponse
utilizzando i relativi metodi Set..
    else if(in0.getLoanDuration()<=maxLoanTerm && in0.getCreditScore()>800 &&
(in0.getCreditRisk()).equals("High") ){//seconda condizione
       int decide = (int)(Math.random() * 1);
       if(decide==0){
       reply.setInterestRate(this.PRIME_RATE + ratePremium +
(in0.getLoanDuration()/12)/10 +
        (new Random()).nextInt(10)/10);
       reply.setErrorCode(new Integer(0));
       reply.setQuoteId(bankName + (new Random()).nextInt(100000) + 99999);
int timestamp=getTimestamp();
try{
this.getConnection ();
Statement statement =
this.connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONC
UR_UPDATABLE);
statement.executeUpdate("INSERT INTO loantb(Ssn,Id,Timestamp,Rate,Stato)
VALUES('"+in0.getSsn()+"','"+reply.getQuoteId()+"','"+timestamp+"','"+reply.getI
nterestRate()+"','"+stato+"') ");
statement.close ();
      }catch (SQLException e){System.out.println("SQLException : " +
e.getMessage());System.exit(0);}
        this.releaseConnection ();
    }
       else{
    reply.setInterestRate(0.0);
    reply.setErrorCode(new Integer(1));
    reply.setQuoteId(bankName + (new Random()).nextInt(100000) + 99999);
```

```
     }
    }
    else{//terza condizione
    reply.setInterestRate(0.0);
    reply.setErrorCode(new Integer(1));
    reply.setQuoteId(bankName + (new Random()).nextInt(100000) + 99999);
    }
      return reply;//BankQuoteResponse
    }
```

Metogo getLoan

Un client utilizza questo metodo per confermare la richiesta di prestito precedentemente effettuata;ad ogni richiesta infatti la banca assegna un identificativo che viene appunto utilizzato dal client in un secondo momento per richiedere effettivamente il prestito.

Questo metodo riceve un object value `BankQuoteRequest` nel quale vengono serializzati/deserializzati i dati inviati da un client e che corrispondono al Security Social Number e all'identificativo della pratica relativa al prestito.

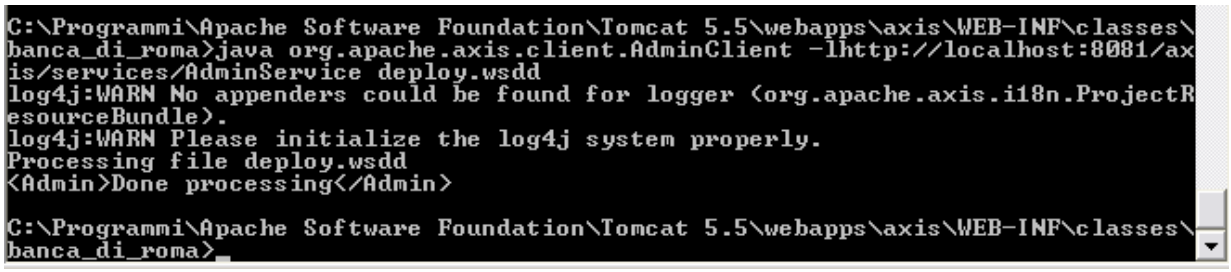
```
//Accedo quindi al database per ottenere le informazioni su eventuali richieste
di prestito di un determinato cliente
PreparedStatement preparedStatement = this.connection.prepareStatement ("SELECT
* FROM loantb WHERE Ssn='"+in0.getSsn()+"' AND Id='"+in0.getQuoteId()+"'");
ResultSet rsltMT = preparedStatement.executeQuery ();

         if(!rsltMT.next())
         result= "Ssn: "+in0.getSsn()+"Nessun prestito richiesto";//Nessuna
richiesta effettuata
         else{
         if((rsltMT.getString(5)).equals("True"))
         result= "Ssn: "+in0.getSsn()+"Prestito già concesso";//Prestito già
erogato

         else if((timestampLoan-rsltMT.getInt(3))>TIME) //La richiesta del
prestito è scaduta

         result="Ssn: "+in0.getSsn()+"Richiesta di prestito scaduta";//Richiesta
scaduta...sono passati più di 10 giorni

         else{
stmtMT=this.connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultS
et.CONCUR_UPDATABLE);
stmtMT.executeUpdate("UPDATE loantb SET Stato='True' WHERE Ssn='"
+in0.getSsn()+"' AND Id='"+in0.getQuoteId()+"'");
stmtMT.close();
result ="Prestito concesso\n"+"Ssn: "+in0.getSsn()+"\n"+"InterestRate:
"+rsltMT.getDouble(4);
             }
         }

preparedStatement.close();

        }catch (SQLException e){System.out.println("SQLException : " +
e.getMessage());System.exit(0);}

this.releaseConnection ();
return result; //Il metodo ritorna una stringa contenente o uno dei messaggi di
errore generati dai casi precedenti oppura la ocnferma che il prestito è stato
effettivamente concesso.In quest'ultimo caso il campo Stato della relativa
tabella viene settato a True per indicare appunto che alla precedente richiesta
di prestito è seguita la conferma da parte del cliente entro i tempi stabiliti.
```

```
}
```

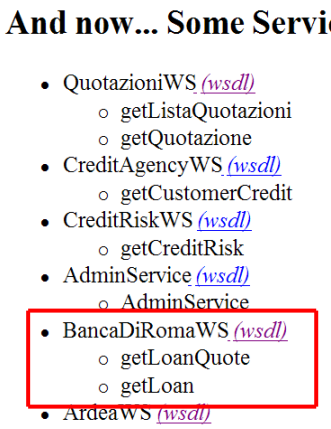
5. Compilare le classi e copiare la cartella `banca_di_roma` in `C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\classes`

6. Effettuare il deploy del Servizio

```
Java org.apache.axis.client.AdminClient -l
http://localhost:8081/axis/services/AdminService deploy.wsdd
```



Per verificare che il servizio BancaDiRomaWS sia stato deployato correttamente basta accedere alla lista dei servizi di axis da un qualsiasi brower:



2.8.2.1 Accesso a database in J2EE:Configurazione di Tomcat

Per accedere ad database utilzzato dal servizio ho seguito le modalità introdotte dalla piattaforma Java 2 Enterprise Edition in cui il ruolo principale è giocato dal container e non dall'applicazione finale: il passaggio a questo nuovo modo di lavorare implica un notevole salto concettuale ma nella pratica la sua adozione è molto semplice ed agevole. I benefici derivanti dall'uso di un container per la gestione delle connessioni si possono raggruppare in due categorie: indirezione e intermediazione della connessione da parte del container. La prima indica che nel servizio non devo specificare nessun parametro specifico del database: niente nome host, nome database e credenziali di accesso. L'applicazione lavora con il concetto di sorgente dati e con un nome astratto che viene associato dal container al database fisico. Tramite la API JNDI l'applicazione in esecuzione all'interno del container deve quindi eseguire una ricerca presso un directory service di un oggetto che implementa l'interfaccia javax.sql.Datasource. Intermediazione del container verso il database engine significa invece che tutte le operazione di gestione della connessione, del flusso dati, autenticazione ed autorizzazione, sono a carico del container il quale si frappone fra il database e l'applicazione finale.
Conseguenza di questo approccio è il fatto di non dover più gestire a livello applicativo del pooling delle connessioni o addirittura della chiusura delle connessioni anche se è da considerarsi una cattiva abitudine lasciare in sospeso volutamente le connessioni. Con Tomcat ho potuto specificare sia le credenziali di accesso ed autenticazione per il database, sia tutti i parametri di connessione. Questi due aspetti (in direzione e intermediazione) permettono alla applicazione di essere del tutto svincolata dalla particolare piattaforma, massimizzando la

portabilità della stessa. Il servizio o un'applicazione in generale J2EE può essere quindi pacchettizzata all'interno di un file con estensione war, jar o ear all'interno del quale non vi è nessun riferimento al nome del server, nome del database, credenziali di autenticazione o parametri di connessione. Una volta che due piattaforme risultano essere completamente e correttamente configurate (dal database all'application server) le applicazioni potranno essere spostati da un server all'altro senza particolari operazioni aggiuntive.

Il concetto è noto come pool di connessioni o Datasource (dall'interfaccia java che lo rappresenterà in fase di sviluppo). Il ciclo di vita di questo nuovo oggetto sarà affidato all'application server, che gestirà le risorse operando su tale pool, garantendo le connessioni al database alle istanze del web service che ne farà richiesta.

I passi da seguire per configurare correttamente Tomcat 5.5.17 e mysql sono:

1. Il primo passo è ottenere il driver dal sito `http://www-it.mysql.com` e copiarlo nella cartella `/lib` di Tomcat. Il driver che ho utilizzato è: `mysql-connector-java-5.1.6-bin.jar`
2. Per configurare una sorgente dati associata a un database MySQL bisogna inserire il seguente frammento di codice xml nel file `context.xml` di Tomcat che rappresenta il file per la configurazione dei datasource:

   Jdbc/ardeadb è il nome JNDI per la sorgente dati: questo nome sarà quello che il servizio utilizzerà per ricavare con una lookup JNDI la datasource e connettersi al database:
   `<Resource name="jdbc/ardeadb"`
   Autenticazione definite a livello di container:
   `auth="Container"`
   Tipo di risorsa che dovrà gestire il Container Tomcat:
   `type="javax.sql.DataSource"`
   Massimo numero di connessioni al database nel pool, compatibilmente con l'attributo *max_connections* di MySQL. Il valore 0 indica nessun limite:
   `maxActive="100"`
   Massimo numero di connessioni inattive da mantenere nel pool. Il valore -1 indica nessun limite:
   `maxIdle="30"`
   massimo tempo di attesa per la ricezione di una connessione al database (in secondi) prima del lancio di un'eccezione. Il valore -1 attende senza limiti:
   `maxWait="10000"`
   `username="alessandro"`
   `password="ale1983"`
   nome del driver utilizzato per la connessione:
   `driverClassName="com.mysql.jdbc.Driver"`
   URL per la connessione JDBC al database. Il parametro `autoReconnect=true` fa sì che il `com.mysql.jdbc.Driver` si riconnetterà automaticamente se MySQL chiude la connessione, cosa che di default avviene dopo 8 ore di connessione inattiva:
   `url="jdbc:mysql://localhost:3306/ardeadb? autoReconnect=true"/>`

Avrei anche potuto mettere queste informazioni nel file `server.xml` o in `minimal-server.xml` ma inserire tali informazioni nel file `context.xml` mi permette di renderle disponibili a tutti gli altri servizi e applicazioni del contenitore e non solo al particolare contesto del servizio BancaDiRomaWS.

3. L'ultimo passaggio nella configurazione del servizio è la registrazione della `resource` nel file `web.xml` di Tomcat:

```
<resource-ref>
<description>DB Connection</description>
<res-ref-name>jdbc/ardeadb</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
```

```
</resource-ref>
```

A questo punto il pool di connessioni è definitivamente attivo e sarà possibile utilizzare il DataSource per gestire in maniera efficiente e pulita il DB della web application.

Il servizio quindi dovrà semplicemente recuperare, ogni qual volta ne avrà la necessità, una connessione dal pool gestito dal Container.

2.8.3 Client Side

1. Costruire un client per invocare i metodi esposti dal servizio

Come detto AXIS (Apache eXtensible Interaction System) è un implementazione di JAX-RPC (ovvero JAVA RMI over soap). Tale sistema fornisce un'interfaccia remota per "messaggi SOAP stile RPC". I modelli di programmazione JAX-RPC si possono riassumere in :

- Generated Stub
- Dynamic Proxy
- DII (Dynamic Invocate Interface)

Generated Stub

Il toolkit JAX-RPC, genera l'interfaccia java RMI e il relativo stub in accordo con la descrizione WSDL, che possono poi essere pubblicati in un JNDI ENC (Environment Naming Context)
In questo modello lo stub è generato a deployment time**.**

Dynamic Proxy

Un dynamic proxy è usato nello stesso modo di generated stub, tranne che l'implementazione dello stub e l'interfaccia remota è generata dinamicamente a run-time.
Come sopra la generazione dell'interfaccia remota avviene in accordo con il documento WSDL, che descrive le interfacce come `porte`; ogni porta può avere uno o più `operazioni`. `Porte e operazioni` sono analoghi a interfacce Java e metodi.

DII (Dynamic Invocate Interface)

JAX-RPC supporta un'altra, ancora più dinamica API, chiamata DII (Dynamic Invocation Interface). DII permette di assemblare chiamate a metodi SOAP dinamicamente a run time.
L'idea è la stessa di CORBA Dynamic Invocation Interface.
JAX-RPC DII permette di creare oggetti che rappresentano singole operazioni di Web Service, altrimenti modellati come metodi di un'interfaccia remota, e di invocare tali operazioni senza la necessità di accedere ad un service factory o di usare uno stub e un'interfaccia remota.

Per connettermi al servizio BancaDiRomaWS ho preferito realizzare un client di tipo Generated Stub attraverso il quale richiamare i metodi remoti come se fossero locali e in modo del tutto trasparente. Questo tipo di soluzione ha reso l'implementazione del client più leggera in quanto altrimenti avrei dovuto oggetto Call, definire l'indirizzo del servizio, definire il nome dell'operazione,chiamare l'invocazione dell'operazione passando i parametri di input (i parametri vanno passati come array di Object) e registrare i mapping per i vari complexType. Avrei dovuto implementare un client del tipo:

```
import org.apache.axis.encoding.ser.*;

Call call = (Call) new Service().createCall();
call.setTargetEndpointAddress(new URL("http://localhost:8081/axis/services/"));
QName qnameBankQuoteRequest = new QName("BancaDiRomaWS", "BankQuoteRequest");
Class classeBankQuoteRequest = BankQuoteRequest.class;
QName qnameBankQuoteResponse = new QName("BancaDiRomaWS", "BankQuoteResponse");
Class classeBankQuoteResponse = BankQuoteResponse.class;
QName qnameBankLoanRequest = new QName("BancaDiRomaWS", "BankLoanRequest");
Class classeBankLoanRequest = BankLoanRequest.class;
```

```
call.registerTypeMapping(classeBankQuoteRequest,     qnameBankQuoteRequest     ,
BeanSerializerFactory.class,BeanDeserializerFactory.class);
call.registerTypeMapping(classeBankQuoteResponse,    qnameBankQuoteResponse    ,
BeanSerializerFactory.class,BeanDeserializerFactory.class);
call.registerTypeMapping(classeBankLoanRequest,      qnameBankLoanRequest      ,
BeanSerializerFactory.class,BeanDeserializerFactory.class);

//richiamo il metodo getLoanQuote
call.setOperationName(new QName("BancaDiRomaWS", "getLoanQuote"));
BankQuoteRequest bpr=new BankQuoteRequest();
bpr.setCreditHistory(10);
bpr.setCreditRisk("Low");
bpr.setCreditScore(1000);
bpr.setLoanAmount(10000);
bpr.setLoanDuration(20);
bpr.setSsn("111-333-555-777");

Object rispostaWS = call.invoke(new Object[]{bpr});
BankQuoteResponse bprs = (BankQuoteResponse) rispostaWS;
```

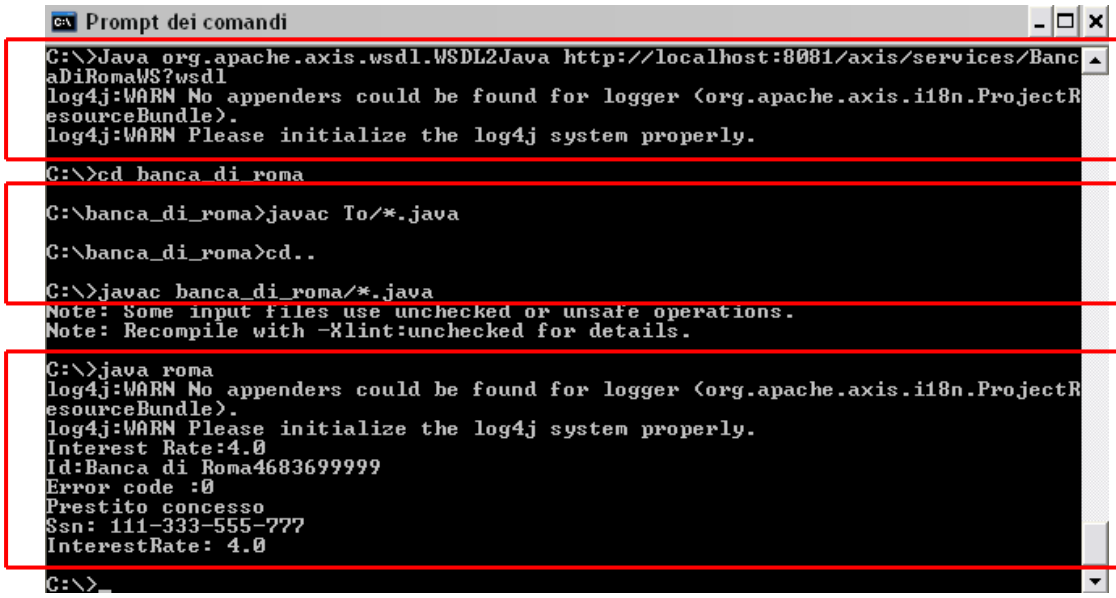
La prima cosa da fare è generare lo stub lato client con il tool WSDL2Java:

```
Java org.apache.axis.wsdl.WSDL2Java
http://localhost:8081/axis/services/BancaDiRomaWS?wsdl
```

1. Genero lo stub

2. Complilo le classi

3. Eseguo il client




```
import java.net.*;
import java.rmi.*;
import java.util.*;
import javax.xml.namespace.*;
import javax.xml.rpc.*;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import java.io.*;

import banca_di_roma.To.*;
import banca_di_roma.*;

public class roma{
public static void main(String[] args)throws Exception{
```

```
int durata = 10;
int ammontare=20000;
String ssn="111-333-555-777";

BancaDiRomaWSService service = new BancaDiRomaWSServiceLocator();
BancaDiRomaWS stub = service.getBancaDiRomaWS();

BankQuoteRequest bpr=new BankQuoteRequest();
bpr.setCreditHistory(15);
bpr.setCreditRisk("Low");
bpr.setCreditScore(1000);
bpr.setLoanAmount(ammontare);
bpr.setLoanDuration(durata);
bpr.setSsn(ssn);

BankQuoteResponse bprs = (BankQuoteResponse)stub.getLoanQuote(bpr);
System.out.println("Interest Rate:"+bprs.getInterestRate());
System.out.println("Id:"+bprs.getQuoteId());
System.out.println("Error code :"+bprs.getErrorCode());

BankLoanRequest blr = new BankLoanRequest();
blr.setSsn(ssn);
blr.setQuoteId(bprs.getQuoteId());

String response = (String)stub.getLoan(blr);
System.out.println(response);
 }
}
```

2.8.4 Messaggi SOAP

Per controllare i messaggi SOAP di richesta e di risposta generati lancio TCPMonitor:

```
java org.apache.axis.utils.tcpmon 7080 localhost 8081
```

dove: 7080 è la porta su cui TCPMonitor si pone in ascolto e con localhost 8081 si indica rispettivamente l'host e la porta a cui inoltrare la richiesta. Per vedere in azione TCPMonitor bisogna modificare la riga di comando nella classe SommaServiceLocator:

```
private java.lang.String BancaDiRomaWS_address =
"http://localhost:8081/axis/services/BancaDiRomaWS";
```

con la seguente:

```
private java.lang.String BancaDiRomaWS_address =
"http://localhost:7080/axis/services/BancaDiRomaWS";
```

In tal modo eseguendo di nuovo la classe Client si vede il passaggio attraverso TCPMonitor della richiesta del client e la risposta del servizio.

Vediamo ora nel dettaglio i messaggi SOAP che vengono scambiati e osserviamo come rispettivo quanto definito precedentemente in [1.4.3].

2.8.4.1 Request message del metodo getLoanQuote

```
==============
Listen Port: 7080
Target Host: localhost
Target Port: 8081
```

```
==== Request ====
```
La prima parte del messaggio coincide con quella di un qualsiasi altro messaggio inviato ad un server con il protocollo HTTP. SOAP è stato infatti esplicitamente pensato per usare HTTP come protocollo di trasporto;utilizzando il framework Axis è possibile trasportare un messaggio SOAP in un messaggio HTTP in modo quasi del tutto trasparente.
Per inviare una richiesta al server con SOAP viene utilizzata la modalità POST:si invia al server il path della risorsa richiesta,seguito da un blocco di dati(payload) che sarà costituito dal messaggio SOAP vero e proprio. Il messaggio HTTP POST che trasporta il payload SOAP deve necessariamente inserire nel suo header(il gruppo di righe che apre la richiesta HTTP) il campo SOAPAction. Il contenuto del campo SOAPAction è un URI(anche vuoto) che dovrebbe fornire delle informazioni riguardanti il contenuto del messaggio SOAP allegato. Per inviare la risposta a un messaggio SOAP il server dovrà attenersi allo standard HTTP. Il contenuto della risposta sarà il messaggio SOAP di ritorno,eventualmente indicante un errore di elaborazione(tramite l'elemento Fault). Se il messaggio SOAP ha generato un errore il server dovrà restituire il messaggio SOAP riguardante l'errore stesso unitamente al codice http 500/Internal Server Error.

```
POST /axis/services/BancaDiRomaWS HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: localhost:7080
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 1639
```

La seconda parte del messaggio è scritta in xml e contiene il payload SOAP.
Bisogna sottolineare il fatto che le regole di codifica utilizzate nei messaggi SOAP sono quelle definite nella sezione 5(`SOAP Encoding`) della specifica di SOAP 1.1[9].
Inizialmente vengono importati i namespace relativi alle codifiche che verranno utilzzate:

```
-xmlns:xsd="http://www.w3.org/2001/XMLSchema"
-xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

Lo Schema XML importato dal namespace`(http://www.w3.org/2001/XMLSchema)` è il linguaggio di definizione XML usato per vincolare la struttura dei documenti XML;consente la definizione di tipi e le dichiarazioni di elementi che possono essere usate per valutare la validità di un documento XML (per la definizione formale di validità si veda [10]).

La specifica XML Schema è suddivisa in due parti [11]:

• Una specifica per le strutture (XML Schema Part 1: Structures), che propone un modello astratto per la definizione di tipi e per la dichiarazione degli attributi ed elementi, che possono essere usati per convalidare un documento XML.

• Una specifica per i tipi di dati (XML Schema Part 2: Datatypes), che fornisce un insieme di tipi built-in e alcuni meccanismi per definire tipi di dati utente.

SOAP segue l'iniziativa di XML Schema importandone il relativo namespace:il riutilizzo di questa tecnologia è infatti una raccomandazione proposta dal consorzio W3C.
Anche se SOAP non richiede l'uso di XML Schema ne utilizza tuttavia i tipi di dati built-in per le definizioni dei tipi che possono essere usati per realizzare il mapping delle chiamate e risposte RPC (serializzazione/deserializzazione).

In altre parole nella costruzione dei messaggi SOAP viene utilizzata sia le codifica XML Schema che la SOAP Encoding in accordo con quanto definito nella specifica SOAP[9].

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

```
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

Il namespace di XML Schema viene associato al prefisso xsd che viene usato per definire i valori dell'attributo xsi:type.

Il namespace di XML Schema-Instance viene utilizzato per associare il documento(ovvero un'istanza di schema) all'XML Schema. A tale namespace è associato il prefisso xsi che permette l'uso dell'attributo `xsi:type` per specificare il tipo di dato di ogni elemento trasmesso;SOAP stabilisce infatti che quando un elemento non è determinato dal suo nome è necessario ricorrere a `xsi:type` per elimare qualsiasi tipo di ambiguità sul tipo di dato trasportato.
Il modo in cui viene costruito un messaggio SOAP dipende da quanto specificato nell'elemento `binding` del file WSDL :
`-style="rpc"`:indica che un messaggio viene inviato utilizzando un formato RPC;di conseguenza il messaggio SOAP contiene un elemento `operation(ns1:getLoanQuote)` che definisce appunto il metodo invocato.
`-use="encoded"`: i valori nel messaggio vengono inviati con informazioni `xsi:type`.

Un valore semplice è definito come un valore appartenente ad un tipo/dominio semplice. I tipi semplici sono tutti quelli definibili come `simpleType` negli Schemi XML. Un valore semplice è serializzato all'interno di un elemento detto `accessor` del valore. Il tipo del valore può essere specificato sia nel messaggio tramite l'attributo `xsi:type`(proveniente dal namespace `schema instance`) che nello schema che definisce la grammatica del messaggio.

Esempio: `<creditRisk xsi:type="soapenc:string">Low</creditRisk>)`

In questo caso `creditRisk` è l'accessor del valore `Low` e il tipo del messaggio è specificato dall'attributo `xsi:type` per indicare che il tipo string è stato codificato con `soapenc` importato dal namespace http`://schemas.xmlsoap.org/soap/encoding/`.

Sempre secondo quanto stabilito nella sezione 5 della specifica SOAP 1.1, alcuni oggetti saranno codificati utilizzando la `multiRef` serialization.In particolari strutture dati`(`come i `complexType BankQuoteRequest| BankQuoteResponse| BankLoanRequest)` un valore può possedere più di un `accessor(creditHistory,creditScore,creditRisk,loanAmount,loanDuration,ssn)`.
Attraverso l'uso degli attributi id e href è possibile referenziare degli accessori in modo da includere il loro valore una sola volta e poi referenziarlo con l'attributo href ove necessario. In questo caso il valore è contenuto effettivamente in uno solo degli elementi `accessor`,dotato di un attributo id. Tutti gli altri accessor saranno vuoti,ma dotati di un attributo `href` che fa riferimento all'`id` dell'accessor contenente il valore.
Un elemento referenzia un valore (contenuto di un altro elemento) se il valore dell'attributo href combacia con il valore dell'attributo id a meno del primo carattere '#'.

La chiamata al metodo remoto `getLoanQuote` viene codificata come una struttura XML nel Body del payload SOAP in cui:

- L'elemento radice della struttura ha lo stesso nome del metodo da chiamare **`(ns1:getLoanQuote)`**;
- I parametri(accessor) sono codificati come elementi figli della radice,dichiarati con lo stesso nome e tipo del corrispondente parametro formale del metodo ed elencati nello stesso ordine con cui compaiono nella signature del metodo.

```
<soapenv:Body>
        <ns1:getLoanQuote
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://banca_di_roma">
```

```
        <in0 href="#id0"/>
    </ns1:getLoanQuote>
```

Nell'elemento radice è contenuto l'attributo `<in0 href="#id0"/>` che fa riferimento all'accessor `multiRef id="id0"` il cui tipo `xsi:type="ns2:BankQuoteRequest"` corrisponde al complexType che rappresenta il parametro d'input del metodo.

```
<multiRef id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:BankQuoteRequest"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="http://To.banca_di_roma">
```

Sono presenti poi tanti accessor quanti sono i paramentri di input del complexType,con lo stesso nome e tipo del corrispondente paramtero formale del tipo complesso:

```
        <creditHistory href="#id1"/>
        <creditRisk xsi:type="soapenc:string">Low</creditRisk>
        <creditScore href="#id2"/>
        <loanAmount href="#id3"/>
        <loanDuration href="#id4"/>
        <ssn xsi:type="soapenc:string">111-333-555-777</ssn>
    </multiRef>
```

Da notare che non tutti gli accessor vengono codificati usando la multi-reference:in questo caso solo gli accessor che corrispondo agli `element` contenuti nella `sequence` del `complexType` nel wsdl i cui dati sono di tipo int `(xsd:int)` vengono rappresentati dalla coppia id/href. I dati contenuti negli `element creditRisk` ed `ssn` sono infatti di tipo string `(type="soapenc:string")` e vengono automaticamente tradotti in un'stanza di XML Schema Instance `xsi:type="soapenc:string"`. La codifica con attributi href/id non viene utilizzata se il tipo di dato dell'elemento è un oggetto appartenente alla sola classe String:se appartiene ad una qualsiasi delle altre classi wrapper per l'elemento verranno creati un accessor con l'attributo href e un accessor con l'attributo id e contenente l'effettivo valore.
Gli accessor `creditHistory,creditScore,loanAmount e loanDuration` contengo quindi il solo attributo `href` che fa riferimento all'`id` dell'`accessor` che contiene concretamente il valore;mentre gli `accessor creditRisk` ed `ssn` contengono direttamente il valore e l'attributo `xsi:type="soapenc:string"` nel quale viene indicato sia il tipo di codifica utilizzata `(soapenc)` che il tipo del dato trasportato `(string)`.

Come previsto gli `accessor` compaiono nello stesso ordine con il quale sono definiti nel file WSDL:

```
<complexType name="BankQuoteRequest">
 <sequence>
  <element name="creditHistory" type="xsd:int" /> //xsd=XML Schema Definition
  <element name="creditRisk" nillable="true" type="soapenc:string" />
  <element name="creditScore" type="xsd:int" />
  <element name="loanAmount" type="xsd:int" />
  <element name="loanDuration" type="xsd:int" />
  <element name="ssn" nillable="true" type="soapenc:string" />
 </sequence>
</complexType>
```

Il valore di ogni `accessor` con attributo href definito precedentemente è contenuto nel corrispondente `accessor multiRef`;negli elementi accessor vengono inseriti:
-attributo `encodingStyle`:per specificare l'URI che identifica le regole di serializzazione utilizzate nella codifica del messaggio del tipo di dato dell'elemento `(SOAP Encoding)`
-attributo `xsi:type`.

```
         <multiRef id="id3" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="xsd:int"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">20000</multiRef>
         <multiRef id="id4" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="xsd:int"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">10</multiRef>
         <multiRef id="id1" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="xsd:int"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">15</multiRef>
         <multiRef id="id2" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="xsd:int"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">1000</multiRef>
      </soapenv:Body>
   </soapenv:Envelope>
```

2.8.4.2 Response message del metodo getLoanQuote

```
==== Response ====
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Wed, 15 Oct 2008 09:53:36 GMT
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

L'esito della chiamata,se positivo,viene codificato come una struttura XML dove:

- L'elemento radice ha convenzionalmente il nome del metodo seguito da "Response" **(<ns1:getLoanQuoteResponse);**
- Il primo elemento della struttura rappresenta il valore di ritorno del metodo **(getLoanQuoteReturn)**. Il suo nome non ha importanza ma ovviamente il tipo **(xsi:type="ns2:BankQuoteResponse")** deve coincidere con quello ritornato dal metodo **(BankQuoteResponse).**

```
<complexType name="BankQuoteResponse">
<sequence>
<element name="errorCode" type="xsd:int" />
<element name="interestRate" type="xsd:double" />
<element name="quoteId" nillable="true" type="soapenc:string" />
  </sequence>
  </complexType>
```

- A seguire vengono inseriti,nell'ordine in cui compaiono della signature,i valori di tutti i parametri di tipo(out) e (in/out) del metodo con le stesse regole di codifica discusse precedentemente per il messaggio di richiesta.

```
      <soapenv:Body>
         <ns1:getLoanQuoteResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://banca_di_roma">
            <getLoanQuoteReturn href="#id0"/>
         </ns1:getLoanQuoteResponse>
```

```
          <multiRef id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:BankQuoteResponse"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="http://To.banca_di_roma">
            <errorCode href="#id1"/>
            <interestRate href="#id2"/>
            <quoteId xsi:type="soapenc:string">Banca di Roma464499999</quoteId>
         </multiRef>

         <multiRef id="id2" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="xsd:double"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">4.0</multiRef>
         <multiRef id="id1" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="xsd:int"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">0</multiRef>
      </soapenv:Body>
   </soapenv:Envelope>
==============
```

2.8.4.3 Request message del metodo getLoan

```
==============
Listen Port: 7080
Target Host: localhost
Target Port: 8081
==== Request ====
POST /axis/services/BancaDiRomaWS HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: localhost:7080
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 763

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <ns1:getLoan
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://banca_di_roma">
            <in0 href="#id0"/>
         </ns1:getLoan>
         <multiRef id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:BankLoanRequest"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="http://To.banca_di_roma">
            <quoteId xsi:type="soapenc:string">Banca di Roma4071299999</quoteId>
            <ssn xsi:type="soapenc:string">111-333-555-777</ssn>
         </multiRef>
      </soapenv:Body>
   </soapenv:Envelope>
```

2.8.4.4 Response message del metodo getLoan

```
==== Response ====
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Wed, 15 Oct 2008 09:53:36 GMT
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <ns1:getLoanResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://banca_di_roma">

<getLoanReturn xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
 Prestito concesso
 Ssn: 111-333-555-777
 InterestRate: 4.0
</getLoanReturn>
         </ns1:getLoanResponse>
      </soapenv:Body>
   </soapenv:Envelope>
==============
```

2.9 Esempio: OrchestratoreWS

Anche se lo standard definisce i WS come Stateless,il che vuol dire che ad ogni invocazione la classe esposta viene istanziata ex-novo,un servizio può essere stateless oppure stateful;
questa caratteristica si riferisce alla capacità di ricordare lo stato di una specifica conversazione (sessione) tra un client ed il servizio;

- Un servizio è *stateless* se non mantiene informazioni di stato su ciò che avviene tra richieste successive di uno stesso client;
- Un servizio è *stateful* se mantiene (qualche) informazione di stato circa le diverse richieste successive da parte di uno stesso client nell'ambito di una sessione (o conversazione).

Abbiamo visto precedentemente come si implementa,si effettua il deploy e si invoca un servizio stateless.

Analizziamo ora un *statefull service* che invocando una serie di *stateless Web Services* implementati con tecnologie differenti fornisce dei servizi bancari ai client.

2.9.1 Server-side

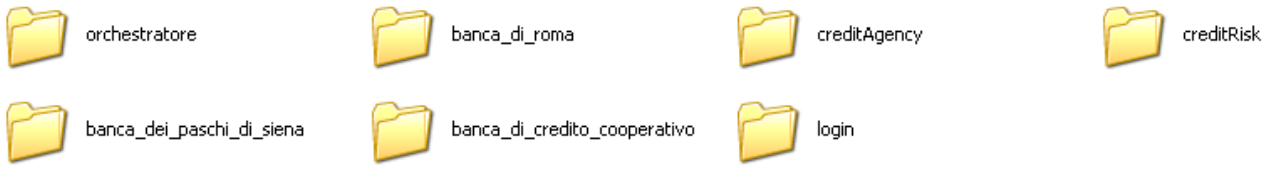
Il deploy del servizio è effettuato con Axis1.4 e viene seguito lo stesso iter presentato con `BancaDiRomaWS`;l'unica differenza consiste nel fatto che il servizio è di tipo statefull pertanto quando viene laniato il tool WSDL2Java va inserita anche l'opzione –d session e nel file di deploy generato automaticamente verrà inserito il parametro scope:

deploy.wsdd

```
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <!-- Services from OrchestratoreWSService WSDL service -->

  <service name="OrchestratoreWS" provider="java:RPC" style="rpc" use="encoded">
      <parameter name="wsdlTargetNamespace"
value="http://orchestratore.orchestrator"/>
      <parameter name="wsdlServiceElement" value="OrchestratoreWSService"/>
      <parameter name="schemaUnqualified" value="http://xml.apache.org/xml-
soap"/>
```

```
      <parameter name="wsdlServicePort" value="OrchestratoreWS"/>
      <parameter name="className"
value="orchestrator.orchestratore.OrchestratoreWSSoapBindingSkeleton"/>
      <parameter name="wsdlPortType" value="OrchestratoreWS"/>
      <parameter name="typeMappingVersion" value="1.2"/>
      <parameter name="allowedMethods" value="*"/>
      <parameter name="scope" value="Session"/>

  </service>
</deployment>
```

Ogni client contatta di conseguenza questo servizio senza sapere che in realtà per il task che ha richiesto ne vengono effettivamente invocati n. Il servizio si comporta quindi come client nei confronti dei Web Services che invoca. Pertanto la cartella `(orchestrator)` contenente i file necessari al deploy include anche le cartelle contente gli stub dei servizi invocati da OrchestratoreWS.
La cartella `C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis\WEB-INF\classes\orchestrator contiene` quindi:



I servizi implementati per realizzare tale scenario sono i quindi seguenti:

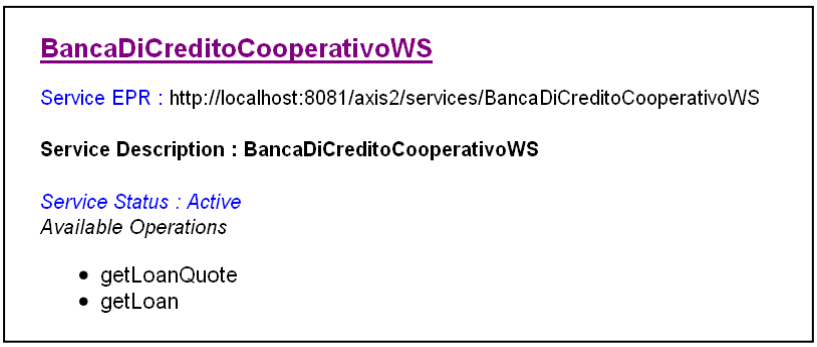
Servizio	Tecnologia	Tipo(in base allo stato)
LoginWS	Axis Apache Tomcat	Stateless
BancaDiRomaWS	Axis Apache Tomcat	Stateless
BancaDiCreditoCooperativoWS	Axis2 Apache Tomcat	Stateless
BancaDeiPaschiDiSiensaWS	Session Bean esposto come servizio su JBoss	Stateless
CreditAgencyWS	Axis Apache Tomcat	Stateless
CreditRiskWS	Axis Apache Tomcat	Stateless
OrchestratoreWS	Axis Apache Tomcat	Statefull

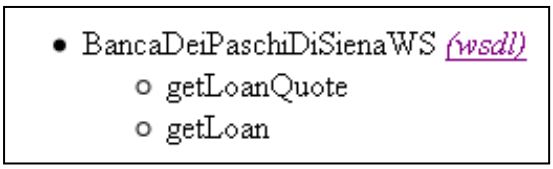
Il Web Service LoginWS viene utilizzato per gestire i login dei client allo statefull service OrchestratoreWS.

- LoginWS *(wsdl)*
  - doLogin

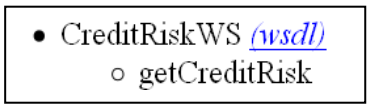
I servizi `BancaDiCreditoCooperativoWS` e `BancaDeiPaschiDiSienaWS` sono equivalenti(cambia solamente il modo in cui gestiscono le concessioni dei prestiti) a `BancaDiRomaWS` visto precedentemente:l'unica differenza consiste nella tecnologia utilzzata per implementarli e verranno presentati rispettivamente nei capitoli 3 e 4.

- BancaDiRomaWS *(wsdl)*
  - getLoanQuote
  - getLoan

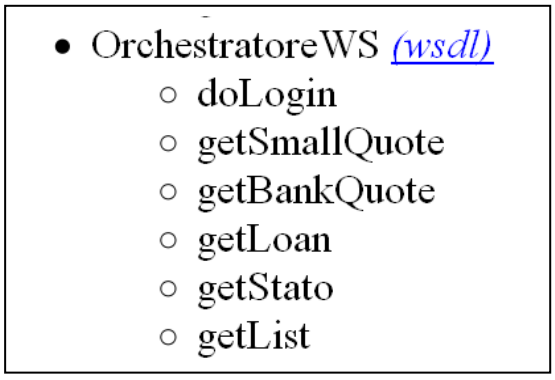




Ai servizi che rappresentano le banche vanno aggiunti i Web Services `CreditAgencyWS` e `CreditRiskWS`. Il primo espone il metodo `getCustomerCredit` che dato un SSN restituisce le informazioni personali di un utente quali Nome,Cognome,Età,creditHistory(da quanto si percepisce un dato creditScore) e creditScore. Il secondo servizio espone il metodo `getCreditRisk` il quale,dati i valori di creditHistory,creditScore ed età calcola il rischio di concedere un prestito ad un determinato utente.



Per realizzare OrchestratoreWS tutti i servizi mostrati sono stati implementati e deployati nei rispettivi application server.



L'interfaccia del servizio è la seguente:

```
package orchestratore;
import java.util.*;
public interface OrchestratoreWS{

public String doLogin(String user);
//Lista delle banche disponibili

public Vector getList();
//Tasso d'interesse migliore
public String getSmallQuote();
//Tasso d'interesse della banca richiesta
public String getBankQuote(String banca);
//Richiesta di prestito ad una banca
```

```
public String getLoan(String banca);
//Stato della conversazione
public String getStato();

}
```

OrchestratoreWS può essere visto come un Web Services in grado di mascherare ai client la presenza degli altri servizi e dare la sensazione di interagire con un'unica entità.

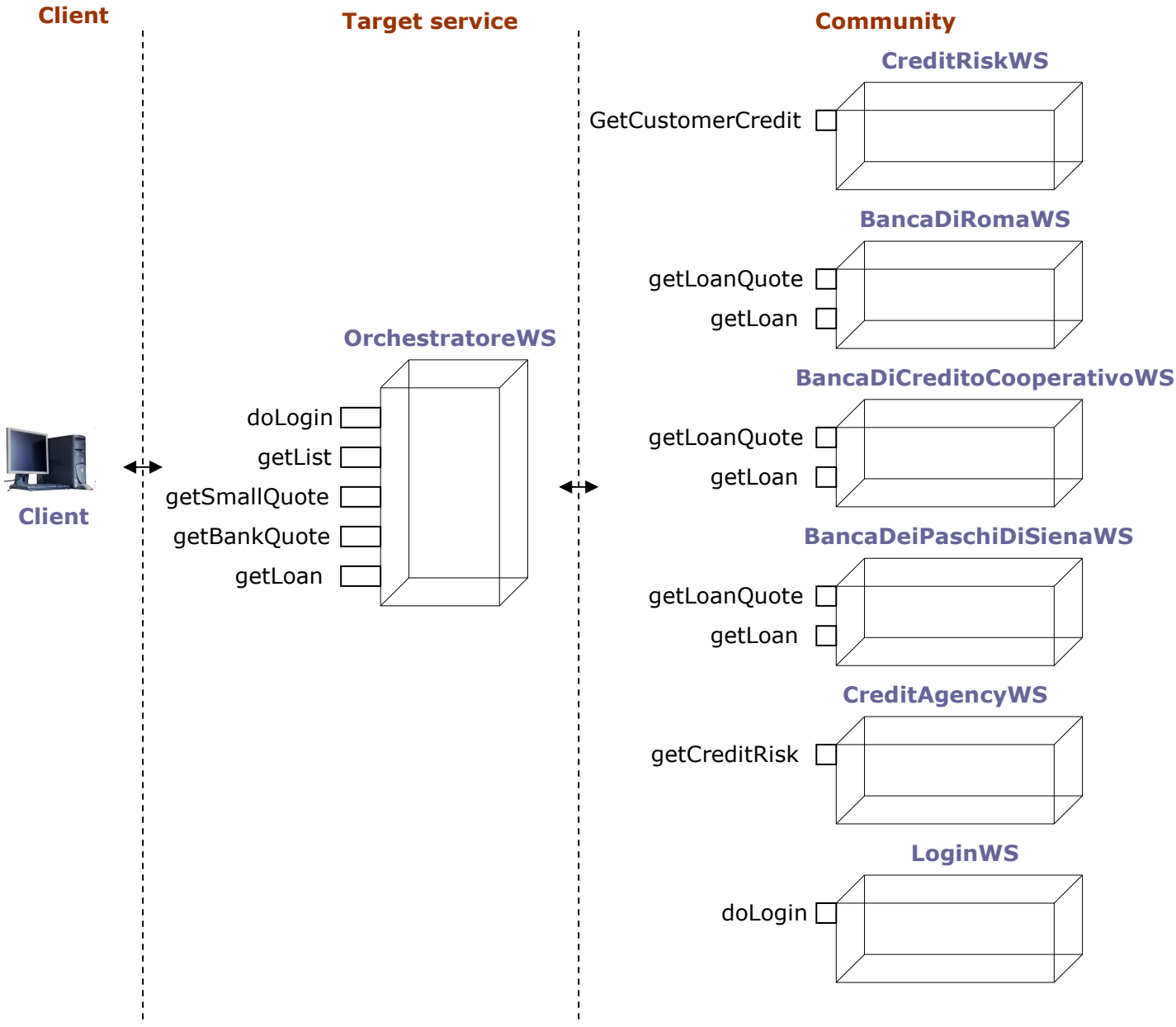


Figura 20:OrchestratoreWS come Target service

Il servizio modellato come un TS < A, S, So, δ, F> con:
- uno stato iniziale
- transazioni deterministiche:dato uno stato,l'esecuzione di un'operazione su questo stato porta il sistema in un solo altro stato. In altre parole non è possibile portare il sistema in due stati differenti eseguendo una stessa azione a partire da uno stato.
è il seguente:

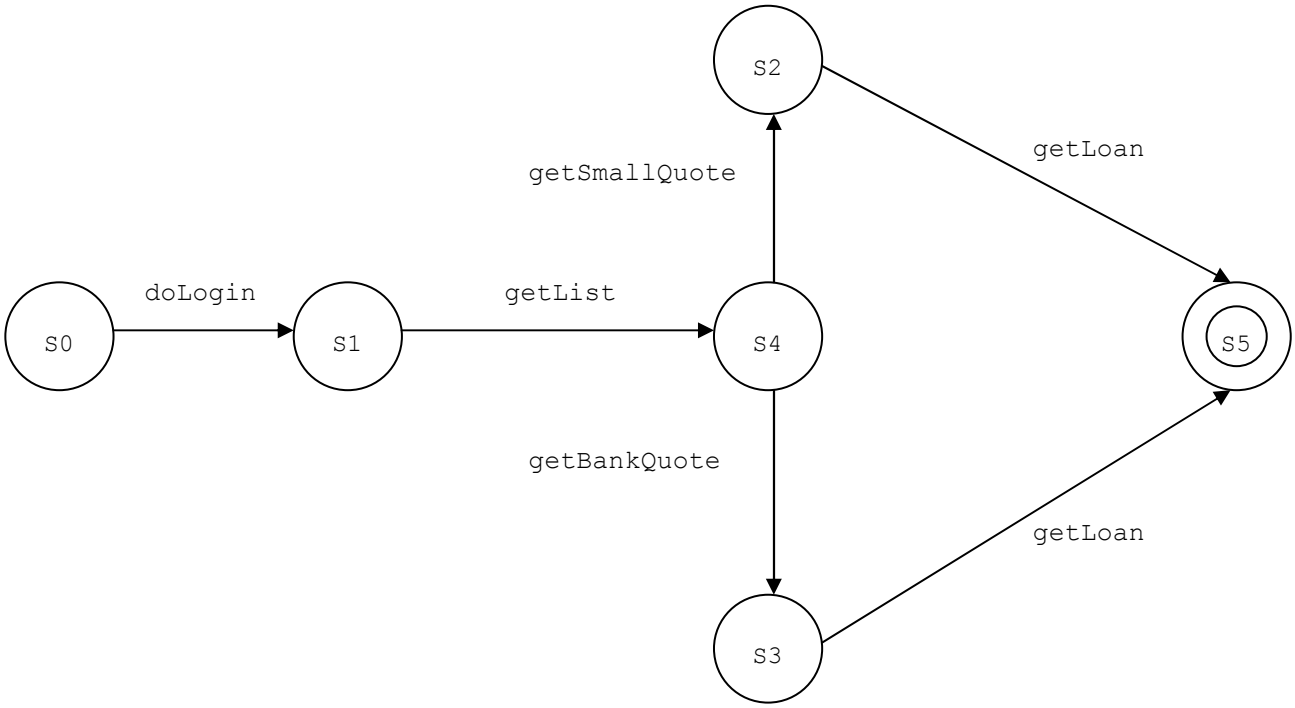


Figura 21:TS del servizio OrchestratoreWS

```
S0------------->Stato iniziale,Operazione:doLogout
S1------------->Operazione:doLogin
S2------------->Operazione:getSmallQuote
S3------------->Operazione:getBankQuote
S4------------->Operazione:getList
S5------------->Operazione:getLoan

A = {doLogin,getSmallQuote,getBankQuote,getList,getLoan}
S = {S0,S1,S2,S3,S4,S5}
So={S0}
δ = {(S0,doLogin,S1),
     (S1,getList,S4),
     (S4,getSmallQuote,S2),
     (S4,getBankQuote,S3),
     (S2,getLoan,S5),
     (S2,getLoan,S5),
    }
F = {S5}
```

Il TS presentato è la rappresentazione astratta del comportamento del servizio;per completezza avrei dovuto prevedere un metodo di Logout,dare la possibilità ai client di richiamare più volte l'operazione `getList,getBankQuote o getSmallQuote` quando si trovano nello stato corrispondente o di eseguire ulteriori operazioni una volta raggiunto lo stato finale corrispondente al task principale di chiedere un prestito ad una banca…..l'obiettivo che volevo raggiungere infatti non era quello di considerare tutte le possibili iterazioni tra i client e il

servizio,bensì dimostrare come a partire da un TS con transazioni deterministiche è possibile implementare un servizio statefull deterministico in grado di mantenere lo stato della conversazione con i client.

Come detto nell'introduzione nessuna delle tecnologie analizzate in questo contesto supporta l'implementazione di servizi statefull in grado di mantenere lo stato della conversazioni con i client. E' compito del programmatore implementare tali tipologie di servizi:nel caso del Web Service OrchestratoreWS ho infatti prima rappresentato il servizio con il relativo TS e successivamente scritto il codice che concretamente implementa i metodi esposti. In effetti ho realizzato un servizio statefull che mantiene lo stato della conversazione ma per il procedimento seguito non si può parlare di tipologia di sviluppo supportata dalla tecnologia:in quel caso,ad esempio,Axis o Axis2 dovrebbero fornire un qualche tool che analizzando il file xml che descrive il TS costruisce automaticamente la classe che implementa il servzio.

Vediamo ora la classe che concretamamente rappresenta il servizio,un possibile client per invocare i metodi da esso esposti e l'output generato dalla relativa comunicazione.

OrchestratoreWSSoapBindingImpl.java

```
/**
 * OrchestratoreWSSoapBindingImpl.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.4 Apr 22, 2006 (06:55:48 PDT) WSDL2Java emitter.
 */

package orchestrator.orchestratore;

import orchestrator.login.*;
import orchestrator.banca_di_roma.*;
import orchestrator.banca_di_credito_cooperativo.*;
import orchestrator.banca_dei_paschi_di_siena.*;
import orchestrator.creditAgency.*;
import orchestrator.creditRisk.*;

import java.sql.SQLException;
import java.io.IOException;

import java.util.*;
import java.net.*;
import java.rmi.*;
import javax.xml.namespace.*;
import javax.xml.rpc.*;

public class OrchestratoreWSSoapBindingImpl implements
orchestrator.orchestratore.OrchestratoreWS{
//La variabile Stato viene utilizzata per gestire lo stato della conversazione
        private int Stato=0;
//La variabile history contiene la sequenza di tutti gli stati correttamente
//visitati durante la comunicazione
private String history="Stato iniziale";
        private String Id="";
        private int durata = 50;
        private int ammontare=10000;
        private String ssn="111-333-555-777";
        private CustomerCredit credit;
        private String risk="";

        private String  Id_Roma="";
        private String  Id_Credito="";
        private String  Id_Siena="";
```

```
//Metodo doLogin()
public java.lang.String doLogin(java.lang.String in0) throws
java.rmi.RemoteException {
        if(Stato!=0){
        return "Impossibile eseguire l'operazione richiesta in questo
stato"+"\n";
        }
//L'unico stato dal quale è possible accedere allo Stato 1 di Login è lo Stato
//iniziale 0
        Stato=1;
//In questo modo mantengo la sequenza degli stati visitati
        history=history.concat("|"+Stato);
//Se LoginWS non è disponibile non è possibile accedere al servizio
        try{
        LoginWSService serviceLogin = new LoginWSServiceLocator();
        LoginWS_PortType stubLogin = serviceLogin.getLoginWS();
        return "User: "+stubLogin.doLogin(in0)+"\n";
        }catch(RemoteException ex){
       return "Servizio non disponibile:LoginWS non attivo";
                              }
         catch(ServiceException ex){
       return "Servizio non disponibile:LoginWS non attivo";
                              }
    }
//Metodoo getSmallQuote()
//Questo metodo verifica inizialmente se entrambi i servizi CreditAgencyWS e
//CreditRiskWS sono disponibili,in caso contrario ritorna un messaggio di
//servizio non disponibile. In caso contrario con le informazioni ottenuti da
//questi servizi costruisce i complexType relativi ai messaggi di richiesta dei
//servizi che rappresentano le banche e utilizza i relativi stub per invocare il
//metodo getBankQuote di ognuna di esse. Vengono quindi inseriti in un Vector i
//nomi delle banche disponibili e restituita quella che propone il tasso
//d'interesse minore.
public java.lang.String getSmallQuote() throws java.rmi.RemoteException {
//L'unico stato dal quale è possibile richiedere tale operazione ed accedere al
//relativo stato è lo stato 4;prima di invocare questo metodo bisogna quindi
//invocare il metodo getList
        if(Stato!=4){
        return "Impossibile eseguire l'operazione richiesta in questo
stato"+"\n";
        }
        String min="";
        int minore=0;

        try{
        CreditAgencyWSService service = new CreditAgencyWSServiceLocator();
        CreditAgencyWS_PortType stub = service.getCreditAgencyWS();
        credit=(CustomerCredit)stub.getCustomerCredit(ssn);
        }catch(ServiceException ex){return "Servizio non
disponibile:CreditAgencyWS non attivo"+"\n";}
         catch(RemoteException ex){return "Servizio non disponibile:CreditAgency
non attivo"+"\n";}
        try{
        CreditRiskWSService service1 = new CreditRiskWSServiceLocator();
        CreditRiskWS_PortType stub1 = service1.getCreditRiskWS();

risk=(String)stub1.getCreditRisk(credit.getCreditScore(),credit.getCreditHistory
(),credit.getAge());
        }catch(ServiceException ex){return "Servizio non
disponibile:CreditRiskWS non attivo"+"\n";}
         catch(RemoteException ex){return "Servizio non disponibile:CreditRiskWS
non attivo"+"\n";}
```

```
        int d=0;
        Vector list=new Vector();
        Vector list_rate=new Vector();
//Imposto le variabili relative ai tassi d'interesse delle banche ad un valore
//alto
        double Rate_Roma=1000.0;
        double Rate_Credito=1000.0;
        double Rate_Siena=1000.0;
//Nel ciclo do-while invoco i servizi delle banche ed inserisco quelle
//attualmente disponibili nel Vector list
        do{
             try{
             if(d==0){
         //richiesta banca di roma
         BancaDiRomaWSService service2 = new BancaDiRomaWSServiceLocator();
         BancaDiRomaWS_PortType stub2 = service2.getBancaDiRomaWS();
         orchestrator.banca_di_roma.BankQuoteRequest bpr=new
orchestrator.banca_di_roma.BankQuoteRequest();
         bpr.setCreditHistory(credit.getCreditHistory());
         bpr.setCreditRisk(risk);
         bpr.setCreditScore(credit.getCreditScore());
         bpr.setLoanAmount(ammontare);
         bpr.setLoanDuration(durata);
         bpr.setSsn(credit.getSsn());
                   orchestrator.banca_di_roma.BankQuoteResponse bprs1
=(orchestrator.banca_di_roma.BankQuoteResponse)stub2.getLoanQuote(bpr);
//Se il servizio è attivo eseguo le seguenti linee di codice:
//aggiungo alla lista dei servizi attivi quello appena contattato
                   list.add("Banca di Roma");
//se la banca ha concesso il prestito
                   if(bprs1.getErrorCode()==0){
//nella variabile Id_Roma inserisco il codice relative al prestito e che veràà
//utilizzato per richiedere concretamente il prestito alla banca
                   Id_Roma=bprs1.getQuoteId();
//mentre in Rate_Roma inserisco il tasso d'interesse associato al prestito che
//può essere concesso
                   Rate_Roma=bprs1.getInterestRate();
                   }
                  d++;
                 }
             if(d<2){
         //richiesta paschi siena
         BancaPSEJBService service3 = new BancaPSEJBServiceLocator();
         BancaPSEJB stub3 = service3.getBancaDeiPaschiDiSienaWS();
         orchestrator.banca_dei_paschi_di_siena.BankQuoteRequest bpr1=new
orchestrator.banca_dei_paschi_di_siena.BankQuoteRequest();
         bpr1.setCreditHistory(credit.getCreditHistory());
         bpr1.setCreditRisk(risk);
         bpr1.setCreditScore(credit.getCreditScore());
         bpr1.setLoanAmount(ammontare);
         bpr1.setLoanDuration(durata);
         bpr1.setSsn(credit.getSsn());
                   orchestrator.banca_dei_paschi_di_siena.BankQuoteResponse bprs2
=(orchestrator.banca_dei_paschi_di_siena.BankQuoteResponse)stub3.getLoanQuote(bp
r1);
                   list.add("Monte dei Paschi di Siena");
                   if(bprs2.getErrorCode()==0){
                   Id_Siena=bprs2.getQuoteId();
                   Rate_Siena=bprs2.getInterestRate();
                   }
                   d++;
                  }
```

```
            if(d<3){
        //richiesta credito cooperativo
        BancaDiCreditoCooperativoWS service4 = new
BancaDiCreditoCooperativoWSLocator();
        BancaDiCreditoCooperativoWSSoap12BindingStub stub4 =
(BancaDiCreditoCooperativoWSSoap12BindingStub)service4.getBancaDiCreditoCooperat
ivoWSHttpSoap12Endpoint();
        orchestrator.banca_di_credito_cooperativo.BankQuoteRequest bpr2=new
orchestrator.banca_di_credito_cooperativo.BankQuoteRequest();
        bpr2.setCreditHistory(credit.getCreditHistory());
        bpr2.setCreditRisk(risk);
        bpr2.setCreditScore(credit.getCreditScore());
        bpr2.setLoanAmount(ammontare);
        bpr2.setLoanDuration(durata);
        bpr2.setSsn(credit.getSsn());
                    orchestrator.banca_di_credito_cooperativo.BankQuoteResponse
bprs3
=(orchestrator.banca_di_credito_cooperativo.BankQuoteResponse)stub4.getLoanQuote
(bpr2);
                     list.add("Banca di Credito Cooperativo");
                     if(bprs3.getErrorCode()==0){
                     Id_Credito=bprs3.getQuoteId();
                    Rate_Credito=bprs3.getInterestRate();
                     }
                     d++;
                }
            }
//Le eccezioni si verificano se un servizio contattato non è disponibile:ogni
//volta che invoco il metogo getBankQuote di un servizio non attivo viene di
//conseguenza incrementata la variabile d attraverso la quale è possibile
//invocare il servizio successivo ed uscire dal ciclo.
             catch(RemoteException ex){d++;}
             catch(ServiceException ex){d++;}
        }while(d<3);
//Il seguente If serve per controllare se c'è almeno una banca disposta a
//concedere il prestito con le condizioni indicate;se infatti tutte le variabili
//che corrispondono ai tassi d'interesse hanno lo stesso valore di default
//impostato precedentemente significa che quando sono stati contattati i
//relativi servizi questi hanno risposto con un codice di errore pari ad 1.
        if(Rate_Roma==1000.0 && Rate_Credito==1000.0 && Rate_Siena==1000.0)
        return "Nessuna banca disposta a concedere il prestito";
//Se c'è almeno una banca disposta ad concedere il prestito…
        if(!list.isEmpty()){
//Aggiorno lo stato
        Stato=2;
        history=history.concat("|"+Stato);
//Restituisco una messaggio che contiene il come della banca che offre il
//miglior tasso d'interesse e il relativo valore.
        if(Rate_Roma<Rate_Credito && Rate_Roma<Rate_Siena)
        min="Banca: Banca di Roma"+"\n"+"Interest_Rate: "+Rate_Roma+"\n";
        else if(Rate_Credito<Rate_Roma && Rate_Credito<Rate_Siena)
        min="Banca: Banca di Credito Cooperativo"+"\n"+"Interest_Rate:
"+Rate_Credito+"\n";
        else if(Rate_Siena<Rate_Roma && Rate_Siena<Rate_Credito)
        min="Banca: Monte dei Paschi di Siena"+"\n"+"Interest_Rate:
"+Rate_Siena+"\n";
        Stato=2;
        return min;
        }
//Se la lista dei servizi invocati è vuota…
        return "Nessun servizio attualmente disponibile"+"\n";
    }
```

```
//Metodo getLoan()
//Per richiedere concretamente un prestito vengono utilizzati i codici inviati
//da ogni banca relativi ad una precedente richiesta,preventivo di prestito e
//memorizzati nelle variabili Id_Roma,Id_Credito e Id_Siena.
 public java.lang.String getLoan(java.lang.String in0) throws
java.rmi.RemoteException {
        if(Stato==0 || Stato==1 || Stato==4){
        return "Impossibile eseguire l'operazione richiesta in questo
stato"+"\n";
        }
//Se viene richiesto un prestito alla Banca di Roma
        if(in0.equals("Banca di Roma")){
        try{
        BancaDiRomaWSService service2 = new BancaDiRomaWSServiceLocator();
        BancaDiRomaWS_PortType stub2 = service2.getBancaDiRomaWS();
        orchestrator.banca_di_roma.BankLoanRequest blr=new
orchestrator.banca_di_roma.BankLoanRequest();
        blr.setQuoteId(""+Id_Roma);
        blr.setSsn(ssn);
        String response_loan=(String)stub2.getLoan(blr);
        Stato=5;
        history=history.concat("|"+Stato);
        return response_loan;
        }
             catch(RemoteException ex){return "Servizio BancaDiRomaWS non
disponibile"+"\n";}
             catch(ServiceException ex){return "Servizio BancaDiRomaWS non
disponibile"+"\n";}
        }
//Se viene richiesto un prestito a Monte dei Paschi di Siena
        else if(in0.equals("Monte dei Paschi di Siena")){
        try{
        BancaPSEJBService service3 = new BancaPSEJBServiceLocator();
        BancaPSEJB stub3 = service3.getBancaDeiPaschiDiSienaWS();
        orchestrator.banca_dei_paschi_di_siena.BankLoanRequest blr=new
orchestrator.banca_dei_paschi_di_siena.BankLoanRequest();
        blr.setQuoteId(""+Id_Siena);
        blr.setSsn(ssn);
      String response_loan=(String)stub3.getLoan(blr);
        Stato=5;
        history=history.concat("|"+Stato);
        return response_loan;
        }
             catch(RemoteException ex){return "Servizio BancaDeiPaschiDiSienaWS
non disponibile"+"\n";}
             catch(ServiceException ex){return "Servizio BancaDeiPaschiDiSienaWS
non disponibile"+"\n";}
        }
//Se viene richiesto un prestito a Banca di Credito Cooperativo
        else if(in0.equals("Banca di Credito Cooperativo")){
        try{
        BancaDiCreditoCooperativoWS service4 = new
BancaDiCreditoCooperativoWSLocator();
        BancaDiCreditoCooperativoWSSoap12BindingStub stub4 =
(BancaDiCreditoCooperativoWSSoap12BindingStub)service4.getBancaDiCreditoCooperat
ivoWSHttpSoap12Endpoint();
        orchestrator.banca_di_credito_cooperativo.BankLoanRequest blr=new
orchestrator.banca_di_credito_cooperativo.BankLoanRequest();
        blr.setQuoteId(""+Id_Credito);
        blr.setSsn(ssn);
      String response_loan=(String)stub4.getLoan(blr);
        Stato=5;
        history=history.concat("|"+Stato);
```

```
        return response_loan;
        }
             catch(RemoteException ex){return "Servizio
BancaDiCreditoCooperativoWS non disponibile"+"\n";}
             catch(ServiceException ex){return "Servizio
BancaDiCreditoCooperativoWS non disponibile"+"\n";}
        }
        return "Banca inclusa nel servizio"+"\n";
    }

//Metodo getBankQuote
//Il metodo ritorna il tasso d'interesse associato ad un prestito richiesto
//presso una banca
    public java.lang.String getBankQuote(java.lang.String in0) throws
java.rmi.RemoteException {
        if(Stato!=4){
        return "Impossibile eseguire l'operazione richiesta in questo
stato"+"\n";
        }
        if(in0.equals("Banca di Roma")){
        try{
        CreditAgencyWSService service = new CreditAgencyWSServiceLocator();
        CreditAgencyWS_PortType stub = service.getCreditAgencyWS();
        credit=(CustomerCredit)stub.getCustomerCredit(ssn);
        }catch(ServiceException ex){return "Servizio non
disponibile:CreditAgencyWS non attivo"+"\n";}
         catch(RemoteException ex){return "Servizio non disponibile:CreditAgency
non attivo"+"\n";}
        try{
        CreditRiskWSService service1 = new CreditRiskWSServiceLocator();
        CreditRiskWS_PortType stub1 = service1.getCreditRiskWS();

risk=(String)stub1.getCreditRisk(credit.getCreditScore(),credit.getCreditHistory
(),credit.getAge());
        }catch(ServiceException ex){return "Servizio non
disponibile:CreditRiskWS non attivo"+"\n";}
         catch(RemoteException ex){return "Servizio non disponibile:CreditRiskWS
non attivo"+"\n";}

        try{
        BancaDiRomaWSService service2 = new BancaDiRomaWSServiceLocator();
        BancaDiRomaWS_PortType stub2 = service2.getBancaDiRomaWS();
        orchestrator.banca_di_roma.BankQuoteRequest bpr=new
orchestrator.banca_di_roma.BankQuoteRequest();
        bpr.setCreditHistory(credit.getCreditHistory());
        bpr.setCreditRisk(risk);
        bpr.setCreditScore(credit.getCreditScore());
        bpr.setLoanAmount(ammontare);
        bpr.setLoanDuration(durata);
        bpr.setSsn(credit.getSsn());
                  orchestrator.banca_di_roma.BankQuoteResponse bprs1
=(orchestrator.banca_di_roma.BankQuoteResponse)stub2.getLoanQuote(bpr);
                  if(bprs1.getErrorCode()==1)
                  return "Banca di Roma non concede il prestito"+"\n";
                  Stato=3;
                 history=history.concat("|"+Stato);
                  Id_Roma=bprs1.getQuoteId();
                  return "Banca di Roma
Interest_Rate:"+bprs1.getInterestRate()+"\n";

             }
             catch(RemoteException ex){return "Servizio BancaDiRomaWS non
disponibile"+"\n";}
```

```
            catch(ServiceException ex){return "Servizio BancaDiRomaWS non
disponibile"+"\n";}
         }
        else if(in0.equals("Monte dei Paschi di Siena")){
        try{
        CreditAgencyWSService service = new CreditAgencyWSServiceLocator();
        CreditAgencyWS_PortType stub = service.getCreditAgencyWS();
        credit=(CustomerCredit)stub.getCustomerCredit(ssn);
        }catch(ServiceException ex){return "Servizio non
disponibile:CreditAgencyWS non attivo"+"\n";}
         catch(RemoteException ex){return "Servizio non disponibile:CreditAgency
non attivo"+"\n";}
        try{
        CreditRiskWSService service1 = new CreditRiskWSServiceLocator();
        CreditRiskWS_PortType stub1 = service1.getCreditRiskWS();

risk=(String)stub1.getCreditRisk(credit.getCreditScore(),credit.getCreditHistory
(),credit.getAge());
        }catch(ServiceException ex){return "Servizio non
disponibile:CreditRiskWS non attivo"+"\n";}
         catch(RemoteException ex){return "Servizio non disponibile:CreditRiskWS
non attivo"+"\n";}

        try{
        BancaPSEJBService service3 = new BancaPSEJBServiceLocator();
        BancaPSEJB stub3 = service3.getBancaDeiPaschiDiSienaWS();
        orchestrator.banca_dei_paschi_di_siena.BankQuoteRequest bpr=new
orchestrator.banca_dei_paschi_di_siena.BankQuoteRequest();
        bpr.setCreditHistory(credit.getCreditHistory());
        bpr.setCreditRisk(risk);
        bpr.setCreditScore(credit.getCreditScore());
        bpr.setLoanAmount(ammontare);
        bpr.setLoanDuration(durata);
        bpr.setSsn(credit.getSsn());
        orchestrator.banca_dei_paschi_di_siena.BankQuoteResponse bprs1
=(orchestrator.banca_dei_paschi_di_siena.BankQuoteResponse)stub3.getLoanQuote(bp
r);
                 if(bprs1.getErrorCode()==1)
                 return "Monte dei Paschi di Siena non concede il
prestito"+"\n";
                 Stato=3;
                history=history.concat("|"+Stato);
                 Id_Siena=bprs1.getQuoteId();
                 return "Monte dei Paschi di Siena
Interest_Rate:"+bprs1.getInterestRate()+"\n";
             }
             catch(RemoteException ex){return "Servizio BancaDeiPaschiDiSienaWS
non disponibile"+"\n";}
             catch(ServiceException ex){return "Servizio BancaDeiPaschiDiSienaWS
non disponibile"+"\n";}
        }
        else if(in0.equals("Banca di Credito Cooperativo")){
        try{
        CreditAgencyWSService service = new CreditAgencyWSServiceLocator();
        CreditAgencyWS_PortType stub = service.getCreditAgencyWS();
        credit=(CustomerCredit)stub.getCustomerCredit(ssn);
        }catch(ServiceException ex){return "Servizio non
disponibile:CreditAgencyWS non attivo"+"\n";}
         catch(RemoteException ex){return "Servizio non disponibile:CreditAgency
non attivo"+"\n";}
        try{
        CreditRiskWSService service1 = new CreditRiskWSServiceLocator();
        CreditRiskWS_PortType stub1 = service1.getCreditRiskWS();
```

```
risk=(String)stub1.getCreditRisk(credit.getCreditScore(),credit.getCreditHistory
(),credit.getAge());
        }catch(ServiceException ex){return "Servizio non
disponibile:CreditRiskWS non attivo"+"\n";}
         catch(RemoteException ex){return "Servizio non disponibile:CreditRiskWS
non attivo"+"\n";}

        try{
        BancaDiCreditoCooperativoWS service4 = new
BancaDiCreditoCooperativoWSLocator();
        BancaDiCreditoCooperativoWSSoap12BindingStub stub4 =
(BancaDiCreditoCooperativoWSSoap12BindingStub)service4.getBancaDiCreditoCooperat
ivoWSHttpSoap12Endpoint();
        orchestrator.banca_di_credito_cooperativo.BankQuoteRequest bpr=new
orchestrator.banca_di_credito_cooperativo.BankQuoteRequest();
        bpr.setCreditHistory(credit.getCreditHistory());
        bpr.setCreditRisk(risk);
        bpr.setCreditScore(credit.getCreditScore());
        bpr.setLoanAmount(ammontare);
        bpr.setLoanDuration(durata);
        bpr.setSsn(credit.getSsn());
        orchestrator.banca_di_credito_cooperativo.BankQuoteResponse bprs1
=(orchestrator.banca_di_credito_cooperativo.BankQuoteResponse)stub4.getLoanQuote
(bpr);
                  if(bprs1.getErrorCode()==1)
                   return "Banca di Credito Cooperativo non concede il
prestito"+"\n";
                   Stato=3;
                  history=history.concat("|"+Stato);
                   Id_Credito=bprs1.getQuoteId();
                   return "Banca di Credito Cooperativo
Interest_Rate:"+bprs1.getInterestRate()+"\n";
             }
             catch(RemoteException ex){return "Servizio
BancaDiCreditoCooperativoWS non disponibile"+"\n";}
             catch(ServiceException ex){return "Servizio
BancaDiCreditoCooperativoWS non disponibile"+"\n";}
         }
         return "Banca non inclusa nel servizio"+"\n";
    }
//metodo getStato
    public java.lang.String getStato() throws java.rmi.RemoteException {
        String response="";
        if(Stato==0)
        response="Stato: "+Stato+"\n"+"Operazione:Stato iniziale,bisogna
effettuare il login" +"\n"+  "History: "+history+"\n";
        else if(Stato==1)
        response="Stato: "+Stato+"\n"+"Operazione:doLogin"+"\n"+  "History:
"+history+"\n";
        else if(Stato==2)
        response="Stato: "+Stato+"\n"+"Operazione:getSmallQuote" +"\n"+
"History: "+history+"\n";
        else if(Stato==3)
        response="Stato: "+Stato+"\n"+"Operazione:getBankQuote" +"\n"+
"History: "+history+"\n";
        else if(Stato==4)
        response="Stato: "+Stato+"\n"+"Operazione:getList" +"\n"+  "History:
"+history+"\n";
        else if(Stato==5)
        response="Stato: "+Stato+"\n"+"Operazione:getLoan" +"\n"+  "History:
"+history+"\n";
        return response;
```

```
    }
//Metodo getList
//Il metodo resituisce un oggetto di tipo Vector contenente la lista di tutti I
servizi attualmente disponibili
    public java.util.Vector getList() throws java.rmi.RemoteException {
      Vector list=new Vector();
        if(Stato!=1){
        list.add("Impossibile eseguire l'operazione richiesta in questo
stato"+"\n");
        return list;
        }
//La prima cosa da verificare è che siano attivi i servizi CreditAgency e
//CreditRisk
        try{
        CreditAgencyWSService service = new CreditAgencyWSServiceLocator();
        CreditAgencyWS_PortType stub = service.getCreditAgencyWS();
        credit=(CustomerCredit)stub.getCustomerCredit(ssn);
        }catch(ServiceException ex){list.add("Servizio non
disponibile:CreditAgencyWS non attivo"+"\n");return list;}
         catch(RemoteException ex){list.add("Servizio non
disponibile:CreditAgency non attivo"+"\n");return list;}
        try{
        CreditRiskWSService service1 = new CreditRiskWSServiceLocator();
        CreditRiskWS_PortType stub1 = service1.getCreditRiskWS();

risk=(String)stub1.getCreditRisk(credit.getCreditScore(),credit.getCreditHistory
(),credit.getAge());
        }catch(ServiceException ex){list.add("Servizio non
disponibile:CreditRiskWS non attivo"+"\n");return list;}
         catch(RemoteException ex){list.add("Servizio non
disponibile:CreditRiskWS non attivo"+"\n");return list;}

       //risposte
       int d=0;
//Nel ciclo do-while viene costruita la lista dei servizi attivi
       do{
            try{
            if(d==0){
        //Richiesta banca di roma
        BancaDiRomaWSService service2 = new BancaDiRomaWSServiceLocator();
        BancaDiRomaWS_PortType stub2 = service2.getBancaDiRomaWS();
        orchestrator.banca_di_roma.BankQuoteRequest bpr=new
orchestrator.banca_di_roma.BankQuoteRequest();
        bpr.setCreditHistory(credit.getCreditHistory());
        bpr.setCreditRisk(risk);
        bpr.setCreditScore(credit.getCreditScore());
        bpr.setLoanAmount(ammontare);
        bpr.setLoanDuration(durata);
        bpr.setSsn(credit.getSsn());
                  orchestrator.banca_di_roma.BankQuoteResponse bprs1
=(orchestrator.banca_di_roma.BankQuoteResponse)stub2.getLoanQuote(bpr);
                  list.add("Banca di Roma");
                  d++;
                  }
            if(d<2){
        //richiesta paschi siena
        BancaPSEJBService service3 = new BancaPSEJBServiceLocator();
        BancaPSEJB stub3 = service3.getBancaDeiPaschiDiSienaWS();
        orchestrator.banca_dei_paschi_di_siena.BankQuoteRequest bpr1=new
orchestrator.banca_dei_paschi_di_siena.BankQuoteRequest();
        bpr1.setCreditHistory(credit.getCreditHistory());
        bpr1.setCreditRisk(risk);
        bpr1.setCreditScore(credit.getCreditScore());
```

```
        bpr1.setLoanAmount(ammontare);
        bpr1.setLoanDuration(durata);
        bpr1.setSsn(credit.getSsn());
                  orchestrator.banca_dei_paschi_di_siena.BankQuoteResponse bprs2
=(orchestrator.banca_dei_paschi_di_siena.BankQuoteResponse)stub3.getLoanQuote(bp
r1);
                  list.add("Monte dei Paschi di Siena");
                  d++;
                  }
            if(d<3){
        //richiesta credito cooperativo
        BancaDiCreditoCooperativoWS service4 = new
BancaDiCreditoCooperativoWSLocator();
        BancaDiCreditoCooperativoWSSoap12BindingStub stub4 =
(BancaDiCreditoCooperativoWSSoap12BindingStub)service4.getBancaDiCreditoCooperat
ivoWSHttpSoap12Endpoint();
        orchestrator.banca_di_credito_cooperativo.BankQuoteRequest bpr2=new
orchestrator.banca_di_credito_cooperativo.BankQuoteRequest();
        bpr2.setCreditHistory(credit.getCreditHistory());
        bpr2.setCreditRisk(risk);
        bpr2.setCreditScore(credit.getCreditScore());
        bpr2.setLoanAmount(ammontare);
        bpr2.setLoanDuration(durata);
        bpr2.setSsn(credit.getSsn());
                   orchestrator.banca_di_credito_cooperativo.BankQuoteResponse
bprs3
=(orchestrator.banca_di_credito_cooperativo.BankQuoteResponse)stub4.getLoanQuote
(bpr2);
                   list.add("Banca di Credito Cooperativo");
                   d++;
                  }
            }
             catch(RemoteException ex){d++;}
             catch(ServiceException ex){d++;}
        }while(d<3);
//Se c'è almeno un servizio attivo
        if(!list.isEmpty()){
//Passo allo stato successivo
        Stato=4;
//Aggiorno la sequenza degli stati visitati
        history= history.concat("|"+Stato);
//Resituisco la lista dei servizi attivi
        return list;
        }
//Altrimenti la lista contiene un semplice messaggio…
        list.add("Nessun servizio attualmente disponibile"+"\n");
        return list;
 }
}
```

2.9.2 Client-side

Un possibile client che invochi i metodi del servizio è il seguente:

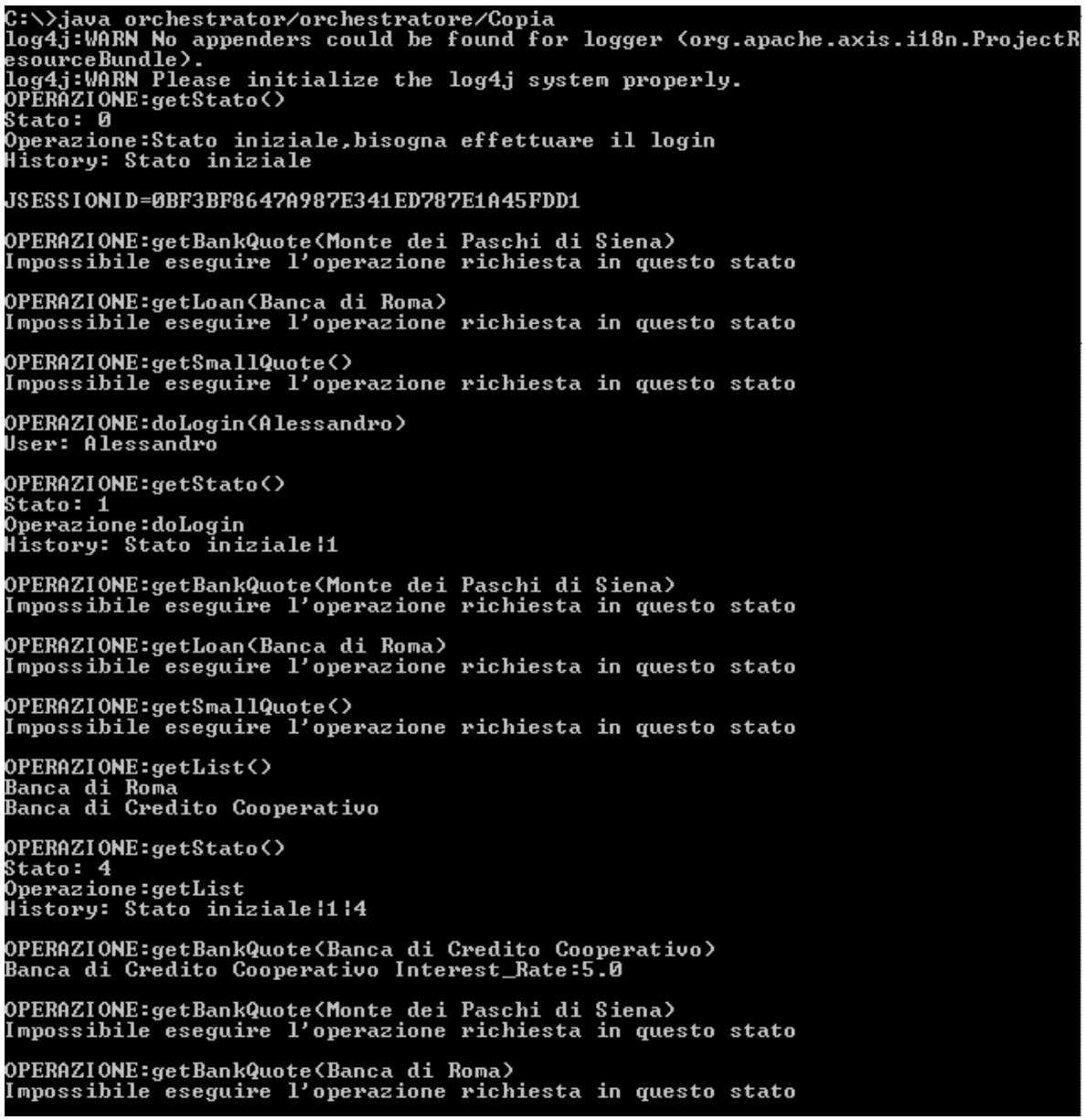
Client.java

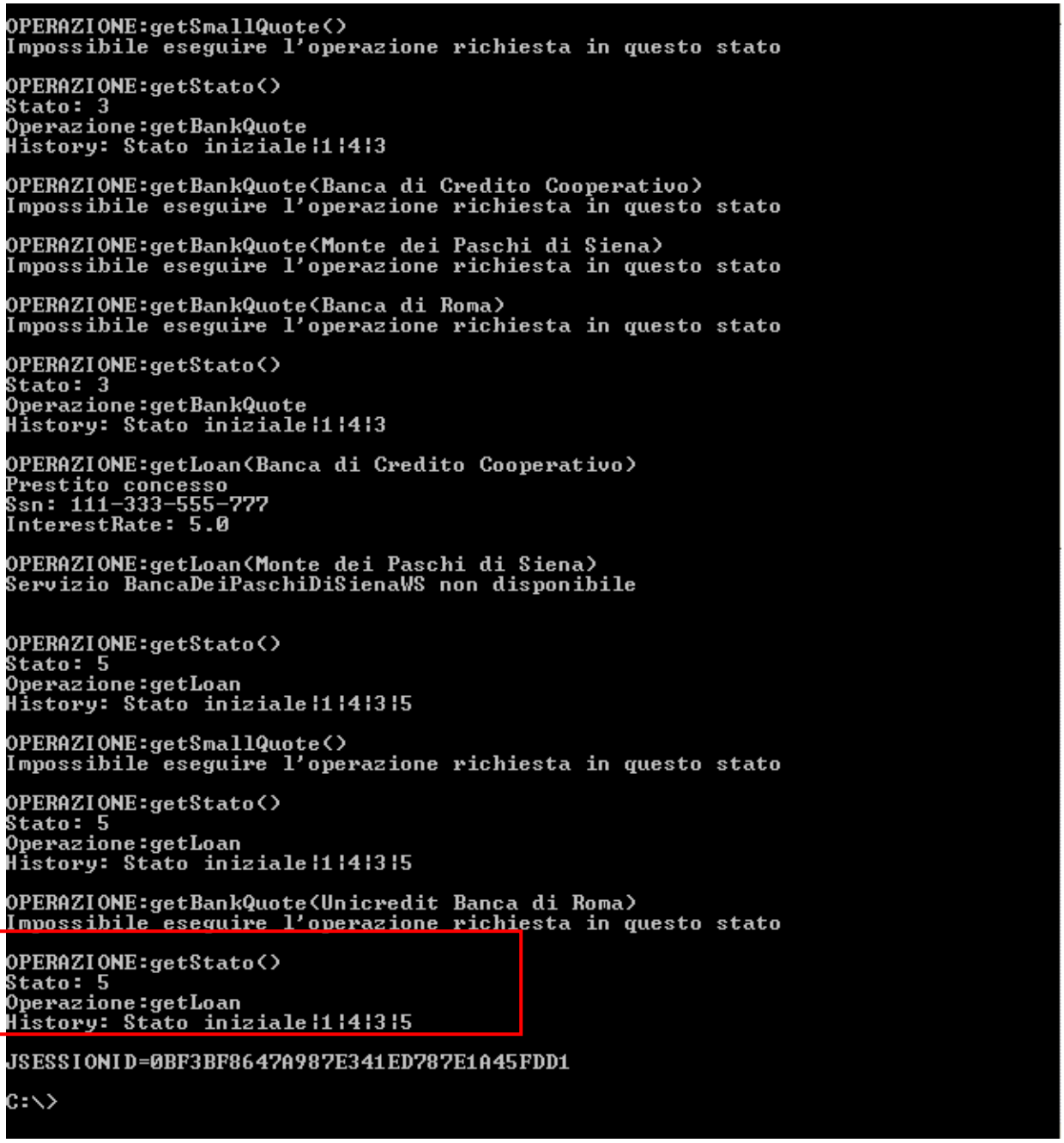
```
package orchestrator.orchestratore;
import java.net.*;
import java.rmi.*;
import java.util.*;
import javax.xml.namespace.*;
import javax.xml.rpc.* ;
import java.util.* ;
```

```
import org.apache.axis.transport.http.HTTPConstants ;
import org.apache.axis.client.Call;

public class Client {
public static void main(String[] args)throws Exception{
Vector list=new Vector();

//Stub
OrchestratoreWSService service = new OrchestratoreWSServiceLocator();
OrchestratoreWS stub = service.getOrchestratoreWS();
//La seguente linea di codice è necessaria per abilitare il meccanismo delle
//sessioni lato client:in questo modo quando si riceve il primo messaggio di
//risposta del servizio contenente il cookie questo viene automaticamente
//estratto dal messaggio ricevuto ed incuso nelle successive invocazioni.
((OrchestratoreWSSoapBindingStub)stub).setMaintainSession(true);
System.out.println("OPERAZIONE:getStato()");
System.out.println(stub.getStato());
//Ottengo il cookie associato all'istanza del servizio che si sta utilizzando
Call authCall = ((OrchestratoreWSServiceLocator)service).getCall();
org.apache.axis.MessageContext msgContext = authCall.getMessageContext();
String cookie = msgContext.getStrProp(HTTPConstants.HEADER_COOKIE);
System.out.println(cookie);
System.out.println("");
System.out.println("OPERAZIONE:getBankQuote(Monte dei Paschi di Siena)");
System.out.println((String)stub.getBankQuote("Monte dei Paschi di Siena"));
System.out.println("OPERAZIONE:getLoan(Banca di Roma)");
System.out.println((String)stub.getBankQuote("Banca di Roma"));
System.out.println("OPERAZIONE:getSmallQuote()");
System.out.println((String)stub.getSmallQuote());
System.out.println("OPERAZIONE:doLogin(Alessandro)");
System.out.println(stub.doLogin("Alessandro"));
System.out.println("OPERAZIONE:getStato()");
System.out.println(stub.getStato());
System.out.println("OPERAZIONE:getBankQuote(Monte dei Paschi di Siena)");
System.out.println((String)stub.getBankQuote("Monte dei Paschi di Siena"));
System.out.println("OPERAZIONE:getLoan(Banca di Roma)");
System.out.println((String)stub.getBankQuote("Banca di Roma"));
System.out.println("OPERAZIONE:getSmallQuote()");
System.out.println((String)stub.getSmallQuote());
System.out.println("OPERAZIONE:getList()");
Vector a=(Vector)stub.getList();
Iterator iter=a.iterator();
while(iter.hasNext()){
String banca=(String)iter.next();
System.out.println(banca);
}
System.out.println("");
System.out.println("OPERAZIONE:getStato()");
System.out.println(stub.getStato());
System.out.println("OPERAZIONE:getBankQuote(Banca di Credito Cooperativo)");
System.out.println((String)stub.getBankQuote("Banca di Credito Cooperativo"));
System.out.println("OPERAZIONE:getBankQuote(Monte dei Paschi di Siena)");
System.out.println((String)stub.getBankQuote("Monte dei Paschi di Siena"));
System.out.println("OPERAZIONE:getBankQuote(Banca di Roma)");
System.out.println((String)stub.getBankQuote("Banca di Roma"));
System.out.println("OPERAZIONE:getSmallQuote()");
System.out.println((String)stub.getSmallQuote());
System.out.println("OPERAZIONE:getStato()");
System.out.println(stub.getStato());
System.out.println("OPERAZIONE:getBankQuote(Banca di Credito Cooperativo)");
System.out.println((String)stub.getBankQuote("Banca di Credito Cooperativo"));
System.out.println("OPERAZIONE:getBankQuote(Monte dei Paschi di Siena)");
System.out.println((String)stub.getBankQuote("Monte dei Paschi di Siena"));
```

```
System.out.println("OPERAZIONE:getBankQuote(Banca di Roma)");
System.out.println((String)stub.getBankQuote("Banca di Roma"));
System.out.println("OPERAZIONE:getStato()");
System.out.println(stub.getStato());
System.out.println("OPERAZIONE:getLoan(Banca di Credito Cooperativo)");
System.out.println((String)stub.getLoan("Banca di Credito Cooperativo"));
System.out.println("");
System.out.println("OPERAZIONE:getLoan(Monte dei Paschi di Siena)");
System.out.println((String)stub.getLoan("Monte dei Paschi di Siena"));
System.out.println("");
System.out.println("OPERAZIONE:getStato()");
System.out.println(stub.getStato());
System.out.println("OPERAZIONE:getSmallQuote()");
System.out.println((String)stub.getSmallQuote());
System.out.println("OPERAZIONE:getStato()");
System.out.println(stub.getStato());
System.out.println("OPERAZIONE:getBankQuote(Unicredit Banca di Roma)");
System.out.println((String)stub.getBankQuote("Unicredit Banca di Roma"));
System.out.println("OPERAZIONE:getStato()");
System.out.println(stub.getStato());
authCall = ((OrchestratoreWSServiceLocator)service).getCall();
msgContext = authCall.getMessageContext();
cookie = msgContext.getStrProp(HTTPConstants.HEADER_COOKIE);
System.out.println(cookie);
 }
}
```

L'output generato è il seguente:


C:\>java orchestrator/orchestratore/Copia
log4j:WARN No appenders could be found for logger (org.apache.axis.i18n.ProjectR
esourceBundle).
log4j:WARN Please initialize the log4j system properly.
OPERAZIONE:getStato()
Stato: 0
Operazione:Stato iniziale,bisogna effettuare il login
History: Stato iniziale
JSESSIONID=0BF3BF8647A987E341ED787E1A45FDD1
OPERAZIONE:getBankQuote(Monte dei Paschi di Siena)
Impossibile eseguire l'operazione richiesta in questo stato
OPERAZIONE:getLoan(Banca di Roma)
Impossibile eseguire l'operazione richiesta in questo stato
OPERAZIONE:getSmallQuote()
Impossibile eseguire l'operazione richiesta in questo stato
OPERAZIONE:doLogin(Alessandro)
User: Alessandro
OPERAZIONE:getStato()
Stato: 1
Operazione:doLogin
History: Stato iniziale¦1
OPERAZIONE:getBankQuote(Monte dei Paschi di Siena)
Impossibile eseguire l'operazione richiesta in questo stato
OPERAZIONE:getLoan(Banca di Roma)
Impossibile eseguire l'operazione richiesta in questo stato
OPERAZIONE:getSmallQuote()
Impossibile eseguire l'operazione richiesta in questo stato
OPERAZIONE:getList()
Banca di Roma
Banca di Credito Cooperativo
OPERAZIONE:getStato()
Stato: 4
Operazione:getList
History: Stato iniziale¦1¦4
OPERAZIONE:getBankQuote(Banca di Credito Cooperativo)
Banca di Credito Cooperativo Interest_Rate:5.0
OPERAZIONE:getBankQuote(Monte dei Paschi di Siena)
Impossibile eseguire l'operazione richiesta in questo stato
OPERAZIONE:getBankQuote(Banca di Roma)
Impossibile eseguire l'operazione richiesta in questo stato



Dall'analisi dell'output generato è possible vedere come sia stata rispettata la rappresentazione TS del servizio nelle successive invocazioni del client. L'ultimo metodo invocato `getStato()` indica che la conversazione si trova allo stato finale `(Stato:5)`,che corrisponde appunto all'operazione `getLoan` e che la sequenza delle azioni correttamente eseguite `(history)` è coerente con quanto stabilito nel relativo TS e corrisponde ad un cammino ammissibile dallo stato iniziale a quello finale.

Le stampe iniziali e finali del cookie dimostrano che la sessione è stata mantenuta durante tutta la comunicazione. Il servizio ha creato una sua nuova istanza solo la prima volta che è stato invocato dal client;nel primo messaggio di richiesta non era infatti contenuto alcun cookie:il servzio ne ha generato uno,lo ha univocamente associato al client e lo ha utilizzato nelle successive invocazioni per restituire al client la stessa istanza ed evitare di crearne una nuova e mantenere di conseguenza lo stato della conversazione.

Capitolo 3 – Apache Axis2

In questo capitolo analizzeremo i motivi che hanno portato alla creazione del nuovo engine Web Service Axis2,la nuova generazione dell'Apache Web Service stack,illustrandone le futures,l'architettura,l'installazione e l'esecuzione. Mostreremo poi come effettuare il deploy di un servizio simile a quello presentato nel capitolo precedente analizzando le differenze presenti in questa fase tra Axis ed Axis2.

3.1 Apache Web Service Stack

La storia dei Web Services è passata attraverso una serie di iterazioni nel corso della sua evoluzione. La prima generazione di Web Services consisteva in iterazioni altamente controllate e poteva essere considerata un semplice test di fattibilità. Apache SOAP è stato uno dei più noti SOAP engine di prima generazione. Era pensato per essere un "proof of concept"e non a tutti interessavano le prestazioni che offriva. L'idea che era alla base della prima generazione dei SOAP engine era quello di convincere il mondo che i Web Services erano una possibile opzione. Ben presto i primi SOAP engine iniziarono a produrre gli aspettati risultati. Numerose aziende mostrarono interesse verso le architetture SOA ed iniziarono a costruire software di conseguenza. Questa fase può essere interpretata come la seconda generazione di Web Services che richiede però SOAP engine migliori e più veloci. Aspetti come la scoperta e la definizione di un servizio sono già standardizzati,e al SOAP engine è quindi richiesto di supportare anche questi standard. Apache Axis può essere considerato un SOAP engine per servizi di seconda generazione. Ora, anche la seconda generazione di Web Services sta volgendo al termine. I servizi stanno diventando sempre più complicati e complessi e viene richiesta continuamente la definizione di nuovi standard che devono convivere con quelli esistenti. La terza generazione di Web Services richiede SOAP engine sempre più veloci e robusti;l'attuale implementazione di Axis 1 non rappresenta la scelta migliore e di conseguenza è nato il nuovo Axis 2.

3.2 Nascita di Axis2

Come discusso precedentemente,i Web Services sono in rapida crescita e un gran numero di organizzazioni si stanno muovendo verso questo tipo di tecnologia. Di conseguenza si riscontrano continuamente nuove esigenze che portano alla definizione di nuovi standard. In aggiunta ai requisiti di affidabilità, sicurezza, prestazioni sono state definite nuove specifiche WS* e Web Service engine necessari a sostenerle. Se consideriamo lo stack di Apache Web Service,Axis 1 rappresenta uno stabile engine Web Service e molte organizzazioni ne fanno uso. Quindi,cambiando l'architettura di Axis 1 per supportare nuovi requisiti e standard Web Services non è una buona idea. Ogni software ha un proprio ciclo di vita;è possibile evolverlo ma ad un certo punto sarà necessaria una rivoluzione. La stessa teoria è stata applicata ad Axis 1. Piuttosto che modificare l'architettura di Axis,il team di sviluppo di Apache Web Service ha preferito implementare una nuova architettura. In aggiunta ai nuovi requisiti e specifiche WS*,le performance erano un'altra area alla quale bisognava fare maggiore attenzione. Modificare l'architettura di Axis 1 per aumentare la performance non era affatto semplice. Axis 1 utilizza DOM come meccanismo di rappresentazione XML. Di conseguenza per processare un messaggio bisognava prima caricarlo in memoria e questo causava un rallentamento del sistema ed un incremento della memoria utilizzata. Uno dei motivi che ha portato all'introduzione di Axis 2 è proprio quello di migliorare le performance del sistema. La comunità di Apache Web Service ha quindi deciso di introdurre un nuovo engine Web Service chiamato Axis 2 che soddisfa un certo numero di nuovi requisiti. Axis 2 presenta un'architettura flessibile e facilmente estendibile in grado di supportare standard WS*e rappresenta l'*Apache third-generation Web Service engine*.

3.3 Installazione di Axis2

Apache Axis2 ha un certo numero di release e la 1.3 può essere considerata una delle più stabili e robuste. Una delle caratteristiche dei progetti open-source è che le relative release includono il source-cose,che viene utilizzato per la creazione di file binari,la compilazione e il linking. In più release Axis2 include la distribuzione source code in aggiunta alla distribuzione binaria. E' possibile effettuare il download dell'ultima release di Axis2 dal sito [17].

Ogni release di Axis2 è composta da quattro principali distribuzioni:

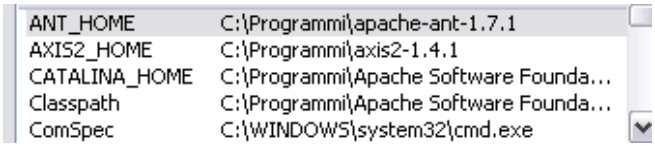
- Binary distribution
- WAR distribution
- Source distribution
- JAR distribution

In questo contesto ho utilizzato la WAR distribution versione 1.4.1. La distribuzione WAR di Axis2 è stata infatti pensata appositamente per deployare Axis2 in application server come Tomcat,JBoss,Weblogic ed altri. E'possibile deployare il file WAR di Axis2 in un application server e verificarne la corretta installazione accedendo da un browser al relativo indirizzo. Nel nostro caso ho effettuato il deploy del file Axis2.war in Apache Tomcat verificandone il corretto funzionamento digitando l'indirizzo http://localhost:8081/Axis2. Per installare la WAR distribution basta seguire i seguenti passi:

Step 1: Intallare un'application server;nel mio caso ho continuato ad usare Apache Tomcat.
Step 2: Copiare il file WAR nella relativa directory Webapps di Tomcat;da notare che la locazione all'interno della quale effettuare il deploy del file WAR dipende comunque dall'application server utilizzato.
Step 3: Verificare la corretta installazione digitando l'inidirzzo http://localhost:8081/axis2. Se si è svolto tutto correttamente è possibile vedere la web application homepage di Axis2;naturalmente l'URL dipende dall'application server utilizzato.

E'inoltre utilie creare le due variabili d'ambiente

```
- ANT_HOME        C:\Programmi\apache-ant-1.7.1
- AXIS2_HOME     C:\Programmi\axis2-1.4.1
```



E aggiungere al `Path` di sistema:

```
- C:\Programmi\apache-ant-1.7.1;
- C:\Programmi\axis2-1.4.1;
```

Da notare che al path ho aggiunto anche la directoy di Ant:ho infatti utilizzato questo programma per la compilazione dei file sia lato cliente che lato server.
Apache Ant è infatti un software per l'automazione del processo di build,è un progetto Apache, open source,ed è rilasciato sotto licenza Apache;è possibile effettuare il download di `ant` da[18]. Per utilizzate il programma è sufficiente scompattare l'archivio in una directory e settare le variabili d'ambiente come mostrato precedentemente.

3.4 Differenze tra Axis2 ed Axis1.x

Vediamo quali sono le differenze più evidenti tra Axis2 ed Axis1.x sotto i seguenti punti di vista:

- Architettura
- Client API
- Deployment di servizi,handler e moduli
- Protocolli di trasporto
- Data Binding
- SOAP

Architettura

Axis2 ed Axis1.x si sviluppano a partire da architetture differenti.

- **Velocità** – Axis2 si basa sulle API StAX,che offrono maggiore velocità rispetto al parsing SAX basato sugli eventi utilizzato in Axis1.x.
- **Stabilità** – Axis2 definisce i concetto di *phase* e di *phase rule* che permettono di avere più stabilità e flessibilità rispetto ad Axis1.x.

- **Transport Framework** – I protocolli di trasporto(i.e.,sender e listener per SOAP su vari tipi di protocollo come HTTP,SMTP,etc.) sono stati astratti dall'Axis2 engine. Avere un Axis2 engine transport-independent consente di avere più flessibilità nelle opzioni di trasporto.
- **WSDL 2.0 support** – Axis2 supporta sia la versione WSDL 1.0 che la 2.0,che sono utilizzate dal tool code generation per creare gli stub e gli skeleton del servizio.
- **Component-oriented architecture** – I componenti di Axis2 consistono in una serie di handler e moduli contenuti in archivi .aar e .mar. Questi componenti sono facilmente riutilizzabili e consentono di estendere le funzionalità di una determinata applicazione.

Client API

Axis2 supporta l'invocazione asincrona di un servizio attraverso la sendReceiveNonBlocking();le invocazione sincrone/asincrone possono inoltre gestire sia singole che doppie connessioni HTTP. Questi vantaggi architetturali permettono ad Axis2 di gestire molte più request e response di quante ne possa supportare Axis1.x.

Deployment di servizi,handler e moduli

In Axis2.x il deployment di un servizio viene effettuato utilizzando il file `.wsdd`. Il deployment in Axis2 utilizza invece il file `services.xml` ed è reso più semplice e veloce grazie alle diverse modalità che si hanno a disposizione(si veda paragrafo 3.5.1-Deployment Model). In Axis2 si passa dal concetto di Handler al concetto di Modulo;un modulo può essere definito come una collezione di handler ed è un file con ha estensione .mar. Il modulo utilizza il file module.xml per specificare la configurazione e l'attivazione degli handler. Quando un servizio è chiamato attraverso un handler bisogna semplicemente passare il riferimento al modulo che lo implementa nel file services.xml. Per capire meglio quanto detto precedentemente vediamo quali differenze ci sono se si vuole attivare il modulo relativo a SOAPMonitor nelle due implementazioni di Axis.
SOAPMonitor è una combinazione di tre componenti:un'applet che visualizza i messaggi di richiesta e di risposta,una servlet che si lega alla porta di default 5001 e si connette all'applet ed una catena di handler utilizzata per intercettare i messaggi.

- In Axis1.x si deve effettuare il deployment di SOAPMonitor attraverso il relativo file .wsdd;bisogna poi inserire il riferimento all'handler soapmonitor:

```
<requestFlow>
  <handler type="soapmonitor"/>
</requestFlow>
<responseFlow>
  <handler type="soapmonitor"/>
</responseFlow>
```

all'interno del file .wsdd del servizio del quale si vogliono visualizzare i messaggi SOAP scambiati(si veda paragrafo 2.6.1).

- In Axis2 l'handler è definito dal file module.xml;tale file è contenuto all'interno del file jar soapmonitor-1.41 con estensione .mar da copiare nella cartella WEB-INF/modules.
- In Axis1.x il SOAPMonitorHandler presenta la seguente signatura:
  `public class SOAPMonitorHandler extends BasicHandler`
- In Axis2 invece la signatura è la seguente:
  `public class SOAPMonitorHandler extends AbstractHandler`
- In Axis2 bisogna effettuare il riferimento al modulo che contiene la catena di handler da utilizzare all'interno del file services.xml.
- Infine Axis2 richiede di inserire il modulo globale `<module ref="soapmonitor"/>` nel file axis2.xml e di definire l'ordine delle fasi per il SOAPMonitorPhase referenziato nel file module.xml.

Per maggiori dettagli si veda[28].

Data Binding

Come Axis1.x anche Axis2 fornisce strumenti automatici che partono dal file WSDL e creano delle classi java stub e skeleton,a cui si appoggieranno client e server (lo strumento anche in questo caso è chiamato WSDL2Java). Al momento della creazione Axis2 permette di decidere quale parser utilizzare di solito a scelta tra ADB, Xmlbeans, Jibx e JAXB-RI. Questi parser trasformano gli elementi XML dichiarati nel file WSDL in oggetti java. Il parser di default è ADB.
Axis1.x utilizza invece un unico di databinding di sistema basato su SAX ed è un opzione non configurabile da parte dell'utente. Anche per questo motivo a volte vengono restituiti messaggi del tipo"`no deserialized found for [qname]`".Axis fornisce infatti dei built-in de/serializers per i java bean e gli array,ed è compito dell'utente implementare opportuni serializzatori/de-serializzatori per rappresentare altri tipi di dato. Bisogna inoltre specificare le classi che implementano i serializzatori/de-serializzatori utilizzati e il mapping tra i qname e i tipi nel file WSDL attraverso le opzioni beanMapping e/o typeMapping.

SOAP

Axis1.x ed Axis2 gestiscono in modo diverso lo stack SOAP. Il modo migliore per comprendere a pieno come Axis2 consenta una migliore e più completa gestione del protocollo SOAP è quello di seguire la Guida dell'Utente[31] e la Guida all' Architettura[30] di Axis2.

3.5 Architettura di Axis2

L'architettura di Axis2 consiste in una serie di moduli core e non-core[25]. Il core engine può essere visto come un pure SOAP monitor processing engine. Ogni messaggio in ingresso al sistema deve essere trasformato in un messaggio SOAP prima di essere passato al core engine. Un messaggio in ingresso comunque può essere sia un SOAP message che un non-SOAP message(REST JSON o JMX). Ma a livello di trasporto questi vengono convertiti in un messaggio SOAP. L'architettura di Axis2 è stata progettata in base a delle regole ben definite. Queste regole sono utilizzate per ottenere un *SOAP processing engine* flessibile ed estendebile:

- Separazione della logica dallo stato per fornire un meccanismo di processamento senza stato(questo perché i Web Services di default sono stateless);il codice che fa il processamento dentro AXIS2 è senza stato. Il codice è eseguito attraverso liberi threads paralleli.
- Tutte le informazioni sono mantenute in un unico information model per abilitare il sistema al suspend e al resume.
- Capacità di estendere il supporto alle nuove specifiche Web Service effettuando solo dei minimi cambiamenti al core architecture.

Progettato partendo dall'esperienza maturata dal progetto Axis 1.0, Apache Axis2 fornisce un Object Model completo e un'architettura modulare che permette di aggiungere facilmente funzionalità e supporto a nuove specifiche collegate ai Web Service. Axis2 permette di:

- Inviare messaggi SOAP
- Ricevere ed elaborare messaggi SOAP
- Creare Web Service partendo da POJO (Plain Old Java Object)
- Creare le classi che implementano sia il lato client sia quello server partendo dal WSDL
- Recuperare facilmente il WSDL di un servizio.
- Asincronia: Supporta la chiamata non bloccante dei client.
- Velocità: attraverso il suo Object Model e lo StAX (Streaming API for XML) Parser, questa è aumentata significativamente.
- AXIOM: è l'Object Model creato appositamente per processare messaggi; estensibile e performante.
- Hot Deployment: è possibile pubblicare servizi ed handler mentre il sistema gira.
- MEP (Message Exchacge Pattern) Support.
- Inviare e ricevere messaggi SOAP con Attachments.
- Creare e utilizzare servizi basati su REST (REpresentational State Transfer)
- Creare o utilizzare servizi che sfruttano specifiche quali WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Coordination, e WS-Atomic Transaction .

- Framework di trasporto: E' possibile spedire e ricevere messaggi SOAP sopra diversi protocolli;il cuore dell'engine è completamente indipendente da questo.
- WSDL Support: Supporta il Web Service Description Language che permette di costruire automaticamente stub per richiedere il servizio ed esportare la descrizione di questo per farsi conoscere.
- Componibilità ed Estendibilità: Attraverso i moduli è possibile aggiungere nuove WS-* (specifiche) che però non saranno "hot deployable".
- Usare l'architettura modulare di Axis2 per aggiungere il supporto a nuove specifiche quando rilasciate.

La figura seguente mostra i componenti principali dell'architettura di Axis2(componenti core e non-core).

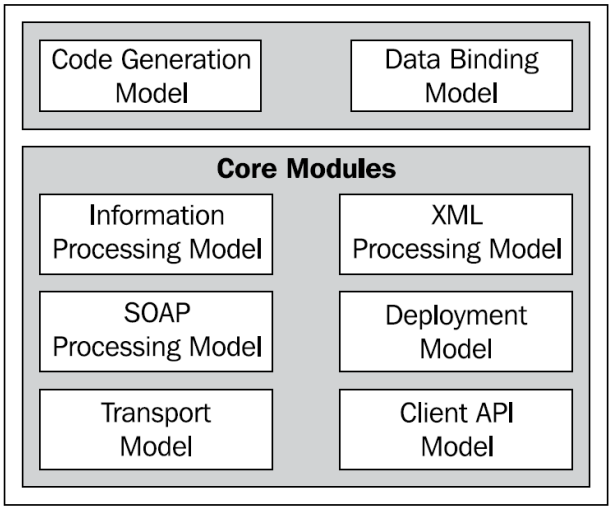


Figura 22:Moduli core e non-core di Axis2

3.5.1 Moduli core

XML Processing Model

Permette di gestire i messaggi SOAP. Questo viene fatto attraverso il nuovo Object Model AXIOM. Infatti è qui che vengono stabilite le performance del sistema. Da qui si sviluppa un sottoprogetto di AXIS2 chè è quello di AXIOM (Axis Object Model) per fornire della API per SOAP e le sue informazioni in formato XML. AXIOM è differente dagli altri standard OM (SAX, DOM, JDOM..), principalmente per la possibilità di aumentare la struttura durante la ricezione dell'XML. AXIOM dipende direttamente da StAX per input e output.

SOAP Processing Model

Questo modello riguarda il processamento degli incoming SOAP message. L'invio e la ricezione di messaggi SOAP possono essere considerate le due chiavi di lavoro del SOAP-processing engine. L'architettura di Axis2 prevede due Pipes (Flows) per gestire le fasi di send e di receive. L'AxisEngine definisce due metodi,send() e receive() per implementare queste due Pipes. Le due pipes sono nominate In*Flow* e Out*Flow*. I complex Message Exchange Patterns (MEPs) sono costruiti combinando questi due tipi di pipes.

Un Message Exchange Pattern(MEP) è un template,sprovvisto di applicazione sematica,che descrive un pattern generico per lo scambio di messaggio fra enti. Questo descrive relazioni (temporali, condizionate, sequenziali) di scambio di messaggi multipli.

Dal punto di vista di Axis2, l'architettura per la gestione dello scambio di messaggi è questa:

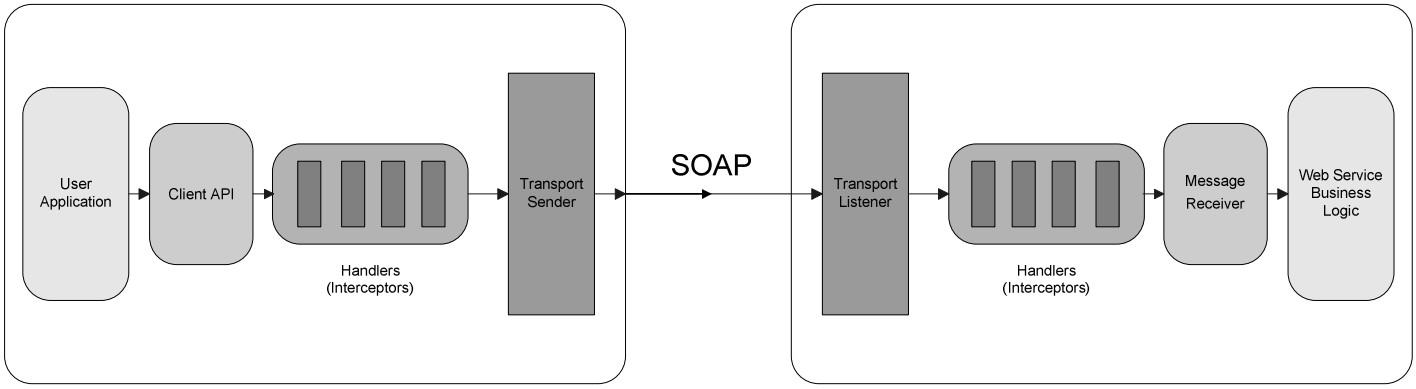


Figura 23:Architettura di Axis2 per lo scambio di messaggi

Supponendo che su ogni lato (client e server) sia in esecuzione Axis2,il processo per lo scambio di messaggi può essere sintetizzato in questo modo:

- Il mittente crea un messaggio SOAP con Axiom.
- Axis2 mantiene il concetto della catena di handler già presente in Axis:gli handlers in Axis2 si occupano di tutte le azioni necessarie al messaggio,come crittografarlo in conformità con WS-Security.
- Il Transport sender invia il messaggio.
- Sul lato ricevente,il Transport listener riceve il messaggio.
- Il Transport listener inoltra il messaggio alla catena di *handlers* definite per il lato server.
- Una volta eseguite le operazioni della fase di *pre-dispatch* il messaggio viene passato ai dispatcher che lo inoltreranno alla giusta applicazione.

In Axis2 tutte queste azioni sono raggruppate in *phases*,molte delle quali pre-definite nell'architettura,come la fase *pre-dispatch,dispatch,message processing.* Ogni fase è una collezione di *handlers*. Axis2 permette di controllare quali handlers si inseriscono in ogni fase e il loro ordine di esecuzione all'interno di una fase. E' anche possibile definire delle fasi e degli handlers aggiuntivi. Uno o più handler compongono un "modulo" che può essere inserito in Axis2. Il concetto di modulo rappresenta il cuore dell'espandibilità di Axis2. Ognuno di questi moduli svolge un compito,come ad esempio Sandesha2 che implementa il WS-ReliableMessaging o Rampart2 che implementa il WS-Security e possono essere inseriti all'interno della catena di handlers in modo da gestire i messaggi e aggiungere le funzionalità implementate in maniera totalmente trasparente all'utente. Inoltre,grazie al modello di deployment di Axis2,è possibile pubblicare un modulo ed 'agganciarlo' a uno o più servizi senza dover fermare Axis2.

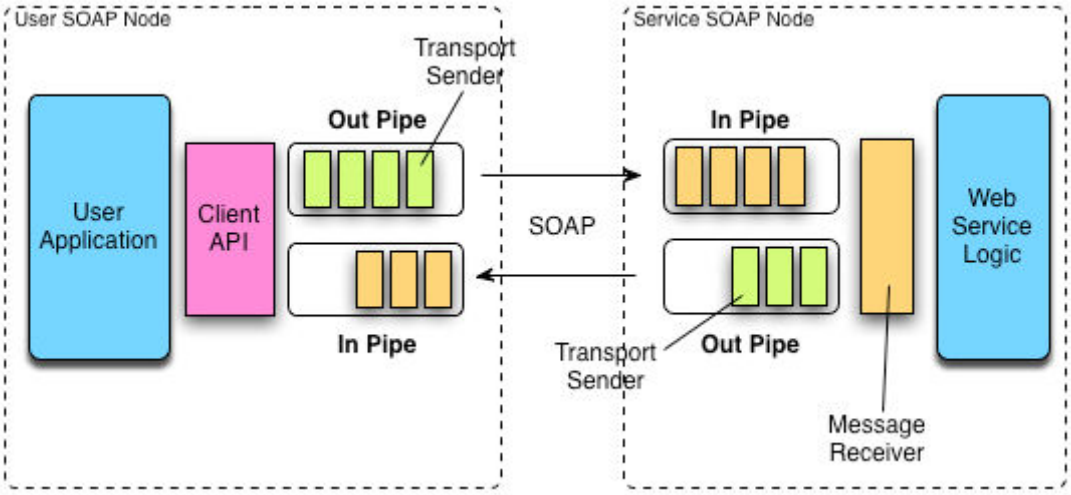


Figura 24:Message processing in Axis2

L'estensibilità del SOAP processing model è garantita dagli handlers. Quando un messaggio viene processato ogni handler che fa parte della catena viene messo in esecuzione.
Gli handler sono come intercettori del messaggio e processano parti di esso (solitamente l'header) per fornire servizi aggiuntivi. Solitamente il percorso effettuato da un messaggio consiste nell'uscire dalla OutPipe del client ed entrare nella InPipe del server ma può capitare al termine di questa la creazione di un nuovo messaggio di risposta da spedire. Si avrà allora un OutPipe lato server e InPipe lato client.

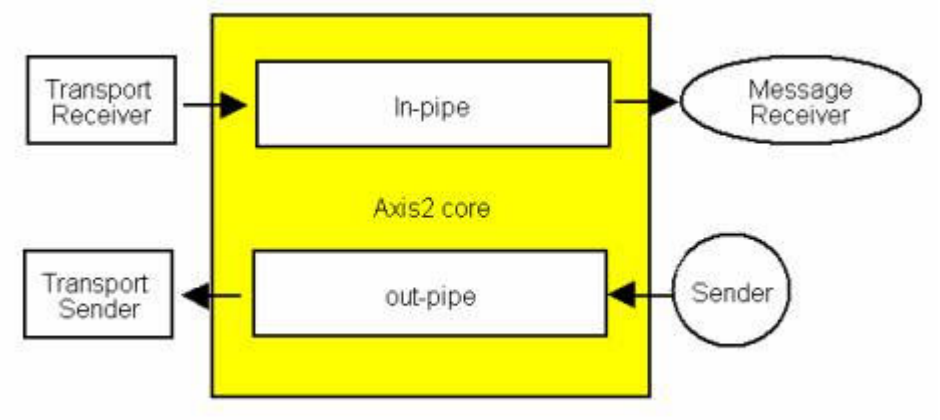


Figura 25:In-Pipe ed Out-Pipe

Un messaggio all'interno delle pipe attraversa diverse fasi. Gli handler sono in esecuzioni all'interno delle fasi che prevedono inoltre un meccanismo per specificare l'ordine di esecuzione degli handlers.

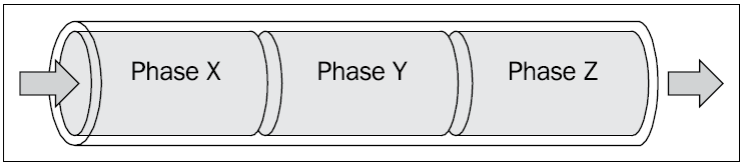


Figura 26:Un flusso e le sue fasi

Gli handler definiti di default da Axis sono i seguenti:

- Dispatchers: cerca il servizio e l'operazione a cui è diretto il messaggio;
- Message Receiver: posizionato alla fine della InPipe questo handler consuma il messaggio;
- Transport Sender: alla fine della OutPipe spedisce il messaggio SOAP;
- Transport Receiver: riceve i messaggi in arrivo dal mittente.

SOAP Processing Model:InPipe(Incoming Message Phases)

Una volta che il Transport Receiver riceve il messaggio viene creato un Message Context per esso. La InPipe viene quindi eseguita con questo contesto. Le fasi dei messaggi in arrivo sono:
– Transport Phase
– Pre-Dispatch Phase
– Dispatch phase
– User phase
– Message validation Phase
– Message Processing Phase

Nella fase di *Transport* gli handler processano informazioni specifiche al trasporto come la validazione di messaggi in entrata osservando gli header dei messaggi, aggiungendo dati nel Message Context.

Nella fase di *Pre-Dispatch* gli handler dovranno riempire il Message Context per la fase successiva. Un esempio di ciò è l'estrarre informazioni dall'header riguardanti l'addressing.

Durante la fase di *Dispatch* vengono ricercati i servizi da mettere in esecuzione data la richiesta del messaggio. La post-condizione di questa fase, permette di lanciare un errore nel caso il servizio non sia stato trovato.

Nell'*User phase* l'utente ha la possibilità di inserire i propri handler. In realtà dalla nuova versione 1.0 di Axis2 l'utente può scegliere di inserire i propri handler in qualsiasi fase, così da poter mettere mano anche a processi quali l'addressing. Questa è stata anche una nostra necessità.

Nella fase di *validazione* del messaggio viene assicurata la corretta esecuzione della fase successiva.

La *Business Logic* del messaggio SOAP è eseguita qui. Ogni operazione associata al messaggio viene messa in esecuzione dal Message Receiver associato.

SOAP Processing Model:OutPipe(Outgoing Message Phases)

La OutPipe è più semplice perchè quando viene messa in esecuzione già i servizi e le operazioni definiti nei messaggi sono conosciuti. Le fasi dei messaggi in uscita sono le seguenti:
– Message Initializate Phase
– User Phases
– Transport phases

La *prima fase* servirà per piazzare gli handler "custom".

La *seconda fase* sarà quella dove l'utente può inserire i propri handler.

La *fase di trasporto* esegue tutti gli hanlder inerenti alla configurazione del trasporto. L'ultimo handler è il Transport Senders che spedisce il messaggio SOAP all'endpoint.

SOAP Processing Model:Handlers

Una caratteristica importante di Axis2 è la possibilità di aggiungere nuovi handlers che dovranno essere inseriti in una specifica fase. Ogni handler deve essere inserito in un proprio modulo per essere configurato in modo da definire la posizione di esecuzione all'interno delle pipe.

SOAP Processing Model:Modules

Un modulo è un pacchetto che contiene handler e un descrittore di essi (regole per le fasi).
Un modulo può essere *available* o *engaged*. Quando un modulo è available i suoi handler non sono nella catena ma sono disponibili. Una volta che il modulo è engaged i moduli al ricevimento del messaggio vanno in esecuzione. (e.g.: WS-Addressing, SoapMonitor).

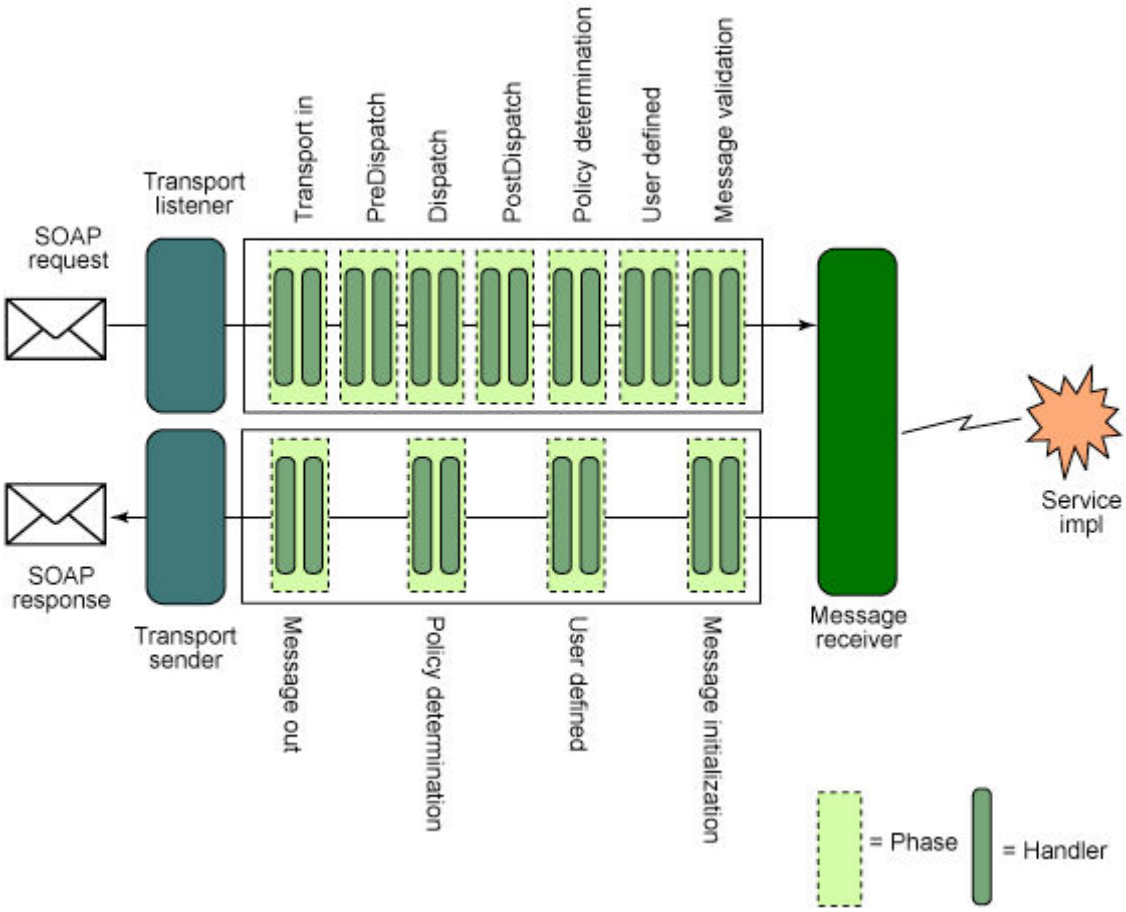


Figura 27: Fasi ed handler nel Message processing

Information Model

Serve per mantenere le informazioni dello stato. Questo modello contiene una gerarchia delle informazioni, più precisamente due, una detta gerarchia di Contesto e l'altra di Descrizione.
La prima rappresenta i dati statici. La seconda mantiene i dati inerenti alle istanze, i contesti dinamici.

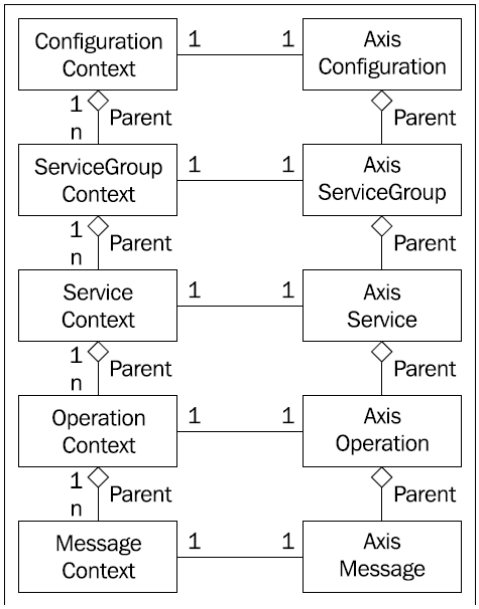


Figura 28:Gerarchia dell'Information Model

Deployment Model

Il deployment model semplifica il deploy dei servizi,constente di configurare il trasporto ed estendere il SOAP Processing Model. Introduce dei nuovi meccanismi di deployment per gestire hot deployment,hot updates, e J2EE-style deployment.

Esistono cinque modalità in Axis2 per creare e deployare un servizio:

- Pojo (Plain Old Java Object);
- Axiom (AXIs Object Model);
- Adb (Axis2 Data Binding);
- XMLBeans(Apache Open Source project);
- JiBX(an API framework for XML-Java data binding, not a JCP standard);

In base alla modalità scelta sono poi disponibili due diversi stili di programmazione:

- WSDL-driven development steps(appicabile ad ADB,XMLBeans e JiBX):
  - Definire il WSDL
  - Generare il Service Skeleton con il tool WSDL2Java
  - Definire il file service.xml
  - Generare l'Axis archive file (.aar) ed effettuare il deploy del servizio nel service container

- Code-driven development steps (applicabile a POJO ed AXIOM):
  - Scrivere l'implementazione del servizio
  - Generare il WSDL con il tool Java2WSDL
  - Definire il file service.xml
  - Geberare l'Axis archive file (.aar) ed effettuare il deploy del servizio nel service

Nel caso in cui si stia utilizzando la distribuzione WAR di Axis2 è possibile effettuare il deploy in due modi:

- Si copia manualmente il file .aar nel repository;
- Si esegue l'upload del servizio attraverso la Web Console.

Client API Model

Fornisce all'utente le API necessarie per utilizzare un servizio,interagire con diversi MEP(Message Exchange Pattern) fra cui in-out e in-only (utilizzabili per costruirne altri). Ci sono tre parametri che decidono la natura dell'interazione fra client e service:

- MEP
- Synchronous/ Asynchronous
- Transport

MEP

In-Only: In questo MEP un client invia un messaggio al server senza ricevere alcuna risposta.

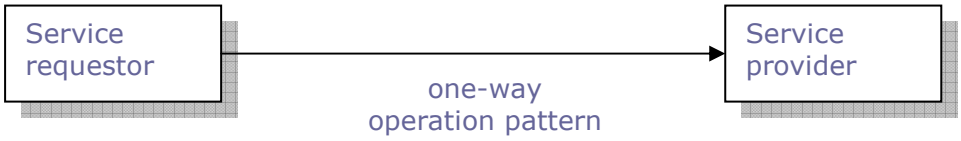


Figura 29:Invocazione in-only di un servizio

In-Out:In questo MEP il client invia un messaggio SOAP al server che lo processa e invia una risposta.

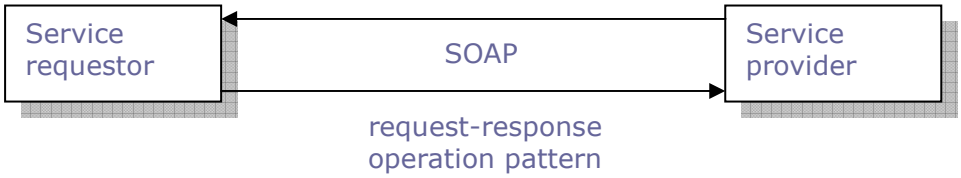


Figura 30:Invocazione in-out di un servizio

Synchronous/ Asynchronous

Blocking API: Quando un client invoca un servizio interrompe la sua esecuzione e rimane in attesa che l'operazione sia completata;il client riprende l'esecuzione solo quando ricevere la risposta o un messaggio di errore da parte del server.

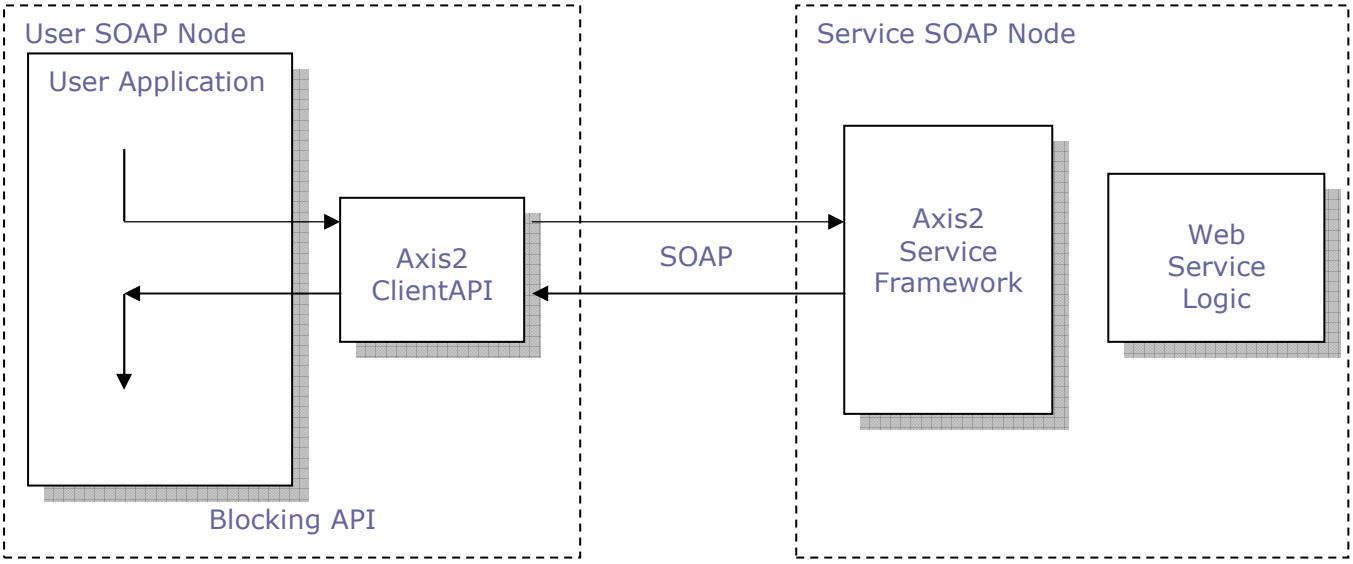


Figura 31:Invocazione Blocking di un servizio

Non-Blocking API: Questo tipo di API si basa sul concetto di callbak o polling. Quando un servizio viene invocato l'applicazione client riprende immediatamente il controllo e la risposta del servizio è gestita utilizzando un apposito oggetto callback. Questo approccio garantisce maggiore flessibilità alle applicazioni client che sono in grado di invocare più di un servizio contemporaneamente senza rimanere bloccati.

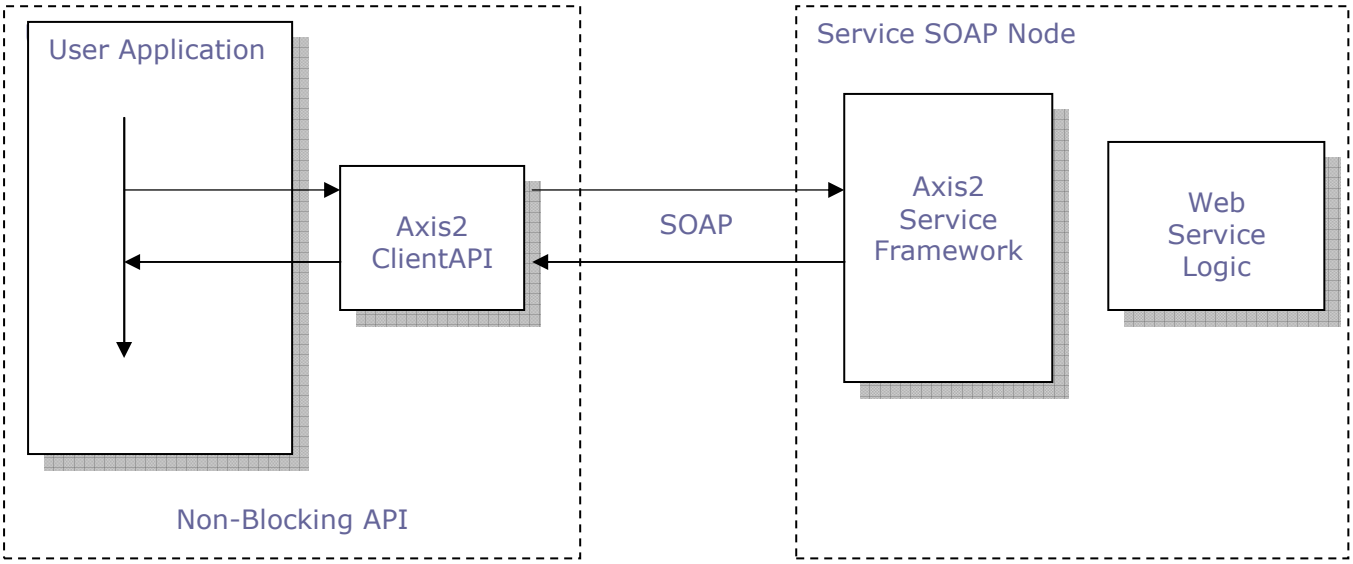


Figura 32:Invocazione Non-Blocking di un servizio

Transport

Entrambi i meccanismi,Blocking API e Non-Blocking API,utilizzano una singola connessione, un unico canale di trasporto per inviare la richiesta e ricevere la risposta e non sono adatti per gestire transazioni che richiedono un lungo tempo di esecuzione. Questo perché la connessione potrebbe terminare prima che arrivi la risposta:il timeout scade!In genere il timeout di default è impostato a 30s e se la risposta da parte del servizio non arriva entro questo arco di tempo la connessione termina e la risposta non viene ricevuta.

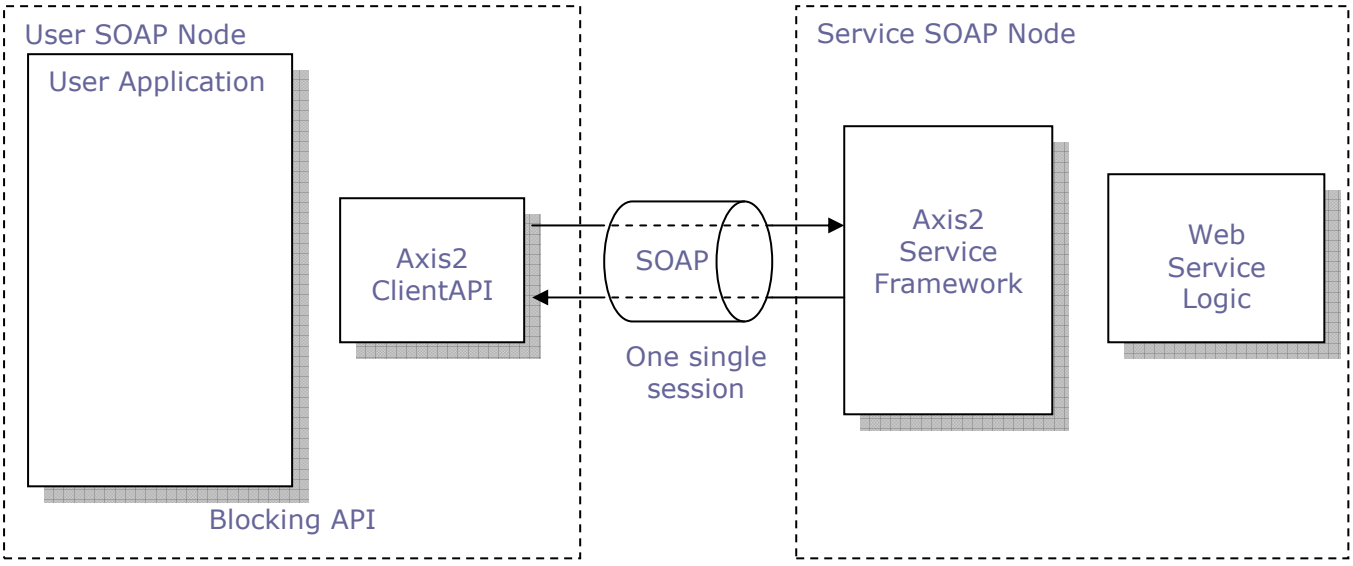


Figura 33:Singol Transport pattern

Una possible soluzione consiste nel creare due connessioni:una per il messaggio di richiesta e l'altra per il messaggio di risposta:in questo caso però richiesta e risposta devono essere correlate;Axis2 supporta WS-Addressing che fornisce una soluzione a questo problema utilizzando gli header <wsa:MessageID> e <wsa:RelatesTo>.

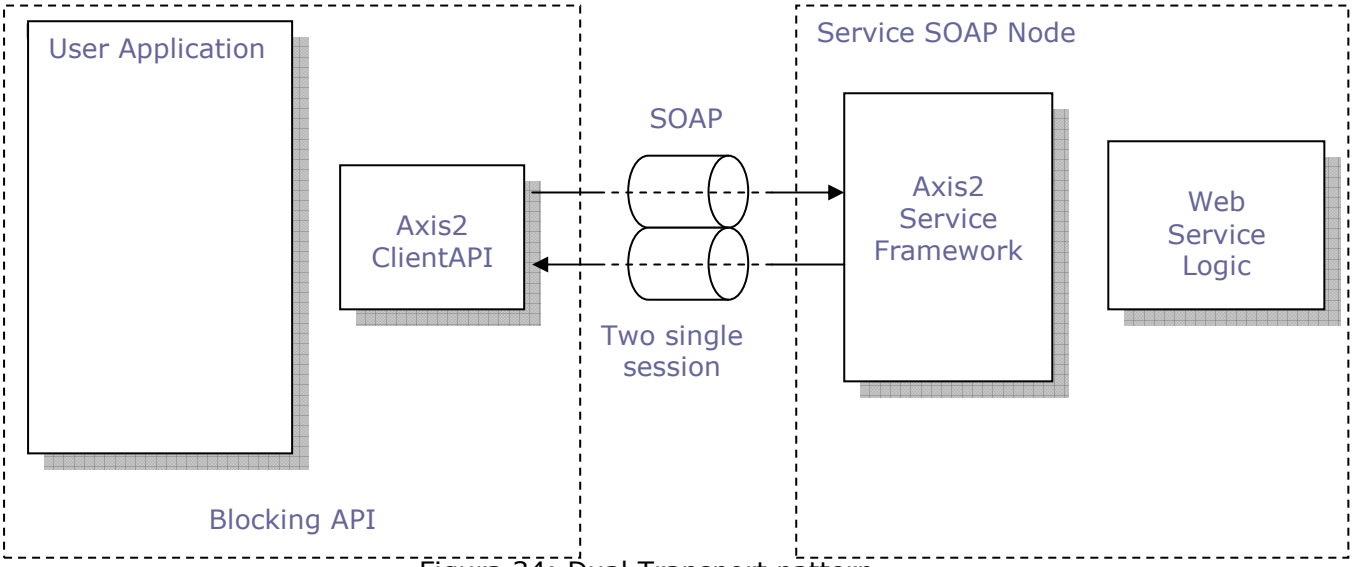


Figura 34: Dual Transport pattern

Combinando API Level Asynchrony e Transport Level Asynchrony è possible quindi ottenere differenti *invocation pattern* per un servizio e in base al livello di asincronia che si vuole ottenere si dovrà abilitare il modulo WS-Addressing:supportato ma di default non registrato.

Transports Model

Questo module contiene gli handler che curano l'interazione con il transport layer. Sono presenti due tipi di handler:TransportListener(TransportReceiver) e TransportSender. Il TransportListener riceve i messaggi SOAP dal transport layer e li passa all'interno della *InPipe* per processarli. Il TransportSender invia i messaggi SOAP ricevuti dall'*OutPipe* su uno specifico trasporto. Axis2 fornisce gli handlers per diversi protocolli. Per il trasporto HTTP il TransportReceiver è implementato server side da un *AxisServlet* e client side da un semplice HTTP server standalone(fornito sempre da Axis2).

In Axis2 è possibile spedire e ricevere messaggi SOAP utilizzando diversi tipi di protocollo:ognuno di essi è caratterizzato da due classi che implementano le interfacce TransportReceiver e TransportSender descritti precedentemente;il cuore dell'engine è completamente indipendente dal protocollo scelto e per questo motivo viene definito transport-independent.
Il trasporto utilizzato per la risposta è deciso attraverso le informazioni di addressing inviate dal mittente. Se il protocollo non è specificato,il server risponderà automaticamente sullo stesso utilizzato per il messaggio di richiesta. Il client è libero di utilizzare il trasporto che preferisce.
Come detto il trasporto può avvenire su diversi protocolli:
– HTTP: protocollo dual channel (two way) che mantiene traccia della sessione;
– TCP: è il più semplice, ma necessita di WS Addressing per essere funzionale;
– SMTP. il transport receiver è un thread che controlla la posta;
- JMS: protocollo di scambio di messaggi asincrono su Java.

Le classi TransportSender e TransportReceiver che caratterizzano ogni protocollo supportato da Axis2 sono definite all'interno del file Axis2.xml;di default sono registrate le classi per il protocollo HTTP,mentre non lo sono quelle relative al protocollo JMS che sono commentate. Per quanto riguarda il protocollo HTTP,il frammento di codice xml utilizzato per registrare la classe che implementa il relativo receiver è il seguente:

```
<transportReceiver name="http"
                        class="org.apache.axis2.transport.http.SimpleHTTPServer">
        <parameter name="port">8080</parameter>
```

Mentre il sender HTTP di default,implementato dalla classe `CommonsHTTPTransportSender`,è registrato in questo modo:

```
<transportSender name="http"
                 class="org.apache.axis2.transport.http.CommonsHTTPTransportSender">
            <parameter name="PROTOCOL">HTTP/1.1</parameter>
            <parameter name="Transfer-Encoding">chunked</parameter>
```

Per specificare il fatto che un servizio possa comunicare utilizzando un solo tipo di protocollo bisogna inserire tale tipo di informazione nel file services.xml del servizio prima di effettuarne il deploy:

```
<transports>
     <transport>https</transport>
</transports>
```

Se nel file services.xml non viene inserito il frammento di XML precedente allora il servizio sarà in grado di utilizzare tutti i protocolli avviabili.

In conclusione Axis2 fornisce il supporto per diversi protocolli di trasporto ma è compito dell'amministratore di sistema configurarli in modo appropriato in base alle proprie esigenze.

3.5.2 Moduli non-core

Code Generation Model

Permette di generare sia a lato server che a lato client automaticamente il codice per semplificare la pubblicazione del servizio e la sua invocazione. Le informazioni vengono estratte da una WSDL che descrive il WS e immesse in un XML indipendente dal linguaggio di output. Una volta "parsato",viene generato il codice nel linguaggio prescelto per fare il servizio.

Data Binding Model

Permette di estendere il modulo precedente per avere più controllo sui dati con la generazione automatica del codice. Ce ne sono di diversi tipi, e permettono il passaggio da dati in formato xml al linguaggio voluto(XMLBeans). Bisogna notare che il Databinding Model non è un modulo core dell'architettura di Axis: in questo modo durante la generazione del codice è possibile decidere quale framework data binding utilizzare. Axis2 supoporta i seguenti framework data binding:

- **ADB:** ADB (Axis2 Data Binding) è un semplice e leggero framework che utilizza StAX ed abbastanza performante. E' il binding di default utilizzato da Axis2.
- **XMLBeans:** permette di manipolare il contenuto di un file XML fornendo una visione a oggetti dei dati ed implementando tutte le funzionalità necessarie allo sviluppatore per gestire in modo automatico le operazioni di lettura,scrittura e validazione di un file XML. Con XMLBeans è possibile generare delle classi java che permettono di leggere, scrivere o validare un file xml. Viene preferito quindi se si vuole avere un supporto completo alla specifica dello schema.
- **JaxMe**
- **JibX**.

3.6 Axis2 Data Binding (ADB)

Come detto precedentemente Apache Axis2 Web services è stato progettato per supportare diversi tipi di data-binding. Nella versione che stiamo analizzando è quindi possibile utilizzare sia XMLBeans che JiBx ma anche un data-binding pensato appositamente per Axis2:Axis2 Data Binding(ADB).

Pensato inizialmente per essere un framework semplice,facile da utilzzare e con un numero limitato di funzionalità ADB viene attualmente utilizzato da molti servizi e costituisce il data-

binding di default di Axis2. A differenza degli altri data-binding framework ADB è comunque vincolato ad Axis2.

Questa restrizione comporta sia limitazioni che vantaggi:essendo integrato con Axis2 il codice di ADB è stato infatti implementato in modo da ottimizzare l'interazione con l'engine.

Anche se progettato per supportare ogni linguaggio di programmazione offre attualmente pieno supporto solo a Java e in modo più limitato al linguaggio C.

Il processo di unmarshaling o meccanismo di parsing in ADB consiste nel creare le Java object structure da uno stream di input XML(XMLStreamReader);viceversa il processo di marshaling o serializzazione in ADB consiste nel tradurre le Java object structure in XML Stream(XMLStreamWriter).

ADB legge direttamente da un oggetto XMLStreamReader e scrive direttamente in un oggetto XMLStreamWriter. Questo è uno dei motivi per cui ADB risulta più veloce rispetto ad altri data-binding come XMLBean e jaxb.

L'architettura di ADB è la seguente[33]:

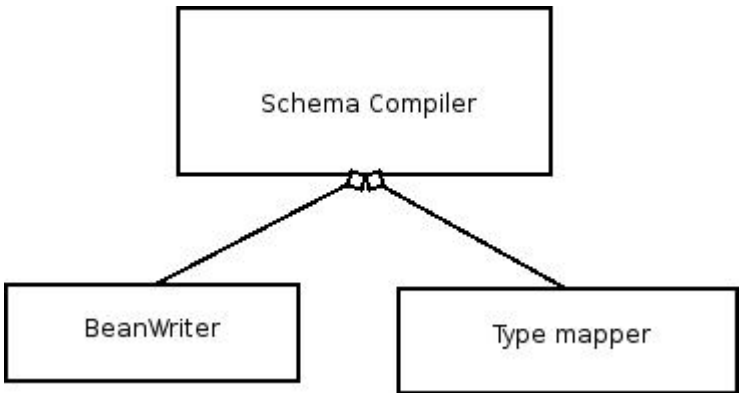


Figura 35: Architettura di ADB

ADB utilizza la libreria WS-commons XmlSchema per leggere uno Schema. Il modello ad oggetti relativo allo schema si traduce in un oggetto di tipo XsmlSchema. Lo schema compiler mantiene un'istanza del writer(JavaBeanWriter per default) che si occupa di scrivere la corrispondente classe. Un writer è libero di utilizzare la tecnica che preferisce:ad esempio JavaBeanWriter utilizza un template XSLT. Lo Schema Compiler utilizza poi un oggetto Type Mapper per conoscere il classname relativo ad ogni QName incontrato.

La trasformazione XSLT dipende dall'implementazione DOM della JVM (crimson o xerces). Il codice dello Schema Compiler è completamente contenuto nel package `org.apache.axis2.databinding.schema.*`. Questo package si trova nel modulo `codegen` di Axis2.

Le classi e i file principali contenuti nel package sono:

1. SchemaCompiler – oggetto principale che compila effettivamente lo schema all'interno delle classi.
2. BeanWriter - BeanWriter rappresenta il tipo d'interfaccia che lo Schema Compiler accetta come writer.
3. JavaBeanWriter – implementazione di default dell'interfaccia BeanWriter.
4. TypeMap – rappresenta l'interfaccia che lo schema compiler utilizza trovare le classi.
5. JavaTypeMap – implementazione di default del the default del TypeMap.
6. BeanTemplate.xsl – template XSL relative al JavaBeanWriter.

7. Schema-compile.properties – il file contenente le proprietà dello schema compiler.

Le regole seguite dallo *schema compiler* per mappare i costrutti dell'XML Schema in classi Java sono le seguenti:

1. Ogni *complex types* si traduce in una classe java bean. Ogni attributo o elemento incapsulato in un complex type si traduce in un campo nella classe generata.
2. Ogni elemento *top level* viene tradotto in una classe.
3. Le restrizioni sui SimpleType sono mantenute sostituendo il tipo con il basetype.

ADB è stato pensato per essere un framework semplice da utilizzare e non è in grado di compilare ogni tipo di schema. Le limitazioni più importanti di ADB sono:

1. Complex Extensions e Restrictions non sono supportate.
2. Choice (Particle) non è supportata.
3. SimpleType Unions non è supportata.

3.7 AXIOM

Uno degli obiettivi principali di ogni middleware Web Service-based consiste nell'evitare di mantenere in memoria l'intero messaggio SOAP. Se da una parte mantenere un intero object model in memoria permette una più semplice gestione degli oggetti XML,dall'altra si penalizza la memoria stessa in quanto l'object model richiede una notevole quantità di memoria di dimensone pari almeno alla dimensione del documento XML che si sta processando.
Le prime generazioni di SOAP engine,come Apache SOAP,utilizzavano un object-model DOM-based per rappresentare un documento XML. Le seconde generazioni di engine,come Apache Axis1.x discusso in precedenza,si basano invece su SAX per evitare di mantenere in memoria l'intera informazione. SAX è un parser di tipo PUSH e una volta iniziato il parser di un documento XML questo non può essere fermato da nessuna entità esterna:nei parser PUSH il controllo del processo di parsing è infatti affidato al parser stesso. Axis1.x mantiene il messaggio XML in memoria sottoforma di eventi SAX:questo comporta comunque un uso intensivo della memoria che diventa evidente quando bisogna gestire un certo numero di file XML. Per documenti XML di media grandezza le performance rimangono accettabili mentre diminuisco se si devono gestire file di grandi dimensioni. Richieste concorrenti con documenti XML di grandi dimensioni potrebbero causare il crash del SOAP engine.

DOM risulta quindi più facile ed immediato da utilizzare rispetto a SAX ed è pertanto preferito per manipolare un file XML;tuttavia l'albero generato da DOM va mantenuto completamente nella memoria RAM e di conseguenza non è possibile utilizzare questa interfaccia per manipolare file che siano più grandi della memoria disponibile sul computer.

Abbiamo visto quindi che per processare un documento XML è possibile utilizzare due modelli[21,22,23,24]:

- *Tree-based*: caricato in memoria e facile da gestire per il programmatore. Costo di memoria elevato e costo computazionale per la costruzione degli oggetti.DOM appartiene a questa categoria: costruisce partendo dal file XML un albero dove ogni nodo dell'albero corrisponde ad un elemento del file.
- *Event-based*: gestisce direttamente lo stream con ottime prestazioni. Difficile gestione per il programmatore, si legge solo "in avanti" e non si può tornare indietro.(SAX)

StAX[21] è un nuovo metodo per elaborare XML,che si affianca a i precedenti DOM e SAX.

Streaming API for XML - StAX - si orienta come dice il nome stesso allo streaming, cercando di mantenere i vantaggi dei precedenti approcci.

Di per sé StAX è molto simile al precedente SAX,come quest'ultimo non richiede di avere una rappresentazione completa del documento XML in memoria (come fa, invece, DOM),ma mantiene in memoria solo quanto sta leggendo,migliorando sensibilmente le prestazioni.

A differenza di SAX cambia il modo con cui gli eventi sono generati. StAX utilizza un metodo *pull*,SAX un metodo *push*. Il metodo pull assicura che sia l'applicazione a generare l'eleaborazione del documento e non il contrario (l'elaborazione del documento genera eventi per l'applicazione, l'applicazione quindi gestisce,non genera gli eventi). In aggiunta a questa opportunità (dipende dal tipo di applicazione se l'opportunità si trasforma in vantaggio) StAX semplifica l'iterazione con il documento.

Uno degli obiettivi di StAX[21] è quello di fornire un modello Event-based,quindi dalle ottime performance,facile da utilizzare come un Tree-based:il limite di un modello basato sugli eventi rimane comunque quello che gli eventi passati non sono riproducibili,lo stream può essere processato una sola volta.

Axis2 per evitare di mantenere il completo messaggio SOAP in memoria e fondere finalmente il modello Event-based con quello Tree-based ha introdotto un nuovo Object Model per rappresentare i messaggi SOAP:AXIOM.

AXIS Object Model [19,20]è un parser XML di tipo pull che si basa sulle API StAX per l'input e l'output dei dati. Il punto innovativo di questo modello è il supporto alla costruzione del modello differita. Questo modello,infatti,non costruisce l'intero documento una volta ricevuto, come avviene ad esempio per il DOM,ma ritarda la costruzione fino a quando non è necessaria. Il modello ad oggetti contiene solo quello che è stato costruito,il resto è ancora nello stream. Inoltre AXIOM può generare eventi StAX partendo da qualsiasi punto del documento ricevuto, indipendentemente dal fatto che sia stato processato o meno. Ancora,AXIOM può generare questi eventi StAX costruendo o meno il modello corrispondente. Questo in AXIOM è chiamato caching.

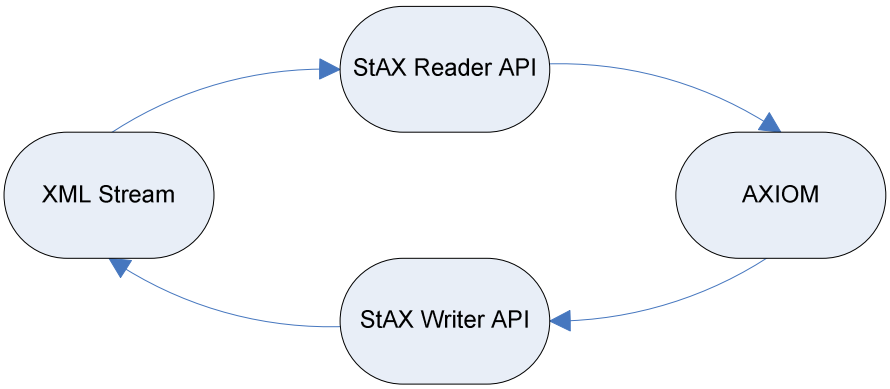


Figura 36:Accesso di AXIOM all'XML Stream

Come mostrato in figura, AXIOM accede allo stream XML attraverso StAX. L'implementazione del parser StAX utilizzata da AXIOM è quella Woodstox,ma ovviante è possibile sostituirla con qualsiasi altra implementazione delle API StAX.

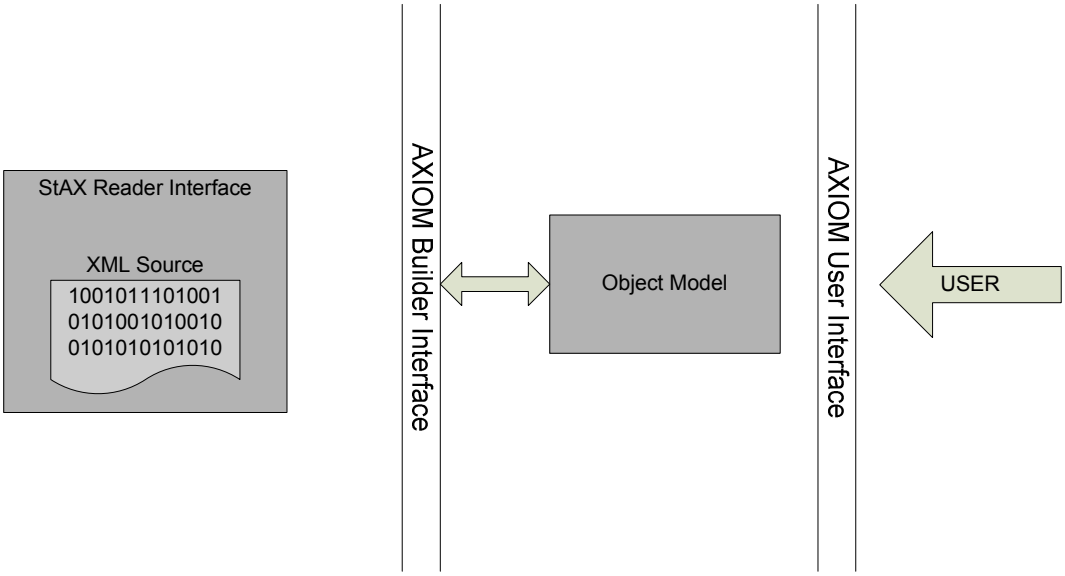


Figura 37:Architettura di AXIOM

AXIOM "vede" l'input stream XML attraverso lo stream reader di StAX,che viene incapsulato dall'interfaccia del costruttore fornito. Ci sono più implementazioni del costruttore, ad esempio StAXOMBuilder che costruisce completamente l'XML info-set model,oppure lo StAXSOAPModelBuilder che costruisce l'object model specifico per SOAP.Ogni implementazione supporta la costruzione differita ed il caching.
L'API AXIOM lavora sopra l'interfaccia dei costruttori e si interfaccia all'utente attraverso una semplice,ma potente, API. Fornisce la massima flessibilità potendo cambiare l'implementazione del costruttore o dell'object model in maniera indipendente dal resto. AXIOM è fornito con due implementazioni di questa API. La prima è basata su un modello a liste collegate (Linked List Object Model, LLOM) considerata l'implementazione di default. L'altra, conosciuta come DOOM (DOM over OM) fornisce un livello di interfaccia DOM su AXIOM. DOOM consente così di poter sfruttare le implementazioni di altre API sviluppate per DOM,come ad esempio WSS4J usato per implementare WS-Security.
Per lavorare con più implementazioni AXIOM usa il concetto di Factory. La factory è concepita in modo che se non è specificata un'implementazione da utilizzare, sarà usata quella di default dal classpath.

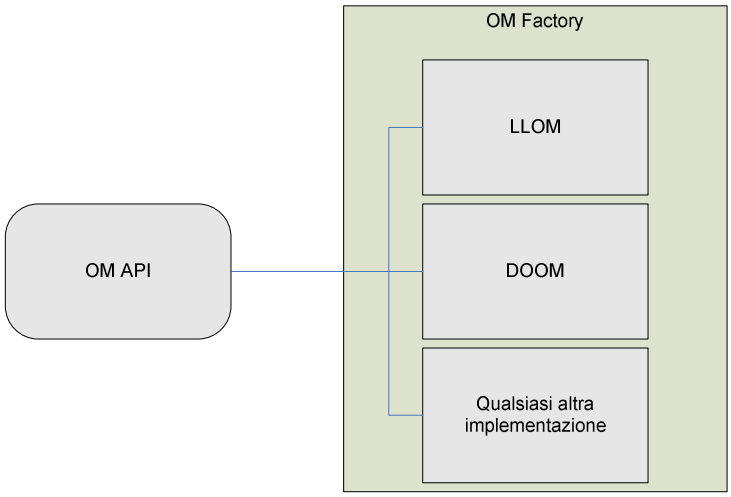


Figura 38:Factory di AXIOM

3.8 Gestione delle sessioni

L'architettura di Axis2 è stata progettata in modo da mantenere separata la logica dallo stato. In Axis2 handlers,MessageReceivers,TransportSenders,TransportReceivers e l'AxisEngine sono stateless:non mantengono lo stato nelle loro classi. Quando si implementa un handler bisogna quindi evitare di gestire lo stato al suo interno per rispettare la natura stateless di Axis2. Ad esempio l'implementazione del seguente handler non è corretta:viene infatti utilizzata la variabile di classe `MessageContext` per memorizzare il `currentMessageContext`. Se si effettua il deploy di Axis2 in un ambiente concorrente avremo dei problemi:nel codice dell'handler non dovrebbe essere inserita alcuna variabile di classe.

```
public class InvalidHandler extends AbstractHandler {
//Variabile di classe per mantenere il message context corrente
private MessageContext currentMessageContext;
public InvocationResponse invoke(MessageContext msgContext) throws AxisFault {
currentMessageContext = msgContext;
return InvocationResponse.CONTINUE;
}
}
```

Questo non significa che non è possible mantenere lo stato in Axis2. Significa semplicemente che non è l'approccio migliore:per gestire le sessioni in Axis2 si ricorre al *context hierarchy*.

Una hierarchy è composta da cinque tipi di context:

- **ConfigurationContext**: E'la rappresentazione run-time dell'intero sistema. Il lifetime del configuration context equivale al lifetime del sistema:ogni proprietà o stato memorizzato durerà per sempre(fino allo shutdown del sistema).
- **ServiceGroupContext**: In Axis2 è possible includere nel file di deploy services.xml più di un servizio nell'elemento `<service Group>`:la relativa rappresentazione run-time è definita ServiceGroupContext.
- **ServiceContext**: Rappresenta il run time di un servizio. Il lifetime del context equivale al lifetime della sessione. Nel sistema è possibile avere più di un service context:dipende dal session scope e dai servizi corrispondenti.
- **OperationContext**: Questo contesto rappresenta il lifetime di un MEP(Message Exchange Pattern). Il lifetime di un operation context è in genere più piccolo del lifetime di un ServiceContext.
- **MessageContext**: Il lifetime di un incoming message è rappresentato dal message context. Se due handler in una data chain vogliono memorizzare un dato,il miglior modo è utilizzare il message context. Un OperationContext può avere uno o più MessageContexts.

Il seguente esempio mostra come accedere al ServiceContext utilizzando il MessageContext all'interno della classe che implementa il servizio. Lo scopo del servizio è semplicemente quello di aggiungere al valore dell'invocazione corrente il valore di quella precedente.

```
public class MyService {
public int add(int value) {
MessageContext messageContext = MessageContext.getCurrentMessageContext();
ServiceContext sc = messageContext.getServiceContext();
Object previousValue = sc.getProperty(«VALUE»);
int previousIntValue = 0;
if (previousValue != null) {
previousIntValue = Integer.parseInt((String)previousValue);
}
int currentValue = previousIntValue + value;
sc.setProperty(«VALUE»,»» + currentValue);
return currentValue;
}
}
```

In questo modo è possible accedere al MessageContext corrente indipendentemente dal session scope utilizzato.

3.8.1 Tipi di Sessioni in Axis2

In Axis2 sono previsti differenti tipi di sessioni ognuna con un proprio lifetime:alcune dureranno pochi secondi mentre altre avranno un lifetime pari al lifetime dell'intero sistema. L'architettura di Axis2 supporta quattro tipi di sessioni:

- Request
- SOAPSession
- Application
- Transport

Request session scope

Request session scope è il default session in Axis. Quando si effetua il deploy di un servizio senza specificare come devono essere gestite le sessioni,il servizio sarò deployato in modalità request session scope. Il lifetime di ogni sessione equivale al tempo richiesto per processare la richiesta:in pratica non viene gestita alcuna sessione e lo stato della conversazione non viene mantenuto.
Quando si effettua il deploy in request session scope viene creata un'implementazione della classe del servizio per ogni invocazione dello stesso:se un client invoca il servizio *n* volte allora saranno create *n* istanze del servizio.
Se si specifica questo *value* nel paramentro *scope* il servizio implementato sarà di tipo stateless.

Per default il file services.xml non contiene il paramtero scope. Per effettuare il deploy del servizio secondo questa modalità si potrebbe quindi sia lasciare inalterato il file generato automaticamente sia inserire esplicitamente il parametro in questo modo:

```
<service name="Service_name" scope="request">
</service>
```

SOAP Session Scope

Nel SOAP Session Scope è Axis2 che mette a disposizione un meccanismo per mantere lo stato della conversazione tra il servizio e il client. Gestire il SOAP session richiede di effettuare delle modifiche sia client side che server side;il client deve includere nei messaggi delle informazioni aggiuntive necessarie al server per il dentificare il client ed associarlo ad una determinata istanza del servizio.

In altre parole per gestire il SOAP session bisogna abilitare il modulo addressing che implementa lo standard WS-Addressing(si veda paragrafo 3.8) sia client side che server side.

Per gestire il SOAP Session un client deve quindi inserire un'informazione aggiuntiva nell'header SOAP: il serviceGroupId. Un client riceverà il valore del serviceGroupId la prima volta che contatterà il servizio deployato in SOAP session scope.
Il servizio infatti genera un serviceGroupId ogni volta che un client lo contatta per la prima volta e lo include nel messaggio di risposta all'interno del paramentro `wsa:ReplyTo`:

```
<wsa:ReplyTo>
<wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
<wsa:ReferenceParameters>
<axis2:ServiceGroupId
xmlns:axis2="http://ws.apache.org/namespaces/axis2">urn:uuid:65E9C56F702A398A8B1
1513011677354</axis2:ServiceGroupId>
</wsa:ReferenceParameters>
</wsa:ReplyTo>
```

Se il client vuole utilizzare una stessa istanza del servizio deve copiare il ReferenceParameter ed includerlo nelle successive invocazioni. In questo modo il servizio riconoscerà che al client è già stato associato un identificatore e non verrà creata nessuna nuova istanza del servizio. A

differenza del request session,nel SOAP Session bisogna considerare il timeout associato alla sessione utilzzata. Se il client non invoca il servizio per più di 30 secondi la sessione terminerà. Se il client invia in un momento successivo il serviceGroupId scaduto verrà generato un AxisFault.
Per modificare il valore del timeout di default bisogna modificare il file di configurazione globale axis2.xml in questo modo:

```
<parameter name="ConfigContextTimeoutInterval">30000</parameter>
```

Per abilitare il client ad estrarre l'identificativo inviato dal servizio e ad includerlo nelle successive invocazioni bisogna inserire il seguente codice:

```
ServiceClient sc = new ServiceClient();
sc.engageModule("addressing");
```

Come si può vedere è necessario caricare il modulo relativo al WS-Addressing anche lato client. In questo modo però non viene abilitato il meccanismo delle sessioni:semplicemente vengono incluse negli header dei messaggi SOAP informazioni relative ai client e al servizio.
Per abilitare concretamente il SOAP Session scope sono necessarie anche le seguenti istruzioni:

```
Options opts = new Options();
opts.setManageSession(true);
sc.setOptions(opts);
```

In questo modo il client specifica che vuole mantenere lo stato della conversazione con il servizio.

Per effettuare il deploy di un servizio secondo questa modalità bisogna modificare il file services.xml in questo modo:

```
<service name="Service_name" scope="soapsession">
</service>
```

Transport Session Scope

Il Transport Session Scope equivale alla modalità HTTP Session descritta nel primo capitolo per Axis1.4:anche in questo caso se si utilizza HTTP come protocollo di trasporto si impiegano i cookie.
Il riultato è lo stesso che si ottiene utilizzando le SOAP Session;in questo caso però le sessioni sono gestite a livello di protocollo e non a livello di engine:le informazioni necessarie al servizio per riconoscere un determinato client e stabilire se deve creare o meno un sua nuova istanza non sono contenute infatti nel messaggio SOAP;di conseguenza non è compito dell'engine modificare tale protocollo con il WS-Addressing per includerle.

A differenza del SOAP Session scope in questo caso è possibile far comunicare più di un *service group* su una stessa transport session:di conseguenza il numero di istanza create del servizio dipende dal numero di transport session generate.

Per effettuare il deploy di un servizio secondo questa modalità bisogna modificare il file services.xml in questo modo:

```
<service name="Service_name" scope=" transportsession">
</service>
```

In questo caso non è necessario l'utilizzo del modulo addressing;nel codice del client basta inserire:

```
((OrchestratoreWSSoapBindingStub)stub).setMaintainSession(true);
```

//per visualizzare il cookie :

```
Call authCall = ((OrchestratoreWSServiceLocator)service).getCall();
org.apache.axis.MessageContext msgContext = authCall.getMessageContext();
String cookie = msgContext.getStrProp(HTTPConstants.HEADER_COOKIE);
```

se viene utilizzato lo stub per contattare il servizio oppure:

```
ServiceClient sc=new ServiceClient();
Options opts=new Options();
opts.setManageSession(true);
sc.setOptions(opts);
```

nelle altre modalità di accesso ad un servizio.
Da notare che in questo caso non è necessario abilitare il WS-Addressing:non bisogna modificare il protocollo SOAP,quello che serve,il cookie,è contenuto nel protocollo di trasporto.

In altre parole SOAP Session scope e Trasport Session scope rappresentano modi diversi per raggiungere uno stesso obiettivo:nel primo si utilizza il WS-Addressing per modificare il protocollo SOAP per includere un ServiceGroupId mentre nel secondo si utilizza un cookie incluso nel protocollo di trasporto e si lascia inalterato il messaggio SOAP.

Application Scope

Application scope presenta il lifetime più lungo rispetto alle modalità precedenti:equivale al lifetime del sistema. I servizi deployati con questa modalità vengono istanziati una sola volta per ogni applicazione all'interno della Java Virtual Machine (in un cluster di N nodi, i servizi saranno istanziati N volte). Anche questi servizi possono essere statefull,ma non devono mantenere informazioni relative agli utenti poichè di ciascun servizio ne esiste solo un'istanza condivisa da tutti gli utenti (sono equivalenti a un Singleton).

In questo caso i client non devono preoccuparsi di inviare alcun tipo di informazione aggiuntiva al servizio per utilizzare la stessa sessione.

Per effettuare il deploy di un servizio secondo questa modalità bisogna modificare il file services.xml in questo modo:

```
<service name="Service_name" scope=" application">
</service>
```

In conclusione è possibile affermare che:

- `Request`(Axis1.4) equivale al `RequestScope(Axis2)`
- `HTTP Cookie`(Axis.4) equivale al `Transport Session scope(Axis2)`
- `SOAP Header`(Axis1.4) equivale al `SOAP Session scope(Axis2)`
- `Application`(Axis1.4) equivale all'`Application Scope`(Axis2)

Cambia naturalmente solo il modo in cui vengono implementati,ad esempio il SOAP Header utilizza il SimpleSessionHeader mentre il SOAP Session scope utlizza il modulo WS-Addressing ma entrambi comunque sono transport-independent.

3.9 Esempio: BancaDiCreditoCooperativoWS

Per capire come si implementa e si effettua il deploy di un servizio con Axis2 riprendo l'esempio presentato nel capitolo precedente;l'idea infatti è quella di deployare tre servizi ai quali corrispondono altrettante banche,utilizzando tre tecnologie differenti:

- Axis – Apache Tomcat
- Axis 2 – Apache Tomcat
- JBoss – Apache Tomcat

Per questo motivo il servizio che presenterò in questo capitolo avrà la stessa struttura,la stessa logica di business e gli stessi metodi esposti da BancaDiRomaWS deployato con Axis;la

differenza è questo nuovo Web Services,chiamato BancaDiCreditoCooperativoWS,sarà implementato e deployato con Axis 2 attraverso l'utilizzo dei relativi tool. Nella prima parte anlalizzeremo quindi passo dopo passo i passi necessari per deployare correttamente il servizio nell'application server Apache Tomcat mentre nella seconda parte vedremo come utilizzare tale servizio attraverso due diversi client che ne invocano i relativi metodi uno in modalità sincrona e l'altro in modalità asincrona. Bisogna comunque notare che il termine asincrono può essere in questo caso fuorviante;è più giusto infatti definire questo tipo di comunicazione come non bloccante piuttosto che come asincrona. La comunicazione è non bloccante in quanto un client può invocare la richiesta al servizio e continuare ad eseguire le proprie operazioni per poi gestire in modo opportuno la relativa risposta;questo scenario è ben diverso da quello previsto per un tipo di comunicazione asincrona che disaccoppia completamente un client dal servizio. Nel tipo di comunicazione non bloccante infatti se la risposta del servizio impiega più del tempo di attesa previsto dal server che ospita il client verrà lanciata un'eccezione di read timeout a dimostrazione del fatto che l'interazione non è asincrona,ma semplicemente non bloccante per il tempo previsto di attesa della risposta. Se non si vuole modificare la configurazione di base di Axis2 le Api Non-Blocking rappresentano il massimo livello di asincronia che si può ottenere.
Per migliorare la qualità della comunizione e disaccoppiare ulteriormente client e servizi, bisogna infatti apportare alcune modifiche sia client side che server side ad Axis2;tali modifiche permettono all'engine di implementare correttamente lo standard del WS-Addressing[32,34].
Per superare quindi il problema del read timeout ed ottenere di conseguenza il vero il massimo livello di asincronia supportato in Axis2 bisogna implementare le API Non-blocking client side e configurare la comunicazione con il server in modo da utilzzare due canali di trasporto differenti per la richiesta e la risposta. In questo modo il client non rimane in attesa di ricevere la risposta di un servizio e non deve preoccuparsi neanche del fatto che il tempo di risposta di quest'ultimo superi il timeout stabilito per la connessione;ricevuta la richiesta il servizio non utilizza lo stesso canale di comunicazione della richiesta,canale che nel frattempo potrebbe non esistere più:terminata l'elaborazione dell'operazione richiesta il servizio crea una nuova connessione verso lo stesso client ed invia la relativa risposta. In questo modo però richiesta e risposta sono completamente disaccoppiate:l'unico modo per dare la possibilità al servizio di conoscere il client che gli ha inviato una determinata richiesta per poi rispondere su altro canale è di inserire le informazioni del client nel messaggio SOAP di richiesta generato;questo meccanismo di correlazione tra request e response consiste appunto nel WS-Addressing che in accordo con l'architettura di Axis2 consiste in un modulo con estensione .mar.
Il WS-Addressing si occupa quindi di definire uno standard per la nomenclatura di client e servizi.
Come detto Axis2 supporta il WS-Addressing ma per utilizzarlo prima configurarlo sia client side che server side.

WS-Addressing server side

Per implementare lo standard si possono seguire la prima cosa da fare è copiare il modulo addressing-1.4.1 nella cartella `webapps\axis2\WEB-INF\lib`.
Si possono poi seguire due diverse modalità di configurazione:
-a livello di servizio
-a livello globale
Nel primo caso è possibile selezionare i servizi che effettivamente utilizzeranno lo standard inserendo un opportuno tag nel file di deploy services.xml:

```
<service>
<module ref="addressing"/>
</service>
```

E solo i servizi deloyati in questo modo implementeranno lo standard.
Mentre nel secondo caso il modulo verrà configurato nel file di configurazione globale axis2.xml e di conseguenza lo standard sarà utilizzato da tutti i servizi deployati nell'application server:

```
 <!-- ================================================= -->
    <!-- Global Modules  -->
    <!-- ================================================= -->
    <!-- Comment this to disable Addressing -->
    <module ref="addressing"/>
```

Se si accede come amministratore alla Web Console di Axis e si segue il link dei servizi avviabli è possibile osservare l'EndPoinReference,la descrizione,lo stato e i moduli attivati per ogni servizio;in questo caso per il servizio BancaDiCreditoCooperativoWS sono stati configurati sia il modulo relativo a SOAPMonitor per vedere i messaggi SOAP scambiati che quello relativo al WS-Addressing.

WS-Addressing client side

Come per il server side anche in questo caso bisogna inserire il modulo addressing-1.4.1 con estenzione .mar nella cartella contenente le librerie necessarie ad Axis2 `C:\Programmi\axis2-1.4.1\lib`. Il passo successivo è quello di inserire alcune righe di codice all'interno del client che permettono di utilizzare concretamente e correttamente lo standard.

3.9.1 Server-side

Per scrivere un servizio occorrono quattro passi:

1) Implementare la classe che svolge il servizio;
2) Scrivere il file *services.xml* che descrive il servizio;
3) Creare un archivio *.aar* (Axis Archive);
4) Fare il deploy del servizio sul Server.

Esistono diversi modi per implementare il servizio: ***Pojo***, *Axiom*, *Adb*, *XMLBeans*, e *JiBX*.
In questo caso ho seguito l'approccio di tipo Pojo(*Plain Old Java Object*). Utilizzare questa tecnologia significa scegliere un modo veloce e facile da implementare, semplice da mantenere. Per implementare il server,si incomincia scrivendo la classe come una comune classe Java,i suoi metodi dichiarati pubblici saranno quelli invocabili.

1. Costruire l'interfaccia del servizio

La classe che rappresenta l'interfaccia del servizio presenta gli stessi metodi di quella relativa all'esempio presentato nel capitolo precedente,cambia naturalmente solo il nome;le classi BankQuoteRequest,BankQuoteResponse e BankLoanRequest che definiscono i complextType sono invece identiche a quelle già analizzate.

BancaDiCreditoCooperativoWS.java

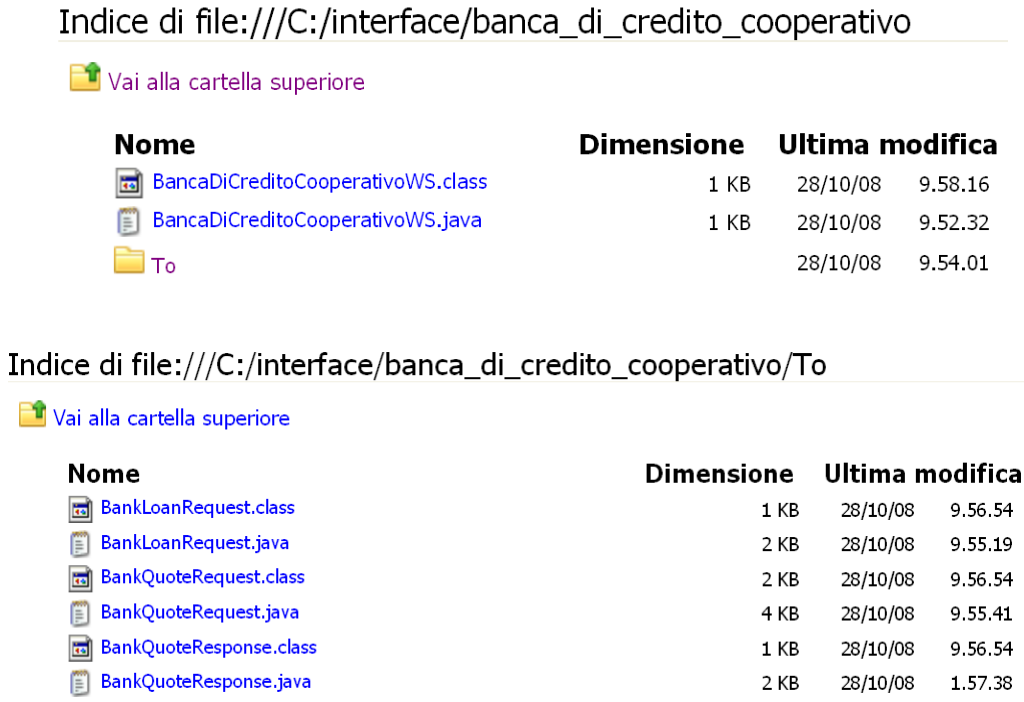
```
package banca_di_credito_cooperativo;
import banca_di_credito_cooperativo.To.*;

public interface BancaDiCreditoCooperativoWS {
    public BankQuoteResponse getLoanQuote(BankQuoteRequest in0) ;
    public String getLoan(BankLoanRequest in0) ;
}
```

Compilo tutti i file:

```
C:\interface>cd banca_di_credito_cooperativo
```

```
C:\interface\banca_di_credito_cooperativo>javac To/*.java
C:\interface\cd banca_di_credito_cooperativo>cd..
C:\interface\javac banca_di_credito_cooperativo/*.java
```

La struttura delle directory `interface` contenente il progetto è la seguente:



2. Creare il file WSDL a partire dall'interfaccia appena creata.

```
C:\interface>java2wsdl
-l "http://localhost:8081/axis2/services/BancaDiCreditoCooperativoWS"
-cn banca_di_credito_cooperativo.BancaDiCreditoCooperativoWS
```

Dove:

```
-cn: fully qualified class name
```

Viene generato il file BancaDiCreditoCooperativo.wsdl nella directory corrente.

La tabella seguente mostra la mappatura da una struttura Java su una struttura WSDL e XML correlata di tipo document/literal.

Struttura Java	Struttura WSDL e XML
SEI (Service endpoint interface)	wsdl:portType
Metodo	wsdl:operation
Parametri	wsdl:input, wsdl:message, wsdl:part
Return	wsdl:output, wsdl:message, wsdl:part
Tipi primitivi	Tipi semplici xsd(no soapenc)
Bean Java	xsd:complexType
Proprietà del bean Java	xsd:elements nidificati di xsd:complexType

La struttura generale del documento WSDL generato è quindila seguente:

```
  <?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://banca_di_credito_cooperativo"
    xmlns:ns1="http://org.apache.axis2/xsd"
    xmlns:ns="http://banca_di_credito_cooperativo"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:ax21="http://To.banca_di_credito_cooperativo/xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
<wsdl:types>
<wsdl:message name="getLoanRequest">
<wsdl:message name="getLoanResponse">
<wsdl:message name="getLoanQuoteRequest">
<wsdl:message name="getLoanQuoteResponse">
<wsdl:portType name="BancaDiCreditoCooperativoWSPortType">
<wsdl:binding name="BancaDiCreditoCooperativoWSSoap12Binding"
    type="ns:BancaDiCreditoCooperativoWSPortType">
<wsdl:binding name="BancaDiCreditoCooperativoWSHttpBinding"
    type="ns:BancaDiCreditoCooperativoWSPortType">
<wsdl:binding name="BancaDiCreditoCooperativoWSSoap11Binding"
    type="ns:BancaDiCreditoCooperativoWSPortType">
<wsdl:service name="BancaDiCreditoCooperativoWS">
    </wsdl:definitions>
```

Come si può vedere presenta alcune differenze rispetto a quello analizzato nel capito precedente con Axis1.4.

`<wsdl:definitions>`
Le prime differenze è possibile notarle già a livello di namespace.

In `definitions` viene importato il namespace:
`xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"`
per abilitare il supporto alla specifica dei metadati WS-Addressing:standard non previsto invece nella versione Axis1.x.

Non viene importato il namespace relativo al SOAP Encoding:tale codifica non viene infatti utilizzata se il WSDL è di tipo document/literal come in questo caso.

Viene importato il namespace:
`xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">`
relativo alla versione SOAP(SOAP 1.2)utilizzata con il prefisso `soap12`.
La presenza del namespace relativo al protocollo HTTP è dovuto al fatto che nel documento WSDL è stato definito,come vedremo tra poco,il binding HTTP:
`xmlns:http=http://schemas.xmlsoap.org/wsdl/http/`
Viene importato poi il nampespace relativo alla codifica MIME che è un componente fondamentale per protocolli come HTTP che prevede che i dati siano trasmetti come messaggi:
`xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"`

`<wsdl:binding>`
Nella specifica del WSDL sono stati definiti tre binding:

```
<wsdl:binding name="BancaDiCreditoCooperativoWSSoap12Binding"
    type="ns:BancaDiCreditoCooperativoWSPortType">
<wsdl:binding name="BancaDiCreditoCooperativoWSHttpBinding"
    type="ns:BancaDiCreditoCooperativoWSPortType">
<wsdl:binding name="BancaDiCreditoCooperativoWSSoap11Binding"
    type="ns:BancaDiCreditoCooperativoWSPortType">
```

per defnire come le operazioni:
<wsdl:operation name="**getLoanQuote**">
<wsdl:operation name="**getLoan**">
definite nel portType saranno trasmesse sulla rete.
La definizione di tre binding differenti consente al servizio di utilizzare altrettanti protocolli per lo scambio di messaggi:
-SOAP1.1
-SOAP1.2
-HTTP
Il binding con SOAP(versione 1.1 e 1.2) è relativo agli endpoint e cioè alle specifiche di invio dei messaggi di input ed output tramite SOAP: le informazioni relative ai messaggi vengono veicolate nella struttura SOAP utilizzando i blocchi Header e Body.

```
<wsdl:binding name="BancaDiCreditoCooperativoWSSoap12Binding"
    type="ns:BancaDiCreditoCooperativoWSPortType">
  <soap12:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="getLoanQuote">
  <soap12:operation soapAction="urn:getLoanQuote" style="document" />
<wsdl:input>
  <soap12:body use="literal" />
    </wsdl:input>
<wsdl:output>
  <soap12:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
<wsdl:operation name="getLoan">
  <soap12:operation soapAction="urn:getLoan" style="document" />
 <wsdl:input>
  <soap12:body use="literal" />
    </wsdl:input>
<wsdl:output>
  <soap12:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
```

Per quanto riguarda HTTP,invece,le specifiche WSDL definiscono collegamenti per i verbi GET e POST (`<http:binding verb="POST" />`) in modo da consentire ad una applicazione sviluppata per WSDL di accedere ad un Web Server come se fosse un Web Service:in questo caso vengono scambiati documenti XML con il nome e gli argomenti del metodo contenuti nel corpo del POST.

```
<wsdl:binding name="BancaDiCreditoCooperativoWSHttpBinding"
    type="ns:BancaDiCreditoCooperativoWSPortType">
  <http:binding verb="POST" />
<wsdl:operation name="getLoanQuote">
  <http:operation location="BancaDiCreditoCooperativoWS/getLoanQuote" />
 <wsdl:input>
  <mime:content part="getLoanQuote" type="text/xml" />
    </wsdl:input>
 <wsdl:output>
  <mime:content part="getLoanQuote" type="text/xml" />
    </wsdl:output>
    </wsdl:operation>
 <wsdl:operation name="getLoan">
  <http:operation location="BancaDiCreditoCooperativoWS/getLoan" />
 <wsdl:input>
  <mime:content part="getLoan" type="text/xml" />
    </wsdl:input>
 <wsdl:output>
```

```
  <mime:content part="getLoan" type="text/xml" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
```

`document/literal`
La differenza più evidente con il WSDL relativo al servizio implementato con Axis1.4 è dovuta al fatto che il documento generato con il tool di Axis2 è di tipo `document/literal` e non `rpc/encode`r:per questo motivo saranno differenti anche i messaggi SOAP di risposta generati dai servizi.
Lo stile `document` è sostanzialmente utilizzato per descrivere i Web Service come entità software orientate al consumo di documenti ed infatti ogni parametro in questo caso è rappresentato dal documento che lo descrive senza nessuna modifica. Lo stile `literal`, prescrive che i dati debbano essere descritti mediante un documento XML-Schema e non fa nessuna supposizione sul modo in cui i dati vengono serializzati-deserializzati; l'unico vincolo che si richiede è che il documento sia sempre descritto mediante XMLSchema.

Ricordando qunto detto su `rpc` ed `encoder` nel capitolo precedente è possibile avere,quindi, quattro possibili combinazioni *di Binding Style, Encoding Style* in un messaggio SOAP:

1) `rpc/encoded`:Web Service visto come procedura remota con serializzazione e deserializzazione standard (di fatto è una formalizzazione XML di una chiamata ad un procedura remota).
2) `rpc/literal`:Web Service visto come procedura remota con serializzazione proprietaria.
3) `document/encoded`:Web Service visto come servizio document oriented, ma in cui la serializzazione e deserializzazione dei parametri segue l'encoding standard. Di fatto non usato.
4) `document/literal`:Web Service visto come servizio document oriented in cui la serializzazione e deserializzazione dei dati avviene in maniera proprietaria.

Questa varietà di combinazioni ha introdotto diversi problemi di interoperabilità tra diverse implementazioni di Web Service SOAP,soprattuto in relazione ai diversi linguaggi di programmazione e piattaforme utilizzati.Per questo motivo si è costituito il consorzio *WS-I (Web Service Interoperability*)[35,36,37] il cui scopo è proprio quello della creazione di specifiche per permettere una reale interoperabilità tra le diverse implementazioni dei Web Service. Per questo motivo l'organizzazione WS-I ha elaborato un set di specifiche chiamato *WS-I Basic Profile* che devono essere supportate dagli implementatori dei servizi affinche si abbia un minimo livello di interoperabilità per i Web Service,tra le altre cose,il WS-I Basic Profile prevede,per esempio,che venga utilizzato il formato *document/literal* come encoding.
In pratica se due vendor affermano di essere WS-I basic profile compliant possiamo essere sicuri che i prodotti saranno interoperabili.

3. Generare lo skeleton del servizio

A partire dal documento WSDL è ora possibile utilizzare un altro tool,wsdl2java, che permette di generare lo skeleton del servizio.

```
C:\interface>wsdl2java -uri BancaDiCreditoCooperativoWS.wsdl -ss -sd
```
Dove:
```
- ss: Generate server-side code(i.e. skeletons)
- sd: Generate service descriptor(i.e. services.xml)
```

Non specificando l'opzione `-d` viene utilizzato il data-binding di default ADB.
In questo modo vengono generati il file *build.xml* e le cartelle *resources* e *src*:

Indice di file:///C:/interface

Vai alla cartella superiore

Nome	Dimensione	Ultima modifica
BancaDiCreditoCooperativoWS.wsdl	9 KB	28/10/08 11.21.31
banca_di_credito_cooperativo		28/10/08 11.20.38
build.xml	6 KB	28/10/08 11.28.32
resources		28/10/08 11.28.30
src		28/10/08 11.28.29

- `build.xml`: il file viene utilizzato per compilare utilizzando il comando *ant* le classi generate includendo tutte le librerie necessarie ad Axis2;la compilazione di tali classi produce anche il file `.aar` da copiare all'interno dell'application server per effettuare il deploy del servizio.
- resources: questa cartella contiene i file *service.xml* e *BancaDiCreditoCooperativo.wsdl* generati.
- *src*: source file.

Clausola WSDL — **Classi Java generate**

Mappatura dei tipi XML definiti nella sezione `<wsdl:types>`:

Per i complexType `<xs:complextType>`: → Classe java:

BankQuoteRequest	BankQuoteRequest.java
BankQuoteResponse	BankQuoteResponse.java
BankLoanRequest	BankLoanRequest.java

Per ogni `<xs:element>`: → Classe java:

getLoanQuote	getLoanQuote.java
getLoanQuoteResponse	getLoanQuoteResponse.java
getLoan	getLoan.java
getLoanResponse	getLoanResponse.java

```
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
    targetNamespace="http://banca_di_credito_cooperativo"
    xmlns:ax22="http://To.banca_di_credito_cooperativo/xsd">
  <xs:import namespace="http://To.banca_di_credito_cooperativo/xsd" />
    <xs:element name="getLoanQuote">
     <xs:element name="getLoanQuoteResponse">
    <xs:element name="getLoan">
    <xs:element name="getLoanResponse">
    </xs:schema>
```

Mappatura di `<wsdl:service>`

Per il servizio: → Implementazione del servizio:

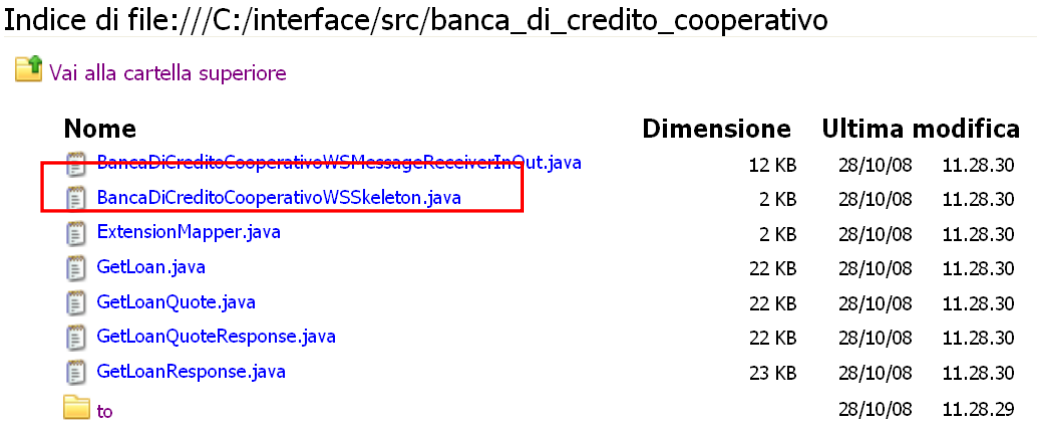
BancaDiCreditoCooperativoWS	BancaDiCreditoCooperativoWSSkeleton.java
	BancaDiCreditoCooperativoWSMessageReceiverInOut.java
	ExtensionMapper.java
	Service.xml

Indice di file:///C:/interface/src

Vai alla cartella superiore

Nome	Dimensione	Ultima modifica
banca_di_credito_cooperativo		28/10/08 11.28.30

All'interno di *banca_di_credito_cooperativo* troveremo:



- `BancaDiCreditoCooperativoWSMessageReceiverInOut.java`:Questa classe ha un metodo che si chiama *invokeBusinessLogi*c che recupera il messaggio in ricezione e invia il messaggio in uscita dal web service una volta richiamato lo skeleton. r
- `BancaDiCreditoCooperativoWSSkeleton.java`: La classe che implementa il sevizio.
- `ExtensionMapper.java`: Utilizzata per implementare il supporto a `xsi:type`;il codice che richiama extension mapper è generato all'interno del metodo Factory.parse del bean e viene attivato quando si trova un attributo xsi:type. In altre parole quando nello stream XML di input viene individuato un `xsi:type` si utilizza la classe ExtensionMapper per trovare la corrispondente classe associata al'elemento.

```
public  class ExtensionMapper{

public static java.lang.Object getTypeObject(java.lang.String namespaceURI,
                                              java.lang.String typeName,

javax.xml.stream.XMLStreamReader reader) throws java.lang.Exception{
 if(                                                                    (
http://To.banca_di_credito_cooperativo/xsd".equals(namespaceURI) &&
                    "BankQuoteResponse".equals(typeName)){
                    return
banca_di_credito_cooperativo.to.xsd.BankQuoteResponse.Factory.parse(reader);
                          }
```

E le seguenti classi Java che corrispondono agli elementi trovati nel `<wsdl:types>` del WSDL:

- `GetLoan.java`
- `GetLoanQuote.java`
- `GetLoanQuoteResponse.java`
- `GetLoanResponse.java`

```
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
    targetNamespace="http://banca_di_credito_cooperativo"
    xmlns:ax22="http://To.banca_di_credito_cooperativo/xsd">
  <xs:import namespace="http://To.banca_di_credito_cooperativo/xsd" />
<xs:element name="getLoanQuote">
<xs:element name="getLoanQuoteResponse">
<xs:element name="getLoan">
  <xs:element name="getLoanResponse">
    </xs:schema>
```

Mentre la cartella `to` contiene la cartella `xsd` che a sua volta contiene le classi che definiscono i complexType `BankQuoteRequest`,`BankQuoteResponse` e `BankLoanRequest`.

In accordo con il binding ADB nella cartella `to` sono state generate le classi corrispondenti ai compexType definiti dal servizio e individuati dal binding durante il parsing del documento. Per

capire come ADB ha tradotto le classi java a partire dal WSDL analizziamo il caso del complexType BankQuoteRequest:le stesse considerazioni valgono comunque anche per gli altri BanQuoteResponse e BankLoanRequest. Il frammento XML nel file WSDL che definisce il complexType è il seguente:

```
<xs:complexType name="BankQuoteRequest">
 <xs:sequence>
  <xs:element minOccurs="0" name="creditHistory" type="xs:int" />
  <xs:element minOccurs="0" name="creditRisk" nillable="true" type="xs:string" />
  <xs:element minOccurs="0" name="creditScore" type="xs:int" />
  <xs:element minOccurs="0" name="loanAmount" type="xs:int" />
  <xs:element minOccurs="0" name="loanDuration" type="xs:int" />
  <xs:element minOccurs="0" name="ssn" nillable="true" type="xs:string" />
    </xs:sequence>
   </xs:complexType>
```

Sia le classi generate da *ADB* nel package *banca_di_credito_cooperativo*:

- GetLoanQuote.java
- GetLoanQuoteResponse.java
- GetLoan.java
- GetLoanResponse.java

sia le classi che corrispondono ai complex type generati nel package *banca_di_credito_cooperativo.to.xsd:*

- BankQuoteRequest.java
- BankQuoteResponse.java
- BankLoanRequest.java

presentano le seguenti caratteristiche:

- implementano l'interfaccia ADBBean:tale interfaccia contiene il solo metodo `getPullParser`. Questo metodo veniva utilizzato nelle versioni precedenti di Axis2 per generare un XMLStreamReader durante la serializzazione:nella versione che stiamo analizzando questo metodo è deprecato per motivi di performance.

  ```
  implements org.apache.axis2.databinding.ADBBean
  ```

- ogni *child element* contiene una variabile locale,metodi getter e setter con il relativo tipo(esempio *BankQuoteRequest*):

  ```
  o  protected int localCreditHistory ;
  o  public  int getCreditHistory();
  o  public void setCreditHistory(int param)
  ```

- contengono i metodi `GetOMElement` e `serialize`:questi metodo sono utilizzati per serializzare le Java object structure in stream XML.
- contengono il metodo `parse`:utilizzato per creare un oggetto ADBEan da un XMLStreamReader.

Per pubblicare il servizio è necessario il file di configurazione (services.xml) generato dal tool e creato nella directory META-INF. Il file è il seguente:

```
<serviceGroup>
    <service name="BancaDiCreditoCooperativoWS">
        <messageReceivers>
            <messageReceiver mep="http://www.w3.org/ns/wsdl/in-out"
class="banca_di_credito_cooperativo.BancaDiCreditoCooperativoWSMessageReceiverIn
Out"/>
        </messageReceivers>
```

```
        <parameter
name="ServiceClass">banca_di_credito_cooperativo.BancaDiCreditoCooperativoWSSkel
eton</parameter>
        <parameter name="useOriginalwsdl">true</parameter>
        <parameter name="modifyUserWSDLPortAddress">true</parameter>
        <operation name="getLoanQuote" mep="http://www.w3.org/ns/wsdl/in-out"
namespace="http://banca_di_credito_cooperativo">
            <actionMapping>urn:getLoanQuote</actionMapping>
            <outputActionMapping>urn:getLoanQuoteResponse</outputActionMapping>
        </operation>
        <operation name="getLoan" mep="http://www.w3.org/ns/wsdl/in-out"
namespace="http://banca_di_credito_cooperativo">
            <actionMapping>urn:getLoan</actionMapping>
            <outputActionMapping>urn:getLoanResponse</outputActionMapping>
        </operation>
    </service>
</serviceGroup>
```

Cerchiamo di capire quanto specificato nel file che è l'equivalente del .wsdd in Axis1.x.

`<serviceGroup>:` rappresenta l'elemento radice del file XML;al suo interno è possible definire tutti i servizi che si vogliono pubblicare:in quanto caso il servizio è uno solo.

`<service name="BancaDiCreditoCooperativoWS">:` contiene il nome del servizio.

`<messageReceivers>:` contiene la classe che si occupa di ricevere i request message,chiamare la classe che implementa lo skeleton del servizio ed inviare il response message.

`<parameter name="ServiceClass">:` contiene la classe nella quale vengono concretamente implementati i metodi esposti dal servizio.

`<operation name="Nome metodo">:` è presente un elemento <operation> per ogni metodo del servizio.

`<actionMapping>:` contiene il nome dell'azione da seguire quando viene richiesto di eseguire il metodo ad essa associato.

`<outputActionMapping>:` contiene il nome dell'azione da seguire per rispondere ad un precedente messaggio di richiesta di esecuzione del metodo ad essa associato.

Quanto definito nel file services.xml viene correttamente riportato nel docuemento WSDL generato:

```
<wsdl:portType name="BancaDiCreditoCooperativoWSPortType">
<wsdl:operation name="getLoanQuote">
  <wsdl:input message="ns:getLoanQuoteRequest" wsaw:Action="urn:getLoanQuote" />
  <wsdl:output message="ns:getLoanQuoteResponse"
    wsaw:Action="urn:getLoanQuoteResponse" />
</wsdl:operation>
<wsdl:operation name="getLoan">
  <wsdl:input message="ns:getLoanRequest" wsaw:Action="urn:getLoan" />
  <wsdl:output message="ns:getLoanResponse" wsaw:Action="urn:getLoanResponse" />
</wsdl:operation>
</wsdl:portType>
```

E nei relative messaggi SOAP generati:

```
==== Request,metodo:getLoanQuote====
POST /axis2/services/BancaDiCreditoCooperativoWS/ HTTP/1.1
Content-Type: application/soap+xml;
charset=UTF-8;
action="urn:getLoanQuote"
```

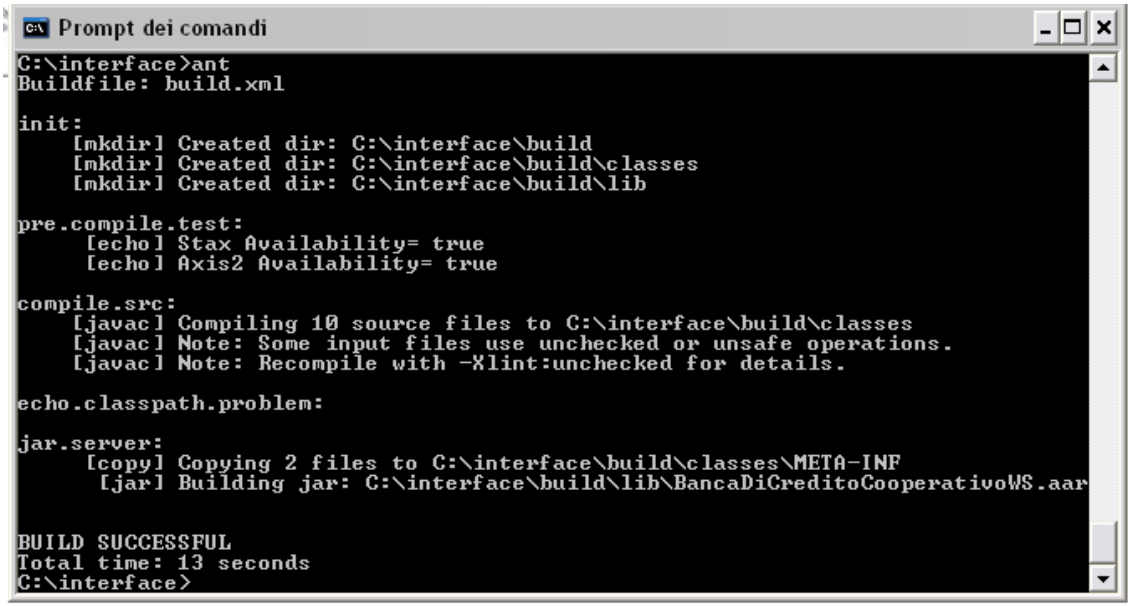
```
==== Response,metodo:getLoanQuote====
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/soap+xml;
action="urn:getLoanQuoteResponse";
```

4. Modificare la classe `BancaDiCreditoCooperativoWSSkeleton.java`

La classe di default generata dal tool non implementa la logica di business:è un template nel quale bisogna concretamente scrivere l'implementazione dei metodi del servizio e che è la stessa(cambiano solamente alcuni parametri)del servizio BancaDiRomaWS presentato precedentemente:

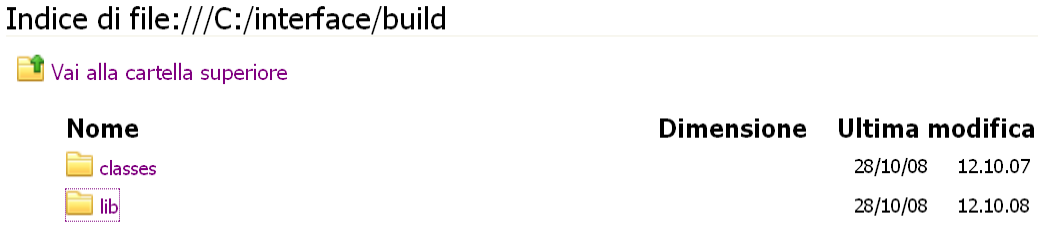
```
/**
 * BancaDiCreditoCooperativoWSSkeleton.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis2 version: 1.4.1  Built on : Aug 13, 2008 (05:03:35 LKT)
 */
    package banca_di_credito_cooperativo;
    /**
     *  BancaDiCreditoCooperativoWSSkeleton java skeleton for the axisService
     */
    public class BancaDiCreditoCooperativoWSSkeleton{


        /**
         * Auto generated method signature
         *
                                     * @param getLoanQuote
         */

                  public banca_di_credito_cooperativo.GetLoanQuoteResponse
getLoanQuote
                    (
                    banca_di_credito_cooperativo.GetLoanQuote getLoanQuote
                    )
            {
                //TODO : fill this with the necessary business logic
                throw new  java.lang.UnsupportedOperationException("Please
implement " + this.getClass().getName() + "#getLoanQuote");
        }


        /**
         * Auto generated method signature
         *
                                     * @param getLoan
         */

                  public banca_di_credito_cooperativo.GetLoanResponse getLoan
                    (
                    banca_di_credito_cooperativo.GetLoan getLoan
                    )
            {
                //TODO : fill this with the necessary business logic
                throw new  java.lang.UnsupportedOperationException("Please
implement " + this.getClass().getName() + "#getLoan");
        }

    }
```

5. Compilo i file generati eseguendo il comando ant nella directory contenente il file build.xml.

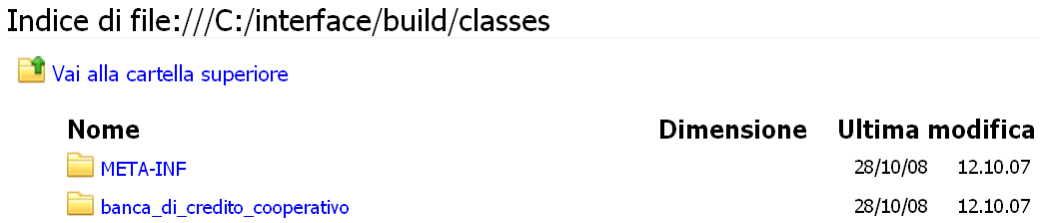


In questo modo viene generata anche la cartella *build* e l'aspetto finale della directory del progetto è:

Nella cartella `build` sono state generare le cartelle:



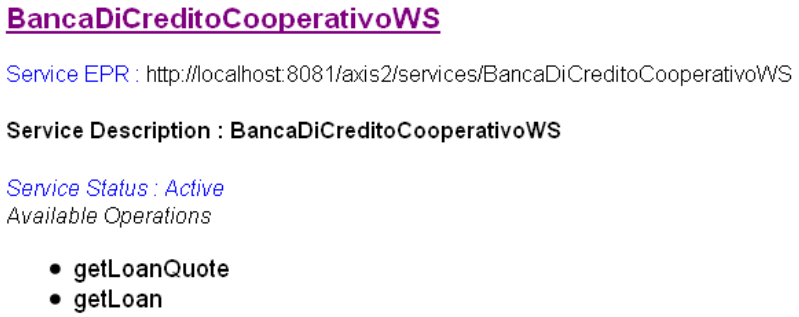
La cartella `classes` contiene la cartella *META-INF* che contiene i file *service.xml* e *BancaDiCreditoCooperativo.wsdl* e la `cartella banca_di_credito_cooperativo` contenenti tutti i file *.class* del progetto.



La cartella `lib` contiene invece il file BancaDiCreditoCooperativo.aar da utilizzare per il deploy.

6. Deploy

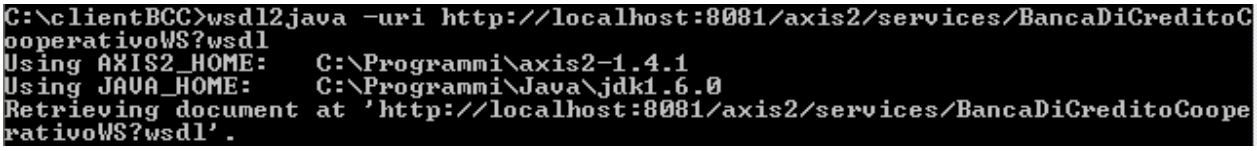
Per effettuare il deploy del nuovo servizio in Axis2 con Apache Tomcat come application server non rimane che copiare il file .aar appena generato in `C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\axis2\WEB-INF\services`.
Accedendo alla pagina http://localhost:8081/axis2/services/listServices contenente la lista dei servizi deployati con Axis2 è possibile verificare che il procedimento seguito è stato svolto correttamente e il nostro servizio è stato deployato nell'application server Apache Tomcat:



3.9.2 Client-side sincrono(Blocking API)

1. Genero lo stub per invocare il serivzio a parire dalla descrizione del file WSDL

```
C:\clientBCC>wsdl2java -uri
http://localhost:8081/axis2/services/BancaDiCreditoCooperativoWS?wsdl
```



In questo modo vengono generati il file `build.xml` e la cartella `src`.
La cartella src contiene la cartella `banca_di_credito_cooperativo` che rappesenta il package del progetto e nella quale sono stati generati i file:

- `BancaDiCreditoCooperativoWSStub`: per un'invocazione sincrona
- `BancaDiCreditoCooperativoWSCallbackHandler`: per un'invocazione asincrona

2. Compilo il client e i file generati

Per compilare il client eseguendo il comando ant bisogna prima copiarlo all'interno della directory di lavoro,ad esempio in C:\clientBCC\src,e poi modificare il file build.xml aggiungendo il seguente frammento di codice:

```
<target if="jars.ok" depends="jar.client" name="run.client">
        <path id="test.class.path">
            <pathelement location="${lib}/${name}-test-client.jar"/>
            <path refid="axis2.class.path"/>
        </path>
        <java fork="true" classname="ClientBcc" >
            <classpath refid="test.class.path"/>
```

```
        </java>
    </target>
```

In altre parole ho inserito il nuovo `target` run.client al progetto all'interno del quale ho defiinito la classe `ClientBcc` che rappresenta appunto il client per il servizio.
In questo modo è possibile eseguire il client utillizando tutte le librerie necessarie con il comando `ant run.client`.
Il client che invoca in modalità sincrona il servizio è il seguente:

```
import banca_di_credito_cooperativo.*;

public class ClientBcc {
public static void main(String[] args)throws Exception{

int durata = 10;
int ammontare=20000;
String ssn="111-333-555-777";


BancaDiCreditoCooperativoWSStub stub2 = new BancaDiCreditoCooperativoWSStub ();


BancaDiCreditoCooperativoWSStub.BankQuoteRequest bpr=new
BancaDiCreditoCooperativoWSStub.BankQuoteRequest();

BancaDiCreditoCooperativoWSStub.GetLoanQuote q=new
BancaDiCreditoCooperativoWSStub.GetLoanQuote();

bpr.setCreditHistory(5);
bpr.setCreditRisk("Low");
bpr.setCreditScore(900);
bpr.setLoanAmount(ammontare);
bpr.setLoanDuration(durata);
bpr.setSsn(ssn);
q.setParam0(bpr);

BancaDiCreditoCooperativoWSStub.BankQuoteResponse
b=stub2.getLoanQuote(q).get_return();

System.out.println("Interest Rate:"+b.getInterestRate());
System.out.println("Id:"+b.getQuoteId());
System.out.println("Error code :"+b.getErrorCode());
 }
}
```

A questo punto posso compilare i file:

```
C:\clientBCC>ant
```

```
C:\clientBCC>ant
Buildfile: build.xml

init:
    [mkdir] Created dir: C:\clientBCC\build
    [mkdir] Created dir: C:\clientBCC\build\classes
    [mkdir] Created dir: C:\clientBCC\build\lib

pre.compile.test:
     [echo] Stax Availability= true
     [echo] Axis2 Availability= true

compile.src:
    [javac] Compiling 3 source files to C:\clientBCC\build\classes
    [javac] Note: C:\clientBCC\src\banca_di_credito_cooperativo\BancaDiCreditoC
operativoWSStub.java uses unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

jar.client:
      [jar] Building jar: C:\clientBCC\build\lib\BancaDiCreditoCooperativoWS-te
t-client.jar

BUILD SUCCESSFUL
Total time: 12 seconds
```

Ed eseguire il client:

```
C:\clientBCC>ant run.client
```

```
C:\clientBCC>ant run.client
Buildfile: build.xml

init:

pre.compile.test:
     [echo] Stax Availability= true
     [echo] Axis2 Availability= true

compile.src:

jar.client:

run.client:
     [java] log4j:WARN No appenders could be found for logger (org.apache.axis2.
description.AxisService).
     [java] log4j:WARN Please initialize the log4j system properly.
     [java] Interest Rate:5.0
     [java] Id:Banca di Credito Cooperativo6200799999
     [java] Error code :0

BUILD SUCCESSFUL
Total time: 2 seconds
C:\clientBCC>
```

Eseguendo TcpMonitor è possibile analizzare i messaggi SOAP di richiesta e di risposta generati :

```
java org.apache.axis.utils.tcpmon 7080 localhost 8081
```

dopo aver modificato la porta del servizio nel file che rappresenta lo stub da 8081 a a7080.

```
==============
Listen Port: 7080
Target Host: localhost
Target Port: 8081
==== Request ====
POST /axis2/services/BancaDiCreditoCooperativoWS/ HTTP/1.1
Content-Type: application/soap+xml; charset=UTF-8; action="urn:getLoanQuote"
User-Agent: Axis2
Host: localhost:7080
Transfer-Encoding: chunked


<?xml version='1.0' encoding='UTF-8'?>
   <soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
      <soapenv:Body>
         <ns2:getLoanQuote xmlns:ns2="http://banca_di_credito_cooperativo">
            <ns2:param0>
```

```
               <ns1:creditHistory
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">5</ns1:creditHistory>
               <ns1:creditRisk
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">Low</ns1:creditRisk>
               <ns1:creditScore
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">900</ns1:creditScore>
               <ns1:loanAmount
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">20000</ns1:loanAmount>
               <ns1:loanDuration
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">10</ns1:loanDuration>
               <ns1:ssn
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">111-222-333-444</ns1:ssn>
            </ns2:param0>
         </ns2:getLoanQuote>
      </soapenv:Body>
   </soapenv:Envelope>

==== Response ====
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/soap+xml;
action="urn:getLoanQuoteResponse";charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 28 Oct 2008 14:29:10 GMT


<?xml version='1.0' encoding='UTF-8'?>
   <soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
      <soapenv:Body>
         <ns2:getLoanQuoteResponse
xmlns:ns2="http://banca_di_credito_cooperativo">
            <ns2:return>
               <ns1:errorCode
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">0</ns1:errorCode>
               <ns1:interestRate
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">5.0</ns1:interestRate>
               <ns1:quoteId
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">Banca di Credito
Cooperativo8149499999</ns1:quoteId>
            </ns2:return>
         </ns2:getLoanQuoteResponse>
      </soapenv:Body>
   </soapenv:Envelope>
==============
```

3.9.3 Client-side asincrono(Non-blocking API)

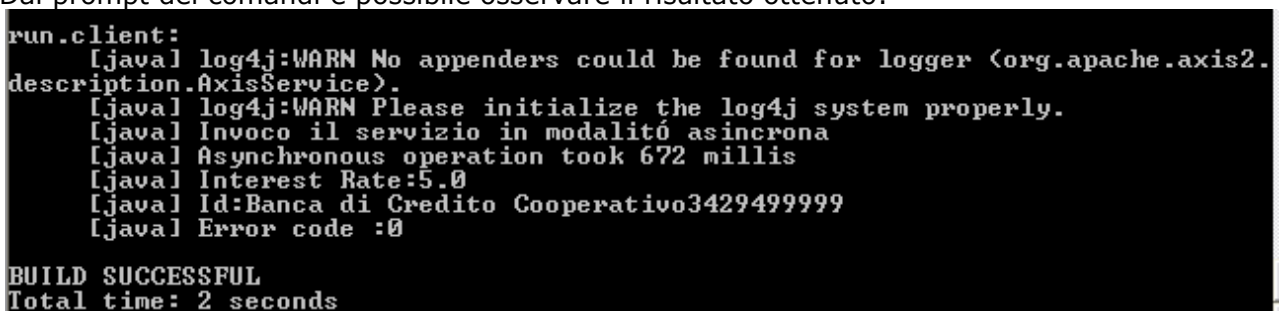
Dopo aver implementato un client che invochi il servizio in modalità sincrona,vediamo ora come invocare lo stesso servizio in modalità non bloccante utilizzando i metodi messi a disposizione dalla classe BancaDiCreditoCooperativoWSCallbackHandler per implementare appunto questo tipo di comunicazione.

```
import banca_di_credito_cooperativo.*;

public class ClientBcc {
public static void main(String[] args)throws Exception{

int durata = 10;
int ammontare=20000;
String ssn="111-222-333-444";

BancaDiCreditoCooperativoWSStub stub2 = new BancaDiCreditoCooperativoWSStub ();

CallbackHandler cb= new CallbackHandler();
```

```
BancaDiCreditoCooperativoWSStub.BankQuoteRequest bpr=new
BancaDiCreditoCooperativoWSStub.BankQuoteRequest();
BancaDiCreditoCooperativoWSStub.GetLoanQuote q=new
BancaDiCreditoCooperativoWSStub.GetLoanQuote();
bpr.setCreditHistory(5);
bpr.setCreditRisk("Low");
bpr.setCreditScore(900);
bpr.setLoanAmount(ammontare);
bpr.setLoanDuration(durata);
bpr.setSsn(ssn);
q.setParam0(bpr);
System.out.println("Invoco il servizio in modalità asincrona");
```
Metodo messo a disposizione dalla classe stub che permette di invocare il servizio in modalità non bloccante.
```
stub2.startgetLoanQuote(q, cb);
        long start = System.currentTimeMillis();
        synchronized (cb) {
            while (!cb.m_done) {
                try {
                    cb.wait(100);
                } catch (Exception e) {}
            }
        }
```
A questo punto ho ricevuto la risposta del servizio che ha impiegato un certo tempo.
```
        System.out.println("Asynchronous operation took " +
            (System.currentTimeMillis()-start) + " millis");
        if (cb.m_response != null) {
```
Gestisco la risposta ricordando che tale metodo ritornava un complexType BankQuoteResponse
```
BancaDiCreditoCooperativoWSStub.BankQuoteResponse b=cb.m_response.get_return();
System.out.println("Interest Rate:"+b.getInterestRate());
System.out.println("Id:"+b.getQuoteId());
System.out.println("Error code :"+b.getErrorCode());
        } else {
            System.out.println("Returned exception:");
            cb.m_exception.printStackTrace(System.out);
        }

}
public static class CallbackHandler extends
BancaDiCreditoCooperativoWSCallbackHandler
    {
        public boolean m_done;
        private Exception m_exception;
        public BancaDiCreditoCooperativoWSStub.GetLoanQuoteResponse m_response;

        public CallbackHandler() {
            super(null);
        }

        public synchronized void
receiveResultgetLoanQuote(BancaDiCreditoCooperativoWSStub.GetLoanQuoteResponse
rsp) {
            m_response = rsp;
            m_done = true;
        }

        public synchronized void receiveErrorgetLoanQuote(Exception e) {
            m_exception = e;
            m_done = true;
```

```
        }
    }
}
```

Dal prompt dei comandi è possibile osservare il risultato ottenuto:



3.9.4 Client-side asincrono(Non-blocking API,Dual Transport)

Un modo per risolvere il problema del read timeout ed utilizzare correttamente lo standard del WS-Addressing è quello di aggiungere le seguenti righe al client Non-Blocking analizzato precedentemente:

```
1. stub2._getServiceClient().getOptions().setUseSeparateListener(true);
2. stub2._getServiceClient().engageModule(new QName("addressing"));
3. stub2._getServiceClient().getOptions().setTransportInfo(Constants.TRANSPORT
   _HTTP,Constants.TRANSPORT_HTTP, true);
```

Nella riga 1. si informa l'engine di Axis2 che i messaggi SOAP di richiesta e di risposta utilizzeranno due canali di trasporto differenti.
Nella riga 2. viene caricato il modulo che implementa lo standard WS-Addressing.
Nella riga 3. viene specificato che devono essere creati due i canali di trasporto(opzione `true`) per i messaggi SOAP di richiesta e risposta e che entrambi utilizzano il protocollo HTTP.

Per capire bene come lo standard è stato effettivamente utilizzato per implementare una comunicazione asincrona analizziamo con TcpMonitor i messaggi generati e scambiati.

Messaggio SOAP di richiesta

```
<?xml version='1.0' encoding='UTF-8'?>
   <soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
```

Una prima differenza con i messaggi SOAP visti in precedenza consiste nel fatto che in questo caso viene importato anche il namespace del WS-Addressing con prefisso `wsa`:

```
      <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
```

Di seguito sono invece riportati gli elementi che caratterizzano lo standard e permettono di creare un meccanismo di correlazione tra la request e la response:

- `<wsa:To>`:indica l'endpoint a cui è destinato il messaggio.
- `<wsa:ReplyTo>`:specifica la locazione a cui inviare la risposta.
- `<wsa:MessageID>`:indica l' identificatore univoco del messaggio.
- `<wsa:Action>`:indica l'operazione o l'azione che deve esser eseguita per il messaggio.

```
<wsa:To>http://192.168.0.4:7080/axis2/services/BancaDiCreditoCooperativoWS/</wsa
:To>
<wsa:ReplyTo>
<wsa:Address>http://192.168.0.4:6060/axis2/services/BancaDiCreditoCooperativoWS1
228577315953_1/</wsa:Address>
         </wsa:ReplyTo>

         <wsa:MessageID>urn:uuid:BB414EA7AB6F3F2F951228577317169</wsa:MessageID>

         <wsa:Action>urn:getLoanQuote</wsa:Action>
      </soapenv:Header>
```

Messaggio HTTP di risposta

Questo messaggio viene inviato dal server al client ed indica che il messaggio di richiesta è stato ricevuto ma non ancora processato.

```
HTTP/1.1 202 Accepted
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=UTF-8
Content-Length: 0
Date: Sat, 06 Dec 2008 15:28:55 GMT
```

Il servizio utilizzerà le informazioni `addressing` contenute nell'header del messaggio SOAP di richiesta per inviare la risposta al corretto client su un altro canale di trasporto.

Messaggio SOAP di risposta

Nel messaggio SOAP di richiesta è possibile notare che la replica al messaggio deve essere inoltrata veso la porta 6060:per intercettare ed analizzare il messaggio di risposta del server ho utilizzato ancora una volta TcpMonitor mettendomi in ascolto sulla porta 6060.
Agli elementi dello standard descritti precedentemente si aggiunge:

- `<wsa:RelatesTo>`: contiene il MessageId della richiesta;indica che il messaggio di risposta si riferisce ad un precedente messaggio di richiesta.

```
==============
Listen Port: 6060
Target Host: localhost
Target Port: 8081
==== Request ====
POST /axis2/services/BancaDiCreditoCooperativoWS1228580490718_1/ HTTP/1.1
Content-Type: application/soap+xml; charset=UTF-8;
action="urn:getLoanQuoteResponse"
User-Agent: Axis2
Host: localhost:6060
Transfer-Encoding: chunked

<?xml version='1.0' encoding='UTF-8'?>
   <soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
      <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">

<wsa:To>http://192.168.0.4:6060/axis2/services/BancaDiCreditoCooperativoWS122858
0490718_1/</wsa:To>
         <wsa:ReplyTo>
            <wsa:Address>http://www.w3.org/2005/08/addressing/none</wsa:Address>
         </wsa:ReplyTo>
         <wsa:MessageID>urn:uuid:5C44DA85AD7ED630291228580492714</wsa:MessageID>
         <wsa:Action>urn:getLoanQuoteResponse</wsa:Action>
         <wsa:RelatesTo>urn:uuid:BB414EA7AB6F3F2F951228577317169</wsa:RelatesTo>
      </soapenv:Header>
      <soapenv:Body>
         <ns2:getLoanQuoteResponse
xmlns:ns2="http://banca_di_credito_cooperativo">
            <ns2:return>
               <ns1:errorCode
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">0</ns1:errorCode>
               <ns1:interestRate
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">5.0</ns1:interestRate>
               <ns1:quoteId
xmlns:ns1="http://To.banca_di_credito_cooperativo/xsd">Banca di Credito
Cooperativo455899999</ns1:quoteId>
            </ns2:return>
         </ns2:getLoanQuoteResponse>
```

```
    </soapenv:Body>
</soapenv:Envelope>
```

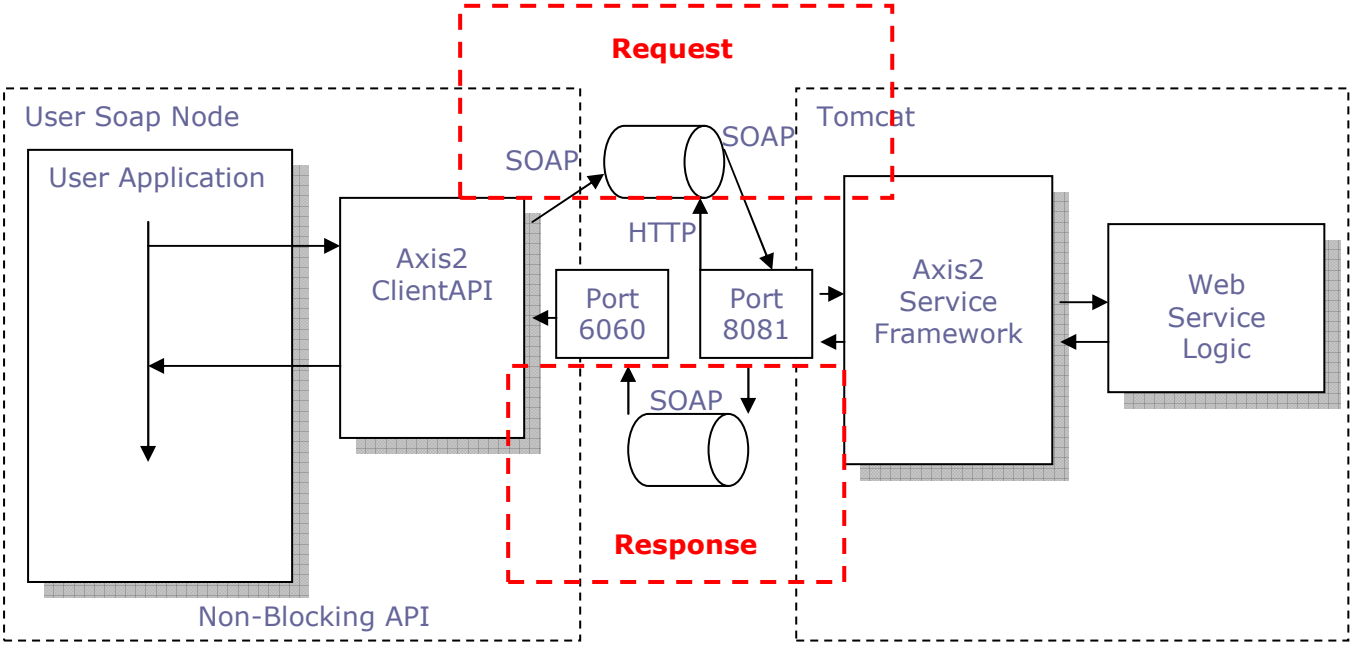


Figura 39: Comunicazione Non-Blocking,Dual Transport

CAPITOLO 4 - Enterprise Java Beans e Web Services

4.1 Introduzione

Nel capitolo precedente sono stati affrontati gli aspetti dell'impiego del tool Apache Axis per la creazione/gestione dei Web services. Con questi concetti è quindi possibile scrivere ex novo applicazioni raggiungibili da ogni punto del Web e con qualsiasi tecnologia: .Net, php, etc. Ora, però, è necessario voltarsi un attimo e soffermarsi sull'utilità dei Web services e sul dilagante successo di SOAP. Infatti bisogna tenere presente l'attitudine all'integrazione dei Web services per cui, ipoteticamente, nessuna applicazione già scritta deve essere buttata via per ricrearla con i nuovi standard. Il tema dell'integrazione rappresenta, in sintesi, il vero stimolo che ha portato all'affermazione delle tecnologie XML, SOAP, WSDL come substrato della filosofia Web services.

Questo capitolo ha l'obiettivo di mostrare il lato d'integrazione di Axis verso verso l'architettura J2EE: Enterprise Java Beans e i Web Services. Nella prima parte vengono descritti gli Enterprise Java Beans, la loro classificazione, il loro ciclo di vita, la loro implementazione e il loro deployment. Nell'ultima sezione viene presentata la possibilità di utilizzare gli EJB come dei Web Services, analizzando una possibile soluzione per implementare tale scenario e fornendo un esempio dettagliato sull'utilizzo degli EJB, in cui AXIS funge da processore SOAP lato Client e Server per la comunicazione tra le due applicazioni. Grazie alla combinazione di AXIS e J2EE è possibile ottenere l'implementazione di un sistema distribuito che può basare:

- la robustezza, la sicurezza, l'affidabilità, la gestione delle transazioni e la persistenza delle informazioni, la gestione delle risorse in generale, baandosi esclusivamnte sui servizi offerti dalla piattaforma J2EE.
- l'interoperabilità tra l'applicazione lato server e una specifica applicazione lato client (realizzata in qualsiasi linguaggio di programmazione) grazie all'utilizzo dei Web Services.

Bisogna infatti sottolineare che J2EE è un infrastruttura affidabile, sicura e di alta qualità, utilizzata per la realizzazione di complesse applicazioni distribuite di business. L'unico limite di questo ambiente era legato al fatto che l'implementazione di un meccanismo di comunicazione tra un Client C/C++/C# e un applicazione Server-Side su J2EE era molto complesso da realizzare. Grazie alla nascita dei Web Services e del protocollo SOAP è possibile interfacciare queste applicazioni di business Server-Side con qualsiasi Client.

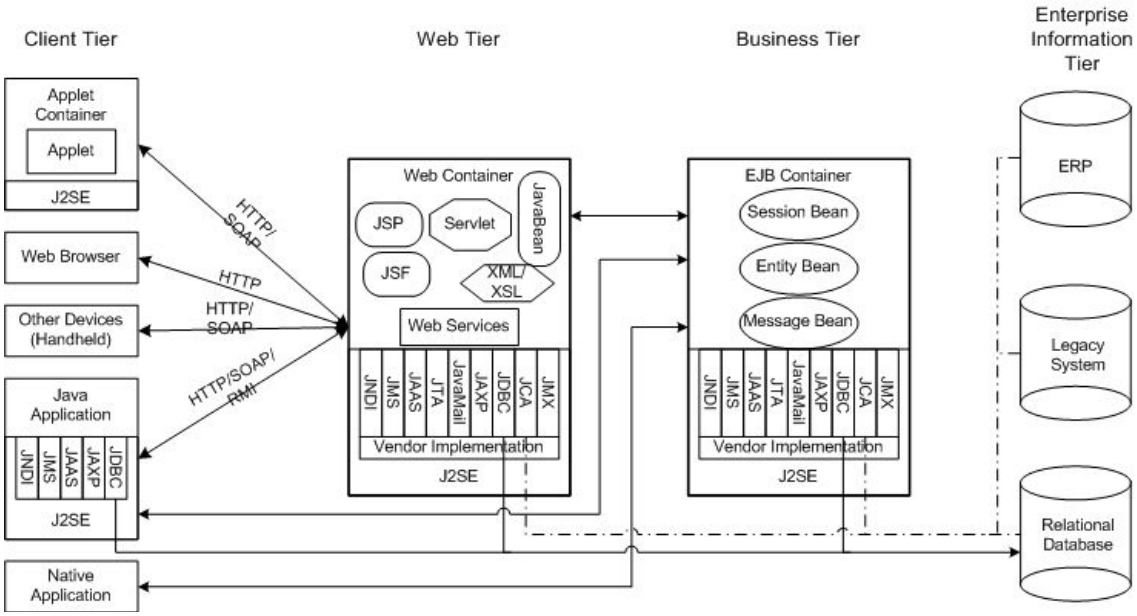


Figura 40:J2EE/Java EE Architecture

Allo stesso tempo è possibile implementare Web Services tramite EJB sia utilizzando una piattaforma come il connubio Application Server Tomcat e Axis sia la piattaforma J2EE.

4.2 Enterprise Java Bean

Gli Enterprise Java Beans sono le componenti più importanti della piattaforma J2EE; rappresentano una vera e propria tecnologia per l'implementazione della logica di business di un sistema distribuito ad oggetti: La SUN definisce così gli EJB:

"La architettura EJB è una architettura a componenti per lo sviluppo e la realizzazioni di applicazioni business distribuite basate su componenti. Le applicazioni scritte con EJB sono scalabili, transazionali e sicure. Queste applicazioni possono essere scritte una volta e poi portate su ogni piattaforma server che supporti lo standard EJB".

Possiamo dare una definizione molto più ridotta ma comunque esplicativa degli EJB:

"EJB è un modello di programmazione a componenti utilizzato per implementare la logica di business delle nostre applicazioni server side ".

Possiamo classificare gli EJB in tre categorie :

- Session bean
- Entity bean
- Message-driven bean

I Session Bean rappresentano logicamente una sessione di operazioni di cui un applicazione client o un altro bean può aver bisogno;praticamente sono delle classi che implementano la logica di business in un' applicazione server-side. Se pensiamo alla descrizione di uno scenario da implementare a livello software i session bean con i loro metodi implementano le azioni, rappresentate dai *verbi* della discussione.
Gli Enity Bean rappresentano logicamente i dati di business tipici di un applicazione distribuita;praticamente sono delle classi che fisicamente sono mappate in tabelle di un database relazionale e le loro istanza sono le tuple delle corrispondenti tabelle. Se pensiamo alla descrizione di uno scenario da implementare a livello software gli entity bean con i loro metodi implementano i soggetti e i sostantivi dell'analisi logica del periodo.
I Message Driver Bean sono molto simili ai Session Bean con la differenza che non hanno metodi che vengono invocati "da remoto" da un applicazione client, ma gestiscono la comunicazione con altri sistemi o all'interno dello stesso container tramite lo scambio di messaggi asincroni e l'utilizzo del protocollo JMS.
Bisogna fare un ulteriore distinzione all'interno degli EJB che riguarda gli EJB locali e quelli remoti**.** Gli EJB locali possono essere chiamati all'interno dello stesso container da altri bean mentre quelli remoti possono essere invocati da altre applicazioni tramite la rete e l'utilizzo di protocolli distribuiti come RMI-IIOP; quest'ultimo lega il modello ad oggetti di JAVA(RMI) e il protocollo di trasporto di CORBA (IIOP). Un EJB può essere sia accessibile in locale che da remoto, sostanzialmente devono essere implementate interfaccie locali e remote (dopo vedremo un esempio). La distinzione tra EJB locali e EJB remoti ha senso perché è possibile garantire la comunicazione tra oggetti all'interno dello stesso container senza ricorrere a protocolli di networking, garantendo prestazioni più efficienti.

Un'applicazione distribuita può essere tipicamente composta da:

- La componente di dialogo con un database (local entity bean)
- La componente di supporto alla logica di business (*stateless* local session bean)
- La componente che implementa la logica di business distribuita (*stateless o stefeful* remote session bean)
- La componente che gestisce l'invio e la ricezione di messaggi con altri sistemi(messagedriven beans)

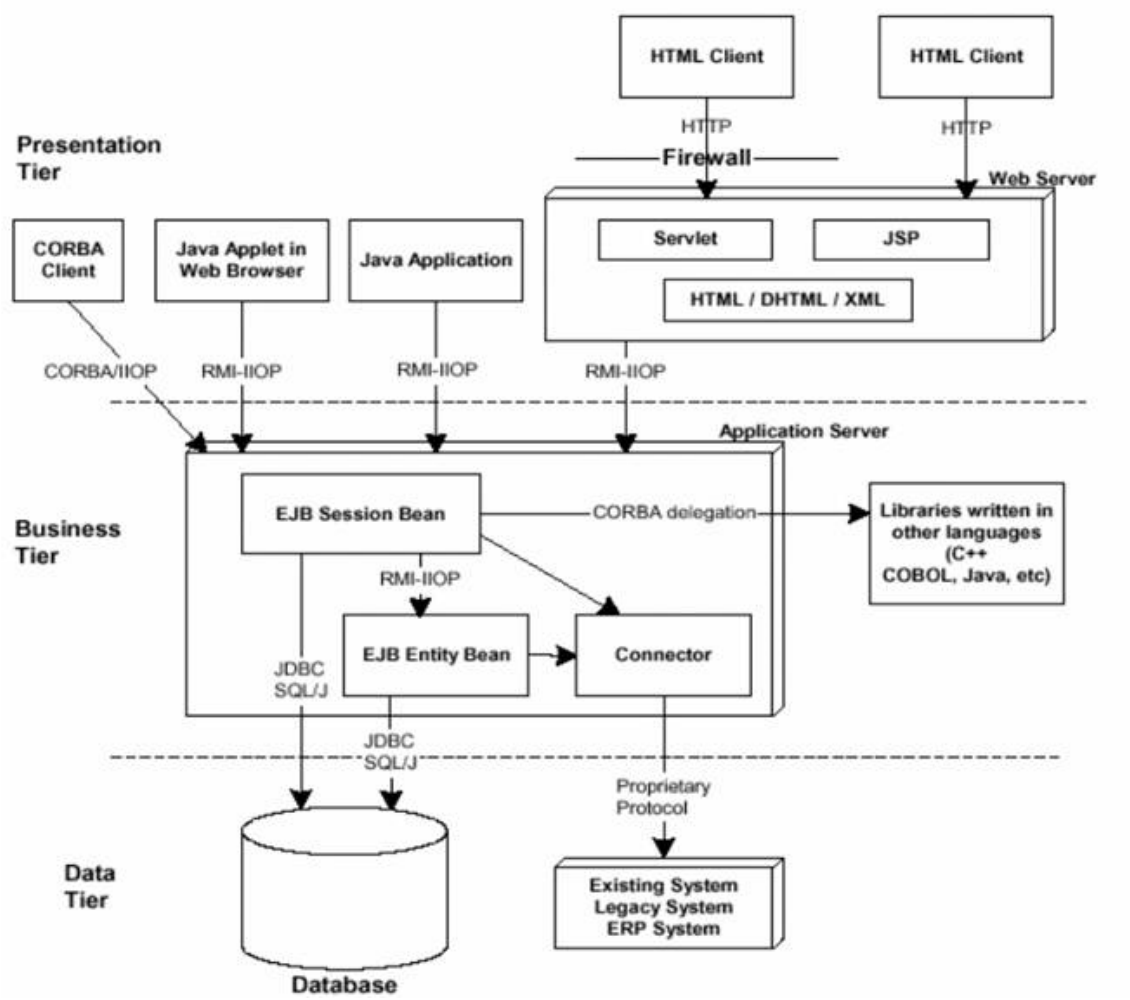


Figura 41: Ejb nel modello 3-tier

4.3 EJB e Web Services

Come detto in precedenza,gli ejb sono stati progettati per avere un alto grado d'interoperabilità verso altri linguaggi di programmazione, affinché sia possibile integrare in un'unica piattaforma sistemi sviluppati con tecnologie diverse, soprattutto per il riuso di sistemi legasy.
In modo nativo un sistema sviluppato con ejb permette di interagire tramite:

- Applicazioni *browser-based,* in cui il lato server è implementato con JSP e Servlets;
- Applicazioni basate sull'utilizzo di RMI-IIOP per la comunicazione tra client e EJB lato Server (implicitamente si ha così l'integrazione in sistemi sviluppati con CORBA);
- Applicazioni basate sullo *scambio di messaggi* utilizzando JMS.

4.3.1 Esportare gli EJB tramite web services

In maniera nativa,l'accesso agli EJB tramite un client java avviene tramite il protocollo RMI (*Remote Method Invocation*) su IIOP (*Internet Inter-ORB Protocol*). Per permettere ad un client non java di accedere all'ambiente degli EJB utilizzando diversi protocolli di rete,in particolare HTTP,si può pensare di introdurre una sintassi XML. Quello che Sun ha realizzato è semplicemente l'incapsulamento degli EJB in un involucro che è compatibile con SOAP: di fatto un EJB viene quindi trasformato in un web service che sfrutta le potenzialità offerte e la flessibilità e portabilità di entrambe le tecnologie, permettendo in particolare che ambienti eterogenei si scambino informazioni in formato XML,usando come trasporto tecnologie Internet (HTTP,ma anche FTP, SMTP).

A causa della natura senza stati e senza connessione del protocollo SOAP,i bean di sessione senza stati (stateless session bean) sono i migliori candidati per essere esposti come web services.

Necessità minima per un EJB al fine di poter essere utilizzato come web service è un sistema di comunicazione XML che supporti SOAP.

Le soluzioni principali per esportare gli ejb tramite web services sono sostanzialmente le seguenti:

Soluzioni proprietarie degli EJB container

Quasi tutti i container permettono di esportare i propri EJB come web services. Nel caso di SUN JavaTM System Application Server Platform Edition 81,all'atto del deployment si specifica di voler esportare l'ejb come web services. Il tool chiederà di inserire il file WSDL che descrive il servizio ed un file XML per ulteriori informazioni che serviranno al container all'atto del deployment (stessa funzionalità del file WSDD di Axis).

Utilizzare un motore SOAP esterno all'EJB container

Wrapper

Utilizzando un motore SOAP esterno all'EJB container,è possibile implementare un web services che fa da wrapper tra gli EJB ed il client SOAP. Il web services implementerà al suo interno tutta la logica per l'accesso all'EJB. La presente soluzione è sempre applicabile,con qualsiasi sistema si utilizza. In realtà è molto più utile quando non si vuole esportare esattamente la logica implementata negli EJB,mascherandone parte di essa.

Soluzione offerta da AXIS

Axis permette d'interagire con gli EJB di tipo Session Stateless in modo semi-automatico, tramite l'utilizzo di uno dei suoi provider. Precisamente il provider "Java:EJB". Non serve altro che scrivere un file WSDD per il deployment su axis dei servizi che si vogliono esportare inserendo come parametri al servizio i seguenti attributi:

- `beanJndiName`*:* il nome registrato nell'architettura JNDI del Session Bean Stateless con cui si vuole interagire;
- `homeInterfaceName`*:* il nome dell'interfaccia home remota;
- `remoteInterfaceName`*:* il nome dell'interfaccia remota del servizio;
- `jndiContextClass`*:* il nome della classe che implementa l'interfaccia JNDI;
- `jndiURL`*:* l'url del server di naming.

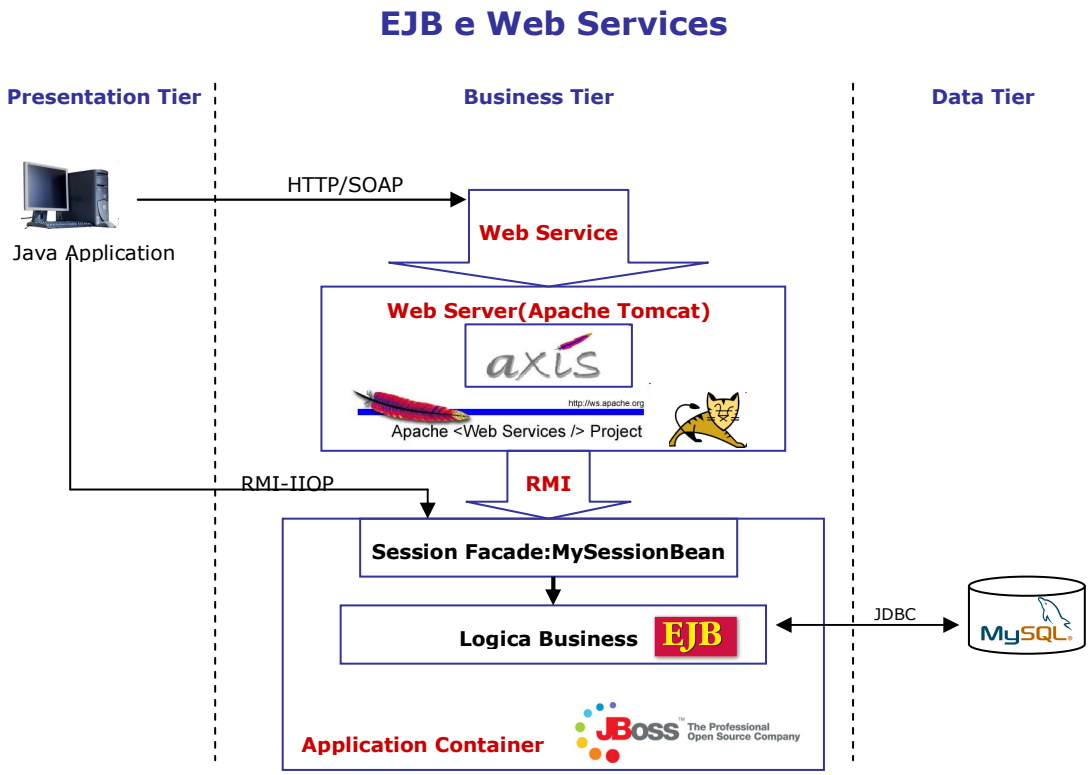


Figura 42:EJB esposto come Web Service

Nota:Attualmente solo gli Stateless Bean possono essere utilizzati come endpoint per un Web Services come specificato in JSR-109.
JSR-109[38] consente di definire i requisiti di deploy per EJB che vogliono essere esposti come Web Service.

4.4 Axis e JBoss

Per implemtare Web Services in JBoss sfruttando le funzionalità offerte da Axis bisogna innanzitutto configurarli correttamente. Per installare Axis all'inerno di JBoss basta copiare la cartella `axis-1_4` all'interno di `C:\Programmi\jboss-4.0.3SP1\server\default\deploy` rinominandola `axis.war`.
`La cartella axis-1_4` deve essere rinominata in `axis.war` per permettere a JBoss di riconoscerla come web application e quindi al web container di effettuarne il deploy a run-time.

Sotto:
`WEB-INF\lib`
Occorre, invece inserire le librerie:

- jboss-client.jar
- jboss-j2ee.jar
- jbossmq-client.jar
- jbosssx-client.jar
- jndi.jar
- jnp-client.jar

con cui Axis è in grado di interrogare l'application container su cui gira l'EJB tramite protocollo RMI.

4.5 JBoss - Database

4.5.1 Configurazione di JBoss

JBoss utilizza una filosofia di gestione delle risorse completamente diversa rispetto a Tomcat. In JBoss una applicazione EJB, una web application o un file di configurazione XML sono gestiti, in prima istanza, nello stesso modo: devono essere deployati all'interno dell'application server il quale potrà prenderli in gestione anche in modo dinamico "a caldo". Senza addentrarmi ulteriormente nella spiegazione del funzionamento di JBoss limiterò la trattazione ai passi necessari per specificare una sorgente dati e deployarla all'interno del server.

4.5.2 Deploy di un file di configurazione -DS

Per poter configurare una sorgente dati è necessario copiare un file XML il cui nome ricalca il seguente schema

```
<nome>-ds.xml
```

dove <nome> per convenzione ricalca quello della sorgente dati che si sta definendo. All'interno di tale file vengono inserite tutte le informazioni necessarie per definire il nome JNDI e per configurare l'accesso al database. All'interno della distribuzione nella directory

```
JBOSS_HOME/docs/examples/jca
```

è possibile trovare molti file di configurazione di esempio per tutti i principali database disponibili in questo momento: da MySQL e Oracle,DB2, Postgres, Hypersonic DB, Informix e perfino SQL Server. Osservando il contenuti di tali file potranno essere facilmente create le configurazioni di per i casi più disparati. Ad esempio per definire la sorgente dati MyDS di un database MySQL si potrà utilizzare un file fatto nel seguente modo;

```
<datasources>
<local-tx-datasource>
<jndi-name>MyDS</jndi-name>
<connection-url>jdbc:mysql://localhost/community</connection-url>
<driver-class>org.gjt.mm.mysql.Driver</driver-class>
<user-name>admin</user-name>
<password>admin</password>
</local-tx-datasource>
</datasources>
```

In questo caso si usa un contesto transazionale locale dato che il driver per MySQL non supporta le transazioni distribuite. Consultando uno dei file di esempio si potranno verificare i parametri tramite i quali configurare la connessione con strumenti alternativi (es. OCI).
Interessante notare come in questo caso si abbia maggior libertà nello specificare funzionalità aggiuntive (come il comando SQL da eseguire alla prima connessione o nel momento dell'ottenimento dal pool). Di fatto queste variazioni sono prerogative del driver utilizzato e non sono in alcun modo dipendenti da JBoss o dalla specifica J2EE.

4.5.3 MySQL Database Configuration

Per utilizzare correttamtne Jboss 4.0 con MySQL,bisogna copiare il file `mysql-connector-java-5.1.6-bin.jar` che rappresenta il driver del database nella cartella `C:\Programmi\jboss-4.0.3SP1\server\default\lib`.
Siccome abbiamo messo un nuovo JAR nella directory lib , riavviamo JBoss per assicurarci che il driver venga inserito nel classpath.
Per utilizzare il data source MyQSL,bisogna poi copiare il file .xml `docs/examples/jca/mysql-ds.xml/` in `C:\Programmi\jboss-4.0.3SP1\server\default` e modificare tale file in questo modo:

```
setting |<driver-class/>| to |com.mysql.jdbc.Driver|
|<connection-url/>| to |jdbc:mysql://<mysqlhost>/<database>|,
dove |<mysqlhost>| è l'host del server MySQL
```

e |<database>| è il database MySQL.

Successivamente bisogna configurare gli elementi |<datasource>| e |<type-mapping>| contenuti all'interno del file /standardjaws.xml/ o /jaws.xml/ nella cartella C:\Programmi\jboss-4.0.3SP1\server\default\conf:

```
|<jaws>
  <datasource>java:/MySqlDS</datasource>
  <type-mapping>mySQL</type-mapping>
</jaws> |
```

Dobbiamo configurare anche gli elementi |<datasource>| e |<datasource-mapping>| elements all'interno del file /standardjbosscmp-jdbc.xml/ o /jbosscmp-jdbc.xml/:

```
|<jbosscmp-jdbc>
    <defaults>
       <datasource>java:/MySqlDS</datasource> java:jdbc/ardeadb
       <datasource-mapping>mySQL</datasource-mapping>
  </defaults>
</jbosscmp-jdbc>|
```

Infine non rimane che modificare il file /login-config.xml/ con i MySQL database settings. Agiungiamo il seguente elemento |<application-policy/>| nel file /login-config.xml/:

```
|<application-policy name = "MySqlDbRealm">
   <authentication>
      <login-module code =
  "org.jboss.resource.security.ConfiguredIdentityLoginModule"
                           flag = "required">
         <module-option name ="principal">sa</module-option>
         <module-option name ="userName">sa</module-option>
         <module-option name ="password"></module-option>
         <module-option name ="managedConnectionFactoryName">
            jboss.jca:service=LocalTxCM,name=MySqlDS
         </module-option>
      </login-module>
   </authentication>
</application-policy>  |
```

Dopo aver modificato i file /mysql-ds.xml/, /standardjaws.xml/, /standardjbosscmp-jdbc.xml/, e /login-config.xml/ files,il server JBoss 4.0 è configurato per essere utilizzato con il database MySQL.

4.5.4 Indirezione a due livelli

Quanto visto fino a questo punto rappresenta la prima parte del processo di configurazione. Il container è infatti in grado adesso di esporre un nome (JNDI) logico al quale viene associata una connessione verso un database. Per migliorare l'accoppiamento fra l'applicazione e l'application server il passo successivo è quello di definire all'interno del file di deploy della applicazione web o EJB un nome che corrisponderà in qualche modo a quanto specificato all'interno dell'application server. In questo modo migliora il disaccoppiamento della applicazione dall'ambiente di esecuzione e si favorisce di gran lunga la portabilità della applicazione stessa. Per fare questo all'interno di ogni applicazione J2EE (web o EJB) si associa il nome pubblico definito nella configurazione dell'application server, con un nome privato che sarà quello acceduto da codice. In questo modo il codice Java scritto farà sempre riferimento in ogni punto della applicazione al nome privato della applicazione e mai a quello definito nel server: se tale nome dovesse cambiare, sarà sufficiente modificare il file di deploy (XML) della applicazione e non sarà necessario in alcun modo modificare il codice applicativo. Questo duplice livello di indirezione si ottiene in modo differente a seconda che si tratti di una web application o di una applicazione EJB.

Configurazione della applicazione: il caso di una applicazione EJB

Per prima cosa è necessario definire all'interno della applicazione il legame con il nome pubblico definito all'interno del container:questa cosa può essere fatta modificando il file di deploy proprietario dell'application server.
La procedura per configurare questo nome interno può essere fatta a livello globale per ogni bean session o entity e varia a seconda che si usi BMP o CMP. Vediamo il caso di un BMP session bean. Nel file proprietario di deploy,il jboss.xml deve essere modificato in modo da inserire il seguente codice

jboss.xml

```
<ejb-name>MySessionBean</ejb-name>
<jndi-name> MySessionBean </jndi-name>
<resource-ref>
<resource-ref-name>jdbc/MyDataSource</resource-ref-name>
<jndi-name>java:/jdbc/MyDS</jndi-name>
</resource-ref>
```

ejb-jar.xml

```
<resource-ref>
     <description>DB Connection</description>
     <res-ref-name>jdbc/ MyDataSource </res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
    </resource-ref>
```

Analogamente al caso delle applicazione web, anche in questo caso si associa il nome pubblico della datasource `java:/jdbc/MyDS` (nome che segue la convenzione JNDI per risorse di tipo datasource) con un nome locale al session `jdbc/MyDataSource`. In quest'ultimo caso il prefisso jdbc non è obbligatorio ma convenzionale. Fatto questo all'interno del codice, per ricavare la sorgente dati (operazione eseguita normalmente all'interno del metodo setSessionContext()) si potrà scrivere:

```
try {
InitialContext initialContext = new InitialContext();
dataSource = (DataSource) initialContext.lookup("java:comp/env/MyDS");
} catch (Exception ex) {
logger.error(ex);
throw new EJBException("Unable to get DataSource");
}
```

Si noti il prefisso env nella istruzione

```
dataSource = (DataSource) initialContext.lookup("java:comp/env/MyDS");
```

che permette di specificare l'accesso all'ENV di esecuzione del bean stesso. Come per i casi precedenti a questo punto la datasource potrà essere utilizzata per ricavare la connessione ed eseguire le operazioni di lettura scrittura verso il database tramite SQL e JDBC.

Nel caso in cui si utilizzi CMP le cose sono leggermente differenti. Il mapping interno fra nome pubblico della sorgente dati e interno nel codice viene risolto dal container in modo trasparente. Definendo il nome pubblico della sorgente dati nel file jbosscmp-jdbc.xml di fatto non vi sono altre operazioni da eseguire:

```
<jbosscmp-jdbc>
<defaults>
<datasource>java:/MyDS</datasource>
<datasource-mapping>mySQL</datasource-mapping>
</defaults>
```

Essendo in questo caso il mapping a carico del container,non è necessario scrivere nessuna riga di codice Java per accedere alla sorgente dati,nè sarebbe possibile.

4.6 Esempio: BancaDeiPaschiDiSienaWS

Come esempio riprendo quello proposto nel capitolo precedente mostrando nella prima parte i passi necessari per deployarlo come Stateless Session Bean all'interno di JBoss, e nella seconda quelli per esporre il Bean come Web Services. Supponiamo infatti che la banca Monte dei paschi di Siena abbia necessità di implementare una procedura specifica per calcolare il tasso d'interesse legato ad un determinato prestito e una che consenta ai clienti di richiedere effettivamente un prestito richiesto;e supponiamo che la prima scelta progettuale sia quella di fornire tali funzionalità attraverso il protocollo di comunicazione RMI per cercare di implementare in maniera semplice e veloce, una struttura a oggetti distribuiti full-Java (sia il lato client che quello server devono essere realizzati obbligatoriamente utilizzando tale linguaggio). Per questo motivo i due 'servizi' richiesti sono stati inizialmente imlplementati in un Session Bean e la comunicazione tra i client che vi accedono e il server che elabora le richieste avviene tramite il protocollo RMI. Suppoiniamo poi che in un secondo momento i progettisti della precedente soluzione decidano che limitare l'utilizzo dei metodi implementati a client che utilizzino il solo linguaggio java per effettuare le richieste RMI e il fatto di dover configurare in modo appropriato il firewall per permettere la corretta comunicazione non rispecchi più la scelta migliore. E supponiamo che decidano quindi di esporre il bean come Web Services per permettere ad un client implementato in un qualsiasi linguaggio di programmazione di interagire con i metodi esposti e per evitare di configurare il firewall;nei Web Service infatti tale problema non esiste,dato che come protocollo di comunicazione utilizzano il connubbio SOAP-HTTP che permette di far passare il traffico sulla porta 80,unica porta lasciata aperta per default dai firewall.

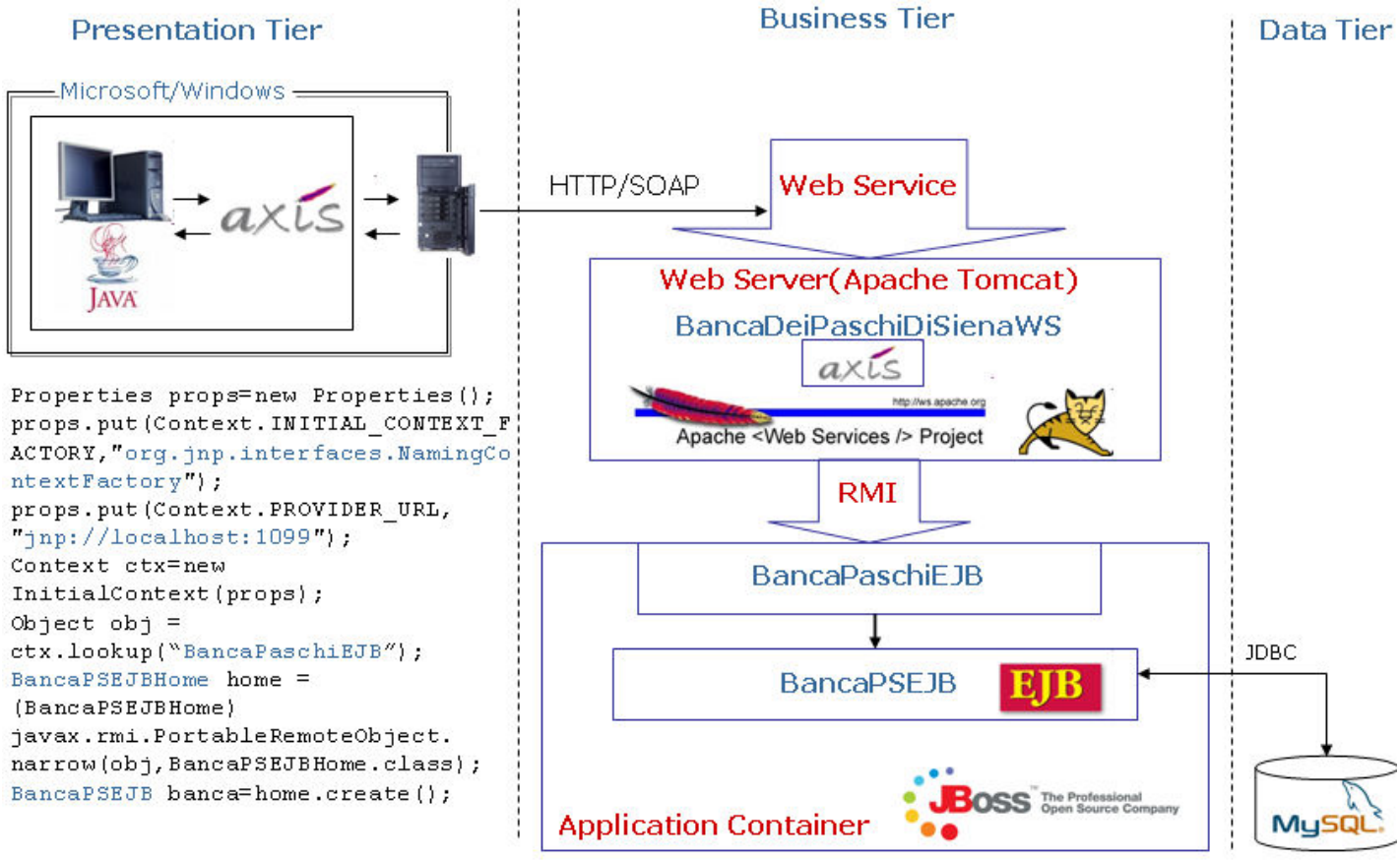


Figura 43:Scenario realizzato

1. Creare lo Stateless Session Bean

BancaPSEJBHome

L'interfaccia Home offre il metodo create, il quale permette di "creare" un istanza del Bean e di accedervi tramite l'interfaccia remota o locale.

```
package banca_dei_paschi_di_siena;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface BancaPSEJBHome extends javax.ejb.EJBHome
{
BancaPSEJB create()
throws java.rmi.RemoteException,
javax.ejb.CreateException;
}
```

BancaPSEJB

```
package banca_dei_paschi_di_siena;
import banca_dei_paschi_di_siena.To.*;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface BancaPSEJB extends javax.ejb.EJBObject
{
BankQuoteResponse getLoanQuote(BankQuoteRequest bqr)throws
java.rmi.RemoteException;
String getLoan(BankLoanRequest blr)throws java.rmi.RemoteException;
}
```

BancaPSEJBBean

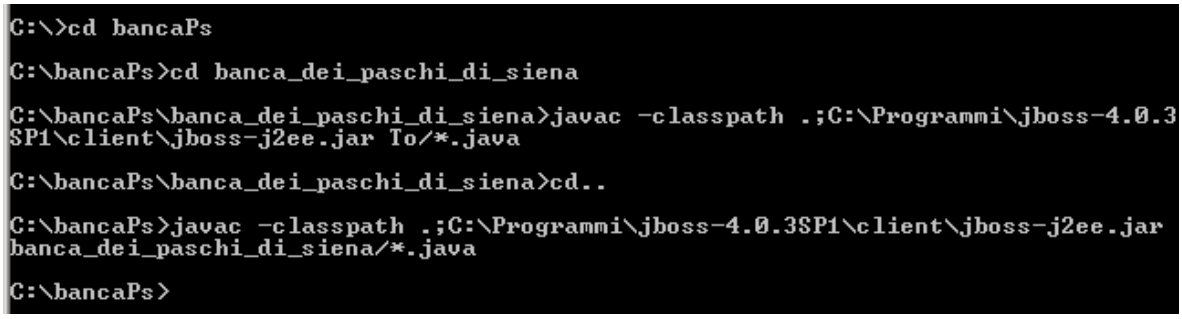
L'implementazione di questi metodi è a carico del container tranne per i metodi di business. A questo punto non ci resta che implementare la logica del bean.

```
package banca_dei_paschi_di_siena;
import banca_dei_paschi_di_siena.To.*;
import javax.ejb.*;
import java.sql.*;
import java.util.*;
import javax.sql.*;
import javax.naming.*;
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class BancaPSEJBBean implements javax.ejb.SessionBean
{

public void ejbCreate(){
System.out.println("Creazione del Bean BancaDeiPaschiDISiena");
}
public void ejbRemove(){
System.out.println("Rimozione del bean BancaDeiPaschiDISiena ");
}
public void ejbActivate() {}
public void ejbPassivate() {}
public void setSessionContext(javax.ejb.SessionContext ctx)
```

```
{}
....Il codice rimanente è lo stesso presentato nell'esempio BancaDiRomaWS
analizzato precedentemente.
```

2. Compilare le classi

La compilazione delle interfacce remota ed home e dell'implementazione richiede le classi definite nel jar:
`{dir. inst}\client\jboss-j2ee.jar`



3. deployment descriptor

Il deployment descriptor è Un file XML che specifica le informazioni sul bean (Deployment descriptor);esistono molti server commerciali EJB che mettono a disposizione dei tools grafici. JBoss non ha un editor XML, ma la costruzione del file, anche se manuale, è veramente semplice:

ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
<enterprise-beans>
<session>
<ejb-name>BancaPSEJB</ejb-name>
<home>banca_dei_paschi_di_siena.BancaPSEJBHome</home>
<remote>banca_dei_paschi_di_siena.BancaPSEJB</remote>
<ejb-class>banca_dei_paschi_di_siena.BancaPSEJBBean</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Container</transaction-type>
<resource-ref>
     <description>DB Connection</description>
     <res-ref-name>jdbc/ardeadb</res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
    </resource-ref>
</session>
</enterprise-beans>
</ejb-jar>
```

Il deployment descriptor specifico per Jboss, il file XML che specifica le informazioni per il look-up con JNDI è invece il seguente:

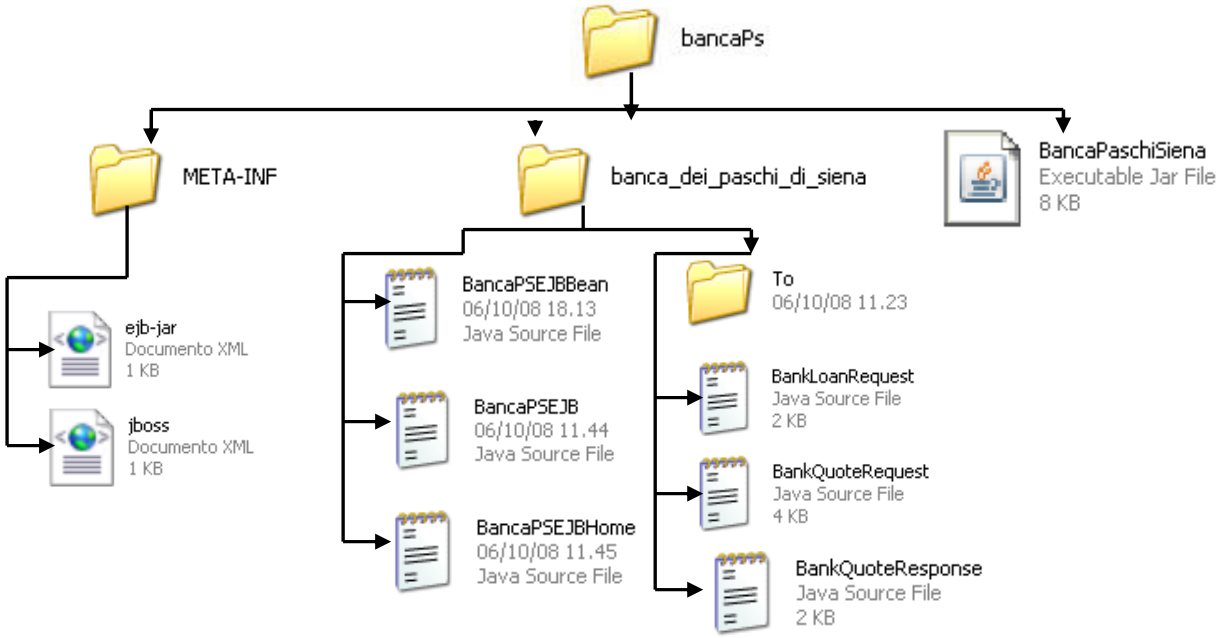
jboss.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<jboss>
<enterprise-beans>
<session>
<ejb-name>BancaPSEJB</ejb-name>
<jndi-name>BancaPaschiEJB</jndi-name>
```

```
<resource-ref>
<res-ref-name>jdbc/ardeadb</res-ref-name>
<jndi-name>java:jdbc/ardeadb</jndi-name>
</resource-ref>
</session>
</enterprise-beans>
</jboss>
```

3. packaging del Bean

Per poter fare il deploy del Session su JBoss occorre creare un file JAR oppure una directory con estensione .jar con la seguente struttura:




```
C:\bancaPs>set JARLIST=banca_dei_paschi_di_siena/BancaPSEJB.class

C:\bancaPs>set JARLIST=%JARLIST% banca_dei_paschi_di_siena/BancaPSEJBHome.class

C:\bancaPs>set JARLIST=%JARLIST% banca_dei_paschi_di_siena/BancaPSEJBBean.class

C:\bancaPs>set JARLIST=%JARLIST% banca_dei_paschi_di_siena/To/*.class

C:\bancaPs>set JARLIST=%JARLIST% META-INF

C:\bancaPs>jar cvf BancaPaschiSiena.jar %JARLIST%
aggiunto manifesto
aggiunta in corso di: banca_dei_paschi_di_siena/BancaPSEJB.class (in = 427) (out
 = 243) (compresso 43%)
aggiunta in corso di: banca_dei_paschi_di_siena/BancaPSEJBHome.class (in = 306)
(out = 207) (compresso 32%)
aggiunta in corso di: banca_dei_paschi_di_siena/BancaPSEJBBean.class (in = 6548)
 (out = 3464) (compresso 47%)
aggiunta in corso di: banca_dei_paschi_di_siena/To/BankLoanRequest.class (in = 7
18) (out = 399) (compresso 44%)
aggiunta in corso di: banca_dei_paschi_di_siena/To/BankQuoteRequest.class (in =
1423) (out = 635) (compresso 55%)
aggiunta in corso di: banca_dei_paschi_di_siena/To/BankQuoteResponse.class (in =
 928) (out = 493) (compresso 46%)
la voce META-INF/ sarÓ ignorata
aggiunta in corso di: META-INF/ejb-jar.xml (in = 775) (out = 384) (compresso 50%
)
aggiunta in corso di: META-INF/jboss.xml (in = 312) (out = 164) (compresso 47%)

C:\bancaPs>
```

4. deploy del Bean

Per effettuare il deploy del Bean sul server è necessario copiare il file .jar della cartella di deploy del server,e.g., `C:\Program Files\jboss-4.0.2\server\default\deploy` e se tutto è andato bene dovremmo visualizzare una schermata di questo tipo:

```
11:23:35,250 INFO  [ProxyFactory] Bound EJB Home 'BancaPSEJB' to jndi 'BancaPasc
hiEJB'
11:23:35,250 INFO  [EJBDeployer] Deployed: file:/C:/Programmi/jboss-4.0.3SP1/ser
ver/default/deploy/BancaPaschiSiena.jar
```

5. Web Services Deployment Descriptor

Dopo aver deployato il Session Stateless Bean all'interno di JBoss non resta che creare il file .wsdd che permette di esporre lo stesso Bean come Web Services.

La struttura generale di un file di deployment nel caso di un EJB è la seguente:

```
<service name="service-name" provider="java:EJB">
<parameter name="jndiURL"
value="naming-service-url"/>
<parameter name="jndiContextClass"
value="initial-context-factory-class-name"/>
<parameter name="beanJndiName"
value="ejb-jndi-name"/>
<parameter name="homeInterfaceName"
value="home-interface-name"/>
<parameter name="remoteInterfaceName"
value="remote-interface-name"/>
<parameter name="className"
value="remote-interface-name"/>
<parameter name="allowedMethods"
value="list-of-methods-to-expose"/>
</service>
```

Il file .wsdd che permetterà di esporre il nostro Session Stateless Bean come Web Service è il seguente:

deploy.wsdd

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<service name="BancaDeiPaschiDiSienaWS" provider="java:EJB">
<requestFlow>
      <handler type="soapmonitor"/>
    </requestFlow>
    <responseFlow>
      <handler type="soapmonitor"/>
    </responseFlow>
```

Per abilitare SOAPMonitor e analizzare i messaggi scambiati

```
<parameter name="wsdlTargetNamespace" value="http://banca_dei_paschi_di_siena"/>
<parameter name="schemaUnqualified"value="http://To.banca_dei_paschi_di_siena"/>
<parameter name="typeMappingVersion" value="1.2"/>

<parameter name="jndiURL" value="jnp://localhost:1099"/>
<parameter name="jndiContextClass"
value="org.jnp.interfaces.NamingContextFactory"/>
<parameter name="beanJndiName" value="BancaPaschiEJB"/>
<parameter name="homeInterfaceName"
value="banca_dei_paschi_di_siena.BancaPSEJBHome"/>
<parameter name="remoteInterfaceName"
value="banca_dei_paschi_di_siena.BancaPSEJB"/>
<parameter name="className" value="banca_dei_paschi_di_siena.BancaPSEJBBean"/>
<parameter name="allowedMethods" value="*"/>
<parameter name="scope" value="Request"/>
```

<typeMapp ng> per la serializzazione/de serializzazione dei ComplexTtpe

```
    <typeMapping
      xmlns:ns="http://To.banca_dei_paschi_di_siena"
      qname="ns:BankQuoteRequest"
```

```
        type="java:banca_dei_paschi_di_siena.To.BankQuoteRequest"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="http://To.banca_dei_paschi_di_siena"
        qname="ns:BankLoanRequest"
        type="java:banca_dei_paschi_di_siena.To.BankLoanRequest"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="http://To.banca_dei_paschi_di_siena"
        qname="ns:BankQuoteResponse"
        type="java:banca_dei_paschi_di_siena.To.BankQuoteResponse"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

</service>
</deployment>
```

Rispetto agli esempi analizzati precedentemente sono presenti alcune differenze. A parte l'attributo *name* che mantiene il nome del servizio,l'attributo *provider* viene ora impostato con il valore *java:EJB*. Ciò indica ad Axis che si tratta del deployment di una EJB e che deve dunque attendersi ulteriori specifici paramentri:

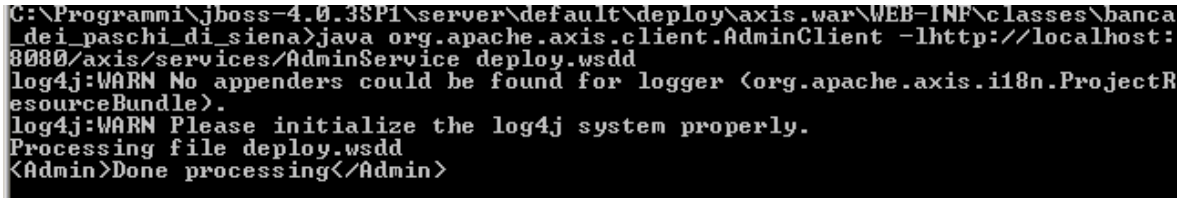
- l'URL su cui risponde il bean tramite protocollo RMI (attributo *jndiURL*),
  la factory per il contesto JNDI (attributo *jndiContextClass*);contiene la Classe per effettuare il binding su JNDI di Jboss
- Il nome JNDI con cui è "deployato" il bean all'interno dell'application container (attributo *beanJndiName*),
- Il nome dell'interfaccia home necessaria ad effettuare il look up del bean (attributo *homeInterfaceName*),
- il nome dell'interfaccia remote (attributo *remoteInterfaceName*).

Gli elementi *beanMapping* sono necessari ad indicare l'impiego dei tipi complessi *BankQuoteRequest,BankQuoteResponse e BankLoanRequest* descritti nei capitoli precedenti.

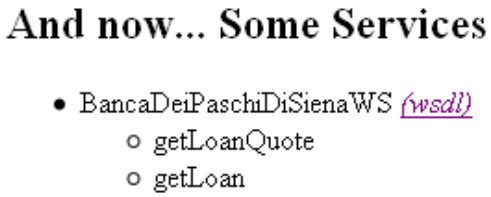
Per quanto riguarda il parametro *scope* il *value* ad esso associato è di tipo Request:in questo modo abbiamo creato un servizio stateless a partire da uno stateless session bean! Ogni client invocherà i metodi esposti dal Web Service senza sapere che questi corrispondono effettivamente alle funzionalità offerte ed implementate da un session bean.

Creato il file deploy.wsdd non rimane che inserirlo nella cartella banca_dei_paschi_di_siena (contenuta nella cartella bancaPs) e copiarla nella cartella axis.war di JBoss,ossia in `C:\Programmi\jboss-4.0.3SP1\server\default\deploy\axis.war\WEB-INF\classes` e deployare l'Ejb come Web Services con il comando:

```
java org.apache.axis.client.AdminClient -
lhttp://localhost:8080/services/AdminService deploy.wsdd
```



Il Bean è stato deployato come Web Services e se si accede alla lista dei servizi offerti da Axis in JBoss dovrebbe comparire:



Cliccando sul link wsdl otteniamo il relativo file che ne descrive l'interfaccia:

BancaDeiPaschiDiSienaWS.wsdl

```
  <?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://banca_dei_paschi_di_siena"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://banca_dei_paschi_di_siena"
    xmlns:intf="http://banca_dei_paschi_di_siena"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns1="http://To.banca_dei_paschi_di_siena"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <!--
      WSDL created by Apache Axis version: 1.4
      Built on Apr 22, 2006 (06:55:48 PDT)
    -->
 <wsdl:types>
 <schema targetNamespace="http://To.banca_dei_paschi_di_siena"
    xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="BankQuoteRequest">
     <sequence>
  <element name="creditHistory" type="xsd:int" />
  <element name="creditRisk" nillable="true" type="xsd:string" />
  <element name="creditScore" type="xsd:int" />
  <element name="loanAmount" type="xsd:int" />
  <element name="loanDuration" type="xsd:int" />
  <element name="ssn" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
 <complexType name="BankQuoteResponse">
 <sequence>
  <element name="errorCode" type="xsd:int" />
  <element name="interestRate" type="xsd:double" />
  <element name="quoteId" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
 <complexType name="BankLoanRequest">
<sequence>
  <element name="quoteId" nillable="true" type="xsd:string" />
  <element name="ssn" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </schema>
    </wsdl:types>
 <wsdl:message name="getLoanQuoteRequest">
  <wsdl:part name="in0" type="tns1:BankQuoteRequest" />
    </wsdl:message>
<wsdl:message name="getLoanQuoteResponse">
```

```
    <wsdl:part name="getLoanQuoteReturn" type="tns1:BankQuoteResponse" />
      </wsdl:message>
   <wsdl:message name="getLoanRequest">
    <wsdl:part name="in0" type="tns1:BankLoanRequest" />
      </wsdl:message>
  <wsdl:message name="getLoanResponse">
    <wsdl:part name="getLoanReturn" type="soapenc:string" />
      </wsdl:message>
   <wsdl:portType name="BancaPSEJB">
   <wsdl:operation name="getLoanQuote" parameterOrder="in0">
    <wsdl:input message="impl:getLoanQuoteRequest" name="getLoanQuoteRequest" />
    <wsdl:output message="impl:getLoanQuoteResponse" name="getLoanQuoteResponse"
      />
      </wsdl:operation>
   <wsdl:operation name="getLoan" parameterOrder="in0">
    <wsdl:input message="impl:getLoanRequest" name="getLoanRequest" />
    <wsdl:output message="impl:getLoanResponse" name="getLoanResponse" />
      </wsdl:operation>
      </wsdl:portType>
  <wsdl:binding name="BancaDeiPaschiDiSienaWSSoapBinding"
      type="impl:BancaPSEJB">
    <wsdlsoap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="getLoanQuote">
    <wsdlsoap:operation soapAction="" />
   <wsdl:input name="getLoanQuoteRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://banca_dei_paschi_di_siena" use="encoded" />
      </wsdl:input>
   <wsdl:output name="getLoanQuoteResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://banca_dei_paschi_di_siena" use="encoded" />
      </wsdl:output>
      </wsdl:operation>
   <wsdl:operation name="getLoan">
    <wsdlsoap:operation soapAction="" />
  <wsdl:input name="getLoanRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://banca_dei_paschi_di_siena" use="encoded" />
      </wsdl:input>
   <wsdl:output name="getLoanResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://banca_dei_paschi_di_siena" use="encoded" />
      </wsdl:output>
      </wsdl:operation>
      </wsdl:binding>
   <wsdl:service name="BancaPSEJBService">
  <wsdl:port binding="impl:BancaDeiPaschiDiSienaWSSoapBinding"
      name="BancaDeiPaschiDiSienaWS">
    <wsdlsoap:address
      location="http://localhost:8080/axis/services/BancaDeiPaschiDiSienaWS" />
      </wsdl:port>
      </wsdl:service>
      </wsdl:definitions>
```

L'esecuzione del deployment ha prodotto gli stessi effetti illustrati precedentemente:

- attivazione del servizio di esecuzione RPC,
- pubblicazione del documento WSDL che descrive il servizio, all'URL: http://localhost:8080/axis/services/BancaDeiPaschiDiSienaWS?wsdl

Occorre sottolineare che per entrambe le fasi viene perso il carattere di provenienza EJB. Dunque, le modalità di accesso al servizio (per esempio tramite client DII) e la tipologia della descrizione WSDL,sono analoghe a quelle descritte in precedenza.

Anche i messagi SOAP scambianti durante la comunicazione presenteranno le stesse caratteristiche di quelli analizzati nel primo capitolo per il servizio BancaDiRomaWS:l'engine SOAP utilizzato è infatti lo stesso come sono gli stessi il binding (style="**rpc**") e l'encoding style(use="**encoded**").

6. Client

Le modalità per accedere al servizio rimangono le stesse analizzate nel capitolo precedente;il bean infatti è ora esposto come Web Service e in quanto tale è possibile creare un client secondo le tipologie:

- Generated Stub
- Dynamic Proxy
- DII (Dynamic Invocate Interface)

Eseguendo un qualsiasi client che richiami i metodi getLoanQuote() e getLoan() implementati dal SessionBean esposto come servizio,nel prompt di JBoss viene evidenziato l'evento di creazione del Bean:

```
public void ejbCreate(){
System.out.println("Creazione del Bean BancaDeiPaschiDISiena");}
```

```
14:02:35,734 INFO  [STDOUT] Creazione del Bean BancaDeiPaschiDISiena
```

Bisogna ricordare che gli Stateless Session Bean non mantengono lo stato conversazionale per il client con cui interagiscono: qualsiasi informazione di stato associata con un bean stateless è locale al metodo invocato e non al Session Bean.
Poiché non è previsto uno stato conversazione,il container può assegnare un qualsiasi session EJB stateless già esiste a qualsiasi client.
Il Container può perfino associare lo stesso Stateless Bean a più client per risparmiare memoria. Questa politica di allocazione dei Stateless Bean dipende dall'implementazione del EJB Container ed è trasparente ai client che "credono" di avere uno stateless session bean dedicato.

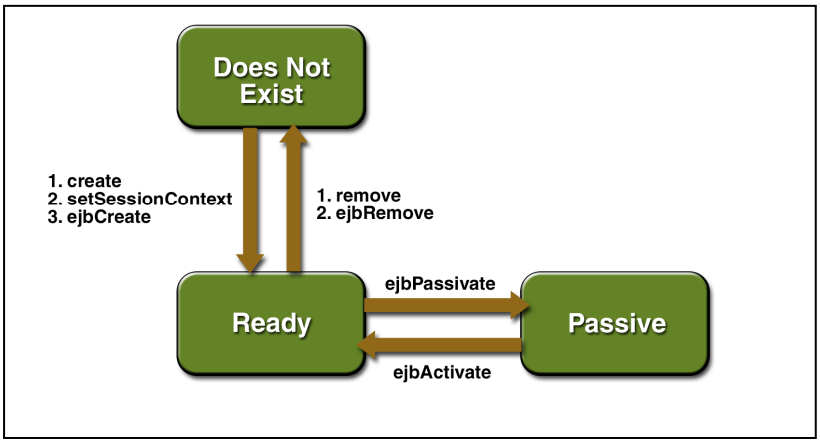


Figura 44:Ciclo di vita di un SessionBean Stateful

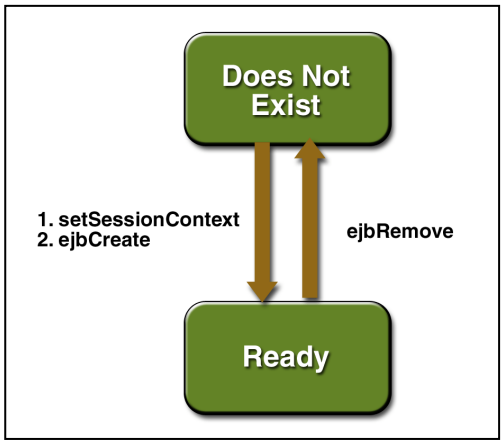


Figura 45:Ciclo di vita di un Session Bean Stateless

Capitolo 5 – Standard per la notifica di eventi nei Web-Services

ABSTRACT
Attualmente l'interazione tra Web-services viene realizzata mediante scambio di messaggi. Un messaggio è in grado di incapsulare qualsiasi tipo di informazione come ad esempio: dati di livello applicativo; errori; faults; risultati di operazioni di "ricerca" e "discovery" di servizi ecc. Nelle architetture orientate ai servizi(SOA) è di fondamentale importanza poter informare singole entità della rete sullo stato di una specifica risorsa. In particolare deve essere possibile per un'entità (un web-service) ricevere messaggi di notifica di evento così come avviene nei sistemi di tipo publish/subscribe. L'obiettivo di questo capitolo è quello di presentare e mettere a confronto metodologie per il design di web services in grado di interagire tramite scambio di notifiche di eventi. Dopo una breve introduzione sull'applicabilità del modello publish/subscribe nel mondo dei web-services,verranno presentate alcune specifiche che consentono di modellare interazioni di tipo public/subscribe tra servizi. In particolare concentreremo l'attenzione sulla specifica WS-BaseNotification dello standard WS-Notification introducendo un framework capace di supportarlo e proponendo un esempio attraverso il quale spiegare i concetti esposti inizialmente a livelo teorico.

5.1 Introduzione

Normalmente le interazioni tra web-services avvengono mediante invocazioni a metodi di tipo one-to-one. Un classico scenario applicativo è il seguente: un web service A invoca un metodo sul web-service B; in seguito B potrebbe spedire un valore di ritorno ad A. Questo semplice scenario riguarda servizi che implementano una logica di tipo request/response. Una logica come questa non garantisce tuttavia un buon livello di coordinazione nel caso i servizi si trovino ad operare in contesti maggiormente complessi in cui è necessario interagire all'interno un processo di business comune. Un paradigma publish/subscribe consentirebbe interazioni di tipo uno-a-molti mediante l'invio (in parallelo a tutte le entità interessate) di notifiche di evento. Un approccio per lo scambio di messaggi di tipo One-to-one non è infatti utilizzato laddove è necessario coordinare più processi che sono condivisi da più parti. Un paradigma publish/subscribe, essendo basato su scambio di messaggi a fronte di generazione di eventi, faciliterebbe il coordinamento tra web-services. Tale approccio infatti consente di separare concettualmente il ruolo del mittente da quello dei possibili (uno o più) destinatari dei messaggi di notifica. Per questo motivo è possibile, per un mittente ed un ricevente, comunicare senza che questi siano necessariamente connessi. L'*Event-driven*, o "*Notification-based, interaction pattern*" è comunemente utilizzato per la comunicazione tra entità in sistemi distribuiti. Esempi esistono in molti domini: ad esempio nei sistemi publish/subscribe messi a disposizione dai vendors di Message Oriented Middleware. Uno scenario publish/subscribe per il mondo dei web services potrebbe essere rappresentato nella seguente maniera: Alcuni Web services (che per semplicità chiameremo *produttori*) avranno l'incarico di disseminare informazioni sottoforma di notifiche di evento ad un insieme di altri Web Services (che chiameremo *consumatori*). Ciascun web service potrà dichiarare interesse al "consumo" di particolari notifiche di evento registrandosi presso un *produttore* che è responsabile della loro spedizione. Come parte del processo di sottoscrizione, un web service interessato al "consumo" di eventi dovrà ovviamente dichiarare il tipo degli eventi che si intende osservare. Al verificarsi di un evento nel sistema, i web services incaricati della distribuzione delle notifiche di evento dovranno inviare "one-way messages" a tutti i consumatori registrati (che hanno esplicitamente dichiarato all'atto della sottoscrizione di voler ricevere notifiche di quel tipo di evento) Sarà possibile per più web service registrarsi al "consumo" di uno stesso tipo di evento. In tal caso ciascuno di essi riceverà una copia separata dell'informazione (una notifica di evento). Così come avviene in tutti i sistemi publish/subscribe,è necessario fare una netta distinzione tra alcune fasi dell'interazione. Per poter ricevere infatti delle notifiche di evento un consumer deve essere sottoscritto al tipo di evento richiesto. Nella fase iniziale, detta di *discovery*, il "consumatore" cercherà di individuare quali siano i servizi in grado di notificare eventi. La fase di discovery è sempre necessaria:infatti è possibile effettuare una sottoscrizione soltanto presso un servizio in grado di poter comunicare notifiche dell'evento desiderato. Una volta individuati tali servizi,deve essere possibile,con un particolare meccanismo di interazione, ricevere informazioni sui singoli tipi di eventi che è possibile osservare. La sottoscrizione

avviene nella seconda fase del processo di interazione tra produttore e consumatore di eventi. Una volta scoperti quali siano i tipi di eventi a cui è possibile sottoscriversi, il consumatore deve poter segnalare al produttore l'intenzione di ricevere delle notifiche riguardanti uno o più tipi di evento supportati. La fase di sottoscrizione, se portata a termine, consiste in una sorta di contratto che vincola il produttore a notificare eventi al consumatore ogni qualvolta essi si verifichino. Come ogni contratto, anche la sottoscrizione può essere soggetta a scadenze di carattere temporale ed in generale dovrebbe essere possibile poterla rinnovare. Deve anche essere possibile per il consumatore esprimere dei "filtri" sul contenuto informativo dei singoli eventi. Una volta completata la fase di sottoscrizione, quello che ci si aspetta è che il produttore di notifiche di evento notifichi il consumatore ogni volta che un event o si venga a verificare. Questo normalmente avviene secondo una logica di tipo push: è il produttore che ha il compito di avvisare tutti i consumatori sottoscritti all'evento. Attualmente per i web-services esistono varie specifiche in grado di supportare scenari publish/subscribe come quello appena descritto[55,56,57]:

- WS-Events;
- WS-Eventing;
- WS-Notification.

Queste tre specifiche sono tra loro molto simili e presentano in realtà pochissime differenze (spesso di carattere puramente terminologico). Ciascuna di queste consente l'implementazione di scenari simili a quello attualmente descritto.
Ciascuna specifica si appoggia a specifiche web già esistenti: in particolare WS-Notification richiede per il suo funzionamento che vengano implementate numerose altre specifiche,a differenza di come accade nel caso di WS-Events e WS-Eventing. Un forte limite di WS-Events e WS_Eventing è quello di non riuscire a modellare scenari che consentano ai servizi di interagire con degli intermediari. Tolta WS-Notification,nessuna delle altre due specifiche standardizza infatti il ruolo dei broker. Altri limiti riscontrati nelle specifiche sono i seguenti:

- Le specifiche analizzate non definiscono dei protocolli sicuri per lo scambio di informazioni tra servizi. Proprio per questo motivo nei documenti ufficiali si raccomanda fortemente di appoggiarsi nell'implementazione ad altre specifiche web come WS-Security per garantire un maggiore livello di sicurezza.
- Nessuna delle tre specifiche ha nei suoi obiettivi quello di garantire affidabilità nello scambio dei messaggi. Un messaggio potrebbe ad esempio andare perso e potrebbe non raggiungere mai il destinatario.
- Nessuna delle specifiche consente inoltre di definire dei timeout sulle richieste di recupero di notifiche di messaggio quando si è in modalità pull.
- Tutte le specifiche assumono che il canale di comunicazione per lo scambio dei messaggi sia affidabile.
- Le specifiche non consentono la definizione di particolari policy per la gestione delle sottoscrizioni.

Nonostante le tre specifiche si assomiglino moltissimo,WSNotification offre in generale maggiore espressività. Un differenza significativa tra le tre specifiche risiede nel modo con cui ciascuna permette di definire categorie di eventi (topics o più generalmente tipi di evento). WS-Notification è l'unica specifica in grado di fornire un sistema di definizione dei tipi di evento basato sul concetto di topic. WS-Events e WS-Eventing non consentono né la definizione di gerarchie di tipi di evento,né operazioni a supporto dell'*advertising* di eventi.

5.2 Il ruolo del broker nei sistemi publish/subscribe

Un *broker* è un'entità in grado di instradare i messaggi di notifica. Tipicamente nei sistemi di tipo publish/subscribe i brokers aggregano e pubblicano eventi provenienti da altri produttori. Un broker può applicare modifiche ai messaggi di notifica che devono essere inoltrati. Detto in termini pù generali,un broker svolge il ruolo di intermediario tra pubblicatori di notifiche di eventi e consumatori di notifiche di eventi[57]. Un broker in un sistema publish/subscribe ha il compito di:

- Pubblicare eventi provenienti da altri produttori
- instradare le notifiche ai consumer di eventi per conto di uno o più produttori
- portare avanti per conto del pubblicatore la fase di sottoscrizione

La presenza di brokers nel sistema publish/subscribe comporta delle variazioni allo scenario proposto nel paragrafo introduttivo. Di seguito vengono riportate le principali modifiche. In

presenza di brokers, i produttori di eventi non avranno più il compito di memorizzare un elenco delle sottoscrizioni avvenute: questo compito verrebbe delegato al broker presso cui il produttore intende registrarsi. La registrazione consiste nel dichiarare al broker quali siano i tipi di evento che si intende pubblicare. Una volta registratosi presso un broker,il produttore non dovrà più interagire direttamente con i singoli consumatori sottoscritti. In fase di discovery il consumatore interessato ad individuare i servizi in grado di notificare eventi effettuerà le proprie ricerche interrogando solo i brokers presenti nella rete. In tale fase i brokers "esporranno" gli eventi a cui è possibile sottoscriversi per conto dei produttori registrati.
La fase di sottoscrizione non prevede interazioni tra produttori e consumatori di eventi: il consumatore segnala direttamente al broker l'intenzione di voler ricevere notifiche di particolari tipi di eventi. Una volta conclusa la trattativa di sottoscrizione, il broker avrà il compito di gestire dunque tutte le risorse di sottoscrizione. Quindi, ogni volta che si verificherà un evento, il produttore dovrà inoltrare una notifica di evento soltanto ai broker presso cui si è registrato. Spetterà ai brokers, una volta ricevuta la notifica di evento, il compito di inoltrarne una copia a tutti i consumatori sottoscritti. Questo particolare scenario non prevede alcuna interazione tra produttori e consumatori: tutto viene svolto interagendo direttamente e soltanto con dei brokers. Il consumatore interagirà con i brokers sia in fase di discovery sia in fase di sottoscrizione.
Il produttore dovrà semplicemente registrarsi presso dei brokers e spedire soltanto ad essi le notifiche di eventi. I consumatori,d'altro canto,non riceveranno più le notifiche di evento direttamente dai produttori bensì dai brokers.

5.3 WS-NOTIFICATION

WS-Notification è una famiglia di specifiche per web-services che definisce un approccio di tipo publish/subscribe alla notifica di eventi. Le specifiche che compongono WS-Notification sono tre:

- WS-BaseNotification[48]
- WS-BrokeredNotification[49]
- WS-Topics[50]

WS-BaseNotification standardizza gli scambi e le interfacce tra produttore e consumatore di notifiche.
WS-BrokeredNotification definisce l'interfaccia web-service dei notification brokers, specifica,dunque,il ruolo del broker.
WS-Topics definisce, invece, un meccanismo per organizzare e categorizzare gli elementi di interesse per la sottoscrizione,ovvero i "topics".
In questo contesto prenderemo in considerazione la specifica del WS-BaseNotification introducendo un framework per realizzarlo(Apache Muse) ed analizzando ogni messaggio scambiato tra gli attori coinvolti(paragrafo 5.3.9).

5.3.1 Note sulla terminologia utilizzata

WS-Notification definisce un *evento* come un cambiamento nello stato di una risorsa. Ogni evento può essere visto come una specifica istanza di un *topic*. Un *topic* rappresenta una categoria ben precisa di eventi (un tipo di eventi) a cui potersi sottoscrivere. Facendo un paragone con l'*object oriented programming*, un topic è molto simile ad una classe: esso è infatti provvisto di proprietà (i.e. attributi) è metodi tramite i quali poter accedere al proprio stato interno. Facendo un confronto con la specifica WSEvents si potrebbe dire che i *topics* di WS-Notification siano concettualmente paragonabili a degli event-types. In realtà i topic garantiscono una maggiore espressività: un event-type,è poco più di un semplice identificativo; un topic è invece una vera e propria risorsa ispezionabile. WS-Notification consente la definizione di gerarchie di topic allo stesso modo con cui è possibile definire gerarchie di classi nei linguaggi di programmazione object oriented. In particolare la specifica WS-Topic tratta in maniera molto specifica il concetto di topic. Per eventuali approfondimenti sulla tematica dei topics e su come possa essere possibile definirli ed accedere alle informazioni mantenute nella loro struttura si rimanda al documento di specifica[6]. Ai fini della comprensione degli argomenti trattati nel paper basta sapere che un topic è una risorsa accessibile esternamente avente una propria struttura dati interna. WS-Notification distingue ben cinque possibili attori di sistema:

- Notification Producer
- Notification Consumer
- Subscriber
- Subscription Manager
- Notification Broker

Un *producer*,detto anche *Notification Producer* è un servizio il cui compito principale è quello di accettare richieste di sottoscrizione da parte di un *consumer* (i.e. *notification consumer*). Ogni richiesta di sottoscrizione identifica uno o più topics di interesse ed un *Notification Consumer*. In assenza di un broker, il Notification Producer ha l'obbligo di mantenere una lista di sottoscrizioni: ciascuna sottoscrizione è caratterizzata dalle seguenti informazioni:

- Topics (su cui è stata effettuata la sottoscrizione)
- Identificativo del *Notification Consumer* interessato alla notifica degli eventi appartenenti ai topics specificati
- Eventuali informazioni addizionali (la cui semantica non è però standardizzata dalla specifica)

Ogni volta che è necessario inoltrare una notifica, il producer confronta le informazioni contenute nella notifica con quelle delle varie sottoscrizioni a lui conosciute. Lo scopo è quello di inoltrare ai notification consumer corretti le notifiche di evento. Il *Subscription Manager* ha il compito di gestire lo stato delle sottoscrizioni. Un producer delega il compito di gestione delle notifiche ad uno specifico Subscription Manager. Un producer può svolgere anche il ruolo di gestore delle sottoscrizioni implementando anche l'interfaccia del Subscription Manager. Il Notification Consumer è semplicemente il servizio a cui sono destinati i messaggi di notifica. Il compito di portare avanti la trattativa di sottoscrizione con il Producer viene normalmente svolto da un *Subscriber*. Il Subscriber ha in delega il compito di spedire messaggi di sottoscrizione al notification producer. Un notification consumer può anche svolgere il ruolo di subscriber. Inoltre Subscriber e notification Producer possono essere due distinte entità.
Infine con il termine *Notification Broker* ci si riferisce ad un intermediario il cui compito è quello di porsi tra il Notification Producer ed il Notification Consumer. Il broker è responsabile della gestione delle sottoscrizioni e dell'instradamento delle notifiche ai vari Notification Consumers.

5.3.2 Cosa definisce

Come accennato in precedenza, quando si fa riferimento a WSNotification, si distingue tra tre diverse specifiche che sono: WSBaseNotification; WS-BrokeredNotification; WS-Topics. WS-BaseNotification standardizza gli scambi e le interfacce tra produttore e consumatore di notifiche. WS-BrokeredNotification definisce l'interfaccia web-service dei notification brokers (di fatto specifica il ruolo del broker). WS-Topics definisce invece un meccanismo per organizzare e categorizzare gli elementi di interesse per la sottoscrizione conosciuti sotto il nome di "topic".
In definitiva si può dire che L'obiettivo di WS-Notification è quello di definire:

- i ruoli
- la terminologia ed i concetti espressi nel modello
- uno standard per lo scambio di messaggi
- un linguaggio per la descrizione di Topics

5.3.3 Cosa non definisce

WS-Notification similmente a WS-Events,non definisce un meccanismo ad eventi "general-purpose". Si inserisce all'interno dell'esistente Web services framework e per questo motivo richiede necessariamente l'utilizzo di specifiche quali WSDL e SOAP per il suo corretto funzionamento. Non è tra gli obbiettivi di WS-Notification quello di definire in che modo Notification consumers e Subscribers possano venire a conoscenza della presenza di potenziali Notification Producers.

5.3.4 Scenario di funzionamento in assenza di broker

Nel caso di funzionamento in *assenza di broker*, le entità coinvolte nella fase di discovery sono solo due: il producer e il subscriber. Responsabile di questa operazione è, generalmente, il subscriber stesso. Una volta individuati dei possibili topics di interesse, si passa alla fase di sottoscrizione, in cui il subscriber inoltra al producer una relativa richiesta. Tale richiesta conterrà informazioni riguardanti i topics, per cui si richiede la sottoscrizione e il consumer a cui dovranno essere inoltrate le notifiche di eventi. Una volta portata a termine con successo la

fase di sottoscrizione, il producer genera la risorsa di sottoscrizione, la memorizza all'interno della lista di sottoscrizioni e delega ad un subscription manager il compito di gestire la specifica sottoscrizione. A questo punto, il producer avrà l'obbligo di inoltrare ai consumer sottoscritti i corretti messaggi di notifica. Il consumer potrà ricevere notifiche dal publisher ed, inoltre, potrà interagire con il subscription manager al fine di modificare lo stato della risorsa sottoscrizione. E' da sottolineare il fatto che nulla vieta al consumer di svolgere anche il ruolo di subscriber, così come nulla vieta al producer di incaricarsi della gestione delle singole sottoscrizioni.

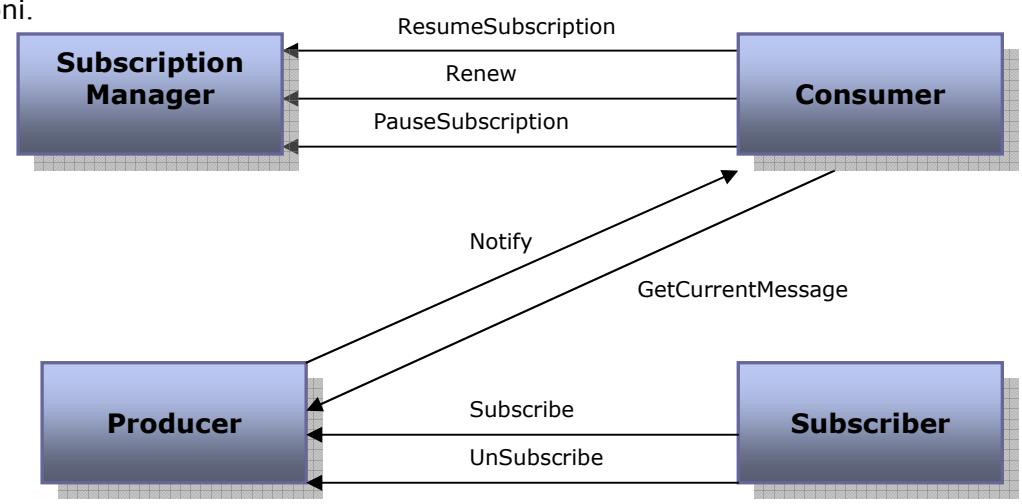


Figura 46:Scenario di funzionamento in assenza di broker

Sono di seguito rappresentati i diagrammi di sequenza relativi alle singole operazioni svolte dagli attori del sistema nel caso di funzionamento in assenza di broker.

Subscribe

Un Notification Producer è in grado di produrre una sequenza di messaggi di notifica. Un Subscriber può registrare l'interesse di un Notification Consumer di ricevere un sottoinsieme di questa sequenza. Un Subscriber invia, dunque, un messaggio di subscribe per registrare tale interesse. Se il messaggio di sottoscrizione ha successo, allora il Notification Producer deve produrre un messaggio di *response* ed inviarlo al subscriber.

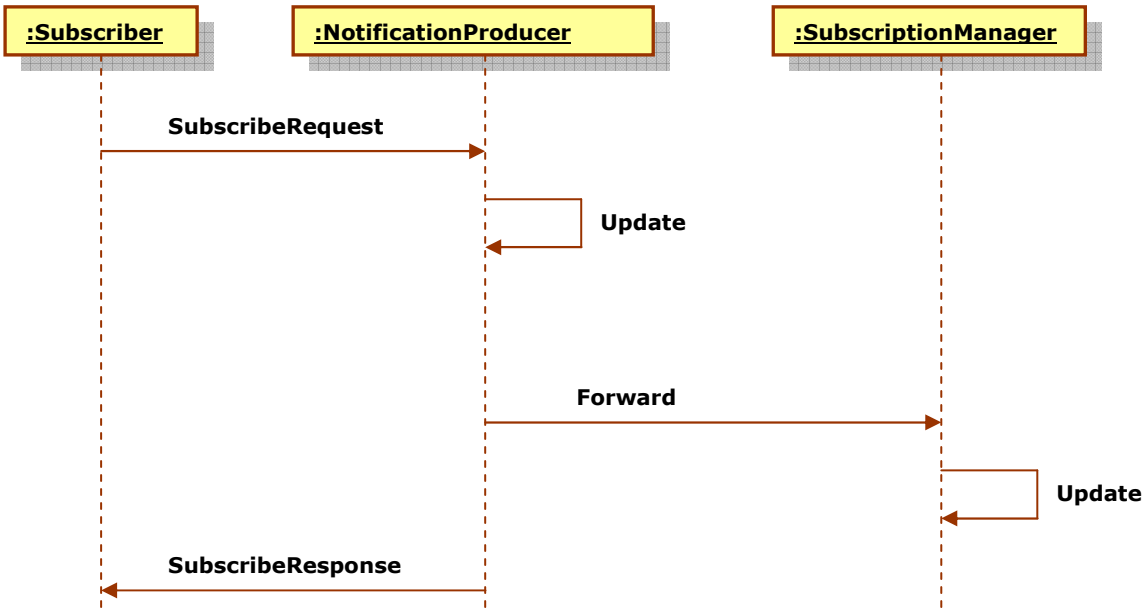


Figura 47:Operazione di Subscribe

GetCurrentMessage

In risposta al messaggio GetCurrentMessage, il NotificationProducer deve ritornare l'ultima notifica pubblicata del topic dato. La funzione principale è informare un consumer appena iscritto, dell'ultimo evento occorso. In alcune circostanze, un NotificationProducer può scegliere di non memorizzare l'ultima *Notifica* per uno o più Topics. In tal caso, il NotificationProducer risponderà con un *fault message*, indicando che non vi sono ultime notifiche per il topic dato.

RenewSubscription

Per modificare il tempo di vita della sottoscrizione, il richiedente deve inviare un messaggio di RenewSubscriptionRequest al SubscriptionManager. Se il SubscriptionManager processa con successo il messaggio di RenewSubscriptionRequest,allora dovrà rispondere con un messaggio di RenewSubscriptionResponse. Se, al contrario,l'elaborazione non dovesse andare a buon fine, verrà ritornato un *fault message.*

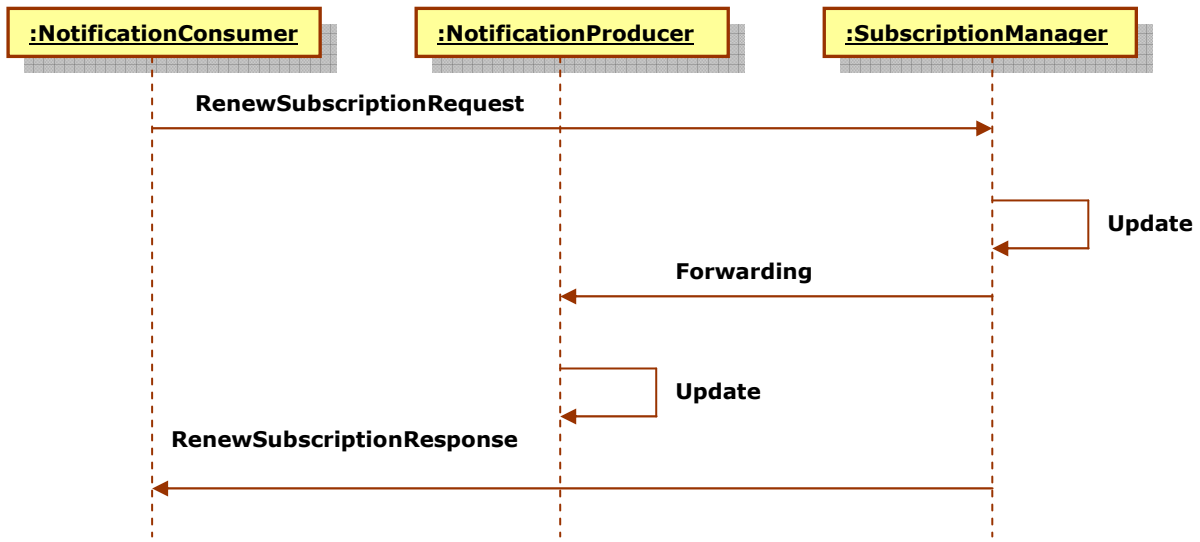


Figura 48:Operazione di Renew

Unsubscribe

Per terminare la sottoscrizione,un richiedente deve inviare un messaggio di UnsubscribeRequest al SubscriptionManager. In seguito ad una richiesta di annullamento della sottoscrizione, il SubscriptionManager deve tentare di distruggere la risorsa Sottoscrizione. Se il SubscriptionManager gestisce con successo il messaggio di UnsubscribeRequest,allora dovrà rispondere con un messaggio di UnsubscribeResponse.

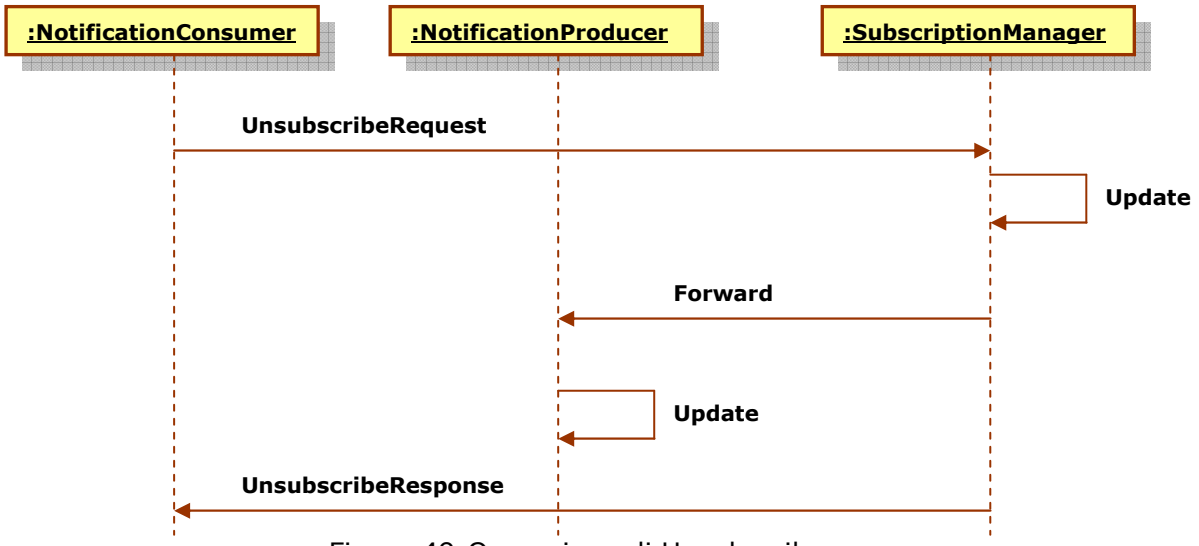


Figura 49:Operazione di Unsubscribe

PauseSubscription

Per sospendere temporaneamente la produzione di messaggi di notifica per una particolare Sottoscrizione, il richiedente deve inviare un messaggio di PauseSubscriptioRequest al SubscriptionManager. In risposta ad un messaggio di richiesta di pausa della Sottoscrizione, il SubscriptionManager dovrà rispondere con un messaggio di PauseSubscriptionResponse.

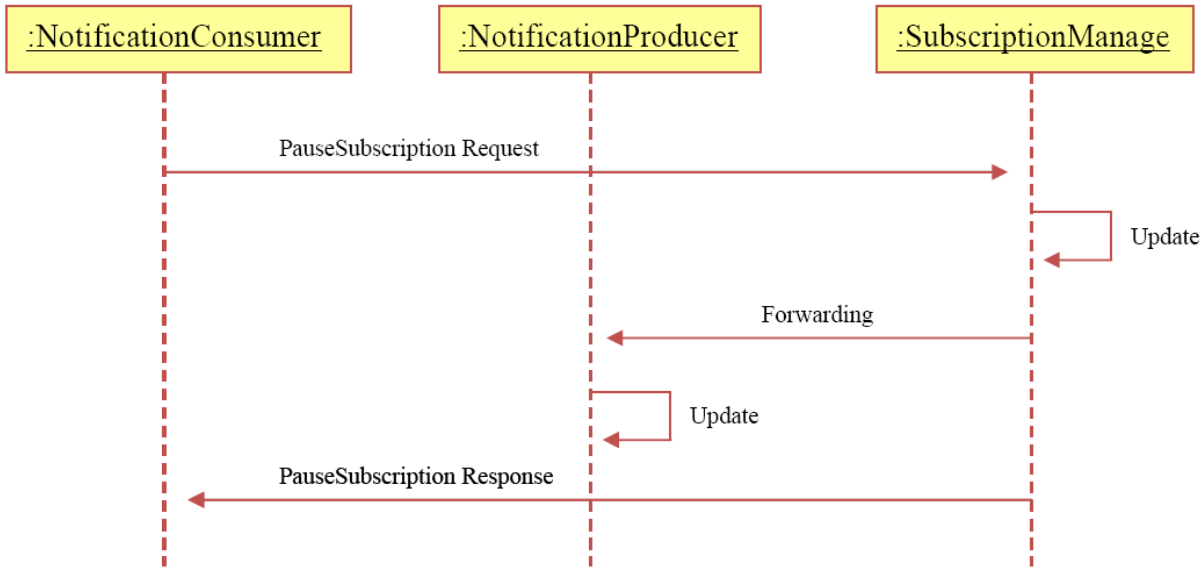


Figura 50:Operazione di PauseSubscripion

ResumeSubscription

Se un richiedente desidera riprendere la produzione dei messaggi di notifica, deve inviare un messaggio di ResumeSubscriptionRequest. In risposta a tale messaggio,il SubscriptionManager dovrà inviarne uno di ResumeSubscriptionResponse. Se l'elaborazione non dovesse andare a buon fine, il messaggio di ritorno sarà un messaggio di *fault message.*

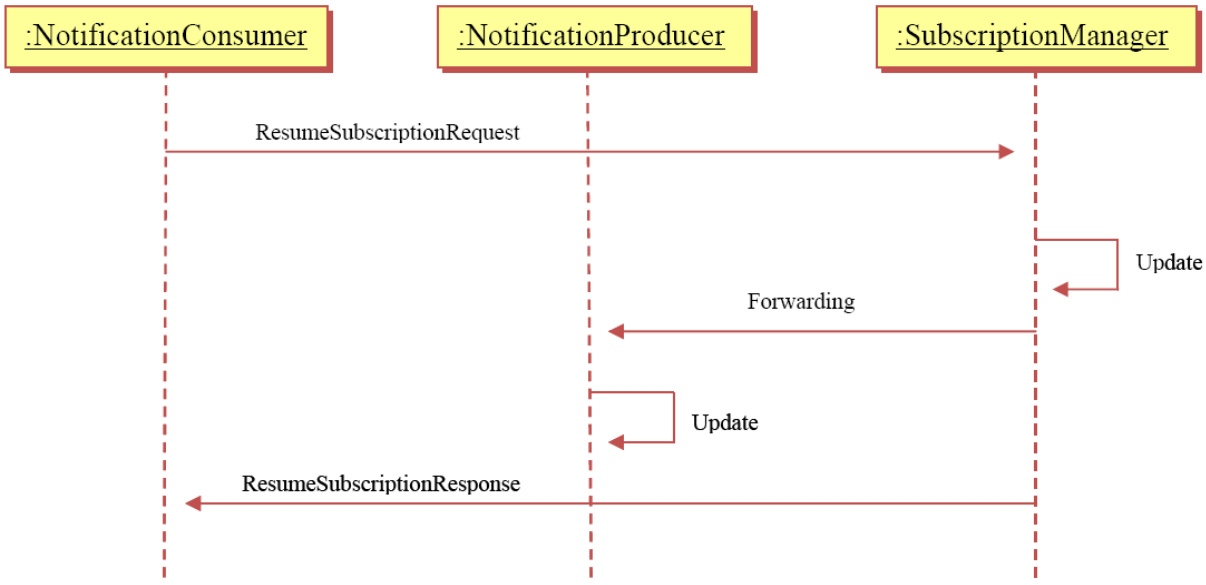


Figura 51:Operazione di ResumeSubscription

Notify

In seguito al verificarsi di un evento, il NotificationProducer dovrà produrre messaggi di Notifica NotifyRequest. In seguito alla ricezione di questo messaggio il NotificationConsumer risponderà con un messaggio NotifyResponse.

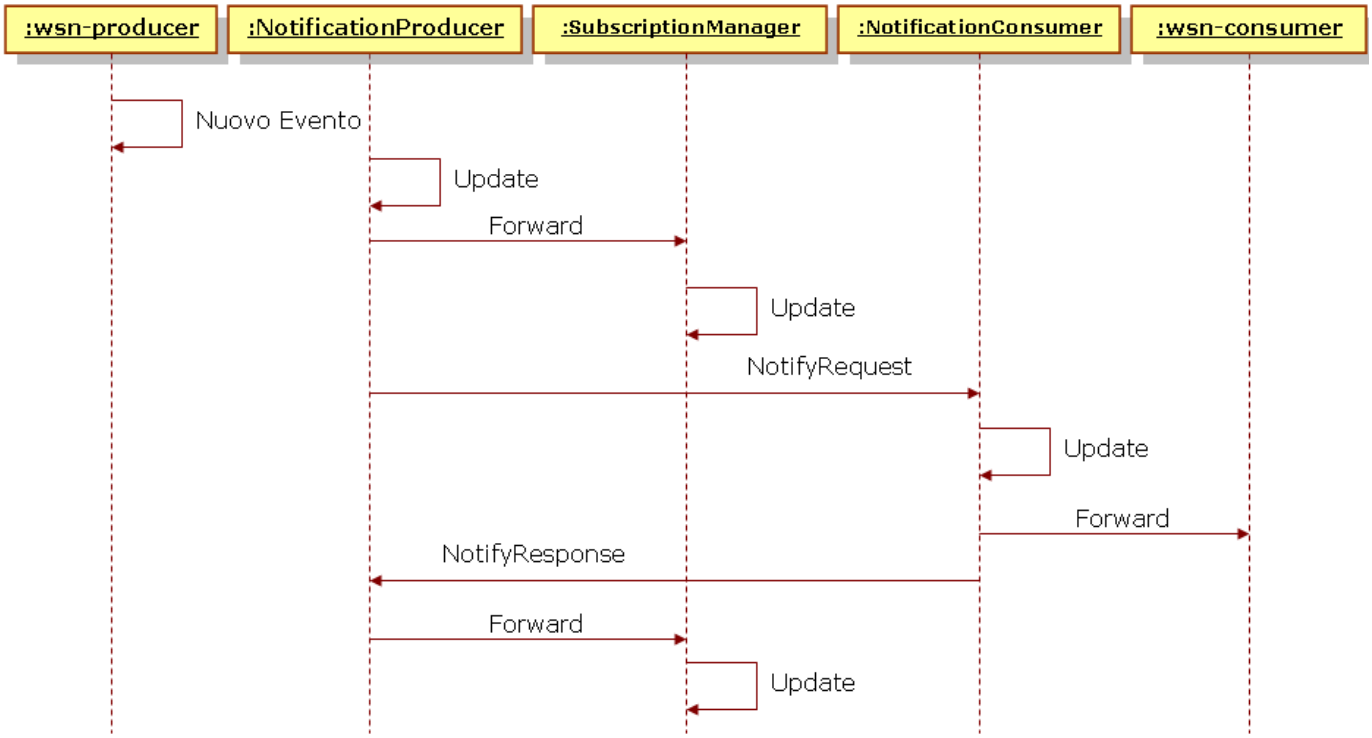


Figura 52:Operazione di Notify

5.3.5 Scenario di funzionamento in presenza di broker

Nel caso di funzionamento in *presenza di broker*, le entità coinvolte nella fase di discovery sono il Subscriber ed il Notification Broker. Il Publisher delega, infatti, al Broker il compito di mantenere le informazioni sui topics. Una volta individuati dei possibili topics di interesse, il Subscriber può passare alla fase di sottoscrizione. Anche questa fase ha come protagonisti il Subscriber ed il Notification Broker. Il Subscriber, infatti, effettuerà la richiesta di sottoscrizione presso il Broker e non più presso il Publisher. Spetterà, dunque, al Broker mantenere memorizzata una lista di sottoscrizioni. Il Publisher, preliminarmente, deve aver contattato il Broker, mediante il metodo RegisterPublisher dell'interfaccia del Broker, dichiarando di essere disposto a pubblicare notifiche di eventi relativi a specifici topics. Una volta registratosi presso il Broker, il Publisher invierà le notifiche degli eventi che verranno prodotti soltanto al Broker presso cui è registrato. Il Broker sarà, quindi, in grado di capire verso quali Consumers dirigere le notifiche di evento, poiché manterrà la lista di tutte le sottoscrizioni. La presenza del Broker non modifica la semantica comportamentale del SubscriptionManager. Infatti, una volta effettuata la sottoscrizione, è possibile accedere alla rispettiva risorsa, mediante i metodi PauseSubscription e ResumeSubscription, messi a disposizione dall'interfaccia del SubscriptionManager.

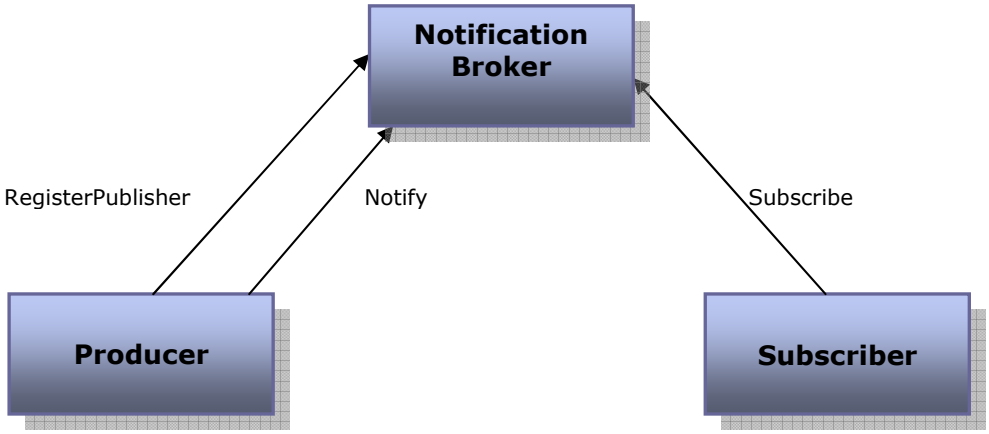


Figura 53:Scenario di funzionamento in presenza di broker

Lo scenario di funzionamento in presenza di broker, permette di diminuire, considerevolmente, l'overhead dovuto alla necessità di spedire notifiche a ciascuno dei singoli consumer della rete. Il publisher dovrà, infatti, comunicare la notifica soltanto ai brokers presso cui è registrato. Secondo le specifiche WS-Notification, un NotificationBroker è rappresentato da un'interfaccia che deve estendere sia l'interfaccia rappresentante il NotificationConsumer,sia quella rappresentante il NotificationProducer. Questo risulta quindi essere un esempio di ereditarietà multipla,in cui una singola interfaccia estende più interfacce,in tal modo l'interfaccia NotificationBroker erediterà i metodi presenti in entrambe le interfacce di livello superiore. In pratica,un NotificationBroker risulta essere,contemporaneamente,sia un NotificationConsumer che un NotificationProducer, e questo avviene in quanto esso è tenuto a eseguire i metodi propri di un NotificationConsumer mentre comunica con un NotificationProducer ed ad eseguire i metodi propri di un NotificationProducer mentre comunica con un NotificationConsumer.

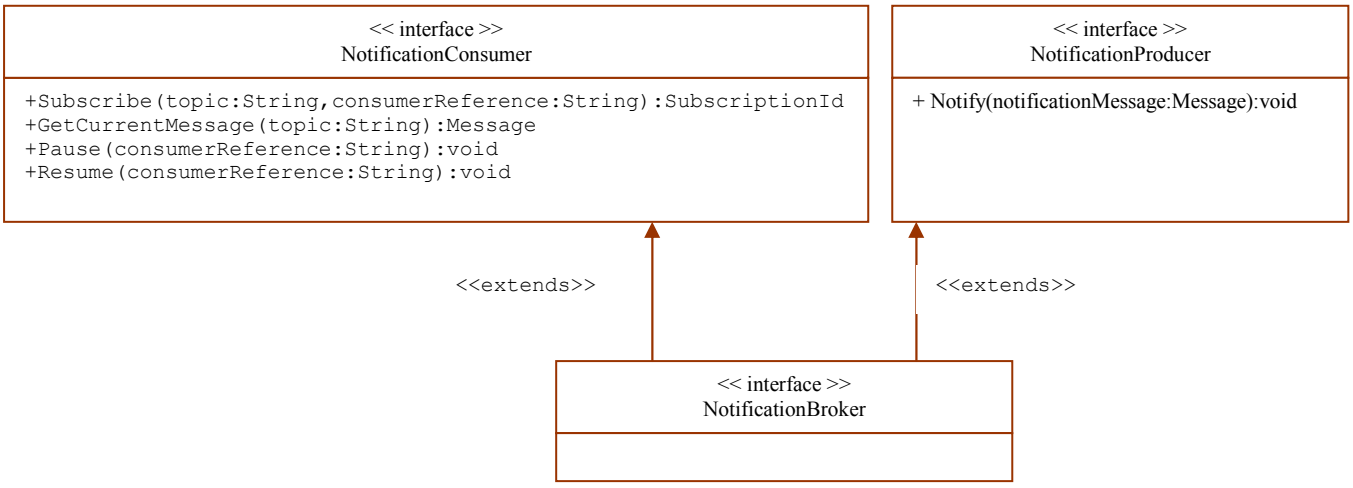


Figura 54:Class diagram che rappresenta le relazioni tra gli oggetti WS-Notification

Sono di seguito rappresentati i diagrammi di sequenza relativi alle singole operazioni svolte dagli attori del sistema nel caso di funzionamento in presenza di broker.

RegisterPublisher

Il messaggio RegisterPublisher è utilizzato dal Publisher per confermare la sua capacità di pubblicare un dato topic, oppure un insieme di topics. Se un'entità desidera registrare un Publisher, deve inviare un messaggio di richiesta di RegisterPublisher al NotificationBroker. Se quest'ultimo accetta il messaggio di richiesta, allora dovrà rispondere con un messaggio di RegisterPublisherResponse. Il NotificationBroker delega ad un PublisherRegistrationManager la gestione delle risorse PublisherRegistration.

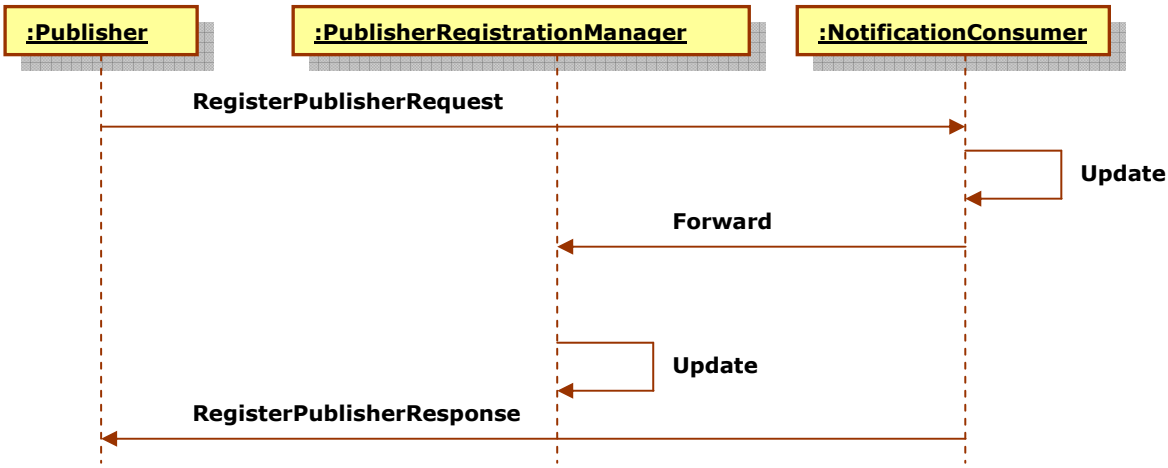


Figura 55:Operazione di RegisterPublisher

Subscribe

Un Subscriber invia un messaggio di SubscribeRequest, per registrare l'interesse a ricevere una sequenza di messaggi di notifica, direttamente al NotificationBroker e non più al NotificationProducer. Se l'operazione va a buon fine, il NotificationBroker restituirà un messaggio di SubscribeResponse, dopo aver aggiornato la lista delle sottoscrizioni.

GetCurrentMessage

Un Subscriber invia un messaggio di GetCurrentMessageRequest al NotificationBroker, per ricevere l'ultima notifica del topic dato. Se l'operazione va a buon fine, il NotificationBroker restituirà un messaggio di GetCurrentMessageResponse.

RenewSubscription

Per modificare il tempo di vita della sottoscrizione, un Subscriber invia un messaggio di RenewSubscriptionRequest al NotificationBroker. Se l'operazione va a buon fine,quest'ultimo restituirà un messaggio di RenewSubscriptionResponse.

Unsubscribe

Per terminare la sottoscrizione, un Subscriber invia un messaggio di UnsubscribeRequest al NotificationBroker. Quest'ultimo aggiornerà la sua lista di sottoscrizioni.

PauseSubscription

Per sospendere temporaneamente la sottoscrizione, il Subscriber invia un messaggio di PauseSubscriptionRequest al NotificationBroker. Se l'operazione va a buon fine,quest'ultimo invierà un messaggio di PauseSubscriptionResponse.

ResumeSubscription

Per riprendere la produzione dei messaggi di notifica,il Subscriber invia un messaggio di ResumeSubscriptionRequest al NotificationBroker. Se l'operazione va a buon fine, quest'ultimo invierà un messaggio ResumeSubscriptionResponse.

Destroy

Se un Publisher decide di non pubblicare più determinati topics, allora invierà un messaggio di DestroyRequest al PublisherRegistrationManager. Se l'operazione ha successo, quest'ultimo invierà un messaggio di DestroyResponse.

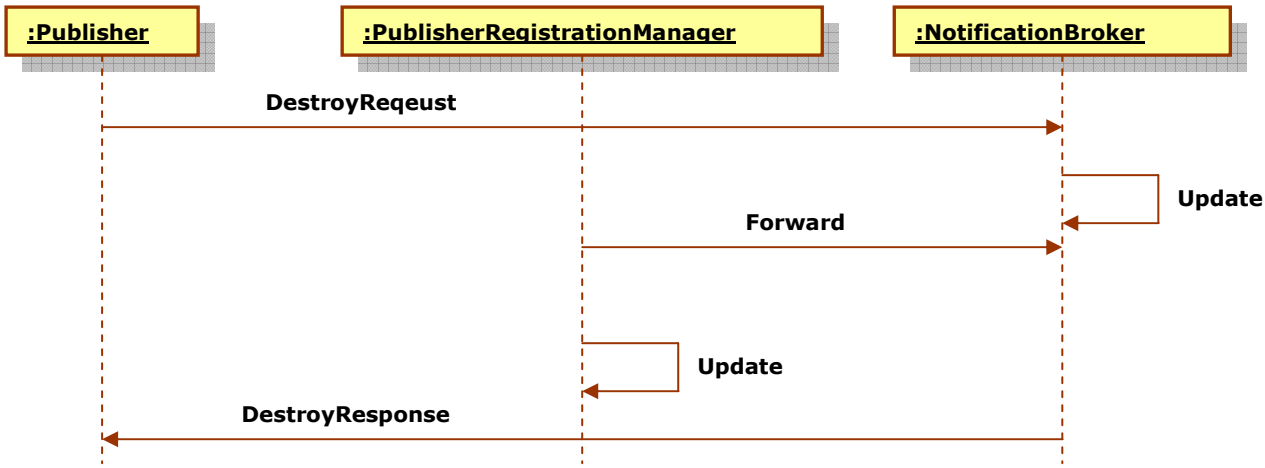


Figura 56:Operazione di Destroy

Notify

All'occorrenza di un nuovo evento, un Publisher manderà una notifica al NotificationBroker. Alla ricezione di un messaggio di Notify, il NotificationBroker inoltrerà la notifica ai Subscribers interessati al determinato topic.

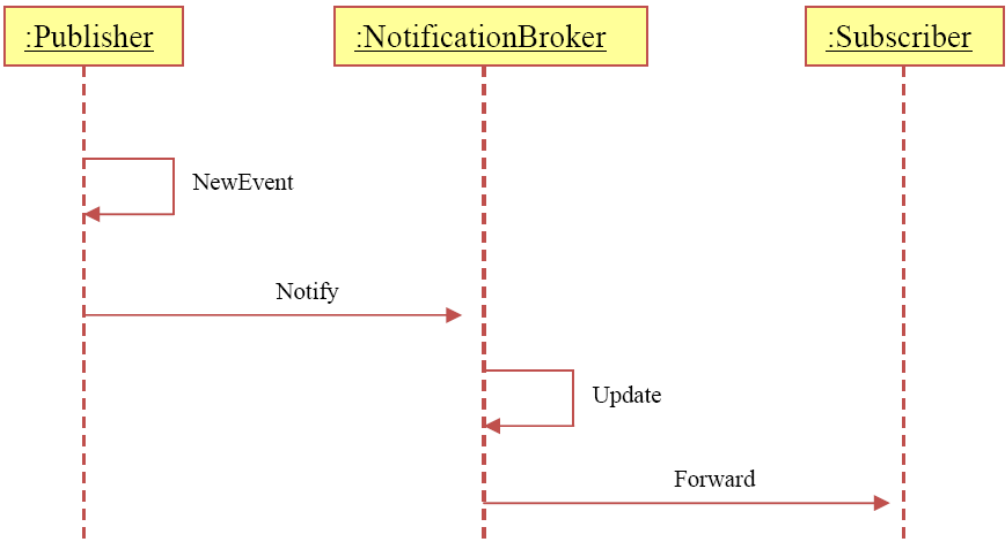


Figura 57:Operazione di Notify

5.3.6 Confronto tra gli scenari

Lo scenario di funzionamento in assenza di Broker e lo scenario di funzionamento in presenza di Broker costituiscono,rispettivamente,una comunicazione diretta e una comunicazione indiretta tra i partecipanti. In particolare lo scenario di funzionamento in presenza di broker, pur aumentando la complessità del sistema,permette di diminuire considerevolmente l'overhead dovuto alla necessità di spedire notifiche a ciascuno dei singoli consumers della rete,aumentando così il livello di scalabilità.

La presenza del Broker aumenta inoltre il livello di disaccoppiamento tra le parti comunicanti, che non necessitano di avere riferimenti l'una dell'altra. Sono proprio queste le caratteristiche principali che spingono a scegliere un modello "brokerato" per l'implementazione.

5.3.7 Server WS-Notification

Per implementare la piattaforma del WS-Notification sono disponibili i seguenti sistemi sviluppati da istituti e contesti differenti,ovvero:

- **`WS-Notification Server`**[57]: è un'applicazione ch implementa precisamente la specifica WS-Notification,la metodologia più espressiva per la notifica degli eventi nei web services.
- **`Globus Toolkit 4 (GT4)`**[52]: un software open source per la costruzione di sistemi *grid*,cioè sistemi che consentono la condivisione di una serie di risorse eterogenee (come server,storage,capacità computazionale,database, applicazioni, ecc.) per fornire agli utenti un unico sistema computazionale virtuale sul quale utilizzare qualsiasi tipo di applicazione. Il toolkit comprende librerie e servizi software sia per il monitoraggio,la ricerca e la gestione delle risorse che per la sicurezza e la gestione dei files. Per queste sue caratteristiche GT4 viene sfruttato come sottostrato nella realizzazione di sistemi grid commerciali da parte delle più importanti compagnie del settore. Tra le varie funzionalità e librerie comprende l'implementazione delle interfacce specificate dallo standard WS-Notification,ma non fornisce alcun software all'utente finale per la gestione delle notifiche. L'implementazione di WS-Notification resta dunque a livello di middleware,le sue funzionalità possono quindi essere usate per porre le basi per la costruzione di un sistema basato su WS-Notification,potendo sfruttare l'implementazione delle API e della messaggistica già presente in GT4.
- **`Servicemix`**[53]: un toolkit Enterprise Service Bus (ESB), cioè un middleware che ha lo scopo di unire assieme l'integrazione delle tecnologie e l'esecuzione dei servizi al fine di rendere i servizi stessi altamente riutilizzabili,distribuito e open source. A tale scopo Servicemix implementa a livello di middleware un insieme di API in grado di fornire servizi fondamentali di messaggistica event-driven basata su XML per architetture orientate al servizio, come i Web Services. Tra le varie funzionalità offerte da Servicemix c'è anche una libreria dedicata all'implementazione delle API e della messaggistica di WS-Notification con strutturazione simile a quella di GT4.
- **`WS-Messenger (WSMG)`**[51]: un software open source che implementa un sistema per la gestione degli eventi tra Web Services basato sugli standard WS-Notification e WS-Eventing. L'applicativo comprende sia la realizzazione di un server testuale (NotificationBroker) in grado di gestire le sottoscrizioni e notificare gli eventi, che l'implementazione di un client (Subscriber) provvisto di interfaccia grafica,con il quale l'utente può gestire le operazioni fondamentali sia per la registrazione agli eventi che per la ricezione dei messaggi di notifica. WSMG rappresenta un software accessibile direttamente dall'utente finale che sfrutta in maniera completa le potenzialità di WS-Notification (oltre all'applicazione discussa in questa tesi).
- **`Apache Muse`**: un framework sul quale è possibile costruire interfacce di Web Services per la gestione di *resource* senza la necessità di implementare l'intera architettura descritta nei relativi standard.

Dato che GT4 e Servicemix implementano WS-Notification solo a livello di middleware,cioè forniscono solo l'implementazione delle API e della messaggistica,possono solamente essere utilizzati come tecnologia intermedia per la realizzazione di architetture basate su WS-Notification.
Per quanto riguarda il WSMG:

- implementa la specifica WS-BrokeredNotification,ossia presenta uno scenario dove agiscono almeno le figure NotificationProducers,NotificationBroker(Server)e NotificationConsumers, cioè uno scenario con la presenza del broker;
- implementa la specifica WS-Topics;
- NotificationProducer,NotificationConsumer e NotificationBroker possono essere eseguiti su macchine differenti.

5.3.8 Apache Muse

Il progetto Apache Muse[39] è un'implementazione Java delle specifiche:

- WS-ResourceFramework(WSRF)[40]
- WS-BaseNotification(WSN)[41]
- WS-DistributedManagement(WSDM)[42]

E rappresenta la soluzione scelta in questo contesto per analizzare una possibile implementazione del WS-BaseNotification introdotto nel paragrafo 4.3.

Lo scenario che vedremo comunque non si basa solo sullo standard WS-BaseNotification:rappresenta un'esempio di come è possibile esporre via Web Service delle *resources*(producer e consumer) che comunicano attraverso lo scambio di messaggi(subscribe,unsubscribe…su protocollo SOAP) utilizzando gli standard WSN,WSRF e WSDM.

Brevemente il WS-Resource Framework consiste in un insieme di specifiche che definiscono un pattern per lo scambio di messaggi per interrogare le proprietà delle risorse stateful e per stabilire come queste possono essere modificate:
- WS-ResourceLifetime,
- WS-ResourceProperties,
- WS-RenewableReferences,
- WS-ServiceGroup
- WS-BaseFaults.

Le specifiche definiscono sia i metodi per l'associazione tra *Web service* e *stateful resource* sia le modalità di creazione, accesso, monitoraggio e distruzione di una *WS-Resource*.
Lo scopo del WS-RF è quello di definire come queste operazioni devono essere specificate all'interno del WSDL (Web Services Description Language) che descrive il servizio.
Un Web Service viene quindi definito come un servizio che espone le proprietà di una resource tramite WS-RF.

Il WSDM è uno standard che consente la gestione delle applicazioni costruite usando Web Services,permettendo alle risorse di essere controllate da diversi gestori attraverso una singola interfaccia.
Il WSDM in realtà è formato da due specifiche,Management Using Web Services (MUWS) e Management Of Web Services (MOWS). MUWS definisce come rappresentare ed accedere alle interfacce di gestione delle risorse come Web Services;definisce un insieme base di capacità di gestione,come l'identificazione di una risorsa,la metrica,la configurazione e le relazioni con le altre risorse.
Il MOWS definisce invece come gestire Web Services come risorse e come descrivere ed accedere alle funzionalità di gestione usando MUWS.

La specifica utilizza alcuni SOAP Header proprietari e altri "adottati" da specifiche pre-esistenti, come WS-Addressing per instradare i messaggi e dichiarare per esempio il destinatario (To), il contesto di dialogo (MessageID e RelatesTo),a chi rispondere (ReplyTo) e a chi inviare eventuali errori (FaultTo),oltre ovviamente all'oggetto del messaggio SOAP (Action).

Le operazioni codificate da WS-Management,con le relative action (come da specifica) sono:

```
Get e GetResponse
Put e PutResponse
Create e CreateResponse
Delete e DeleteResponse
Rename e RenameResponse
Enumerate ed EnumerateResponse
Pull e PullResponse
Renew e RenewResponse
GetStatus e GetStatusResponse
Release e ReleaseResponse
EnumerationEnd
```

```
Subscribe e SubscribeResponse
Renew e RenewResponse
GetStatus e GetStatusResponse
Unsubscribe e UnsubscribeResponse
SubscriptionEnd
Events
Heartbeat
DroppedEvents
Ack
Event
```

Queste operazioni consentono di costruire dialoghi o comunicazioni tra un client e un servizio che espone le risorse di un sistema,per monitorarne lo stato,per leggerne,scriverne o modificarne delle informazioni,per abbonarsi a degli eventi di notifica, eventualmente resi sequenziali dalla possibilità di inviare un Ack di conferma ricezione,prima dell'invio di eventuali ulteriori eventi.
La specifica prevede inoltre la possibilità di inviare (tramite il SOAP Header Options) una serie di parametri (options) al sistema,per argomentare l'oggetto della richiesta inviatagli.

Apache Muse può essere quindi definito come un framework che attraverso la definizione di intefacce ed implementazioni di capability WSRF,WSN e WSDM permette sia di costruire interfacce Web Services per la gestione di `resource` che effettuare il deploy di applicazioni in Apache Axis2 e in ambienti OSGi;il progetto comprende infatti una serie di tools che permettono di generare tutto quello che occorre per la costruzione di un ipotetico scenario. Sono disponibili le versioni:

- Apache Muse 2.2.0 source/binary distribution
- Apache Muse 2.1.0 source/binary distribution
- Apache Muse 2.0.0 source/binary distribution
- Apache Muse 1.0.0 source/binary distribution

In questo contesto verrà presa in considerazione ed analizzata l'architettura della versione 2.2.0.
Per installare *muse* basta scompattare il file della distribuzione e copiare la cartella in una qualsiasi directory,ad esempio in `C:\Programmi\muse-2.2.0-bin`.

5.3.9 Caratteristiche

Apache Muse 2.2.0 include[39]:

- Implementazione di WSRF 1.2,WSN 1.3 e WSDM 1.1.
- Un' implementazione standalone del WSDM Event Format 1.1.
- Conformità con WS-Addressing 1.0 e SOAP 1.2.
- Deployment su piattaforme J2EE e OSGi-based.
- Modello di programmazione comune per la definizione dei tipi di risorse su differenti piattaforme host.
- Separazione del packaging delle Api dall'implementazione;in questo modo è possibile scrivere implementazioni diverse di interface Muse e di caricarle senza includere codice inutilizzato.
- Aggregazione delle classi Java Bean in un singolo modello di stato per WSRF.
- Insieme di *utility APIs* per costruire *common tasks* associati con *resource properties,service groups,relationships,*scenari publish-subscribe.
- A persistence API per permettere agli utenti di recuperare lo stato di una WS-resource a seguito di shutdown dell'host.
- Un WSDL-to-Java client generation tool.
- Completo WSDL-to-Java tooling che esegue il parses WSDL e crea codice client-side e server-side.

5.3.10 Architettura

Apache Muse definice il concetto di *capability* in questo modo:

Ogni resource consiste in una collezione di piccole unità di funzione chiamate capabilities. Il concetto di capability è stato descritto inizialmente in WSDM MUWS 1.0 come mezzo per informare i client circa i dati,le operazioni e i comportamenti che devono aspettarsi da una resource. Il modo in cui sono state definite nella specifica WSDM MUWS ricorda il modo in cui gli sviluppatori Java dividono parti di software in moduli più piccoli ognuno con un proprio task;l'aggregazione di capability come Identity,Configuration e Relationship in singole interfacce web Service suggerisce un modello simile per l'implementazione server-side. Muse ha generalizzato il concetto di capability presentato nel WSDM MUWS per rendere più semplice ai programmatori l'implementazione di resource come unità di funzioni riutilizzabili piuttosto che come singole classi Java monolitiche.

L'architettura di Apache Muse si basa su quattro concetti base[43]:

- **`The Capability`** – Una *resource capability* è una collezione di dati ed operazioni esposti via Web Services. Rappresenta un'unita atomica dell'architettura di Muse. Le *capabilities* sono aggregate per definire i *resource types* e sono implementate come semplici classi java. Le classi *capability* sono analizzate *at initialization time* per determinare il modo in cui si inseriscono nell'interfaccia mostrata dal WSDL della *resource*.
- **`The Resource`** – Una *resource* è una collezione di *capabilities* esposte attraverso un'interfaccia Web Services. Rappresentano anche l'interfaccia utilizzata dalle capabilities che intendono comunicare con altre capabilities.
- **`The Implied Resource Pattern`** – Ogni *resource* ha un unico *endpoint reference (EPR)*. Questo *data type* è definito dallo standard *WS-Addressing* e contiene le basic *networking* e le *runtime-specific data* necessari a localizzare e ad invocare le operazioni. L' *implied resource pattern* consente agli utenti di aggiungere coppie *name-value* di un EPR per distinguere tra più istanze di un tipo di risorsa comuni ad un indirizzo(URL).
- **`The Isolation Layer`** – Le *Muse resources* possono essere deployate in differenti ambienti - J2EE application servers, OSGi, etc. – ma è previsto un unico modello di programmazione. E'possibile implementare una *resource type* ed effettuare il relativo deploy in ambienti eterogenei con delle semplici modifiche al *deployment artifacts* senza modificare il codice della *resource*.

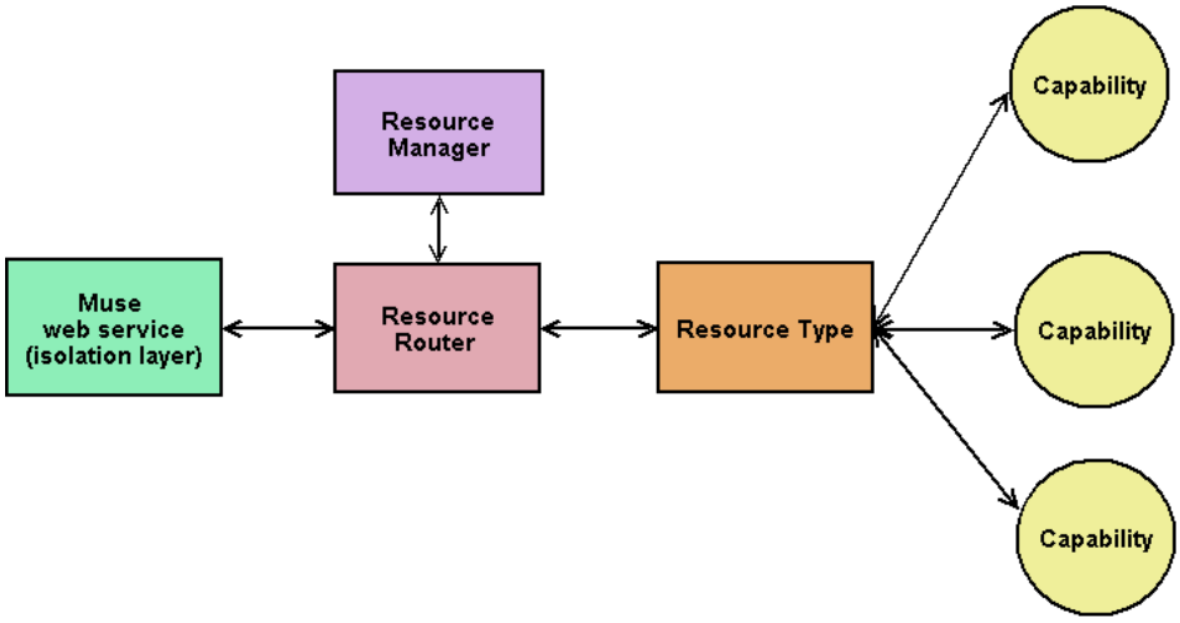


Figura 58:Architettura di Apache Muse

La logica del servizio viene implementata nel capability layer;quello che serve è un modo per combinare insieme le unità prodotte per rispecchiare quanto definito nell' interfaccia WSDL che i *client* utilizzano per analizzare la *resource*. La *resource* esegue effettivamente l'aggregazione e fornisce ai web services una singola interfaccia per gestire le richieste SOAP in arrivo. Muse utilizza quindi il concetto di capability come strategia d'implementazione di un servizio;il *SOAP engine*,l' *isolation layer* e il *router* non sono a conoscenza di questo concetto di modello di programmazione. Ad esempio un'*application server resource* implementata con le capability *WSN NotificationProducer,WSRL ImmediateResourceTermination* e *AppServerDeployment* nasconde l'aggregazione in questo modo:

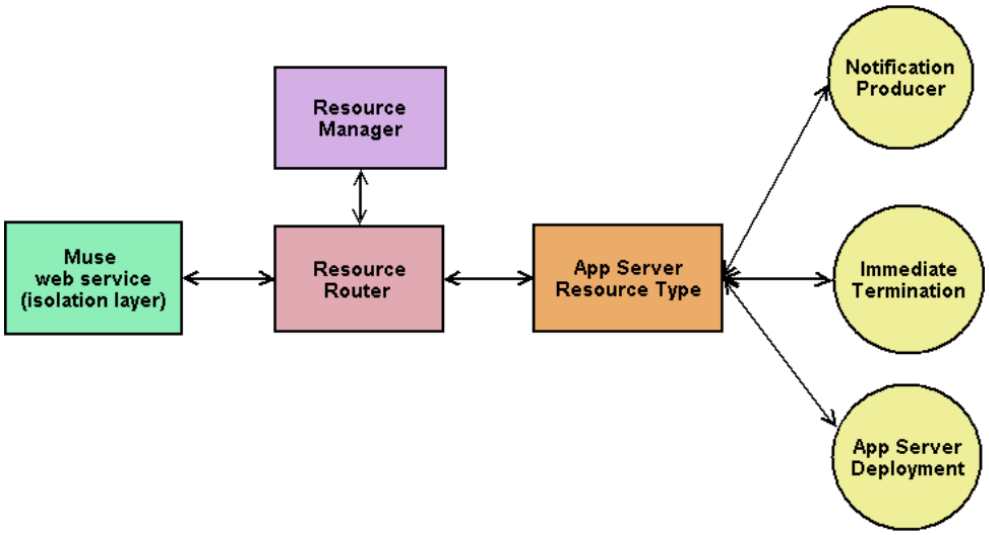


Figura 59:Costruire una *resource* attraverso le *capability*

Il fatto che l'operazione di *Subscribe* sia gestita dalla capability *WSN NotificationProducer* e che l'operazione di *DeployApplication* sia invece gestita dalla capability *AppServerDeployment* è irrilevante per il *resource router* e per tutti gli componenti che gestiscono le richieste. Una volta che la richiesta è stata delegata all'appropriata *resource instance*,solo la *resource* sa che l'operazione in corso è implementata da una altro layer:il *capability object* che definisce il metodo Java equivalente all'operazione richiesta. Il *Java object* rappresenta la *resource instance* che eseguirà i task e che delegherà le rishieste alle relative *capability* sulla base delle informazioni scoperte *at initialization time*. Quando una *resource* viene creata,il *Muse framework* utilizza il WSDL della *resource* per determinare cosa la *resource* ha promesso di sostenere e se è in grado effettivamente di farlo;tale verifica viene effettuata analizzando le definizioni delle capability della resource(incluse le implementazioni delle loro classi java) per essere sicuri che le operazioni previste nel *WSDL port type* presentino equivalenti metodi Java nella collezione di *capablities*.

Infine una *resource* deve garantire che tutte le *capabilities* ricevano la corretta notifica circa gli eventi del proprio ciclo di vita,per dare loro modo di prendere le misure adeguate nelle fasi di *initialization* e di *shutdown*. Ogni interfaccia Capability definita da Muse è composta da quattro *lifecycle events*:

- **`Initialization`** - *Capability.initialize()* – Quando la *capability* è in grado di effettuare tutti i self-contained startup tasks,e non richiede l'utilizzo delle altre *capabilities* presenti nella *resource*.
- **`Post-initialization`** - *Capability.initializeCompleted()* – Quando la *capability* può effettuare tutti i rimanenti startup tasks;in particolare,può interrogare e manipolare altre *capabilities* che si trovano in uno stato stabile.
- **`Pre-shutdown`** - *Capability.prepareShutdown()* – Quando la *capability* può effettuare tutti gli shutdown tasks che riguardano l'interrogazione e la manipolazione di altre

capabilities. Questo fase è definita come *last call* - possibilità per la *capability* di ottenere ciò di cui ha bisogno dagli altri componenti prima del loro shutdown e quindi prima che la loro stabilità non sia più garantita.

- **Shutdown** - *Capability.shutdown()* – Disponendo dei dati necessary per il pre-shutdown, la *capability* può ora effettuare il self-contained shutdown tasks,includendo persistence, configuration, o notifications.

Anche in Apache Muse il WSDL è un documento utilizzato per definire l'interfaccia di un Web Services che client remoti utilizzano per comunicare con le *resource*. Oltre a fornire l'interfaccia di un servizio al client,il WSDL di una resource viene utilizzato *at initialization time* per determinare quali richieste sono valide,quali Api dovrebbero essere utilizzate e come convertire i dati da elementi XML a oggetti Java e viceversa. Il wsdl conferma quanto contenuto nel deployment descriptor muse.xml e fornisce i dettagli necessari per processare le richieste SOAP.

Ad esempio,il WSDL relativo al publisher utilizzato nello scenario che analizzaremo più avanti,è il seguente:

```
    <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://ws.apache.org/muse/test/wsrf"
    xmlns:tns="http://ws.apache.org/muse/test/wsrf"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdl-
    soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex" xmlns:wsrf-
    r="http://docs.oasis-open.org/wsrf/r-2" xmlns:wsrf-rl="http://docs.oasis-
    open.org/wsrf/rl-2" xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
    xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
    xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
    xmlns:wsntw="http://docs.oasis-open.org/wsn/bw-2"
    xmlns:wst="http://docs.oasis-open.org/wsn/t-1"
    xmlns:wsrmd="http://docs.oasis-open.org/wsrf/rmd-1"
    xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"
    xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd"
    name="WsResource">
  + <wsdl:types>
  + <wsdl:message name="GetMetadataMsg">
  + <wsdl:message name="GetMetadataResponseMsg">
  + <wsdl:message name="DestroyRequest">
  + <wsdl:message name="DestroyResponse">
  + <wsdl:message name="ResourceNotDestroyedFault">
  + <wsdl:message name="ResourceUnknownFault">
  + <wsdl:message name="ResourceUnavailableFault">
  + <wsdl:message name="SetTerminationTimeRequest">
  + <wsdl:message name="SetTerminationTimeResponse">
  + <wsdl:message name="UnableToSetTerminationTimeFault">
  + <wsdl:message name="TerminationTimeChangeRejectedFault">
  + <wsdl:message name="GetResourcePropertyDocumentRequest">
  + <wsdl:message name="GetResourcePropertyDocumentResponse">
  + <wsdl:message name="GetResourcePropertyRequest">
  + <wsdl:message name="GetResourcePropertyResponse">
  + <wsdl:message name="InvalidResourcePropertyQNameFault">
  + <wsdl:message name="GetMultipleResourcePropertiesRequest">
  + <wsdl:message name="GetMultipleResourcePropertiesResponse">
  + <wsdl:message name="QueryResourcePropertiesRequest">
  + <wsdl:message name="QueryResourcePropertiesResponse">
  + <wsdl:message name="UnknownQueryExpressionDialectFault">
  + <wsdl:message name="InvalidQueryExpressionFault">
  + <wsdl:message name="QueryEvaluationErrorFault">
```

```
+ <wsdl:message name="SetResourcePropertiesRequest">
+ <wsdl:message name="SetResourcePropertiesResponse">
+ <wsdl:message name="InvalidModificationFault">
+ <wsdl:message name="UnableToModifyResourcePropertyFault">
+ <wsdl:message name="SetResourcePropertyRequestFailedFault">
+ <wsdl:message name="SubscribeRequest">
+ <wsdl:message name="SubscribeResponse">
+ <wsdl:message name="SubscribeCreationFailedFault">
+ <wsdl:message name="TopicExpressionDialectUnknownFault">
+ <wsdl:message name="InvalidFilterFault">
+ <wsdl:message name="InvalidProducerPropertiesExpressionFault">
+ <wsdl:message name="InvalidMessageContentExpressionFault">
+ <wsdl:message name="UnrecognizedPolicyRequestFault">
+ <wsdl:message name="UnsupportedPolicyRequestFault">
+ <wsdl:message name="NotifyMessageNotSupportedFault">
+ <wsdl:message name="UnacceptableInitialTerminationTimeFault">
+ <wsdl:message name="GetCurrentMessageRequest">
+ <wsdl:message name="GetCurrentMessageResponse">
+ <wsdl:message name="InvalidTopicExpressionFault">
+ <wsdl:message name="TopicNotSupportedFault">
+ <wsdl:message name="MultipleTopicsSpecifiedFault">
+ <wsdl:message name="NoCurrentMessageOnTopicFault">
+ <wsdl:portType name="WsResourcePortType" wsrf-
    rp:ResourceProperties="tns:WsResourceProperties"
    wsrmd:Descriptor="WsResourceMetadata"
    wsrmd:DescriptorLocation="WsResource.rmd">
+ <wsdl:binding name="WsResourceBinding" type="tns:WsResourcePortType">
+ <wsdl:service name="WsResourceService">
    </wsdl:definitions>
```

Come è logico aspettarsi il documento presenta la struttura classica di ogni file WSDL:pertanto valgono le stesse considerazioni effettuate precedentemente(paragrafo 1.4.2.1).

`<wsdl:types>`
Nell'elemento types vengono importati gli XML Schema files utilizzati dalla resource.

```
<xsd:schema elementFormDefault="qualified"
    targetNamespace="http://www.w3.org/2005/08/addressing">
  <xsd:include schemaLocation="WS-Addressing-2005_08.xsd" />
    </xsd:schema>
```

`<wsdl:message>`
Gli elementi `message` definiscono i messaggi scambiati tra il servizio e i client. Da notare il prefisso utilizzato per specificare lo standard al quale appartiene l'operazione indicata.
Ad esempio il messaggio `getMetadataMsg|SubscribeRequest|DestroyRequest` è generato quando viene richiesto di eseguire l'operazione `GetMetadata|Subscribe|Destroy` definita dallo standard `WS-MetadataExchange|WS-NotificationProducer|WS-ResourceFramework` con prefisso `wsx|wsnt|wsrf-rl`. WSRF-RL è il prefisso della specifica WS-ResourceLifetime dello standard WS-ResourceFramework:come detto in precedenza è appunto una delle cinque specifiche di cui è composto WS-RF.

```
<wsdl:message name="GetMetadataMsg">
  <wsdl:part name="GetMetadataMsg" element="wsx:GetMetadata" />
    </wsdl:message>
<wsdl:message name="SubscribeRequest">
  <wsdl:part name="SubscribeRequest" element="wsnt:Subscribe" />
    </wsdl:message>
<wsdl:message name="DestroyRequest">
  <wsdl:part name="DestroyRequest" element="wsrf-rl:Destroy" />
    </wsdl:message>
```

```
<wsdl:portType>
<wsdl:portType name="WsResourcePortType" wsrf-
    rp:ResourceProperties="tns:WsResourceProperties"
```

```
    wsrmd:Descriptor="WsResourceMetadata"
    wsrmd:DescriptorLocation="WsResource.rmd">
  <wsdl:operation name="GetMetadata">
  <wsdl:input wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata"
    name="GetMetadataMsg" message="tns:GetMetadataMsg" />
  <wsdl:output
    wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadataResponse"
    name="GetMetadataResponseMsg" message="tns:GetMetadataResponseMsg" />
    </wsdl:operation>
```

`<wsdl:binding>`

Da notare che lo stile di codifica è `document/encoder` e il trasporto utilizzato è `HTTP`.

```
<wsdl:binding name="WsResourceBinding" type="tns:WsResourcePortType">
  <wsdl-soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="GetMetadata">
  <wsdl-soap:operation soapAction="GetMetadata" />
  <wsdl:input>
  <wsdl-soap:body use="encoded"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </wsdl:input>
  <wsdl:output>
  <wsdl-soap:body use="encoded"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </wsdl:output>
    </wsdl:operation>
```

`<wsdl:service>`

```
<wsdl:service name="WsResourceService">
 <wsdl:port name="WsResourcePort" binding="tns:WsResourceBinding">
  <wsdl-soap:address location="http://localhost:8080/wsn-
    producer/services/WsResource" />
    </wsdl:port>
    </wsdl:service>
```

5.3.11 Deployment Descriptor

Il Muse deployment descriptor è chiamato muse.xml. Il file *muse.xml* contiene informazioni circa le *WS-resources* che sono state implementate e viene utilizzato per inizializzare le strutture dati necessarie per validare e girare le richieste ai corretti *capability objects*. In altre parole,è il file che abilita Muse ad aggregare le varie *capability class* in un *resource type(s).*
Per analizzare la struttura generale del file prendo in considerazione lo stesso muse.xml del publisher wsn-producer che presenterò successivamente(nel paragrafo 5.3.13.2).

L'elemento radice *-root element-* del *Muse deployment descriptor* è <muse/>. Contiene una sequenza di tre *child elements*:

- `<router/>` - I dati di configurazione del core engine's router, il quale *maps* le rischieste SOAP all'interno di *actual Java method calls*.
- `<resource-type/>` - I dati di configurazione di una *resource* esposa come servizio. E' qui che si mettono insieme le informazioni provenienti da altri deployment artifacts (come WSDL e XSD) e Java code per permettere il runtime con le informazioni necessarie a creare e manipolare le istanze di un *resource type*.
- `<custom-serializer/>` - Funzione opzionale che consente di estendere il sistema di serializzazione Muse XML per gestire i tipi complessi.

5.3.11.1 Il Router

L'elemento <router/>, contiene i dati relativi alla configurazione delle seguenti cose:

- `<logging/>` - Il sistema di logging dove i messaggi sono memorizzati.
- `<java-router-class/>` - Le classi Java che implementano il router.

- `<persistence/>` - Il meccanismo (opzionale) di persistenza dove memorizzare le router entries.

```
      <router>
            <java-router-
class>org.apache.muse.core.routing.SimpleResourceRouter</java-router-class>
            <logging>
                  <log-file>log/muse.log</log-file>
                  <log-level>FINE</log-level>
            </logging>
            <persistence>
                  <java-persistence-
class>org.apache.muse.core.routing.RouterFilePersistence</java-persistence-
class>
                  <persistence-location>router-entries</persistence-location>
            </persistence>

      </router>
```

Logging

Il sistema di Logging è realizzato tramite le JDK logging API. I due elementi <logging/> - <log-file/> e <log-level/> - consentono di specificare rispettivamente se il file di log Muse sarà scritto e a quale livello livello di dettaglio.

Il *log file path* deve essere relativo alla directory di lavoro corrente dell'applicazione;nel caso di J2EE,questo è la root del WAR. Il nostro progetto creerà infatti un log file nella cartella */WEB-INF/services/muse/log/muse.log*. Questo file sarà sovrascritto(overwritten) ogni volta che l'applicazione *restarted*.

Il log level deve essere uno dei valori previsti nel JDK log level enumeration: OFF, SEVERE, WARNING, etc. La lista completa si trova nel *descriptor's XML schema* come anche le *API documentation* per il java.util.logging.Level.

```
 <xs:element name="log-level">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="OFF"/>
        <xs:enumeration value="SEVERE"/>
        <xs:enumeration value="WARNING"/>
        <xs:enumeration value="INFO"/>
        <xs:enumeration value="CONFIG"/>
        <xs:enumeration value="FINE"/>
        <xs:enumeration value="FINER"/>
        <xs:enumeration value="FINEST"/>
        <xs:enumeration value="ALL"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
```

Il livello di logging di default è INFO. In questo caso verranno stampati sia messaggi di errore e *stack trace* per ogni errore verificatosi durante la fase di inizializzazione o *request handling*,sia messaggi di conferma relativi sempre alla fase di inizializzazioen. Incrementando il livello di logging a FINE(un livello piiù alto,come nel nostro esempio) sarà possibile memorizzare anche ogni messaggio SOAP di richiesta e di risposta processato. In questo caso *log messages* da entrambi i router e *resource instances* verranno visualizzate side-by-side.

Il fatto che il sistema di logging è configurato al livello di router significa che queste impostazioni vengono applicate a tutti i *resource types* menzionate nel descrittore.

The Router Class

Il componente router di Muse maps *WS-Addressing headers* da *SOAP request* in *actual Java method call* di *resource instance*. Quando un messaggio SOAP arriva all'applicazione,la classe che implementa ail servizio eseguirà i seguenti passi:

- Effettua il parse del messaggio SOAP all'interno di una strutttura dati ed estrae i *WS-Addressing headers.*
- Fornisce un *per-request (per-thread) context* dove il router e gli altri componenti possono condividere l'informazione su queste richieste.
- Viene estratto il *WS-Addressing information* e il *SOAP body* (XML) dal router.

Una volta ottenute queste informazioni il router può svolgere i seguenti tasks:

- Ricerca la collezione di *resource instance* per trovare quella associata con il dato *addressing data*.
- Verifica se la *resource* è in grado di gestire l'azione specificata nel messaggio;in questo caso, chiede alla *resource* di invocare tale l'azione usando il SOAP body come input.
- Il router prende il valore di ritorno del metodo e lo passa di nuovo al servizio che lo inserisce nel SOAP envelope e invia la risposta al chiamante.

Si noti che la *resource* gestisce tutti i processamenti del SOAP body - il *router* si occupa solo delle intestazioni SOAP e della semantica di WS-Addressing. In questo modo è possibile limitare le modifiche da apportare all'implementazione del router,anche se questa operazione risulta essere particolarmente semplice se si utilizza il *deployment descriptor*.

L'elemento <java-router-class/> specifica il nome della classe Java che implementa il router component. Questa classe deve implementare l'interfaccia `org.apache.muse.core.routing.ResourceRouter`. L'implementazione di default di questa interfaccia è `org.apache.muse.core.routing.SimpleResourceRouter`.

Persistence

Il *Muse deployment descriptor* permette di configurare un *persistence layer* per memorizzare le *router's entry* (mapping EPR-to-resource) e per effettuarne il *reload* dopo che l'host viene riattivato(restate).

Questo meccanismo è generico in quanto non definisce uno specifico *data store* – per memorizzare le *entries* è possibile usare un semplice file oppure utilizzare un sistema più complesso basato su *Database*. Muse offre una implementazione di default del router di persistenza che si basa sul file system - memorizza un file unico per ogni EPR nel router e ordina i file per *resource type*. È possibile utilizzare questo tipo di soluzione per iniziare per poi sostituirla con qualcosa di più performante,se necessario.

Per abilitare il meccanismo di persistenza per le *router entries*,bisogna inizialmente aggiungere l'elemento opzionale <persistence/> sotto <router/>. Questo elemento è composto da due *child elements*:

- `<persistence-location/>` - L'URI che descrive dove le router entries saranno memorizzate.

```
<persistence-location>router-entries</persistence-location>
```

- `<java-persistence-class/>` - La classe Java che implementa le RouterPersistence API.

```
<java-persistence-class>
org.apache.muse.core.routing.RouterFilePersistence
```

```
        </java-persistence-class>
```

Nel caso si utilizzi l'implementazione di default del meccanismo della persistenza, la *location* è data dal path relativo della directory dove si vogliono memorizzare le *entries*. Se viene configurato il valore di <persistence-location/> a *router-entries*, allora un'applicazione basata su Axis2-based creerà la directory */WEB-INF/services/muse/router-entries*.

Il nome della classe Java responsabile dell'implementazione di default è org.apache.muse.core.routing.RouterFilePersistence. E'possibile estendere o sostituire tale classe per modificare rispettivamente il tipo e il comportamento del meccanismo di persistenza.

5.3.11.2 The Resource Types

L'elemento <resource-type/> indica dove vengono specificati i dati relativi al deployment delle *resources* definite dalle Api Muse e dagli altri web services artifacts. La maggior parte delle *descriptor customization* verrà effettuata qui,anche se in molti i casi basteranno i valori di default specificati nel *sample descriptor* per soddisfare le esigenze richieste.

L'elemento<resource-type/> include:

- `<context-path/>` - Il context path (unique URI) del *resource type*.
- `<wsdl/>` - Il WSDL del *resource type*.
- <java-id-factory-class/> - La classe Java usata per definire *l'endpoint reference* per ogni *resource instance*.
- `<java-resource-class/>` - La classe Java usata per implementare le *Muse resource*, il cui compito è quello di aggregare le capabilities.
- `<capability/>` - Una *capability* è definita da un unico URI ed è implementata da una classe Java.

The Context Path

Ogni *resource type* esposta come web service deve avere un proprio URI ed un proprio WSDL port type. Per applicazioni che espongono più di un *resource type,*l'elemento <context-path/> rappresenta un modo per mappare i vari suffissi URI per una stessa applicazione web service. Ogni *resource type* condividerà un comune URI che individua l'host dell'applicazione;il suffisso farà in modo che ogni *resource type* avrà un proprio endpoint SOAP dal punto di vista del *client*.

Se il context-path è */WsResource*:tutti gli EPR creati per l'istanza di questa *resource type* avranno il seguente indirizzo:

```
http://[host]/wsn-producer/services/WsResource
```

Anche nel caso in cui si sia più di un *resource type*(e quindi,più di un URI),esiste comunque una sola istanza del servizio Muse,e una sola istanza del router Muse. L'elemento <context-path/> fornisce una funzione simile al mapping servlet caratteristico del file *web.xml* di J2EE. In questo modo è possibile mantenere interfacce di risorse remote completamente separate nonostante il fatto alcune *resource* potrebbero condividere lo stesso *application space*.
I context-path del nostro file muse.xml sono infatti due:

```
<context-path>SubscriptionManager</context-path>
<context-path>WsResource</context-path>
```

The WSDL File

Nell'elemento <wsdl/> è possible distinguere due tipi d'informazione – la locazione attuale del file WSDL (in <wsdl-file/>) e il nome del *port type* che definisce l'interfaccia della *resource* (in <wsdl-port-type/>).

```
<wsdl>
      <wsdl-file>wsdl/WS-BaseNotification-1_3.wsdl</wsdl-file>
      <wsdl-port-type>wsntw:SubscriptionManager</wsdl-port-type>
</wsdl>
<wsdl>
      <wsdl-file>wsdl/WsResource.wsdl</wsdl-file>
      <wsdl-port-type>test:WsResourcePortType</wsdl-port-type>
</wsdl>
```

Come nel *log file path*, il path del WSDL dipenderà dalla directory di lavoro dell'applicazione. should. Il *sample WSDL* è in */WEB-INF/services/muse/wsdl* insieme agli schemi che importa. Tutti gli elementi importati contenuti nel WSDL devono avere il path relativo ad esso.

Muse analizzaza il file WSDL con il WSDL4J API in cerca dei *port type* specificati nell'elemento <wsdl-port-type/>. Si assicurerà quindi che tutte le operazioni definite nel *port type* corrisponderanno a metodi Java nella *resource* analizzando le relative classi *capability* finchè non trova una corrispondenza. Il matching viene effettuato convertendo in minuscolo il nome locale del tipo usato per il messaggio di richiesta e confrondandolo con il nome del metodo ritornato dal *Class.getMethods() method*.

The Resource ID Factory Class

Il resource pattern suggerisce l'utilizzo del WS-Addressing reference parameters per distinguere tra EPR che presentano uno stesso indirizzo. Questo è importante quando si effettua il deploy di un *resource type* che presenta istanze multiple perché il router Muse, quando processa una richiesta SOAP,si basa sul mapping dagli EPR alle *resources.*

Muse offre l'elemento <java-id-factory-class/> al fine di agevolare la creazione di EPR senza richiedere agli utenti di costruire l'intero EPR:

```
<java-id-factory-class>
org.apache.muse.core.routing.CounterResourceIdFactory
</java-id-factory-class>

<java-id-factory-class>
org.apache.muse.core.routing.CounterResourceIdFactory
</java-id-factory-class>
```

Il valore di questo elemento corrisponde alla classe Java che implementa l'interfaccia the `org.apache.muse.core.routing.ResourceIdFactory`. Questa interfaccia presenta due semplici metodi: *getIdentifierName()* e *getNextIdentifier()*. Il primo consente agli utenti di specificare il nome del parametro di riferimento EPR; il secondo permette di generare un unico valore per questo parametro nel modo che si ritiene più opportuno.

Muse fornisce due *factory implementations* da usare in situazioni in cui i paramentri EPR non corrispondono allo stesso set di identificatori esistenti oppure quando vengono estratti da un certo *data store*. Le classi di default sono CounterResourceIdFactory e RandomResourceIdFactory. La prima crea identificatori del tipo *MuseResource-N* (dove *N* è un intero monotonicamente crescente); la seconda genera un random UUID usando il JDK. Entrambe le classi utilizzano il nome di default fornito da Muse, *muse-wsa:ResourceIdentifier.*

The Resource Class

L'elemento <java-resource-class/> contiene il nome della classe Java che rappresenta la *resource type*. Questa classe deve implementare l'interfaccia `org.apache.muse.core.Resource`.

```
<java-resource-class>
org.apache.muse.ws.resource.impl.SimpleWsResource
</java-resource-class>
```

```
<java-resource-class>
org.apache.muse.ws.resource.impl.SimpleWsResource
</java-resource-class>
```

La *resource class* ha il compito di istanziare ed inizializzare tutte le *resource's capabilities* e di delegargli la responsabilità di invocare una determinata operazione nel momento in cui il router lo richieda. La *resource class* contiene tutte le strutture dati comuni che saranno necessarie alle capabilities(il log writer, il resource manager, etc.) e permette di avere più controllo sulle routine di inizializzazione e distruzione nel caso si stia lavorando con capabilities non eccessivamente flessibili.

Resource è anche l'interfaccia che la classe capability utilizza per accedere ad altre capability. Ad esempio,la capability WS-resource deve accedere alla capability WSN NotificationProducer per effettuare la sottoscrizione di un altra *resource* per gli eventi che sta producendo.

Spesso le capabilities saranno costruite al di sopra di altre per eseguire correttamente il loro lavoro;l'interfaccia Resource rappresenta quindi un *central hub* che permette di ottenere un simile comportamento nel contesto in cui queste capabilities si trovano.

Il *sample project* utilizza *org.apache.muse.core.SimpleResource* che rappresenta l'implementazione di default di Resource. I metodi *initialize()* e *shutdown()* inizializzano e terminano le capabilities elencate sotto l'elemento <resource-type/>; l'implementazione corrente inizializza le capabilities nell'ordine in cui sono elencate, ma il design del processo di inizializzazione è pensato per gestire un ordinamento non deterministico delle capabilities, e non è possibile scrivere *capability code* che dipende dall'ordine del deployment descriptor.

Capabilities

All'interno di ogni elemento <resource-type/> è possible trovare uno o più elementi <capability/>. Questi elementi definiscono standard e *custom capabilities*. Ogni capability è caratterizzata da un unico URI(usando l'elemento <capability-uri/>) ed è implementata da una classe Java (specificata dall'elemento<java-capability-class/>).

Le *capability URIs* sono usate da *Capability* per interrogare l'oggetto *Resource* che le contiene e per cercare quali altre *capabilities* sono disponibili. La *Resource* fornisce un sistema di lookup *URI-to-Capability* che permette a parti di codice di accedere a specifiche funzionalità delle resource. Ad esempio, l'implementazione della capability WS-Notification NotificationProducer che crea istanze del resource SubscriptionManager resource type avrà bisogno di accedere alla capability WSRL ScheduledResourceTermination dell'oggetto resource per configurare le proprietà *wsrf-rl:TerminationTime* durante l'inizializzazione. E'possibile fare questo chiamando il metodo Resource.getCapability() su *Resource* usando l'URI ScheduledResourceTermination. Ottenuta il *capability object* e inserito nell'appropriata interfaccia, è possibile chiamare i metodi o leggere i dati necessari.

Le *capabilities* che derivano dalle interface standard nelle specifiche OASIS o W3C presentano URI standard o pseudo-standard che dovrebbero essere riutilizzabili. Esempi di *capabilities* con URI standard sono le *capabilities* WSDM MUWS; esempi di *capabilities* con URI pseudo-standard URIs sono WSRF o WSN – le specifiche di questi standard non definiscono speficamente una "capability URI", ma forniscono dei concetti abbastanza simili che Muse può riutilizzare.

Le classi *Capability* dovrebbero avere un'interfaccia Java alla quale l'utente fà riferimento quando si chiamano i metodi della *capability* in modo che questa sia indipendente dall'implementazione della *capability* stessa. Questo permette agli utenti di avere diverse implementazioni di una o più *capabilities* senza creare un effetto domino nel caso si decida di apportare delle modifiche al codice. L'implementazione Muse delle *standard capabilities* fornisce un pattern consistente che può essere esteso nel relativo codice.

5.3.11.3 Creating Custom Serializers

Per gestire tipi di dato particolari come ad esempio complexType bisogna creare una classe che implementa org.apache.muse.core.serializer.Serializer e metterla a disposizione con l'elemento <custom-serializer/> sotto <muse/>.

```
<custom-serializer>
   <java-serializable-type>the new type</java-concrete-class>
   <java-serializer-class>the type's serializer</java-serializer-class>
</custom-serializer>
```

Serializer è una semplice interfaccia per la serializzazione e la deserializzazione di oggetti in XML e viceversa.

Questa versione di Muse contiene molti Serializer *for built-in types*, inclusi primitive e tipi semplici. I *Serializer* di questi *built-in types* sono caricati nel momento in cui Muse esegue lo startup.

Nell'esempio che vedremo la serializzazione è gestita aggiungendo:

```
<custom-serializer>
                <java-serializable-
type>org.apache.ws.muse.test.wsrf.Pratica</java-serializable-type>
                <java-serializer-
class>org.apache.ws.muse.test.wsrf.PraticaSerializer</java-serializer-class>
      </custom-serializer>
```

La *basic rule* seguita da Muse quando processa un messaggio SOAP è che il body dovrebbe contenere *one child element* per ogni *Java method parameter*. Questo non è uno standard,ma è un'utile convenzione che Muse adotta per rendere il parsing il più semplice possible. Usando la regola *one-element-one-parameter*,Muse può fornire facilmente il codice XML corrispondente ai tipi complessi dei quali è necessario eseguire il parse. In questo modo è possibile concentrare l'attenzione sul tipo complesso senza preoccuparsi del resto del body SOAP.

L'interfaccia *Serializer* contiene due metodi - *fromXML()* e *toXML()* – che consentono di convertire istanze di oggetti Java in formato XML e viceversa.

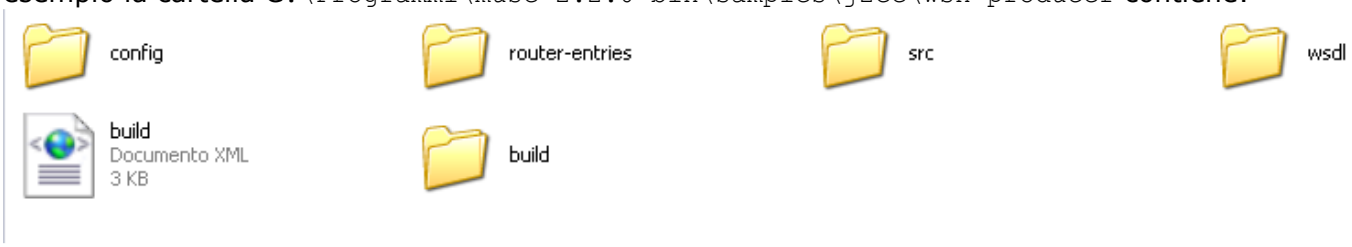
Il serializzatore verrà utilizzato in tutte quelle situazioni nelle quali sarà necessario convertire un frammento XML in un determinato tipo.

5.3.12 Esempio

Per comprendere meglio quanto detto precedentemente vediamo ora come implementare la specifica del WS-BaseNotification in un ipotetico scenario. L'architettura dell'esempio proposto è composta da tre entità principali:

- un producer - pubblica messaggi in due topic differenti e ad intervalli regolari
- due subscriber - che ricevono le informazioni pubblicate nei topic ai quali si sottoscrivono.

L'esempio è stato sviluppato a partire dai Sample Project WSN-Producer e WSN-Consumer messi a disposizione da Apache Muse. Entrambe le directory presentano la stessa struttura;ad esempio la cartella C:`\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-producer` contiene:



- **`/src/package`** – contiene il source code;nel package `org.apache.ws.muse.test.wsrf` sono presenti i seguenti file :
  - MyCapability.java
  - MyCapabilityImpl.java
  - Pratica
  - PraticaSerializer
  - IndiceSerializer
  - La cartella quotazioni contenente lo stub per invocare il servizio QuotazioniWS
- **`/wsdl`** – contiene i file WSDL ed XML Schema della resource implementata. Il file WsResource.wsdl è l'interfaccia WSDL della nostra. Gli XML Schema files definiscono gli standard data type utilizzati nell'interfaccia della resource;tutti gli schemi definiti nello scenario che analizzaremo sono autorizzati da OASIS o W3C e rappresentano le ultime versioni dei rispettivi standard.
- **`/router-entries`** – contiene i file XML relativi agli endpoint references (EPRs) di ogni resource.
- **`/config/muse.xml`** – il descrittore della *resource*.
- **`/config/web.xml`** – il file WAR di configurazione standard
- **`/build.xml`** – il file utilizzato da ant per compilare il package presente in **`/src`** e creare un file WAR con lo stesso nome della cartella dalla quale si esegue il programma e che verrà utilizzato per effettuare il deploy su Tomcat.

La cartella `/build` viene generata dopo aver eseguito il comando ant e presenta la seguente struttura:

```
/wsn-producer
    /build
        wsn-producer.war
        /WEB-INF
                web.xml
                /lib
                /classes
                        /org
                        /quotazioni
                        /wsdl
                        /router-entries
                        muse.xml
```

In questi esempi il producer pubblicava delle semplici stringhe in un topic ad intervalli regolari mentre il consumer dopo aver effettuato la sottoscrizione si metteva in ascolto sul relativo topic e stampava tutte le informazioni ricevute;la sottoscrizione non veniva effettuata direttamente dal consumer al producer ma da una classe java che aveva appunto il compito di notificare ad un determinato producer il fatto che un certo consumer intendeva ricevere tutte le informazioni pubblicate in un topic. Tale classe rappresentava quindi un intermediario tra le due entità con il compito di effettuare le sottoscrizioni.

L'architettura del sistema era quindi la seguente e corrispondeva allo scenario di funzionamento in assenza di un broker:

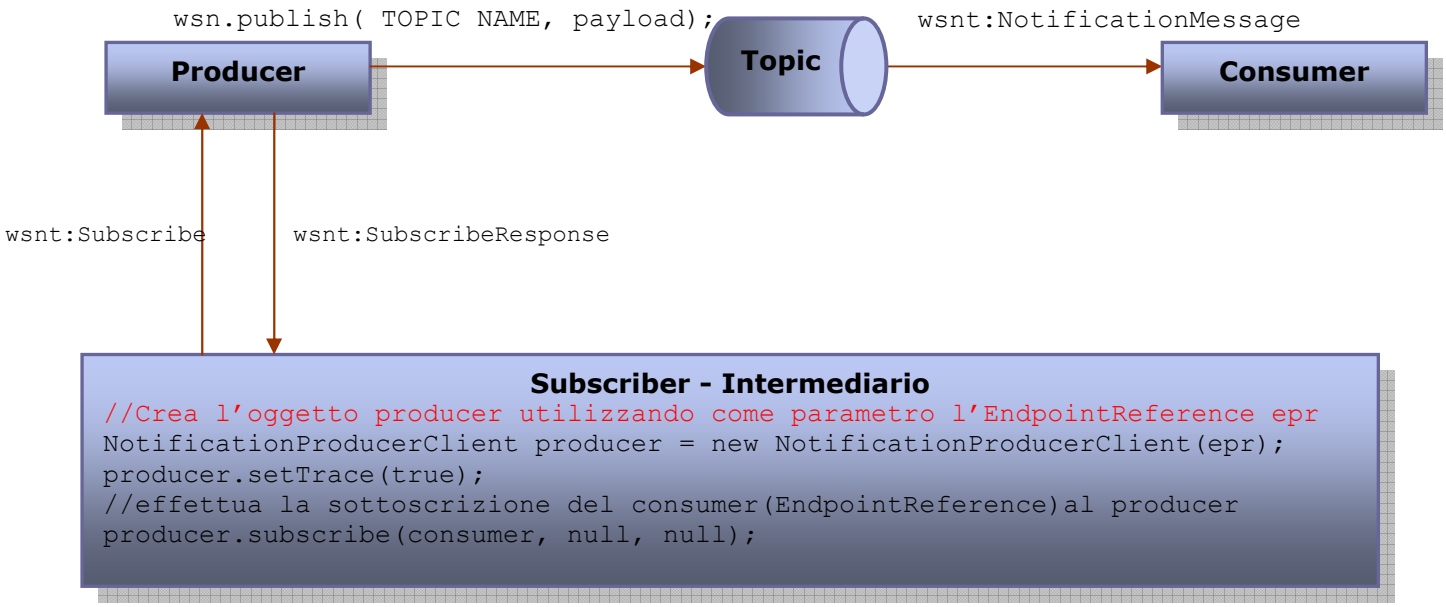


Figura 60:Subscribe,Publish e NotificationMessage in assenza di un broker

Sono quindi partito da questi progetti template per implementare uno scenario leggermente più complesso. Lo scenario realizzato consiste in producer che pubblica in un primo topic le informazioni estratte da un database e relative alle pratiche gestite dal Comune di Ardea e in un secondo le quotazioni di alcuni indici di borsa ottenute dal Web Services QuotazioniWS;il producer si comporta quindi come client di un servizio per utilizzare alcuni metodi dei metodi esposti e pubblicare le informazioni ottenute nel relativo topic. I due consumer,uno deployato su JBoss e l'altro su Tomcat,sottoscrivono il loro interesse a ricevere tali tipi di informazioni mettendosi in ascolto sui topic. Ci sono tre importanti differenze rispetto agli esempi precedenti:

- Non è previsto alcun intermediario:ogni consumer effetturà la propria sottoscrizione ad un determinato producer;
- I messaggi pubblicati non sono semplici stringhe ma complexType:bisogna quindi gestire in modo opportuno la serializzazione/deserializzazione di tali oggetti;
- Viene gestita la persistenza delle sottoscrizioni:viene memorizzata ogni subscribe per permettere ai consumer di continuare a ricevere gli eventi pubblicati nei topic ai quali si sono sottoscritti anche a seguito di uno shutdown e relativo startup del server.

L'architettura è quindi la seguente:

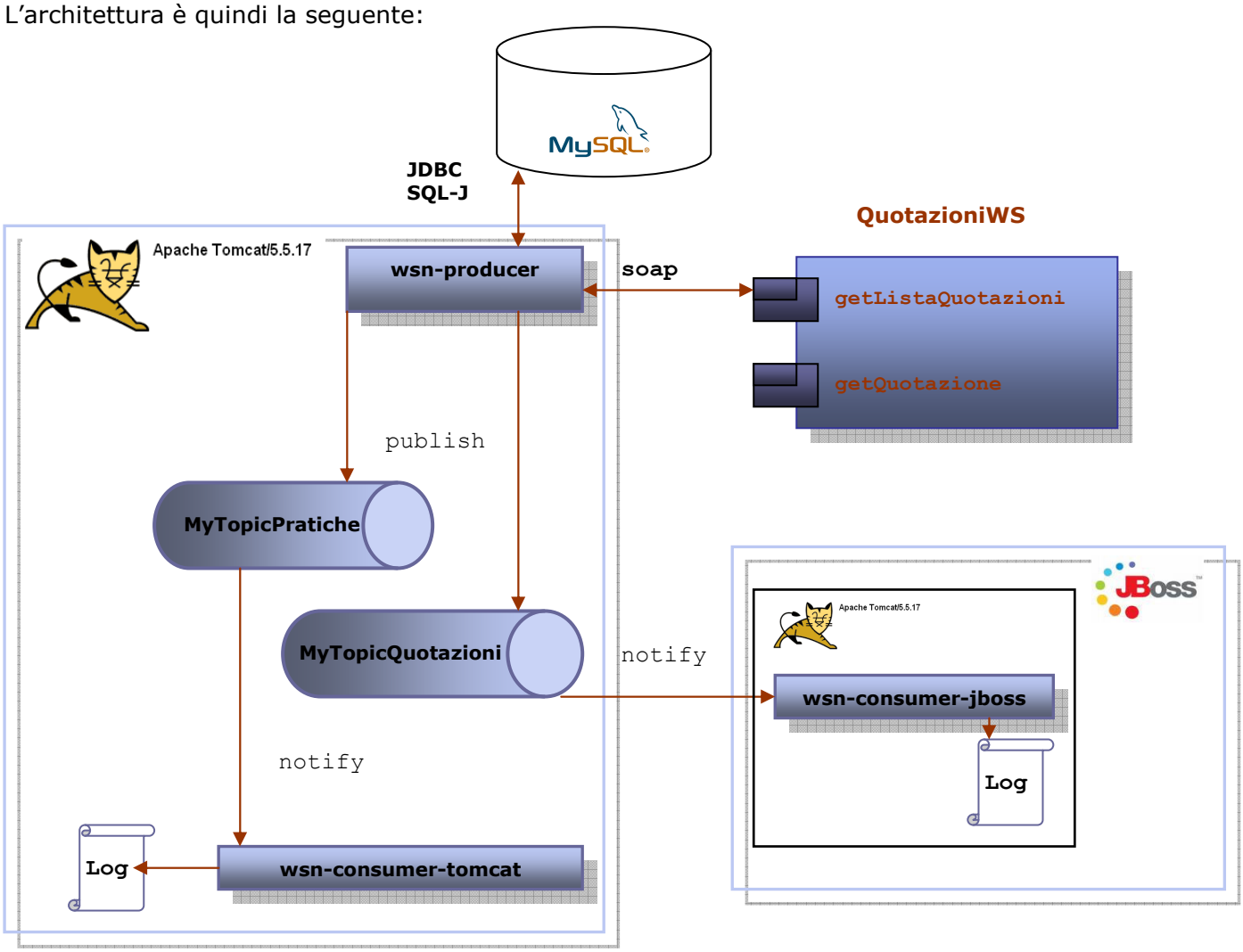


Figura 61:Architettura dello scenario

5.3.13.1 Il Pattern Observer

Una cosa importante da notare è che il modello Publish/Subscribe corrisponde all'implementazione del pattern Observer:i sottoscrittori si registrano presso un pubblicatore e quest'ultimo li informa ogni volta che ci sono nuove notizie. La figura è una schematizzazione del pattern dal punto di vista operativo.

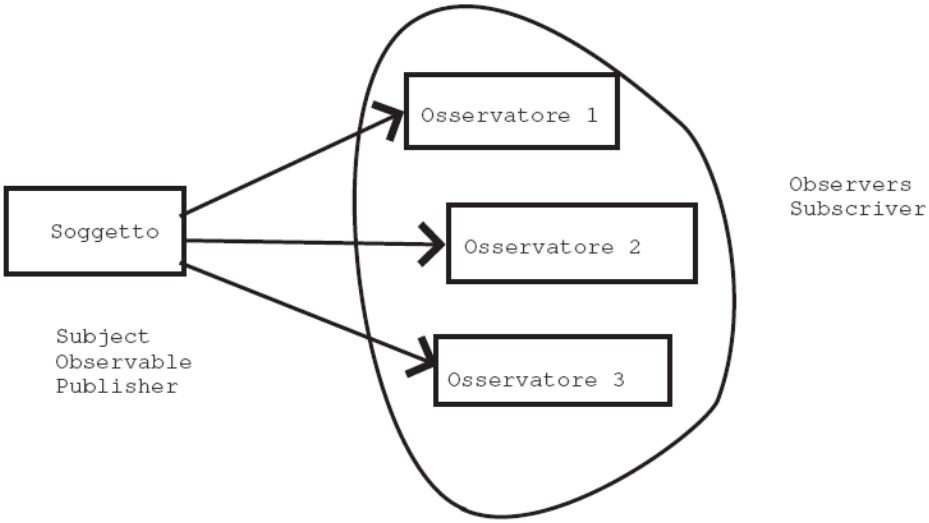


Figura 62:Il pattern Observer in azione

L'Observer Pattern è infatti utilizzato per osservare lo stato di un oggetto da parte di diversi "osservatori" (o simmetricamente, per distribuire un evento a diversi destinatari). L'intento del pattern è quello di definire una dipendenza uno-a-molti tale che quando un oggetto cambia stato tutti quelli che ne dipendono vengono automaticamente notificati del fatto ed aggiornati di conseguenza. L'oggetto osservato è chiamato *Subject*(soggetto) mentre gli oggetti osservatori sono noti come *Observer.*
In risposta alla notifica, ogni osservatore richiederà al soggetto le informazioni necessarie per sincronizzare il proprio stato con il nuovo stato del soggetto.
L'idea alla base del pattern è che gli *Observer* offrono un metodo invocabile dal *Subject* per la notifica, e richiedono ad esso la loro registrazione, mentre il *Subject* mantiene una lista di tutti gli *Observer*. Nel momento in cui si verifica una condizione che implichi un'alterazione dello stato corrente,il *Subject* richiama su ogni elemento della lista di *Observer* il loro metodo preposto a ricevere questo tipo di avvisi.

- il *Subject* non sa come reagiranno gli *Observer* alle notifiche, ma sa che tutti hanno un metodo apposto per riceverle;
- gli *Observer* sanno del *Subject* che esso tiene nota del fatto di avvisarli quando necessario, ma non si interessano del *come.*

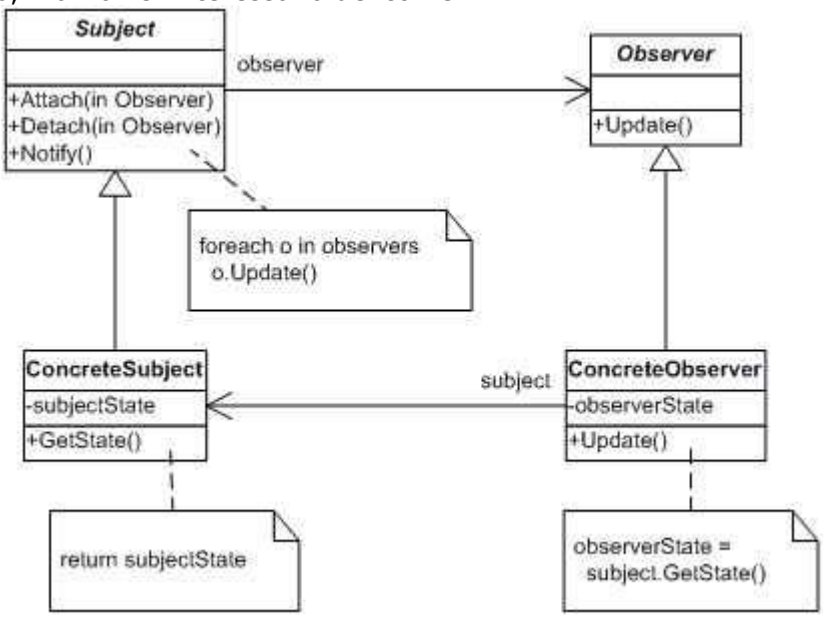


Figura 63:Diagramma UML del pattern

Subject
Una classe che fornisce interfacce per registrare o rimuovere gli observer e che implementa le seguenti funzioni:

- `Attach`
- `Detach`
- `Notify`

Soggetto Concreto
Classe che fornisce lo stato dell'oggetto agli observer e che si occupa di effettuare le notifiche chiamando la funzione notify nella classe padre (Soggetto).
Contiene la funzione: `GetState` - che restituisce lo stato del soggetto.

Observer
Questa classe definisce un'interfaccia per tutti gli observers, per ricevere le notifiche dal soggetto. È utilizzata come classe astratta per implementare i veri Observer, ossia i ConcreteObserver. Coniene la funzione *Update* - una funzione astratta che deve essere implementata dagli observer.

ConcreteObserver
Questa classe mantiene un riferimento al Subject (Concreto), per ricevere lo stato quando avviene una notifica. La funzione non astratta è Update, quando questa viene chiamata dal

Soggetto, il ConcreteObserver chiama la getstate sul soggetto per aggiornare l'informazione su di esso.
Nel paradigma Publish/Subscribe,il soggetto *pubblica* le modifiche al proprio stato e la pubblicazione agli osservatori è fatta chiamando il metodo *notify* che internamente chiama il metodo *update* di tutti gli Observer registrati;gli osservatori *si sottoscrivono* per ottenere gli aggiornamenti e la sottoscrizione di un Observer(Consumer) x è realizzata da x invocando sul subject il metodo *add(x).* In accordo con il pattern un producer avrà una lista contenente le sottoscrizioni ricevute dai vari consumer e nel momento in cui viene generato un evento lo pubblica nel relativo topic;di conseguenza i consumer verranno notificati del fatto che nel topic al quale si sono sottoscritti è stata generato un nuovo tipo d'informazione. I consumer si comportano quindi come gli observers ai quali viene notificato il cambio di stato del subject,il producer che pubblica eventi nei topic.

5.3.12.2 wsn-producer

Dopo aver illustrato l'architettura dello scenario e la relazione tra il paradigma Publish/Subscrive,il Ws-Notification e il Pattern Observer,andiamo ora analizzare in dettaglio l'implementazione del producer. Per iniziare partiamo dalla struttura della directory e dalle cartelle e i file contenuti in essa. Come detto precedentemente per sviluppare lo scenario sono partito dai progetti template messi a disposizione da Muse quindi,sia per il producer che per i due consumer,ho utilizzato la stessa directory contenenti gli stessi file,cartelle,file WSDL ed XML Schema. L'unico file modificato è quindi il solo MyCapabilityImpl che corrisponde all'implementazione vera e propria del producer mentre ho lasciato inalterato il file Capability. Una soluzione alternativa sarebbe stata quella di generare tutto il necessario utilizzando il tool wsdl2java e scrivere a mano il completo WSDL,oppure partire dal sample WSDL messo a disposizione sempre da Apache Muse.

Il file muse.xml è il seguente:

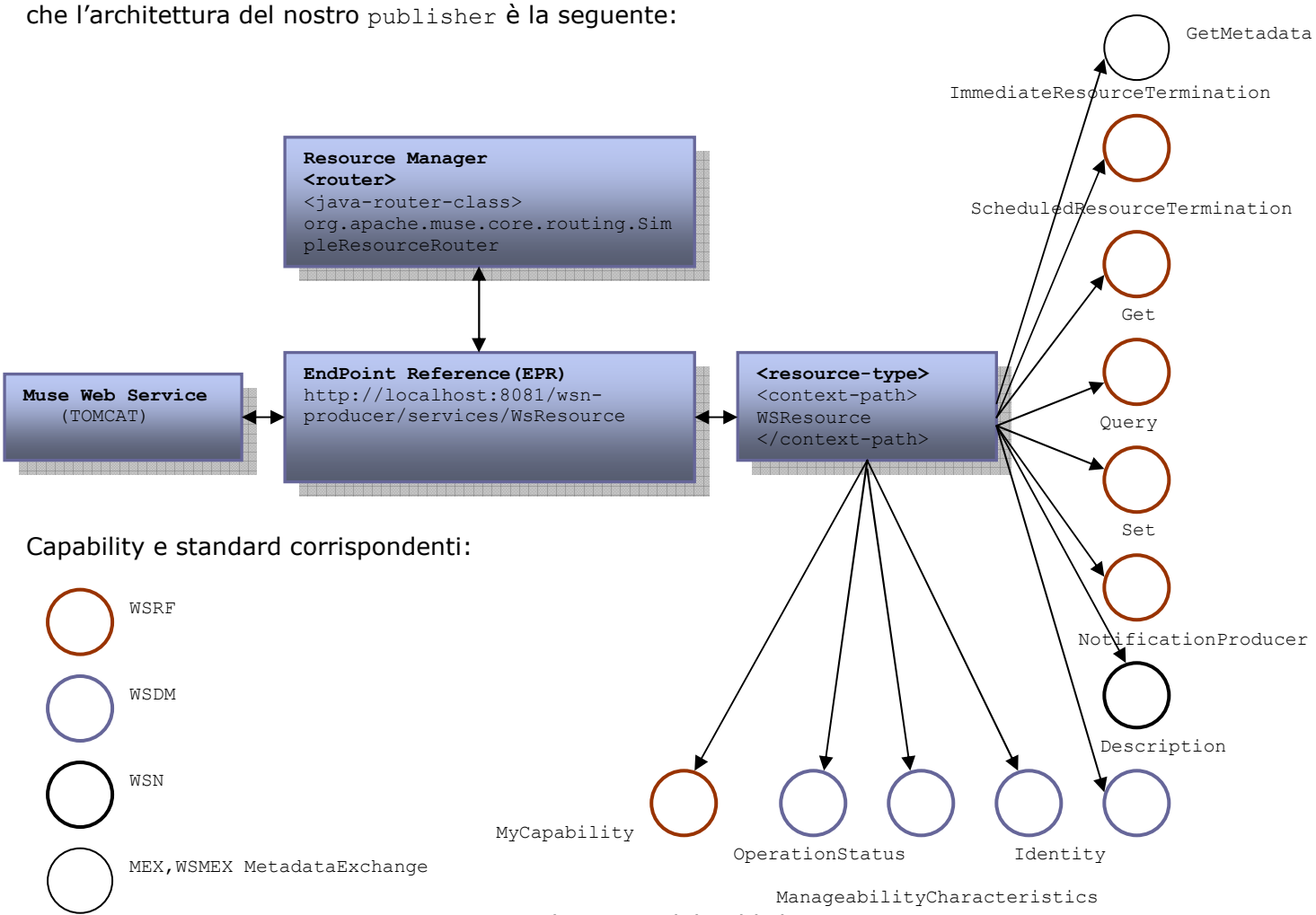
```
<?xml version="1.0" encoding="UTF-8" ?>
<muse xmlns="http://ws.apache.org/muse/descriptor"
      xmlns:wsntw="http://docs.oasis-open.org/wsn/bw-2"
      xmlns:test="http://ws.apache.org/muse/test/wsrf"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://ws.apache.org/muse/descriptor muse-
descriptor.xsd">
      <router>
            <java-router-
class>org.apache.muse.core.routing.SimpleResourceRouter</java-router-class>
            <logging>
                  <log-file>log/muse.log</log-file>
                  <log-level>FINE</log-level>
            </logging>
            <persistence>
                  <java-persistence-
class>org.apache.muse.core.routing.RouterFilePersistence</java-persistence-
class>
                  <persistence-location>router-entries</persistence-location>
            </persistence>
      </router>
      <resource-type use-router-persistence="true">
            <context-path>SubscriptionManager</context-path>
            <wsdl>
                  <wsdl-file>wsdl/WS-BaseNotification-1_3.wsdl</wsdl-file>
                  <wsdl-port-type>wsntw:SubscriptionManager</wsdl-port-type>
            </wsdl>
            <java-id-factory-
class>org.apache.muse.core.routing.CounterResourceIdFactory</java-id-factory-
class>
            <java-resource-
class>org.apache.muse.ws.resource.impl.SimpleWsResource</java-resource-class>
            <capability>
```

```
                  <capability-
uri>http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata</capability-uri>
                  <java-capability-
class>org.apache.muse.ws.metadata.impl.SimpleMetadataExchange</java-capability-
class>
            </capability>
            <capability>
                  <capability-uri>http://docs.oasis-open.org/wsrf/rpw-
2/Get</capability-uri>
                  <java-capability-
class>org.apache.muse.ws.resource.properties.get.impl.SimpleGetCapability</java-
capability-class>
            </capability>
            <capability>
                  <capability-uri>http://docs.oasis-open.org/wsn/bw-
2/SubscriptionManager</capability-uri>
                  <java-capability-
class>org.apache.muse.ws.notification.impl.SimpleSubscriptionManager</java-
capability-class>
                  <init-param>
                        <param-name>trace-notifications</param-name>
                        <param-value>true</param-value>
                  </init-param>
            </capability>
            <capability>
                  <capability-uri>http://docs.oasis-open.org/wsrf/rlw-
2/ImmediateResourceTermination</capability-uri>
                  <java-capability-
class>org.apache.muse.ws.resource.lifetime.impl.SimpleImmediateTermination</java
-capability-class>
            </capability>
            <capability>
                  <capability-uri>http://docs.oasis-open.org/wsrf/rlw-
2/ScheduledResourceTermination</capability-uri>
                  <java-capability-
class>org.apache.muse.ws.resource.lifetime.impl.SimpleScheduledTermination</java
-capability-class>
            </capability>
            <init-param>
                  <param-name>validate-wsrp-schema</param-name>
                  <param-value>false</param-value>
            </init-param>
      </resource-type>
      <resource-type use-router-persistence="true">
            <context-path>WsResource</context-path>
            <wsdl>
                  <wsdl-file>wsdl/WsResource.wsdl</wsdl-file>
                  <wsdl-port-type>test:WsResourcePortType</wsdl-port-type>
            </wsdl>
            <java-id-factory-
class>org.apache.muse.core.routing.CounterResourceIdFactory</java-id-factory-
class>
            <java-resource-
class>org.apache.muse.ws.resource.impl.SimpleWsResource</java-resource-class>
            <capability>
                  <capability-
uri>http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata</capability-uri>
                  <java-capability-
class>org.apache.muse.ws.metadata.impl.SimpleMetadataExchange</java-capability-
class>
            </capability>
            <capability>
```

```
                <capability-uri>http://docs.oasis-open.org/wsrf/rlw-
2/ImmediateResourceTermination</capability-uri>
                <java-capability-
class>org.apache.muse.ws.resource.lifetime.impl.SimpleImmediateTermination</java
-capability-class>
            </capability>
            <capability>
                <capability-uri>http://docs.oasis-open.org/wsrf/rlw-
2/ScheduledResourceTermination</capability-uri>
                <java-capability-
class>org.apache.muse.ws.resource.lifetime.impl.SimpleScheduledTermination</java
-capability-class>
            </capability>
            <capability>
                <capability-uri>http://docs.oasis-open.org/wsrf/rpw-
2/Get</capability-uri>
                <java-capability-
class>org.apache.muse.ws.resource.properties.get.impl.SimpleGetCapability</java-
capability-class>
            </capability>
            <capability>
                <capability-uri>http://docs.oasis-open.org/wsrf/rpw-
2/Query</capability-uri>
                <java-capability-
class>org.apache.muse.ws.resource.properties.query.impl.SimpleQueryCapability</j
ava-capability-class>
            </capability>
            <capability>
                <capability-uri>http://docs.oasis-open.org/wsrf/rpw-
2/Set</capability-uri>
                <java-capability-
class>org.apache.muse.ws.resource.properties.set.impl.SimpleSetCapability</java-
capability-class>
            </capability>
            <capability>
                <capability-uri>http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer</capability-uri>
                <java-capability-
class>org.apache.muse.ws.notification.impl.SimpleNotificationProducer</java-
capability-class>
                 <persistence>
                 <java-persistence-
class>org.apache.muse.ws.notification.impl.NotificationProducerFilePersistence</
java-persistence-class>
                 <persistence-location>subscriptions</persistence-location>
                 </persistence>
            </capability>
            <capability>
                <capability-uri>http://docs.oasis-
open.org/wsdm/muws/capabilities/Identity</capability-uri>
                <java-capability-
class>org.apache.muse.ws.dm.muws.impl.SimpleIdentity</java-capability-class>
            </capability>
            <capability>
                <capability-uri>http://docs.oasis-
open.org/wsdm/muws/capabilities/ManageabilityCharacteristics</capability-uri>
                <java-capability-
class>org.apache.muse.ws.dm.muws.impl.SimpleManageabilityCharacteristics</java-
capability-class>
            </capability>
            <capability>
                <capability-uri>http://docs.oasis-
open.org/wsdm/muws/capabilities/Description</capability-uri>
```

```
                    <java-capability-
class>org.apache.muse.ws.dm.muws.impl.SimpleDescription</java-capability-class>
             </capability>
             <capability>
                    <capability-uri>http://docs.oasis-
open.org/wsdm/muws/capabilities/OperationalStatus</capability-uri>
                    <java-capability-
class>org.apache.muse.ws.dm.muws.impl.SimpleOperationalStatus</java-capability-
class>
             </capability>
             <capability>
                    <capability-
uri>http://ws.apache.org/muse/test/wsrf/MyCapability</capability-uri>
                    <java-capability-
class>org.apache.ws.muse.test.wsrf.MyCapabilityImpl</java-capability-class>
             </capability>
      </resource-type>
      <custom-serializer>
                  <java-serializable-
type>org.apache.ws.muse.test.wsrf.Pratica</java-serializable-type>
                  <java-serializer-
class>org.apache.ws.muse.test.wsrf.PraticaSerializer</java-serializer-class>
      </custom-serializer>
</muse>
```

Dall'analisi del file muse.xml descritto precedentemente e relativo al `wsn-producer` si deduce che l'architettura del nostro `publisher` è la seguente:



Capability e standard corrispondenti:

Figura 64:Architettura del Publisher

Analizziamo le classi che compongono il producer. Come detto,la classe MyCapability è lasciata inalterata:

MyCapability.java

```
package org.apache.ws.muse.test.wsrf;
public interface MyCapability
{
    String PREFIX = "tns";
    String NAMESPACE_URI = "http://ws.apache.org/muse/test/wsrf";
    public int getMessageInterval();
    public void setMessageInterval(int param0);
    public String getServerName();
    public void setServerName(String param0);
}
```

Tale classe viene generata dal tool wsdl2java che oltre ai metodi getter e setter ha creato anche la costante NAMESPACE_URI che contiene l'URI che identifica la capability.

MyCapabilityImpl.java

```
package org.apache.ws.muse.test.wsrf;
import javax.xml.namespace.QName;
import org.w3c.dom.Element;
import org.apache.muse.util.xml.XmlUtils;
import org.apache.muse.ws.addressing.soap.SoapFault;
import org.apache.muse.ws.notification.NotificationProducer;
import org.apache.muse.ws.notification.WsnConstants;
import org.apache.muse.ws.resource.impl.AbstractWsResourceCapability;
import java.util.*;
import java.sql.*;
import java.util.*;
import javax.sql.*;
import java.io.*;
import javax.naming.*;
import java.net.*;
import java.rmi.*;
import javax.xml.namespace.*;
import javax.xml.rpc.*;
import quotazioni.*;importo il package quotazioni contenente lo stub per
accedere al Web Services QuotazioniWS
import quotazioni.pws.*;Importo la classe che definisce il complexType Indice
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.ser.*;
import java.util.Collection;

public class MyCapabilityImpl extends AbstractWsResourceCapability implements
MyCapability{
Connection connection;
private boolean connectionFree = true;
private Vector<Pratica> lista;

-----Gestisco la connessione al database per ottenere i dati delle Pratiche-----
public MyCapabilityImpl()
throws Exception
    {
        try
        {
            InitialContext initialContext = new InitialContext ();
            Context envContext = (Context) initialContext.lookup
("java:comp/env");
            DataSource dataSource = (DataSource) envContext.lookup
("jdbc/ardeadb");
```

```
            this.connection = dataSource.getConnection ();
        }
        catch (Exception e)
        {
            throw new Exception ("Couldn't open connection to Public database: "
+ e.getMessage ());
        }
    }

protected synchronized Connection getConnection ()
    {
        while (this.connectionFree == false)
        {
            try
            {
                wait ();
            }
            catch (InterruptedException e)
            {
            }
        }
        this.connectionFree = false;
        notify ();
        return this.connection;
    }

    protected synchronized void releaseConnection ()
    {
        while (this.connectionFree == true)
        {
            try
            {
                wait ();
            }
            catch (InterruptedException e)
            {
            }
        }
        this.connectionFree = true;
        notify ();
    }

public void close ()
    {
        try
        {
            this.connection.close ();
        }
        catch (SQLException e)
        {
            System.out.println (e.getMessage ());
        }
    }
Metodo che ritorna un object di tipo Vector contenente la lista dei complexType
Pratiche estratte dal database.
public java.util.Vector getListaPratiche() {
        try
        {
            this.getConnection ();


PreparedStatement preparedStatement = this.connection.prepareStatement ("SELECT
* FROM pratichetb");
```

```
ResultSet rsltMT = preparedStatement.executeQuery ();
lista=new Vector<Pratica>();
Creo il vettore scorrendo tutto il ResultSet.
         while(rsltMT.next()) {
         String id =("Id: "+rsltMT.getString(1));
         String dataimm =("Data immissione: "+rsltMT.getString(2));
         String datascad = ("Data scadenza: "+rsltMT.getString(3));
         String stato = ("Stato: "+rsltMT.getString(4));
         String ente = ("Ente: "+rsltMT.getString(7));
         String richiedente = ("Richiedente: "+rsltMT.getString(8));
         Pratica pratica=new
Pratica(id,dataimm,datascad,stato,ente,richiedente);
         lista.add(pratica);
         }

preparedStatement.close ();
        }
        catch (SQLException e)
        {
            System.out.println("SQLException : " + e.getMessage());
            System.exit(0);
        }
        this.releaseConnection ();

return lista;
    }
----------------------------------------------------------Fine gestione Database---
---Implementazione effettiva del wsn-producer-----------------------------------
Costruisco il resource'properties set
private static final QName[] _PROPERTIES = new QName[] {
            new QName(NAMESPACE_URI, "MessageInterval", PREFIX),
            new QName(NAMESPACE_URI, "ServerName", PREFIX)
    };
Creo i due topic nei quali pubblicare l'informazione.
Nel topic MyTopicPratiche veranno pubblicati i dati realativi ai complexType
Pratiche.
    private static final QName _TOPIC_NAME_PRATICHE = new QName(NAMESPACE_URI,
"MyTopicPratiche", PREFIX);
Nel topic MyTopicQuotazioni veranno pubblicati i dati realativi ad alcuni indici
di borsa;tali informazioni sono dapprima ottenute ed elaborati richiamando il
metodo getListaQuotazioni() del Web Service QuotazioniWS Pratiche.
    private static final QName _TOPIC_NAME_QUOTAZIONI = new QName(NAMESPACE_URI,
"MyTopicQuotazioni", PREFIX);

public QName[] getPropertyNames(){return _PROPERTIES; }
private int _MessageInterval = 10;tempo d'attesa per l'invio di ogni messaggio
private String _ServerName = "muse-test.apache.org";nome del server del producer
public int getMessageInterval(){return _MessageInterval; }
public void setMessageInterval(int param0){ _MessageInterval = param0;}
public String getServerName(){return _ServerName;}
public void setServerName(String param0) {_ServerName = param0; }
```

Questo metodo viene chiamato dal containing Resource dopo che tutte le altre capabilities hanno terminato l'inizializzazione con il metodo inizialize().
In questo caso la nostra capability è **MyCapability** mentre le altre,dichiarate nel file muse.xml,sono **GetMetadata, ImmediateResourceTermination, ScheduledResourceTermination,Get,Query,Set,NotificationProducer,Identity,ManageabilityCharacteristics,Description** e **OperationalStatus.** Questo consente alla MyCapability di eseguire l'inizializzazione che richiede la presenza di altre capabilities. Questo stato "post-initialization" è necessario in quanto la definizione e la creazione di una capability sono non-deterministic, nel senso che non è possibile fare alcuna assunzione sull'ordine con le quali queste operazioni vengono eseguite.

Nel file di log viene ad esempio stampato:

```
24-ott-2008 0.45.55 org.apache.muse.core.SimpleResource initializeCapabilities
FINE: [ID = 'CapabilityInitialized'] The resource at 'WsResource' has started
initialization of capability
'http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata'.
24-ott-2008 0.45.55 org.apache.muse.core.SimpleResource initializeCapabilities
FINE: [ID = 'CapabilityInitializationComplete'] The resource at 'WsResource' has
completed initialization of capability
'http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata.
```

```
public void initializeCompleted()
        throws SoapFault
    {
        super.initializeCompleted();
```

Accedo alle resource's WSN capability.
La NotificationProducer è l'interfaccia che rappresenta il WS-Notification NotificationProducer port type; è basata sul WS-N 1.3. Fornisce sia il supporto per le NotificationProducer properties (e I relative topic) che le operazioni: GetCurrentMessage e Subscribe. Contiene anche i metodi necessari per configurare i topic attraverso le Muse's Topic API.
L'API più significativa in questo caso è sicuramente il metodo publish();questo metodo,che presenta diverse segnature,consente di inviare messaggi ai subscribers senza doversi preoccupare di come le sottoscrizioni vengono mantenute e neanche delle modalità per determinare quali messaggi deve effettivamente ricevere un determinato sunscriber. Il metodo permette di concentrare l'attenzione sulla generazione del contenuto degli eventi lasciando all'implementazione del WS-Notification come e dove inviare l'informazione nella rete.

```
        final NotificationProducer wsn =
(NotificationProducer)getResource().getCapability(WsnConstants.PRODUCER_URI);
```

Aggiungo i topic al resource'topic set con gli appropriati topic namespace utilizzando il metodo----Topic addTopic(QName topicName)----messo a disposizione dalla classe NotificationProducer

```
wsn.addTopic(_TOPIC_NAME_PRATICHE);
wsn.addTopic(_TOPIC_NAME_QUOTAZIONI);
```

Il thread è situato all'interno del metodo inizializeCompleted() perché in questo modo è possibile accedere al WSN NotificationProducer capability con la sicurezza che questa abbia terminato la fase d'inizializzazione e sia pronta a gestire le richieste.

```
       Thread producer = new Thread() {
            public void run() {

               String message=null;
               Element payload=null;payload per il topic MyTopicPratiche
               Element payloadIndice=null;payload per il topic MyTopicQuotazioni

                int i=1;
                while (true)
                {
                    try
                    {
```

Ottengo il tempo d'attesa prima di pubblicare un messaggio.

```
int currentInterval = getMessageInterval();
```

Stampa nel file di log.

```
getLog().info("Waiting " + currentInterval + " seconds before sending
message...");
```

Attendo il tempo richiesto.

```
Thread.currentThread().sleep(currentInterval * 1000);
```

Costruisco gli elementi radice dei documenti XML relativi ai complexType Pratica ed Indice.

```
QName rootQName= new QName(NAMESPACE_URI, "Pratica", PREFIX);
```

```
QName rootQNameIndice= new QName(NAMESPACE_URI, "Indice", PREFIX);
Costruisco le istanze delle classi PraticaSerializer e IndiceSerializer
responsabili della serializzazione dei complexType in elementi XML.
PraticaSerializer serializer = new PraticaSerializer();
IndiceSerializer serializerIndice = new IndiceSerializer();
Creo l'oggetto Qualified Name MsgRootQName contenente il NAMESPACE_URI,il nome
del complexType:Message e il PREFIX
QName MsgRootQName= new QName(NAMESPACE_URI, "Message", PREFIX);
XmlUtils è una collezione di metodi relativi al parsing e all manipolazione XML.
Si basa su JAXP,anche se alcuni suoi metodi richiedono caratteristiche
aggiuntive che si trovano solo in Apache Xerces.is a collection of utility
methods related to XML parsing and manipulation.Utilizzando il metodo---
static org.w3c.dom.Element createElement(QName qname)---
creo due nuovi elementi vuoti passandogli il QName "Message".
Vengono quindi creati gli elementi XML:
<tns:Message xmlns:tns="http://ws.apache.org/muse/test/wsrf"></tns:Message>
Element root = XmlUtils.createElement(MsgRootQName);
Element rootIndice = XmlUtils.createElement(MsgRootQName);
-----Topic MyTopicPratiche-------------------------------------------------------
Il metodo getListaPratiche ritorna un Vector contenente appunto la lista delle
Pratiche estratte dal database.
Vector lista = (Vector)getListaPratiche();
Per inserire il complexType nel documento XML utilizzo l'oggetto Iterator
associandolo al Vector lista e creandolo con il metodo iterator().
Iterator iter = lista.iterator();
while(iter.hasNext()){
Ricostruisco il complexType per ogni oggetto Pratica nella lista
Pratica pratica = (Pratica)iter.next();
Aggiungo al payload il complexType serializzato utilizzando il metodo toXML
della classe PraticaSerializer
payload = serializer.toXML(pratica,rootQName);
Il payload costruito è il seguente:

<tns:Pratica>
<tns:Id>Id: 123axkdjf453d</tns:Id>
<tns:Data_Immissione>Data immissione: 7 settembre 2008</tns:Data_Immissione>
<tns:Data_Scadenza>Data scadenza: 1 gennaio 2009</tns:Data_Scadenza>
<tns:Stato>Stato: Eseguita</tns:Stato>
<tns:Ente>Ente: Comune di Nettuno</tns:Ente>
<tns:Richiedente>Richiedente: Alessandro Gabrielli</tns:Richiedente>
</tns:Pratica>

Aggiungo il payload alla radice del documento XML
root.appendChild(payload);

<tns:Message xmlns:tns="http://ws.apache.org/muse/test/wsrf">
<tns:Pratica>
<tns:Id>Id: 123axkdjf453d</tns:Id>
<tns:Data_Immissione>Data immissione: 7 settembre 2008</tns:Data_Immissione>
<tns:Data_Scadenza>Data scadenza: 1 gennaio 2009</tns:Data_Scadenza>
<tns:Stato>Stato: Eseguita</tns:Stato>
<tns:Ente>Ente: Comune di Nettuno</tns:Ente>
<tns:Richiedente>Richiedente: Alessandro Gabrielli</tns:Richiedente>
</tns:Pratica>
</tns:Message>
}
Pubblico le Pratiche nel relativo topic utilizzando il metodo---
----void publish(QName topicName, org.w3c.dom.Element content)----
wsn.publish(_TOPIC_NAME_PRATICHE, root);
getLog().info("Pratiche pubblicate correttamente");
-------------------------------------------------------Fine MyTopicPratiche----

-----Topic MyTopicQuotazioni-----------------------------------------------------
```

Questa parte di codice si occupa della gestione degli eventi da pubblicare nel topic MyTopicQuotazioni. La prima cosa fare è ottenere,compliare ed utilizzare lo stub per contattare il Web Service QuotazioniWS;anche in questo caso il metodo richiesto dal producer ed esposto dal servizio ritornerà un vettore contenente una lista di complexType Indice per cui il procedimento di serializzazione è identico a quello utilizzato precedentemente.

```
getLog().info("Contatto il servizio QuotazioniWS per ottenere le quotazioni e
pubblicarli");
            QuotazioniWSService service = new QuotazioniWSServiceLocator();
            QuotazioniWS stub = service.getQuotazioniWS();

            Vector listaIndice=(Vector)stub.getListaQuotazioni();
            Iterator iterIndice = listaIndice.iterator();

            while (iterIndice.hasNext()) {
            Indice indice = (Indice) iterIndice.next();
            payloadIndice = serializerIndice.toXML(indice,rootQNameIndice);
```

```
<tns:Indice>
<tns:Indice>Indice: MIBTEL</tns:Indice>
<tns:Valore>Valore attuale: 19.623</tns:Valore>
<tns:Apertura_Odierna>Apertura Odierna: 19.375</tns:Apertura_Odierna>
<tns:Max_Oggi>Max Oggi: 19.375</tns:Max_Oggi>
<tns:Min_Oggi>Min Oggi: 19.651</tns:Min_Oggi>
<tns:Variazione_Percentuale>Variazione percentuale:
+1</tns:Variazione_Percentuale>
<tns:Dta_Ora_Ultimo_Valore>Data e Ora dell'ultimo valore: 30/09/08 -
12.05.00</tns:Dta_Ora_Ultimo_Valore>
<tns:Chiusura>Chiusura: 19.011</tns:Chiusura>
</tns:Indice>
```

```
            rootIndice.appendChild(payloadIndice);
            }
```

Pubblico l'evento appena descritto e gestito.

```
wsn.publish(_TOPIC_NAME_QUOTAZIONI, rootIndice);
```

```
<tns:Message xmlns:tns="http://ws.apache.org/muse/test/wsrf">
<tns:Indice>
<tns:Indice>Indice: MIBTEL</tns:Indice>
<tns:Valore>Valore attuale: 19.623</tns:Valore>
<tns:Apertura_Odierna>Apertura Odierna: 19.375</tns:Apertura_Odierna>
<tns:Max_Oggi>Max Oggi: 19.375</tns:Max_Oggi>
<tns:Min_Oggi>Min Oggi: 19.651</tns:Min_Oggi>
<tns:Variazione_Percentuale>Variazione percentuale:
+1</tns:Variazione_Percentuale>
<tns:Dta_Ora_Ultimo_Valore>Data e Ora dell'ultimo valore: 30/09/08 -
12.05.00</tns:Dta_Ora_Ultimo_Valore>
<tns:Chiusura>Chiusura: 19.011</tns:Chiusura>
</tns:Indice>
</tns:Message>
```

```
            getLog().info("Quotazioni degli indici pubblicate correttamente");
----------------------------------------------------Fine MyTopicQuotazioni----
```

Nel log file del producer è possibile osservare tra le atre cose le seguenti stampe:

```
24-ott-2008 0.46.05 org.apache.ws.muse.test.wsrf.MyCapabilityImpl$1 run
INFO: Pratiche pubblicate correttamente
24-ott-2008 0.46.05 org.apache.ws.muse.test.wsrf.MyCapabilityImpl$1 run
INFO: Contatto il servizio QuotazioniWS per ottenere le quotazioni e pubblicarli
24-ott-2008 0.46.09 org.apache.ws.muse.test.wsrf.MyCapabilityImpl$1 run
INFO: Quotazioni degli indici pubblicate correttamente
24-ott-2008 0.46.09 org.apache.ws.muse.test.wsrf.MyCapabilityImpl$1 run
INFO: Waiting 10 seconds before sending message...
                    }
```

```
                    catch (Throwable error)
                    {
            getLog().info("Errore: "+error);
            error.printStackTrace();
                    }
                }
            }
        };
        producer.start();
    }
}
```

Quando il NotificationProducer riceve una subscribeRequest da parte di un consumer,tale sottoscrizione dovrebbe in qualche modo essere memorizzata per crare delle sottoscrizioni durevoli;in questo modo a seguito di uno shutdown e conseguente startup del server sul quale è deployato il producer,alla richiesta di una nuova sottoscrizione,il producer ricomincia a pubblicare gli eventi anche verso tutti i consumer che hanno precedentemente effettuato le sottoscrizioni,sottoscrizioni opportunamente gestite e memorizzate. Una cosa importante da sottolineare è che il producer,eseguito lo startup a seguito di uno shutdown server,è in grado di ricominciare la pubblicazione degli eventi verso i consumer le cui sottoscrizioni sono state memorizzate solo a seguito di una nuova richiesta per la Muse-based application implementata;tale richiesta causerà la re-inizializzazione delle capability terminata la quale la resource SubscriptionManager sarà nuovamente inizializzata e il producer potrà continuare la pubblicazione dei messaggi.

Per abilitare il meccanismo della persistenza delle sottoscrizioni bisogna:

- Abilitare la persistenza per il resource type nel file *muse.xml* – Per abilitare la persistenza di un dato resource type's EPR,si deve aggiungere l'attributo *use-router-persistence* all'elemento <resource-type/> e settarlo a *true* (il valore de defaultt è *false*).
  ```
  <resource-type use-router-persistence="true">
  ```

- Descrivere la persistenza nel file *muse.xml* – Nel file *muse.xml*, sotto l'elemento *router*, si deve creare la seguente entry:

```
<persistence>
    <persistence-location>router-entries</persistence-location>
    <java-persistence-
class>org.apache.muse.core.routing.RouterFilePersistence</java-persistence-
class>
<persistence>
```

Il primo valore, *persistence-location*, informa su come fare riferimento al meccanismo della persistenza;contiene un riferimento generico che può presentare differenti semantiche in base al tipo di persistenza utilizzato,ma in questo sistema basato su file,è una semplice directory. I file di router sono quindi memorizzati in */router-entries*. Il secondo valore, *java-persistence-class*, è il nome della classe che implementa le Muse router persistence API e che è in grado di gestire la serializzazione delle router table entries (EPRs). In questo caso si utilizza un Muse's file-based persistence class.
La persistenza è in questo modo abilitata;per creare anche le cartelle *subscriptions* e *router-entries* nella cartella del produer deployato contenenti rispettivamente le sottoscrizioni ricevute e i file *resource-instance-**N**.xml*(dove N è monoticamente crescente) che rappresentano i reference parameters di ogni EPR,bisogna inserire l'elemento <persistence> all'interno della capability NotificationProducer nel file *muse.xml*:

```
<capability>
<capability-uri>
http://docs.oasis-open.org/wsn/bw-2/NotificationProducer
</capability-uri>
<java-capability class>
```

```
org.apache.muse.ws.notification.impl.SimpleNotificationProducer
</java-capability-class>
<persistence>
<java-persistence-class>
org.apache.muse.ws.notification.impl.NotificationProducerFilePersistence
</java-persistence-class>
<persistence-location>subscriptions</persistence-location>
</persistence>
</capability>
```

In questo modo quando il NorificationProducer riceve un messaggio di requestSubscribe verranno creati i seguenti file:

subscription-1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wsnt:Subscribe xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
    <wsnt:SubscriptionReference>
        <wsa:Address
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:Address>
        <wsa:ReferenceParameters
xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <muse-wsa:ResourceId xmlns:muse-
wsa="http://ws.apache.org/muse/addressing">MuseResource-1</muse-wsa:ResourceId>
        </wsa:ReferenceParameters>
    </wsnt:SubscriptionReference>
    <wsnt:ConsumerReference>
        <wsa:Address
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
consumer-tomcat/services/consumer</wsa:Address>
    </wsnt:ConsumerReference>
    <wsnt:ProducerReference>
        <wsa:ReferenceParameters
xmlns:wsa="http://www.w3.org/2005/08/addressing"/>
        <wsa:Address
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/WsResource</wsa:Address>
    </wsnt:ProducerReference>
    <wsnt:Filter>
        <wsnt:TopicExpression
            Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Concrete"
xmlns:tns="http://ws.apache.org/muse/test/wsrf">tns:MyTopicQuotazioni</wsnt:Topi
cExpression>
    </wsnt:Filter>
</wsnt:Subscribe>
```

resource-instance-1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <muse-wsa:ResourceId xmlns:muse-
wsa="http://ws.apache.org/muse/addressing">MuseResource-1</muse-wsa:ResourceId>
</wsa:ReferenceParameters>
```

L'elemento <muse-wsa:ResourceId>MuseResource-1</muse-wsa:ResourceId> rappresenta il nome dell'EPR relativo ad ogni sottoscrizione memorizzata. I nomi univoci associati agli EPR hanno una forma del tipo MuseResource-N,dove N anche in questo caso è monotonamente crescente.

Per come è implementato il consumer,se si effettua più di una sottoscrizione, per ognuna di esse il NotificationProducer creerà un file subscription-x.xml e router-entries-x.xml. Dal punto di vista del consumer ogni sottoscrizione creata è come se fosse l'unica effettuata nel senso che non tiene traccia di quelle precedenti;ad esempio se il consumer effettua due sottoscrizioni,le successive richieste di unsubscibe o view si riferiranno all'ultima subscribe effettuata. Questo significa che se si effettua una unsubscribe a seguito di due subscribe,dal

punto di vista del consumer è come se si fosse eseguita l'unsubscribe di tutte le subscribe effettuate,infatti se si invoca il "metodo" view a seguito di due subscribe successive e di una unsubscribe verrà generato un messaggio SOAP di errore del tipo:

```
<soap:Reason>
                <soap:Text>[ID = 'DestinationUnreachable'] There is no resource
available at the given EPR:

&lt;wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"&gt;
    &lt;wsa:Address&gt;http://localhost:8081/wsn-
producer/services/SubscriptionManager&lt;/wsa:Address&gt;
    &lt;wsa:ReferenceParameters&gt;
        &lt;muse-wsa:ResourceId
            xmlns:muse-wsa="http://ws.apache.org/muse/addressing"
            wsa:IsReferenceParameter="true"
xmlns:wsa="http://www.w3.org/2005/08/addressing"&gt;MuseResource-3&lt;/muse-
wsa:ResourceId&gt;
    &lt;/wsa:ReferenceParameters&gt;
&lt;/wsa:EndpointReference&gt;
```

Quando il consumer effettua una subscribe infatti non viene ritornata quella eventualmente già eseguita;come detto ne viene creata un'altra e si perde ogni riferimento a quella precedente.
Dal punto del producer invece ogni sottoscrizione effettuata da un consumer viene memorizzata;per questo motivo,anche se viene effettuata una unsubscribe il consumer continuerà a ricevere gli eventi al quale si era sottoscritto con una subscribe precedente a quella sulla quale è stata invocate l'unsubscribe. Infatti se un consumer effettua due subscribe ed un unsubscribe,il producer continuerà ad avere in memoria i file subscription-1.xml e router-entries-1.xml che utilizzerà per inviare gli eventi verso l'EPR specificati in essi.

Dopo aver analizzato la classe che implementa il producer vediamo come sono state realizzate le classi PraticaSerializer ed IndiceSerializaer responsabili della serializzazione/de serializzazione degli eventi pubblicati nei topic. L'utilizzo di queste due classi è necessario in quanto il producer non è esposto come la maggior parte dei servizi web tramite un'unica interfaccia WSDL attraverso la quale indicare gli eventuali complexType presenti per permettere agli utenti di ricrearsi le classi necessarie alla deserializzazione. Non è quindi possibile pubblicare semplicemente i complexType nel topic in quanto i consumer non hanno modo di riconoscerli e gestirli ed estraggono dal topic l'informazione così come viene inviata dal producer. E' necessario quindi costruire ed inserire un documento XML all'interno nel body SOAP del payload da pubblicare contenente tutte le informazioni del complexType. I metodi principali sono fromXML e toXML che permettono rispettivamente la deserializzazione e la serializzazione dell'informazione. Le classi PraticaSerializer ed IndiceSerializer presentano gli stessi metodi,cambia naturalmente solo il complexType. La classe IndiceSerializer deve importare il package contenente la classe definisce complexType Indice ottenuta generando lo stub per chiamare il metodo getListaQuotazioni del servizio QuotazioniWs;è in tale classe che sono definiti infatti i metodi get da utilizzare nel metodo toXML per costruire gli elementi XML associati ai campi del tipo complesso.

Apache muse 2.2.0 fornisce l'interfaccia *org.apache.muse.core.serializer* per la serializzazione e la deserializzazione di object in XML e viceversa. Questa interfaccia consiste di tre metodi:
- Object fromXML(Element xml) throws SoapFault;
  Questo metodo prende un DOM element e lo deserializza in un oggetto Java.
- Element toXML(Object obj, QName qname) throws SoapFault;
  Questo meotdo serializza un oggetto Java in un DOM element e tale elemento viene wrappato in un root element con il relativo Qname;
- Class getSerializableType();
Restituisce la classe Java che definisce il complexType.

PraticaSerializer.java

```
package org.apache.ws.muse.test.wsrf;
```

```
import javax.xml.namespace.QName;
import org.apache.muse.core.serializer.Serializer;
import org.apache.muse.util.xml.XmlUtils;
import org.apache.muse.ws.addressing.soap.SoapFault;
import org.w3c.dom.Element;

public class PraticaSerializer implements Serializer
{
```

Creo dei Qualified Name passandogli il Namespace Uri del producer,il nome dell'elemento ed il prefisso.

```
QName idQName= new QName("http://ws.apache.org/muse/test/wsrf", "Id", "tns");
QName statoQName= new QName("http://ws.apache.org/muse/test/wsrf", "Stato",
"tns");
QName dataimmQName= new QName("http://ws.apache.org/muse/test/wsrf",
"Data_Immissione", "tns");
QName datascadQName= new QName("http://ws.apache.org/muse/test/wsrf",
"Data_Scadenza", "tns");
QName enteQName= new QName("http://ws.apache.org/muse/test/wsrf", "Ente",
"tns");
QName richiedenteQName = new QName("http://ws.apache.org/muse/test/wsrf",
"Richiedente", "tns");
```

Qusto metodo è stato implementato per un'eventuale deserializzazione XML in Java object lato producer;in paratica,nell'esempio mostrato non viene però mai utilizzato. Questo metodo prende in ingresso un elemento XML del tipo:

```
<tns:Pratica>
<tns:Id>Id: 123axkdjf453d</tns:Id>
<tns:Data_Immissione>Data immissione: 7 settembre 2008</tns:Data_Immissione>
<tns:Data_Scadenza>Data scadenza: 1 gennaio 2009</tns:Data_Scadenza>
<tns:Stato>Stato: Eseguita</tns:Stato>
<tns:Ente>Ente: Comune di Nettuno</tns:Ente>
<tns:Richiedente>Richiedente: Alessandro Gabrielli</tns:Richiedente>
</tns:Pratica>
```

e restituisce un complexType del tipo:

Pratica(123axkdjf453d,7 settembre 2008,1 gennaio 2009,Eseguita,Comune di Nettuno,Alessandro Gabrielli)

```
      public Object fromXML(Element arg0) throws SoapFault
      {
            String id = XmlUtils.getElementText(arg0, idQName);
            String dataimm = XmlUtils.getElementText(arg0, dataimmQName);
            String datascad = XmlUtils.getElementText(arg0, datascadQName);
            String stato = XmlUtils.getElementText(arg0, statoQName);
            String ente = XmlUtils.getElementText(arg0, enteQName);
            String richiedente = XmlUtils.getElementText(arg0,
richiedenteQName);

            Pratica pratica = new
Pratica(id,dataimm,datascad,stato,ente,richiedente);
            return pratica;
      }

      public Class getSerializableType()
      {
            return Pratica.class;
      }
```

Questo è il metodo utilizzato per costruire il documento XML che rappresenta il cmomplexType da inserire nel payload dell'evento da pubblicare. Prende in ingresso una pratica ed un---QName MsgRootQName= new QName(NAMESPACE_URI, "Message", PREFIX);---ed utilizza i metodi get messi a disposizione dalla classe Pratica(implements serizlizable e definisce i metodi get e set)per ottenere il valore dei campi che compongono il complexType.

```
	public Element toXML(Object arg0, QName arg1) throws SoapFault
	{
		Pratica pratica = (Pratica) arg0;

		Element root = XmlUtils.createElement(arg1);
		XmlUtils.setElement(root, idQName, pratica.getId());
		 XmlUtils.setElement(root, dataimmQName,
pratica.getDataImmissione());
		XmlUtils.setElement(root, datascadQName, pratica.getDataScadenza());
		XmlUtils.setElement(root, statoQName, pratica.getStato());
		XmlUtils.setElement(root, enteQName, pratica.getEnte());
		XmlUtils.setElement(root, richiedenteQName,
pratica.getRichiedente());

		return root;

	}

}
```

Infine per informare Muse della presenza del nuovo serializzatore bisogna aggiungere il seguente frammento di codice XML nel Muse deployment descriptor(muse.xml)

```
<custom-serializer>
  <java-serializable-type>
   org.apache.ws.muse.test.wsrf.Pratica
  </java-serializable-type>
  <java-serializer-class>
   org.apache.ws.muse.test.wsrf.PraticaSerializer
  </java-serializer-class>
</custom-serializer>
```

5.3.12.3 wsn-consumer

Nello scenario proposto sono presenti due consumer:

- wsn-consumer-tomcat
- wsn-consumer-jboss

Nell'implementazione l'unica cosa che cambia è che il primo intende ricevere gli eventi pubblicati nel topic MyTopicPratiche mentre il secondo quelli pubblicati in MyTopicQuotazioni, gestiti comunque dallo stesso producer. Per questo motivo per illustrare la logica di un wsn-consumer è sufficiente prenderne in considerazione uno sapendo che le uniche cose da modificare saranno il topic al quale si sottoscrive e la porta utilizzata dal server sul quale vengono deployati,porta 8080 per JBoss e 8081 per Apache Tomcat.

Per entrambi i consumer,sia quello deployato su jboss che quello deployato su Tomcat,il file muse.xml utilizzato è il seguente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<muse xmlns="http://ws.apache.org/muse/descriptor"
	xmlns:wsntw="http://docs.oasis-open.org/wsn/bw-2"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://ws.apache.org/muse/descriptor muse-
descriptor.xsd">
	<router>
		<java-router-
class>org.apache.muse.core.routing.SimpleResourceRouter</java-router-class>
		<logging>
			<log-file>log/muse.log</log-file>
			<log-level>FINE</log-level>
		</logging>
		<persistence>
```

```
                  <java-persistence-
class>org.apache.muse.core.routing.RouterFilePersistence</java-persistence-
class>
                  <persistence-location>router-entries</persistence-location>
            </persistence>
      </router>
      <resource-type use-router-persistence="true">
            <context-path>consumer</context-path>
            <wsdl>
                  <wsdl-file>wsdl/WS-BaseNotification-1_3.wsdl</wsdl-file>
                  <wsdl-port-type>wsntw:NotificationConsumer</wsdl-port-type>
            </wsdl>
            <java-id-factory-
class>org.apache.muse.core.routing.CounterResourceIdFactory</java-id-factory-
class>
            <java-resource-class>org.apache.muse.core.SimpleResource</java-
resource-class>
            <capability>
                  <capability-
uri>http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata</capability-uri>
                  <java-capability-
class>org.apache.muse.ws.metadata.impl.SimpleMetadataExchange</java-capability-
class>
            </capability>
            <capability>
                  <capability-uri>http://docs.oasis-open.org/wsn/bw-
2/NotificationConsumer</capability-uri>
                  <java-capability-
class>org.apache.muse.ws.notification.impl.SimpleNotificationConsumer</java-
capability-class>
            </capability>
            <capability>
                  <capability-
uri>http://ws.apache.org/muse/test/wsn/consumer</capability-uri>
                   <java-capability-
class>org.apache.muse.test.wsn.impl.ConsumerCapabilityImplJBoss</java-
capability-class>
            </capability>
      </resource-type>

</muse>
```

E l'architettura è riportata di seguito:

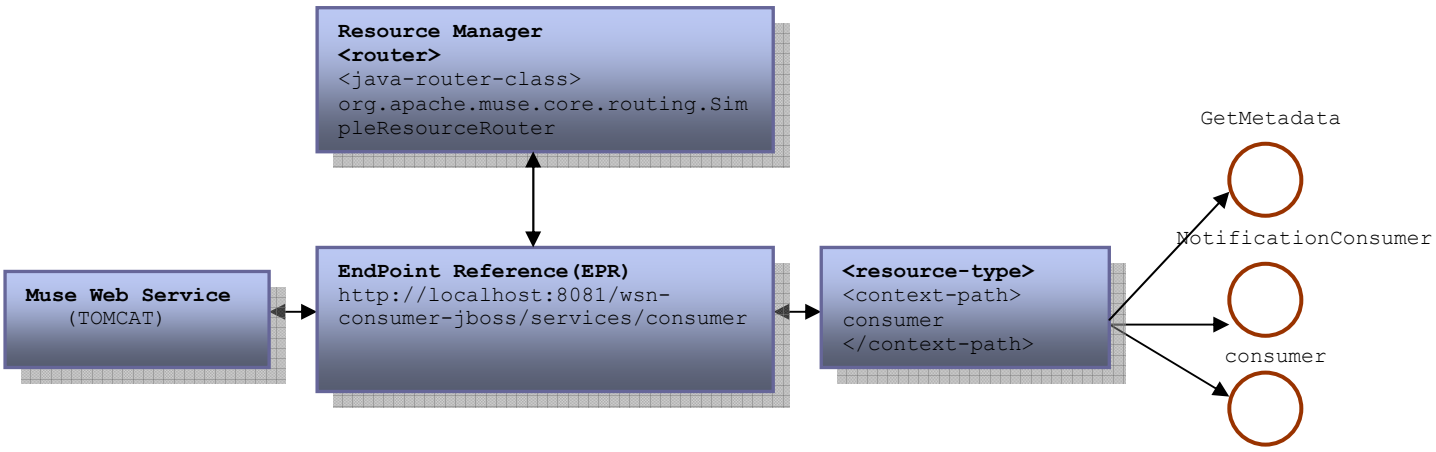


Figura 65:Architettura del Consumer

ConsumerCapability.java

```
package org.apache.muse.test.wsn;
import org.apache.muse.core.Capability;

public interface ConsumerCapability extends Capability
{
    //
    // no additional methods
    //
}
```

ConsumerCapabilityImpl.java

```
package org.apache.muse.test.wsn.impl;

import org.apache.muse.core.AbstractCapability;
import org.apache.muse.test.wsn.ConsumerCapability;
import org.apache.muse.ws.addressing.soap.SoapFault;
import org.apache.muse.ws.notification.NotificationConsumer;
import org.apache.muse.ws.notification.NotificationMessage;
import org.apache.muse.ws.notification.NotificationMessageListener;
import org.apache.muse.ws.notification.WsnConstants;

import org.apache.muse.ws.notification.impl.TopicFilter;
import org.apache.muse.ws.notification.remote.SubscriptionClient;
import org.apache.muse.ws.notification.remote.NotificationProducerClient;
import org.apache.muse.ws.addressing.EndpointReference;
import java.net.URI;
import javax.xml.namespace.QName;
import org.apache.muse.ws.dm.muws.events.WefConstants;
import org.apache.muse.util.xml.XmlUtils;
import org.w3c.dom.Element;
import org.apache.muse.util.xml.*;

import org.apache.muse.ws.dm.muws.*;
import org.apache.muse.ws.resource.*;
import org.apache.muse.ws.resource.impl.*;
import org.apache.muse.ws.addressing.soap.SoapFault;

import java.util.Collection;
import java.util.Iterator;

public class ConsumerCapabilityImplTomcat extends AbstractCapability implements
ConsumerCapability{

private static final QName topicName = new
QName("http://ws.apache.org/muse/test/wsrf", "MyTopicQuotazioni", "tns");
------Inizio del metodo main-----------------------------------------------------
Nel main vengono gestite le sottoscrizioni del consumer al NotificationProducer.
    public static void main(String[] args){
    try{

Costruisco l'oggetto URI che rappresenta l'indirizzo del producer;quest'URI è
composto da tre parti differenti:
- http://localhost:8081   Hostname e porta del container J2EE(Tomcat)
- wsn-producer   Il nome del file WAR con il quale la resource è stata deployata
- services/WsResource      Dove WsResource è il nome del servizio generato
definito nell'elemento---<context-path>WsResource</context-path>---del file
muse.mxl.
URI address = URI.create("http://localhost:8081/wsn-
producer/services/WsResource" );
Costruisco l'EndPointReference con l'inidirizzo appena configurato
EndpointReference producerEPR = new EndpointReference(address);
```

Stesso procedimento per il consumer.

```
address = URI.create("http://localhost:8081/wsn-consumer-
tomcat/services/consumer" );
EndpointReference myEPR = new EndpointReference(address);
```

NotificationProducerClient è un client Web Service per una resources che implementa il WS-N NotificationProducer port type. Fornisce i metodi subscribe() getCurrentMessage() alla base del WS-RF interfaces.

```
NotificationProducerClient producer = new
NotificationProducerClient(producerEPR);
```

SubscriptionClient è un client Web Service per una resources che implementa il WS-N NotificationProducer port type. Fornisce i metodi pauseSubscription() resumeSubscription() alla base del WS-RF interfaces. In questo caso viene utilizzata per invocare i metodi inerenti alla classe org.apache.muse.ws.resource.remote.WsResourceClient getResourcePropertyDocument(),destroy() e getResourceProperty().

```
SubscriptionClient subscription = null;
```

All'esecuzione del consuemer viene richiesto di effettuare una delle seguenti operazioni;

- subscribe Effettuo la sottoscrizione al topic
- unscubscribe Effettuo la unsubscribe al topic
- view Visualizzo la sottoscrizione al topic
- quit

```
while( true ){
System.out.println( "s - subscribe, u - unsubscribe, v -view, q - quit" );
char choice = (char)System.in.read();
if( choice == 's' ){
```

Un Notification Producer è in grado di produrre una sequenza di messaggi di notifica. Un subscriber può registrare l'interesse di un Notification Consumer di ricevere un sottoinsieme di questa sequenza. Un subscriber invia un messaggio di subscribe per registrare tale interesse.
Per effettuare la sottoscrizione utilizzo la funzione-- SubscriptionClient subscribe(EndpointReference consumer,Filter filter,java.util.Date termination) Che ritorna appunto un oggetto della classe SubscriptionClient.Il primo paramento corrisponde all'EndpointReference del consumer,il secondo permette di creare un nuovo oggetto della classe TopicFilter per definire il topic definire un filtro sui topic ai quali è interessato il consumer,mentre il terzo parametro indica.In questo caso viene effettuata la sottoscrizione del consumer al topic MyTopicPratiche.Nel caso in cui come secondo parametro venga passato un null il consumer riceverà gli eventi pubblicati in ogni topic del producer.

```
subscription = producer.subscribe(myEPR,new TopicFilter(topicName), null);
```

Con tale metodo vengono generate due messaggi SOAP:

- subscriptionRequest(consumer)
- subscriptionResponse(producer)

Il messaggio SOAP contenente la sottoscrizione appena effettuata presenta un formato di questo tipo:

- **/wsnt:ConsumerReference:**

Rappresenta il riferimento del Notification Consumer, e tale riferimento è definito dalla specifica WS-Addressing.

- **/wsnt:Filter:**

Con la componente filtro un Subsciber esprime il sottoinsieme di messaggi di notifica che il Notification Consumer può ricevere.

- **/wsnt:Filter /wsnt:TopicExpression/@Dialect:**

E' un attributo richiesto e rappresenta il linguaggio che viene usato per descrivere le TopicExpression.

- **/wsnt:Filter /wsnt:ProducerProperties:**

Questa componente contiene un'espressione filtro valutata sulla ResourceProperties (specifica WS-ResourceProperties)del Notification Producer. Questa espressione deve essere un'espressione boolean.

- **/wsnt:Filter /wsnt:ProducerProperties/@Dialect:**

Rappresenta la dialettica utilizzata.

- **/wsnt:Filter /wsnt:MessageContent:**

Rappresenta un'espressione booleana che definisce il filtro.

- /wsnt:Filter /wsnt:MessageContent/@Dialect:
Rappresenta la dialettica usata.
- /wsnt:InitialTerminationTime:
Questa componente rappresenta la durata della validità della sottoscrizione.
- /wsnt:SubscriptionPolicy:
Questa componente permette di usare policy (requisiti-asserzioni)nella richiesta di sottoscrizione.
- /wsnt:SubscriptionPolicy /wsnt:UseRaw:
La presenza di questo elemento indica che il Notification Producer deve produrre notifiche senza l'utilizzo del Notify wrapper.
- /wsnt:SubscriptionReference:
E' un riferimento WS-Resource della sottoscrizione creata come risultato del messaggio di sottoscrizione.
- /wsnt:CurrentTime:
Questa componente è opzionale ed è presente se il SubscriptionManager usa una terminazione schedulata da WS-ResourceLifeTime. Rappresenta il tempo in cui il messaggio di risposta è stato creato.
- /wsnt:TerminationTime:
Rappresenta il tempo di terminazione della risorsa.

Il messaggio SOAP che ne deriva e che viene stampato ad esempio nel file di log del producer è il seguente:

```
24-ott-2008 14.58.46 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (incoming):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/WsResource</wsa:To>
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsn/bw-2/NotificationProducer/SubscribeRequest</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:c53c4ba0-9e4c-1be1-1c63-
421620f01db3</wsa:MessageID>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
<wsa:Address>http://www.w3.org/2005/08/addressing/role/anonymous</wsa:Address>
        </wsa:From>
    </soap:Header>
    <soap:Body>
        <wsnt:Subscribe xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
            <wsnt:ConsumerReference>
                <wsa:Address
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
consumer-tomcat/services/consumer</wsa:Address>
            </wsnt:ConsumerReference>
            <wsnt:Filter>
                <wsnt:TopicExpression
                    Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Concrete"
xmlns:tns="http://ws.apache.org/muse/test/wsrf">tns:MyTopicQuotazioni</wsnt:Topi
cExpression>
            </wsnt:Filter>
        </wsnt:Subscribe>
    </soap:Body>
</soap:Envelope>
```

Successivamente la resource SubscriptionManager comincia ad inizializzare le capability richieste:

```
24-ott-2008 17.14.30 org.apache.muse.core.SimpleResource initializeCapabilities
FINE: [ID = 'CapabilityInitialized'] The resource at 'SubscriptionManager' has
started initialization of capability……
24-ott-2008 17.14.30 org.apache.muse.core.SimpleResource initializeCapabilities
```

```
FINE: [ID = 'CapabilityInitializationComplete'] The resource at
'SubscriptionManager' has completed initialization of capability……
24-ott-2008 17.14.30 org.apache.muse.core.SimpleResource initialize
INFO: [ID = 'ResourceInitialized'] The resource at 'SubscriptionManager' has
been initialized.
```

Se il messaggio di sottoscrizione ha successo allora il Notification Producer deve produrre un messaggio di response ed inviarlo al subscriber:

```
24-ott-2008 18.10.54 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (outgoing):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addre
ssing/role/anonymous</wsa:To>
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsn/bw-2/NotificationProducer/SubscribeResponse</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:bd23628b-1b63-7b42-61fd-
2d385e68835b</wsa:MessageID>
        <wsa:RelatesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
RelationshipType="wsa:Reply">uuid:1337fd5b-ffc2-7b11-4c2b-
81c27a2a0e2c</wsa:RelatesTo>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:Address>http://localhost:8081/wsn-
producer/services/WsResource</wsa:Address>
        </wsa:From>
    </soap:Header>
    <soap:Body>
<wsnt:SubscribeResponse xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
            <wsnt:SubscriptionReference>
                <wsa:Address
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:Address>
                <wsa:ReferenceParameters
xmlns:wsa="http://www.w3.org/2005/08/addressing">
                    <muse-wsa:ResourceId xmlns:muse-
wsa="http://ws.apache.org/muse/addressing">MuseResource-1</muse-wsa:ResourceId>
                </wsa:ReferenceParameters>
            </wsnt:SubscriptionReference>
            <wsnt:CurrentTime>2008-10-24T14:58:47+02:00</wsnt:CurrentTime>
        </wsnt:SubscribeResponse>
</soap:Body>
</soap:Envelope>
System.out.println("Subscribe effettuata");
System.out.println(XmlUtils.toString(subscription.getResourcePropertyDocument())
);}
```

Con tale metodo vengono generate due messaggi SOAP:

- GetResourcePropertyDocumentRequest(consumer)
- GetResourcePropertyDocumentResponse(producer)

Il messaggio SOAP generato dall'invocazione del metodo GetResourcePropertyDocumentRequest appartenente alla classe SubscriptionClient è il seguente:

```
24-ott-2008 14.58.48 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (incoming):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:To>
```

```
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsrf/rpw-
2/GetResourcePropertyDocument/GetResourcePropertyDocumentRequest</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:b02da209-0ec4-fbdc-db91-
86a4ba3cde47</wsa:MessageID>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">

<wsa:Address>http://www.w3.org/2005/08/addressing/role/anonymous</wsa:Address>
        </wsa:From>
        <muse-wsa:ResourceId wsa:IsReferenceParameter="true"
            xmlns:muse-wsa="http://ws.apache.org/muse/addressing"
xmlns:wsa="http://www.w3.org/2005/08/addressing">MuseResource-1</muse-
wsa:ResourceId>
    </soap:Header>
    <soap:Body>
        <wsrf-rp:GetResourcePropertyDocument xmlns:wsrf-rp="http://docs.oasis-
open.org/wsrf/rp-2"/>
    </soap:Body>
</soap:Envelope>
```

Infine il messaggio SOAP di risposta al metodo GetResourcePropertyDocumentRequest inviato dal Notification Producer al consumer è il seguente:

```
24-ott-2008 14.58.48 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (outgoing):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addre
ssing/role/anonymous</wsa:To>
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsrf/rpw-
2/GetResourcePropertyDocument/GetResourcePropertyDocumentResponse</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:d4e5dba1-503a-f11d-7ef0-
2f7b5f6af8d0</wsa:MessageID>
        <wsa:RelatesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
RelationshipType="wsa:Reply">uuid:b02da209-0ec4-fbdc-db91-
86a4ba3cde47</wsa:RelatesTo>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:Address>http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:Address>
            <wsa:ReferenceParameters>
                <muse-wsa:ResourceId
                    xmlns:muse-wsa="http://ws.apache.org/muse/addressing"
                    wsa:IsReferenceParameter="true"
xmlns:wsa="http://www.w3.org/2005/08/addressing">MuseResource-1</muse-
wsa:ResourceId>
            </wsa:ReferenceParameters>
        </wsa:From>
    </soap:Header>
    <soap:Body>
        <wsrf-rp:GetResourcePropertyDocumentResponse xmlns:wsrf-
rp="http://docs.oasis-open.org/wsrf/rp-2">
            <wsnt:SubscriptionManagerRP xmlns:wsnt="http://docs.oasis-
open.org/wsn/b-2">
                <wsnt:ConsumerReference>
                    <wsa:Address
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
consumer-tomcat/services/consumer</wsa:Address>
```

```
                </wsnt:ConsumerReference>
                <wsrf-rl:CurrentTime xmlns:wsrf-rl="http://docs.oasis-
open.org/wsrf/rl-2">2008-10-24T14:58:48+02:00</wsrf-rl:CurrentTime>
                <wsnt:Filter>
                    <wsnt:TopicExpression
                        Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Concrete"
xmlns:tns="http://ws.apache.org/muse/test/wsrf">tns:MyTopicQuotazioni</wsnt:Topi
cExpression>
                </wsnt:Filter>
                <wsnt:CreationTime>2008-10-24T14:58:46+02:00</wsnt:CreationTime>
                <wsrf-rl:TerminationTime xmlns:wsrf-rl="http://docs.oasis-
open.org/wsrf/rl-2"/>
            </wsnt:SubscriptionManagerRP>
        </wsrf-rp:GetResourcePropertyDocumentResponse>
    </soap:Body>
</soap:Envelope>
```

Parte del contenuto del precedente messaggio SOAP viene stampato sullo standard output del consumer a dimostrazione del fatto che la sottoscrizione è avvenuta con successo:

```
     [java] s - subscribe, u - unsubscribe, v -view, q - quit
s
     [java] Subscribe effettuata
     [java] <?xml version="1.0" encoding="UTF-8"?>
     [java] <wsnt:SubscriptionManagerRP xmlns:wsnt="http://docs.oasis-open.org/w
sn/b-2">
     [java]     <wsnt:ConsumerReference>
     [java]         <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing
">http://localhost:8081/wsn-consumer-tomcat/services/consumer</wsa:Address>
     [java]     </wsnt:ConsumerReference>
     [java]     <wsrf-rl:CurrentTime xmlns:wsrf-rl="http://docs.oasis-open.org/w
srf/rl-2">2008-10-24T11:52:51+02:00</wsrf-rl:CurrentTime>
     [java]     <wsnt:Filter>
     [java]         <wsnt:TopicExpression
     [java]             Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpress
ion/Concrete" xmlns:tns="http://ws.apache.org/muse/test/wsrf">tns:MyTopicQuotazi
oni</wsnt:TopicExpression>
     [java]     </wsnt:Filter>
     [java]     <wsnt:CreationTime>2008-10-24T11:52:50+02:00</wsnt:CreationTime>

     [java]     <wsrf-rl:TerminationTime xmlns:wsrf-rl="http://docs.oasis-open.o
rg/wsrf/rl-2"/>
     [java] </wsnt:SubscriptionManagerRP>

     [java] s - subscribe, u - unsubscribe, v -view, q - quit
```

```
else if( choice == 'u' ){subscription.destroy();
```

Con tale metodo vengono generate due messaggi SOAP:

- DestroyRequest(consumer)
- DestroyResponse(producer)

Con l'invocazione del metodo destroy() il consumer informa il producer che non è più interessato a ricevere gli eventi da lui pubblicati;il messaggio SOAP generato e stampato nel file di log del producer è il seguente:

```
25-ott-2008 14.59.52 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (incoming):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:To>
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsrf/rlw-2/ImmediateResourceTermination/DestroyRequest</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:63c1f1fe-3d01-d11c-884b-
c364f6a960de</wsa:MessageID>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">

<wsa:Address>http://www.w3.org/2005/08/addressing/role/anonymous</wsa:Address>
```

```
        </wsa:From>
        <muse-wsa:ResourceId wsa:IsReferenceParameter="true"
            xmlns:muse-wsa="http://ws.apache.org/muse/addressing"
xmlns:wsa="http://www.w3.org/2005/08/addressing">MuseResource-1</muse-
wsa:ResourceId>
    </soap:Header>
    <soap:Body>
        <wsrf-rl:Destroy xmlns:wsrf-rl="http://docs.oasis-open.org/wsrf/rl-2"/>
    </soap:Body>
</soap:Envelope>
```

Ricevuto tale messaggio SOAP il NotificationProducer inizia lo shutdown delle capabilities:

```
25-ott-2008 14.59.52 org.apache.muse.core.SimpleResource shutdownCapabilities
FINE: [ID = 'CapabilityPreparedForShutdown'] The resource at
'SubscriptionManager' has started the shutdown process on capability
'http://docs.oasis-open.org/wsrf/rlw-2/ScheduledResourceTermination'.
25-ott-2008 14.59.52 org.apache.muse.core.SimpleResource shutdownCapabilities
FINE: [ID = 'CapabilityShutdown'] The resource at 'SubscriptionManager' has
completed the shutdown process for capability
'http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata.
……
25-ott-2008 14.59.52 org.apache.muse.core.SimpleResource shutdown
INFO: [ID = 'ResourceDestroyed'] The resource at 'SubscriptionManager' has been
destroyed.
```

Terminato il quale invia al consumer il seguente messaggio SOAP di responseDestroy:

```
25-ott-2008 14.59.52 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (outgoing):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addre
ssing/role/anonymous</wsa:To>
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsrf/rlw-2/ImmediateResourceTermination/DestroyResponse</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:36fa073f-99a5-0f42-2e0e-
62d87af2298c</wsa:MessageID>
        <wsa:RelatesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
RelationshipType="wsa:Reply">uuid:63c1f1fe-3d01-d11c-884b-
c364f6a960de</wsa:RelatesTo>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:Address>http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:Address>
            <wsa:ReferenceParameters>
                <muse-wsa:ResourceId
                    xmlns:muse-wsa="http://ws.apache.org/muse/addressing"
                    wsa:IsReferenceParameter="true"
xmlns:wsa="http://www.w3.org/2005/08/addressing">MuseResource-1</muse-
wsa:ResourceId>
            </wsa:ReferenceParameters>
        </wsa:From>
    </soap:Header>
    <soap:Body>
        <muse-op:DestroyResponse xmlns:muse-op="http://docs.oasis-
open.org/wsrf/rl-2"/>
    </soap:Body>
</soap:Envelope>
```

E i relativi file subscription-1.xml e router-entries-1.xml saranno cancellati dalle relative cartelle subscriptions e router-entries.

```
System.out.println("Unsubscribe effettuata");}
else if( choice =='v'){
```

Con tale metodo vengono generate due messaggi SOAP:

- GetResourcePropertyRequest(consumer)
- GetResourcePropertytResponse(producer)

Il metodo getResourceProperty ritorna nessuna o più istanza di una property.Viene sollevata un eccezione se la property non è definita nel document's schema e tale situazione non equivale a non trovare alcuna istanza della property definita. WsnConstants è una collezione di properties e metodi come FILTER_QNAME da utilizzare quando si programma con le specifiche del WS-Notification.

```
System.out.println(XmlUtils.toString(subscription.getResourceProperty(WsnConstan
ts.FILTER_QNAME)[0]));}
```

Il consumer per ottenere le informazioni riguardanti il topic al quale si è sottoscritto invia quindi al producer un messaggio SOAP di questo tipo:

```
26-ott-2008 15.00.49 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (incoming):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:To>
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:a0e7c31b-4458-f9c3-28d5-
0f0b862120b5</wsa:MessageID>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">

<wsa:Address>http://www.w3.org/2005/08/addressing/role/anonymous</wsa:Address>
        </wsa:From>
        <muse-wsa:ResourceId wsa:IsReferenceParameter="true"
            xmlns:muse-wsa="http://ws.apache.org/muse/addressing"
xmlns:wsa="http://www.w3.org/2005/08/addressing">MuseResource-2</muse-
wsa:ResourceId>
    </soap:Header>
    <soap:Body>
        <wsrf-rp:GetResourceProperty
            xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2" xmlns:wsrf-
rp="http://docs.oasis-open.org/wsrf/rp-2">wsnt:Filter</wsrf-
rp:GetResourceProperty>
    </soap:Body>
</soap:Envelope>
```

E il producer risponderà al consumer con il seguente messaggio SOAP di risposta:

```
26-ott-2008 15.00.49 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (outgoing):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addre
ssing/role/anonymous</wsa:To>
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyResponse</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:d0e49fd0-60da-b85c-08cf-
1e8a4097c8b1</wsa:MessageID>
```

```
        <wsa:RelatesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
RelationshipType="wsa:Reply">uuid:a0e7c31b-4458-f9c3-28d5-
0f0b862120b5</wsa:RelatesTo>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:Address>http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:Address>
            <wsa:ReferenceParameters>
                <muse-wsa:ResourceId
                    xmlns:muse-wsa="http://ws.apache.org/muse/addressing"
                    wsa:IsReferenceParameter="true"
xmlns:wsa="http://www.w3.org/2005/08/addressing">MuseResource-2</muse-
wsa:ResourceId>
            </wsa:ReferenceParameters>
        </wsa:From>
    </soap:Header>
    <soap:Body>
        <wsrf-rp:GetResourcePropertyResponse xmlns:wsrf-rp="http://docs.oasis-
open.org/wsrf/rp-2">
            <wsnt:Filter xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
                <wsnt:TopicExpression
                    Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Concrete"
xmlns:tns="http://ws.apache.org/muse/test/wsrf">tns:MyTopicQuotazioni</wsnt:Topi
cExpression>
            </wsnt:Filter>
        </wsrf-rp:GetResourcePropertyResponse>
    </soap:Body>
</soap:Envelope>
```

```
 <wsnt:Filter>
 <wsnt:TopicExpression
     Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete"
xmlns:tns="http://ws.apache.org/muse/test/wsrf">tns:MyTopicQuotazioni
</wsnt:TopicExpression>
 </wsnt:Filter>
                <wsnt:CreationTime>2008-10-24T14:58:46+02:00</wsnt:CreationTime>
                <wsrf-rl:TerminationTime xmlns:wsrf-rl="http://docs.oasis-
open.org/wsrf/rl-2"/>
            </wsnt:SubscriptionManagerRP>
        </wsrf-rp:GetResourcePropertyDocumentResponse>
    </soap:Body>
</soap:Envelope>
```

Il consumer dopo aver ricevuto il precedente messaggio stamperà sullo standard output ad esempio solo una parte del contenuto:

```
     [java] <?xml version="1.0" encoding="UTF-8"?>
     [java] <wsnt:Filter xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
     [java]     <wsnt:TopicExpression
     [java]         Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/
Concrete" xmlns:tns="http://ws.apache.org/muse/test/wsrf">tns:MyTopicQuotazioni<
/wsnt:TopicExpression>
     [java] </wsnt:Filter>
```

```
else if( choice == 'q' ){break;}
                    }
         }catch (Throwable error){
            error.printStackTrace();
        }
}
```

```
    private class MyListener implements NotificationMessageListener {
    private QName _myTopicName = null;
    private java.util.logging.Logger _log = null;

    public MyListener(QName topicName,java.util.logging.Logger log) {
      _myTopicName = topicName;
      _log = log;
    }
```

Accetto solamente i messaggi provenienti dal topic al quale mi sono sottoscritto

```
    public boolean accepts(NotificationMessage message){
        QName topicName = message.getTopic();
        return _myTopicName.equals(topicName);
    }
```

Il Notification Producer, per potere informare il relativo Consumer sul cambiamento di una situazione e di permettergli la modifica della propria sottoscrizione, deve includere nelle notifiche che produce i seguenti elementi metadata:

- wsnt:SubscriptionReference

Rappresenta il riferimento della sottoscrizione del Consumer a cui viene inviato il relativo messaggio di notifica.

-wsnt:Topic

Un TopicExpression descrive esattamente un topic. Tale topic rappresenta il motivo per cui il messaggio di notifica è stato prodotto dal Producer per inviarlo al relativo Consumer.

- wsnt:Topic/@Dialect

Rappresenta la dialettica utilizzata per le TopicExpression.

- wsnt:ProducerReference

Rappresenta il riferimento del Notification Producer che ha prodotto il messaggio di notifica.Il messaggio stampato nel file di log del consumer che rappresenta il NotificationMessage è quindi simile al seguente:

FINE: [SERVER TRACE] SOAP envelope contents (incoming):

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
<soap:Header>
<wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
consumer-tomcat/services/consumer</wsa:To>
<wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsn/bw-2/NotificationConsumer/NotifyRequest</wsa:Action>
<wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:bb726018-
6161-3b8c-3cd9-b815872a5239</wsa:MessageID>
<wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
<wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing"/>
<wsa:Address>http://localhost:8081/wsn-
producer/services/WsResource</wsa:Address>
</wsa:From>
</soap:Header>
<soap:Body>
        <wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
            <wsnt:NotificationMessage
                xmlns:muse-wsa="http://ws.apache.org/muse/addressing"
                 xmlns:tns="http://ws.apache.org/muse/test/wsrf"
                 xmlns:wsa="http://www.w3.org/2005/08/addressing"
                 xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
                <wsnt:SubscriptionReference>
                    <wsa:Address
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/SubscriptionManager</wsa:Address>
```

SubscriptionManager è un'interfaccia che rappresenta il WS-Notification SubscriptionManager port type;è basata sul WS-N v1.3. Le resources che utilizzano questo tipo di capability rappresentano rappresentano un accordo tra la resource e i consumer che sono in attesa di ricevere le relative

notifications. Se si distrugge una subscription resource termina questa relazione e il consumer non riceverà più alcun messaggio dal producer resource.

```
<wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing">
<muse-wsa:ResourceId xmlns:muse-
wsa="http://ws.apache.org/muse/addressing">MuseResource-2</muse-wsa:ResourceId>
</wsa:ReferenceParameters>
</wsnt:SubscriptionReference>
<wsnt:Topic
Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete"
xmlns:tns="http://ws.apache.org/muse/test/wsrf">tns:MyTopicQuotazioni</wsnt:Topi
c>
<wsnt:ProducerReference>
<wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing"/>
<wsa:Address
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/WsResource</wsa:Address>
</wsnt:ProducerReference>
<wsnt:Message>
 <tns:Message xmlns:tns="http://ws.apache.org/muse/test/wsrf">
 <tns:Indice>
```

Una volta ricevuto il messaggio SOAP NotifyRequest pubblicato dal producer nel topic il consumer deve rispondere al NotificationProducer con un messaggio NotifyResponse per informare il producer che l'evento è stato ricevuto correttamente. Il messaggio SOAP generato è il seguente:

```
25-ott-2008 16.02.18 org.apache.muse.util.LoggingUtils logMessage
FINE: [SERVER TRACE] SOAP envelope contents (outgoing):

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://localhost:8081/wsn-
producer/services/WsResource</wsa:To>
        <wsa:Action
xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/wsn/bw-2/NotificationConsumer/NotifyResponse</wsa:Action>
        <wsa:MessageID
xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:1593460f-d048-3529-30ff-
f022adc6daaf</wsa:MessageID>
        <wsa:RelatesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
RelationshipType="wsa:Reply">uuid:d530b5e2-0377-58d0-d5a3-
2d71191af355</wsa:RelatesTo>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:Address>http://localhost:8081/wsn-consumer-
tomcat/services/consumer</wsa:Address>
        </wsa:From>
    </soap:Header>
    <soap:Body>
        <wsnt:NotifyResponse xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"/>
    </soap:Body>
</soap:Envelope>
```

Se gli eventi pubblicati dal NotificationProducer non raggiungono per un qualche motivo i consumer che hanno precedentemente effettuato una subscribe,viene generato un messaggio di errore del tipo:

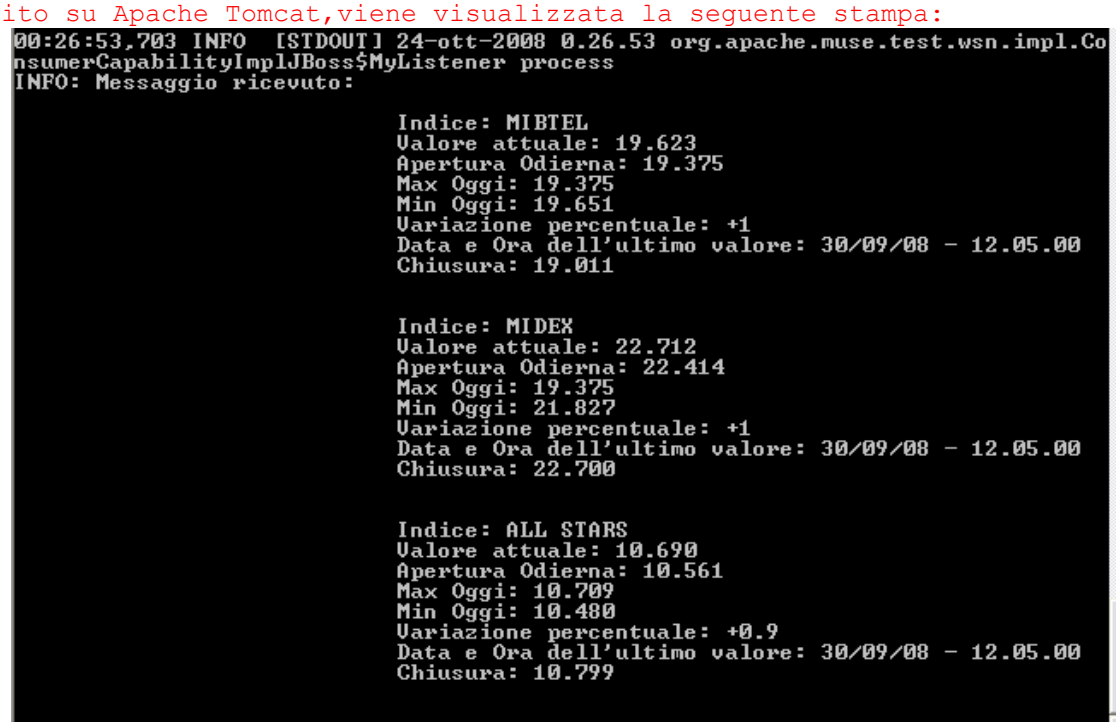
```
25-ott-2008 16.09.57
org.apache.muse.ws.notification.impl.SimpleSubscriptionManager publish
INFO: [ID = 'LastPublishFailed'] The last notification published via wsnt:Notify
failed to reach its destination. The consumer may be unavailable. The original
error was: The element type "HR" must be terminated by the matching end-tag
"</HR>".
```

Questo situazione si verifica ad esempio se il server sul quale è deployato un consumer si guasta o non è temporaneamente raggiungibile. Tale comportamento deriva dal fatto che anche il WS-Notification con Apache Muse implementa il

pattern Observer come spiegato precedentemente.In questo caso infatti il wsn-producer corrisponde all'Observable,i due wsn-consumer-Tomcat/JBoss agli Observer e la pubblicazione degli eventi nei topic al cambio di stato nel oggetto osservato,ossia il producer.La pubblicazione degli eventi in un determinato topic è quindi equivalente alla situazione in cui un Observable invoca il metodo notifyObservers() che invoca il metodo update() di ogni osservatore registrato(con i file subscriptio-i.xml e router-entries-i.xml)per notificare il fatto che ha cambiato stato.

```
    public void process(NotificationMessage message){
        //getLog().info("Received message:\n\n" + message);
        Collection c = message.getMessageContentNames();
        Iterator i = c.iterator();
        while(i.hasNext()){
        Element element = message.getMessageContent((QName)i.next());
        String text = element.getTextContent();
        getLog().info("Messaggio ricevuto: "+text);
           }
```

Sullo standard output di Jboss o nel file di log nel caso in cui il deploy viene eseguito su Apache Tomcat,viene visualizzata la seguente stampa:




```
    }

}
        public void initializeCompleted()throws SoapFault{
        super.initializeCompleted();
        MyListener listener = new MyListener(topicName,this.getLog());
```

NotificationConsumer è un'interfaccia che rappresenta il WS-Notification NotificationConsumer port type ed è basata sul WS-N 1.3.
Questa interfaccia include le observer-style API to the port type operation (wsnt:Notify) in modo che i comsumer possono rispondere ai messaggi che ricevono senza preoccuparsi riscrivere l'implementazione del NotificationConusmer. so that consumers can respond to incoming messages without having to rewrite the NotificationConsumer implementation code.
Nel seguente frammento di codice:

- Creo un oggetto NotificationConsumer;
- utilizzo il metodo---Resource getResource()-------- inherited dall'interfaccia interface org.apache.muse.core.Capability per ottenere la resource che contiene l'istanza della capability;

- utilizzo il metodo----Capability getCapability(java.lang.String capabilityURI)--- per ottenere la Capability associata al dato URI;
- utilizzo il metodo addMessageListener che permette di aggiungere il listener alla lista di listener che verranno notificati ogni volta che la resource riceve un nuovo messaggio.

```
        NotificationConsumer consumerCap =
(NotificationConsumer)getResource().getCapability(WsnConstants.CONSUMER_URI);
        consumerCap.addMessageListener(listener);}}
```

5.3.12.4 Compilazione,deploy ed esecuzione

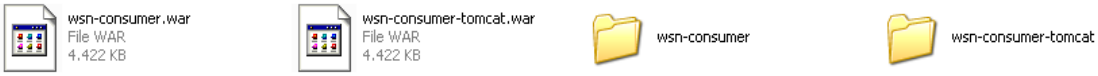
In qust'ultima parte vediamo come effettuare la compilazione e il deploy del wsn-producer e dei wsn-consumer-tomcat e wsn-consumer-jboss implementati.

Per compilare i progetti creati bisogna eseguire i seguenti passi dal prompt dei comandi:
Compilo il producer con il comando ant posizionandomi nella cartella contenente il file build.xml

```
C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-producer\ant
```

```
C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-producer>ant
Buildfile: build.xml

init:

clean:
   [delete] Deleting directory C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-prod
ucer\build

layout:
     [copy] Copying 1 file to C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-produ
cer\build\WEB-INF
     [copy] Copying 24 files to C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-pro
ducer\build\WEB-INF\classes
     [copy] Copying 23 files to C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-pro
ducer\build\WEB-INF\lib

java:
    [javac] Compiling 10 source files to C:\Programmi\muse-2.2.0-bin\samples\j2e
e\wsn-producer\build\WEB-INF\classes
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

war:
      [jar] Building jar: C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-producer\
build\wsn-producer.war

BUILD SUCCESSFUL
Total time: 4 seconds
C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-producer>
```

Compilo i due consumer allo stesso modo lanciando il comando ant dalle cartelle wsn-consumer-tomcat e wsn-consumer-jboss contenenti i relativi file build.xml.
A questo punto sono stai generati i tre file .WAR da utilizzare per il deploy;non rimane che copiare i file wsn-producer.war e wsn-consumer-tomcat.war in C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps ed attendere che Apache Tomcat li riconosca come Web Application creando le relative cartelle wsn-procuder e wsn-consumer-tomcat nella medesima directory.



Infine per deployare wsn-consumer-jboss in JBoss basta copiare il file wsn-consumer-jboss.WAR nella cartella di deploy dell'Application Server.

Per eseguire i consumer implementati e deployati nei rispettivi Server container bisogna eseguire i seguenti comandi:

```
C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-consumer-tomcat\ant run -
Dmain=org.apache.muse.test.wsn.impl.ConsumerCapabilityImplTomcat

C:\Programmi\muse-2.2.0-bin\samples\j2ee\wsn-consumer-JBoss\ant run -
Dmain=org.apache.muse.test.wsn.impl.ConsumerCapabilityImplJBoss
```

Conclusioni

In questa tesina ho approfondito lo studio su problematiche attuali nei sistemi distribuiti come la riusabilità di componenti software e l'interoperabilità tra piattaforme,tecnologie e linguaggi di programmazione eterogenei. A partire dal concetto di servizio e dalla sua rappresentazione atratta con i Transition System ho analizzato il legame tra architetture di tipo SOA e i Web Services,che rappresentano appunto una tipologia particolare di servizi e una possibile tecnologia attraverso la quale realizzare SOA. Ho presentato gli standard che costituiscono i Web Services illustrando gli enormi vantaggi derivanti dal loro utilizzo per poi introdurre ed analizzare nel dettaglio alcune tecnologie come Axis1.4 ed Axis2 per la realizzazione dei servizi stessi. Ho approfondito lo studio sul concetto di servizio stateful,in grado di gestire lo stato della conversazione con i client,e servizi di tipo stateless,confrontandomi con i limiti presenti nelle stesse tecnologie utilizzate e legati alla capacità di supportare l'implementazione di servizi che mantengano lo stato conversazionale. Un altro aspetto approfondito è stato quello della distinzione tra servizi sincroni ed asincroni e dei vari livelli di asincronia che si possono ottenere. Sempre per migliorare la conoscenza sui servizi asincroni mi sono documentato sui vari standard e specifiche che permettono di realizzare questo tipo di comunicazione presentando un framework attraverso il quale ho concretamente implementato la specifica del WS-BaseNotification.

Sviluppi futuri

Nel paragrafo 2.9 ho presentato il servizio OrchestratoreWS;questo servizio è di tipo stateful ed invoca i metodi esposti da altri servizi stateless implementati con tecnoglie differenti per offrire un servizio più complesso e dare ai client la sensazione di interagire con un unico Web Service. Lo scenario proposto mostra come è possibile raggiungere un buon livello di interoperabilità tra servizi eterogenei.
Si potrebbe sostituire il servizio OrchestratoreWS con un processo Bpel utilizzando quindi lo standard BPEL4WS o WS-BPEL che definisce l'uso di Bpel nelle interazioni tra Web Services.
BPEL costituisce infatti il linguaggio standard per la Process Orchestration e rappresenta uno dei componenti fondamentali per realizzare delle Service Oriented Architecture:permette infatti l'integrazione e la cooperazione di diverse componenti,generando così dei servizi web che mantengono le caratteristiche di modularità e scalabilità.
L'importanza del linguaggio BPEL come strumento per l'orchestrazione dei web service si evidenzia principalmente effettuando un parallelo con la specifica da cui deriva, ossia lo standard WSDL. Infatti, la specifica WSDL permette di definire l'interfaccia pubblica dei servizi web,quindi la modalità con cui effettuare operazioni con il servizio stesso; ma, nella definizione di questa interfaccia vi è la completa assenza del concetto di stato, che porta quindi a definire l'interazione con i servizi web come una pura interazione Client-Server, che rende impossibile la creazione di processi di business complessi. Il linguaggio BPEL, invece, sopperisce a questa mancanza fornendo la possibilità di definire variabili che persistono durante tutta l'esecuzione del processo: questa caratteristica è realizzata rendendo anche il processo BPEL un web service, che interagisce con l'esterno tramite un'interfaccia WSDL, che presenta dei tag particolari per la concretizzazione del servizio. Si evidenzia quindi la natura ricorsiva di BPEL: BPEL consente, cioè di modellare le collaborazioni tra servizi esterni o interni, ma il processo da lui gestito viene anch'esso visto dall'esterno come un servizio Web. In questo modo si hanno notevoli vantaggi, prima fra tutte l'estrema modularità con cui possono essere costruiti diversi processi a partire dalle medesime attività atomiche.
Un'altra caratteristica fondamentale è che BPEL è sostanzialmente un'applicazione XML-based. Questa peculiarità gli permette di integrarsi e collaborare con altre applicazioni basate su XML; inoltre, risulta necessaria per l'adempimento della principale funzione di questo linguaggio, ossia l'orchestrazione dei web service. Questi ultimi, infatti, come già detto, basano la loro comunicazione sullo standard WSDL, che è anch'esso XML-based, per cui la comunicazione risulta più agevole se fondata su standard comuni ad entrambi i partecipanti.
È importante sottolineare inoltre come BPEL si distanzi ancora dallo standard WSDL nella gestione di interazioni in modalità sincrona e asincrona, cosa che invece i servizi web semplici non rendono possibile o difficile da implementare. In particolare, la modalità asincrona prevede che il chiamante effettui la chiamata senza allocare risorse condivise (e per questo viene anche chiamata shared nothing), e successivamente continui a eseguire il proprio lavoro senza attendere la risposta del chiamato. Una volta che quest'ultimo abbia terminato il proprio processo, può restituire il risultato delle proprie operazioni a BPEL.
Per tutti questi motivi la soluzione migliore anche nel nostro scenario sarebbe quella di sostituire il servizio OrchetratoreWS con un processo Bpel OrchestratoreWS.bpel

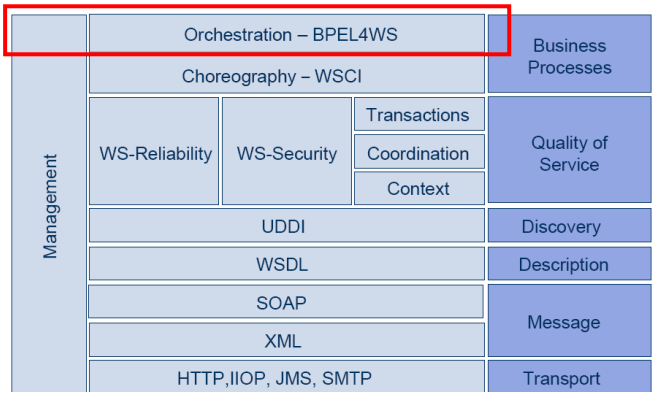


Figura 66:Il ruolo di Bpel nell'architettura dei Web Services

Bibliografia

[1] W3C : Web Services Architecture. On line at: http://www.w3.org/TR/ws-arch/.
[2] MokaByte : Service-Oriented Architecture. On line at: http://www.mokabyte.it/2004/09/soa.htm.
[3] W3C : Web Services Architecture Requirements. On line at: http://www.w3.org/TR/wsa-reqs/.
[4] The Java Web Services Tutorial. Sun Microsystems, 2003.
[5] W3Schools : XML Schema Tutorial. On line at: http://www.w3schools.com/schema/.
[6] MokaByte : Articles. On line at: http://www.mokabyte.it/.
[7] David S. Linthicum. Next Generation Application Integration. Addison-Wesley, 2003.
[8] UDDI Version 2.04 API Specification. On line at: http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm.
[9] Simple Object Access Protocol (SOAP) 1.1, W3C working draft. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
[10] Extensible Markup Language (XML) 1.0, W3C Recommendation 10 Febbraio 1998. http://www.w3.org/TR/1998/REC-xml-19980210
[11] XML Schema , W3C Raccomandation 2 Maggio 2001 Primer: http://www.w3.org/TR/xmlschema-0/
[12] Apache Axis - Web Services Details. On line at: http://www.dsg.cs.tcd.ie/ dowlingj/teaching/ds/lectures/- ws details.pdf.
[13] Greg Barish. Getting started with Web Services Using Apache Axis. On line at: http://www.javaranch.com/newsletter/May2002/newslettermay2002.jsp#axis,Maggio 2002.
[14] AxisWSDD Reference. On line at: http://www.osmoticweb.com/axiswsdd/.
[15] Apache Axis: Sito ufficiale. On line at: http://ws.apache.org/axis/.
[16] Apache Tomcat: Sito ufficiale. On line at: http://jakarta.apache.org/tomcat/.
[17] Apache Axis2: Sito ufficiale. On line at: http://ws.apache.org/axis2/.
[18] Apache Ant: Sito ufficiale. On line at: http://ant.apache.org/
[19] AXIOM: http://today.java.net/pub/a/today/2005/05/10/axiom.html
[20] AXIOM : http://ws.apache.org/commons/axiom/OMTutorial.html
[21] StAX : http://today.java.net/pub/a/today/2006/07/20/introduction-to-stax.html
[22] "Introducing AXIOM: The Axis Object Model" S. W. Eran Chinthaka http://wso2.org/library/26
[23] "Fast and Lighweight Object Model for XML" S. W. Eran Chinthaka http://wso2.org/library/291 http://wso2.org/library/351
[24] "Digging into Axis2: AXIOM". Dennis Sosnosky http://www-128.ibm.com/developerworks/java/library/ws-java2/index.html
[25] Architettura di Axis2 http://ws.apache.org/axis2/1_2/Axis2ArchitectureGuide.html
[26] Thilo Frotscher,Marc Teufel,Dapeng Wang: Java Web Services mit Apache Axis2 2007 entwickler.press http://www.entwickler-press.de http://www.software-support.biz
[27] http://today.java.net/pub/a/today/2006/12/13/invoking-web-services-using-apache-axis2.html
[28] Migrating from Apache Axis1.x to Axis2 http://ws.apache.org/axis2/1_4_1/migration.html
[29] http://www.developer.com/services/article.php/3606466
[30] http://ws.apache.org/axis2/1_0/Axis2ArchitectureGuide.html
[31] http://ws.apache.org/axis2/1_4_1/userguide.html
[32] http://www.w3.org/2002/ws/addr/
[33] http://ws.apache.org/axis2/0_93/adb/adb-howto.html
[34] http://www.w3.org/Submission/ws-addressing/
[35] http://www.ws-i.org/
[36] http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[37] http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/
[38] http://www.jcp.org/en/jsr/detail?id=109
[39] http://ws.apache.org/muse/
[40] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
[41] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
[42] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm
[43] http://ws.apache.org/muse/docs/2.2.0/manual/architecture/programming-model.html
[44] http://www.developer.com/services/article.php/10928_3620661_1
[45] M. Colan, "Service-Oriented Architecture expands the vision of Web services
[46] http://www.stylusstudio.com/api/axis
1_1/org/apache/axis/handlers/SimpleSessionHandler.htm
[48] "WS-BaseNotification", 09/08/2006.
http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-cs-01.pdf
[49] "WS-BrokeredNotification", 09/08/2006.
http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-cs-01.pdf
[50] "WS-Topics", 1/10/2006
http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf
[51] "WS-Messenger (WSMG)" Indiana University.
http://www.extreme.indiana.edu/xgws/messenger/userGuide.html
[52] "Globus Toolkit 4", 2005 http://www.globus.org
[53] http://servicemix.apache.org/home.html
[54] http://home.dei.polimi.it/ghezzi/_PRIVATE/DiBiagio.pdf
[55] http://www.mobilab.unina.it/tesi/Tesi_Martina_Muscariello.pdf
[56] http://www.cs.rug.nl/~aiellom/tesi/zanoni2.pdf
[57] http://dit.unitn.it/~aiellom/tesi/osmic.pdf