UNIVERSITA' DEGLI STUDI DI NAPOLI

"Federico II"

Corso di Laurea in Ingegneria Informatica

DIPARTIMENTO DI INFORMATICA E SISTEMISTICA

TESI DI LAUREA IN SISTEMI INFORMATIVI

"Web Services"

Tecnologie abilitanti e prospettive: un'applicazione per il riutilizzo del software aziendale

RELATORE

Ch.mo Prof.

Lucio Sansone

CANDIDATO

Antonio Valentino

matr. 41/2991

CORRELATORI

Ch.mo Ing. Antonio d'Acierno

Ch.mo Dott. Roberto Capobianco

A.A 2000-2001

A tutti quelli che hanno creduto in me, sostenendomi ed aiutandomi nel corso di questi anni.

SOMMARIO

PREFAZIONE.

Si desidera ringraziare il professore Lucio Sansone dell'Università degli Studi di Napoli docente del corso di Sistemi Informativi, oltre che per le conoscenze infusemi, per la grande disponibilità con cui si pone nei confronti di noi studenti e tesisti.

Per quanto riguarda nello specifico questo lavoro di tesi, esso non sarebbe stato portato a termine senza il contributo fondamentale apportato dall'ingegnere Antonio d'Acierno e il Dottore Roberto Capobianco per l'aiuto fornitomi nella raccolta e nell'analisi del materiale di studio.

Un ultimo ringraziamento lo devo alle persone che mi sono state maggiormente vicine in questi anni, infondendomi quella fiducia in me stesso che tante volte mi è venuta a mancare: i miei genitori, Susy, Patrizia e tutti quelli che non hanno mai smesso di credere in me.

Ringraziamenti anche ai membri della commissione d'esame per l'attenzione prestata all'elaborato.

INTRODUZIONE

In questo lavoro di tesi, sviluppato presso l'Italdata S.p.A, sono state approfondite le tematiche legate alla tecnologia dei Web Services.

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).

Come è noto, XML (eXensible Makup Language) è un metalinguaggio di markup completamente estensibile ed è una raccomandazione del gruppo W3C dal febbraio del 1998. SOAP (Simple Object Access Protocol) definisce un modello per realizzare, mediante l'uso di semplici messaggi di richiesta e risposta, un protocollo di base (ad esempio simile all'RPC) per lo scambio di informazioni in formato XML in ambiente distribuito, tipicamente utilizzando HTTP a livello di trasporto. WSDL (Web Service Description Language) è un particolare linguaggio XML che consente di definire le interfacce dei servizi Web. Infine, UDDI (Universal Description, Discovery and Integration) definisce strutture dati e API per realizzare un registro in

cui i fornitori possono pubblicare le informazione sul proprio conto e sui propri servizi, al fine di facilitarne la scoperta da parte di un potenziale consumatore.

Va notato come l'insieme di queste tecnologie di base lasci scoperte alcune funzionalità fondamentali (sicurezza, routing ed affidabilità dei messaggi, gestione delle transazioni distribuite etc.) che gli sviluppatori implementano in modo proprietario e spesso in maniera non interoperabile. D'altro canto queste caratteristiche avanzate sono fondamentali per realizzare servizi forniti all'esterno, a cui Internet può fornire un valore aggiunto. In questo lavoro saranno approfonditi le sole baseline specifications dell'architettura basata sui servizi Web.

La prima fase di questo lavoro di tesi è stata un'attenta e scrupolosa scelta del materiale necessario per raccogliere informazioni sui Web Services e sulle tecnologie abilitanti: XML, SOAP, WSDL e UDDI. Successivamente, dopo un'ulteriore ricerca bibliografica sugli aspetti essenziali del riuso del software aziendale, si è approfondito il discorso dello sviluppo basato sui servizi Web interni (all'azienda) ed il loro riutilizzo nelle applicazioni. A tale proposito, è stato configurato un ambiente per eseguirli ed è stato sviluppato un prototipo che consente di classificarli, di pubblicarli, di scoprirli e di utilizzarli. Il prototipo realizzato, in particolare, è costituito da vari tool. Il primo, riservato ai consumatori dei servizi, consente di cercarli nel registro UDDI in conformità a vari criteri di ricerca (per nome, per categoria, per unità organizzativa aziendale, etc). Oltre alla consultazione di tutte le informazioni esaustive inerenti al servizio trovato, dal programma client è possibile scaricare sia il documento WSDL che descrive il

servizio (ed a partire dal quale, mediante un ulteriore tool, è possibile generarne il proxy per il collegamento al servizio) sia i proxy disponibili, sia l'implementazione stessa del servizio. Un'ulteriore applicazione, destinata ad un comitato direttivo aziendale e a progettisti dei servizi, consente di gestire gli utenti del registro UDDI e di pubblicare tassonomie e descrizioni dei servizi. Al fine di testare il prototipo è stato sviluppato un semplice Web Service, e prodotte le relative descrizioni.

Capitolo I

PANORAMICA SUI WEB SERVICES

1. Introduzione.

Le tecnologie emergenti negli ultimi cinque anni hanno giocato un ruolo chiave nell'evoluzione del mondo Internet, fornendo nuove soluzioni alla crescente domanda dell'e-business e nella comunicazione business-to-business. Tra queste, Java consente di realizzare codice indipendente dalle piattaforme, XML è la fucina della portabilità dei dati, e infine, il Pervasive Computing estende la connessione a una vasta gamma di dispositivi digitali.

Le applicazioni Web, oggi, devono fronteggiare una serie di problemi che dieci anni fa non erano neppure immaginabili. I sistemi distribuiti su aree enormemente vaste, devono assicurare un alto livello di prestazioni e un funzionamento impeccabile. I dati provenienti da database, da sistemi di directory e da applicazioni assai eterogenei devono poter essere distribuiti senza alcun deterioramento e possedere un formato riconosciuto da tutti i potenziali utilizzatori. Le applicazioni devono essere in grado di comunicare con altre componenti aziendali e con altri sistemi appartenenti a strutture aziendali differenti. Nell'era in cui Internet è diventato uno strumento di commercio, le compagnie investono in linee di comunicazioni più di quanto non facciano sui loro prodotti. Per le grandi aziende e per le

piccole e medie imprese il Web offre nuove opportunità di business, fornendo una gamma più ampia di soluzioni operative e di sviluppo del canale d'erogazione. Ma in questo scenario, dove il cliente può passare da una proposta commerciare ad un'altra senza costi aggiuntivi, la concorrenza è spietata. La necessità di fornire velocemente servizi di qualità costringe le aziende a muoversi in tempi più rapidi, e impone di abbandonare la tradizionale metodologia di sviluppo software, basata su progettazione, implementazione, test ed erogazione. L'uscita annuale dei prodotti è storia passata per parecchie aziende di e-business. La dinamicità insita nel mondo del commercio, richiede un meccanismo standard e flessibile per integrare funzionalità in modo più rapido e semplice, capace di disaccoppiare le capacità business dall'implementazione e dai meccanismi di deployment dei servizi, in maniera tale che qualunque azienda possa diventare una rete di servizi dislocati ovunque nel Web e che possono eventualmente cambiare posizione senza comprometterne il funzionamento. Chiaramente, questo fornisce maggiore agilità e opportunità per le organizzazioni di ripensare radicalmente al modo di implementare le proprie soluzioni business [1]; inoltre, l'adozione di un'infrastruttura comune per l'integrazione rende possibile la riduzione dei costi per stabile e mantenere la collaborazione, oltre il firewall, con i propri partner.

XML, HTTP e altri standard emergenti forniscono ai programmatori i mezzi per soddisfare tutte queste nuove esigenze. Non per niente, lo sforzo comune da parte delle maggiori compagnie produttrice di software del calibro della Microsoft, IBM, Ariba, HP, etc, converge nella direzione della realizzazione di un framework, basato

esclusivamente su tali standard, in cui una qualunque funzionalità software può diventare accessibile, pubblicabile e localizzabile sul Web. Queste funzionalità sono esposte mediante i Web Services che rappresentano una rivoluzione per le capacità e-business, e nello stesso tempo un semplice gradino evolutivo nello sviluppo software, perché possono essere costruiti sul software esistente ed estendono standard esistenti [2].

2. Un po' di storia.

Negli ultimi anni si è assistito all'evoluzione di nuovi standard progettati per migliorare l'integrazione tra le compagnie, i partner e i customer.

Oggi Internet può fare affidamento sulle seguenti tecnologie, oramai consolidate:

- **TCP/IP** (Transmission control protocol/Internet Protocol), è un protocollo universale per la comunicazione in rete, qualunque dispositivo dai cellulari ai laptop ai mainframe ai PC possono comunicare attraverso questo protocollo.
- **HTTP** (HyperText Transfer Protocol), è Protocollo di comunicazione a livello d'applicazione progettato per il trasferimento d'informazioni ipermediali tra sistemi distribuiti e collaborativi.
- **HTML** (HyperText Markup Language), è un linguaggio standard di markup che fornisce un'interfaccia per il contenuto delle

informazioni trasmesse sul Web. Si possono usare tag HTML per pubblicare dati su qualunque dispositivo digitale provvisto di browser HTML.

- **XML** (eXtensible Markup Language), è un metalinguaggio di markup per descrivere dati in maniera universale. I documenti XML rendono semplice il movimento di dati strutturati attraverso il Web.
- **JAVA**, è un linguaggio orientato agli oggetti che consente d'implementare codice portabile, eseguibile su qualunque piattaforma.

TCP/IP e HTTP definiscono il protocollo di comunicazione richiesto per la rete pubblica Internet. Java Technology e XML permettono di realizzare moduli software e rappresentare dati entrambi indipendenti dalle piattaforme hardware e software, e in più, essendo costruiti su UNICODE, sono pienamente internazionalizzabili.

Questi standard permettono di progettare applicazioni software lascamente accoppiate e quindi di ridurre le restrizione imposte dai requisiti di similitudine tra i vari partner, risolvendo i problemi legati all'eterogeneità degli ambienti hardware e software e ai cambiamenti delle implementazione delle applicazioni. E' chiaro che tali tecnologie sono il punto di partenza per la realizzazione di un substrato software per l'integrazione di servizi accessibili ovunque nel Web, ma da sole non sono sufficienti. In generale, un'architettura distribuita basata su componenti necessita di un linguaggio per la definizione delle interfacce dei componenti, un protocollo per la comunicazione tra il modulo utilizzatore e l'endpoint, un catalogo che tiene traccia dei

componenti disponibili e un ambiente di esecuzione dei componenti (CEE, Component Excution Environment) capace di nascondere agli sviluppatori i dettagli tecnologici a basso livello. Ora, pensando ad un componente in chiave di funzionalità offerte attraverso la sua interfaccia, si può estendere l'idea di componenti distribuiti a servizi distribuiti che risiedono ovunque su Internet, e cioè a Web Services. Comunque, realizzare un modello di elaborazione orientato ai servizi accettato da tutte le parti in gioco non è una cosa così facile. Tanto è vero che sono stati proposti e adottati parecchie soluzioni spesso contrastanti e in competizione. In ogni modo, la seguente terna di standard sembra mettere d'accordo i maggiori produttori di software:

- SOAP, un semplice modello per incapsulare messaggi di richiesta e risposta codificati in XML.
- WSDL, linguaggio di markup per la descrizione delle interfaccie dei componenti.
- UDDI, registro in cui è possibile pubblicare e scoprire servizi.

L'ambiente d'esecuzione dei servizi Web è un Web Application Server che supporti il protocollo SOAP.

3. I Web Services.

Oggi, i Web Services sono in via di standardizzazione grazie agli sforzi del World Wide Consortium (W3C) e di alcuni dei maggiori produttori software. L'intento è proporre un paradigma service-oriented per l'elaborazione distribuita su Internet che utilizzi esclusivamente

tecnologie standard per invocare, pubblicare e cercare servizi. Tuttavia, per raggiungere quest'obiettivo bisognerà attendere che alcuni standard emergenti siano migliorati ed approvati come raccomandazioni dal W3C. Nel frattempo, si possono utilizzare alcune proposte di infrastrutture basate sui Web Services per cercare nuove soluzione per l'integrazione business-to-business, e, più in generale, nell'esposizione di servizi e nel loro uso in applicazioni di qualunque genere.

3.1. Definizione.

Un Web Service è un servizio disponibile in rete ad uso di altri programmi, che può essere pubblicato, localizzato e invocato attraverso il Web. A sua volta, un servizio espone una o più funzionalità che devono essere specificate in termini di contratto tra il fornitore e l'utilizzatore [9]. Una funzionalità offerta da un Web Service può andare da una semplice richiesta a complesse elaborazioni.

I Web Services sono lascamente accoppiati e comunicano direttamente con altri Web Services attraverso Internet usando tecnologia basata esclusivamente su standard [3]. Essi, dunque, possono essere utilizzati come building block per realizzare applicazioni in sistemi distribuiti aperti, senza preoccuparci dell'implementazione e della localizzazione dei componenti software che forniscono i servizi.

Diversamente dalle attuali tecnologie basate su componenti, i Web Services non sono accessibili mediante un protocollo specifico di modello ad oggetti, come Distributed Component Object Model (DCOM), Remote Method Invocation (RMI), o Internet Inter-ORB Protocol (IIOP), bensì utilizzano protocolli e formati dati che sono standard per il Web, come HTTP (Hypertext Transfer Protocol) e

XML (eXtensible Markup Language). Inoltre, l'applicazione client che consuma i Web Services può essere implementata in un qualsiasi linguaggio di programmazione capace di creare e interpretare i messaggi definiti per l'interfaccia dei Web Services.

L'uso di questa nuova tecnologia, può consentire alle compagnie di esporre, in modo veloce ed economicamente vantaggioso, le proprie applicazioni correnti e future come Web Services che possono essere scoperti e consumati dai partner esterni, oltre il firewall. In definitiva, i Web Services possono essere utilizzati da customer, fornitori e partner commerciali indipendentemente dalla piattaforma hardware, dal sistema operativo e dall'ambiente di programmazione utilizzato.

Per meglio comprendere quanto detto, faccio un semplice esempio di come, nel prossimo futuro, un'applicazione Web può trarre vantaggio dall'uso di Web Services. Consideriamo un sito Web che vende componenti per PC. Questo desidera inserire nelle sue offerte due nuovi servizi di cui non ha il controllo diretto. Il primo è un servizio di spedizione, fornito da un corriere espresso, e il secondo un servizio d'accredito, fornito da un'istituzione di credito.

Una possibile soluzione consisterebbe nello stabilire relazioni EDI (Electronic Data Interchange) con i fornitori e, conseguentemente, ottenere i moduli da integrare per lo specifico ambiente di sviluppo. Questa soluzione richiede un investimento non indifferente e una stretta collaborazione con l'infrastruttura IT dei partner.

Un'altra soluzione potrebbe essere quella di usare un collegamento ipertestuale che rimandi il visitatore al Web site dei partner. Questa

soluzione, meno complessa e più veloce, è sconsigliabile perché poco flessibile e comporta la perdita di controllo sul cliente.

Adottando l'approccio basato sui Web Services , invece, il sito, che vende componenti per PC al dettaglio, può cercare i due servizi desiderati in un catalogo, contattare i fornitori, provare, eventualmente, i servizi e, infine, se li ritiene soddisfacenti può decidere di integrarli nella propria applicazione Web e usarli attraverso Internet.

3.2. Motivazioni.

I benefici derivanti dall'uso dei Web Services sia per l'interazioni interne (Intranet aziendale) sia per quelle su Internet con i partner commerciali, sono indubbi. Questi includono:

- **Interoperabilità** – Le funzionalità dei componenti di piattaforme di sviluppo eterogenee (.NET, CORBA, J2EE) possono essere velocemente integrate nelle applicazioni business. Tutti i Web Services possono interagire tra loro.
- **Accessibilità** – Le correnti applicazioni, sviluppate in qualsivoglia linguaggio di programmazione, possono essere esposte come Web Services velocemente usando appositi tools, che possono essere inseriti in qualsiasi ambiente di sviluppo. Tutti i dispositivi, che supportano fondamentalmente le tecnologie HTTP e XML, possono accedere ai Web Services e consumarli.
- **Riuso di componenti –** Gli sviluppatori potranno usare un ampio numero di Web Services, forniti da terzi parti, da integrare nelle proprie applicazioni. Un'azienda può decidere di riusare il proprio

patrimonio software esistente, trasformando i propri componenti software in Web Services.

- **Nuovi canali di comunicazione** – Si può far leva sugli investimenti fatti nell'infrastruttura Internet per realizzare nuovi canali di comunicazione per raggiungere i propri clienti.

3.3. Un modello concettuale.

Il modello dei Web Services, da un punto di vista orientato ai servizi, è basato su tre ruoli fondamentali e sulle necessarie interazioni tra ciascun ruolo [2]. In figura 1.1 sono illustrati ruoli e interazioni.

I tre ruoli sono:

- **Service Provider** – E' una piattaforma che fornisce l'accesso al servizio. Per esporre un'applicazione come un servizio, il provider deve fornire un accesso basato su un protocollo standard e una descrizione standard del servizio (meta data).
- **Service requestor o user** – In generale, è un'applicazione che invoca o avvia un'interazione con un servizio. Il service requestor e il service provider devono condividere le stesse modalità di accesso e interazione col servizio.
- **Service registry** – E' un registro presso il quale i service provider possono pubblicare il servizio e i service requestor possono trovare i servizi desiderati. Il service registry deve fornire una tassonomia consistente dei Web Services per facilitarne la scoperta, e una descrizione dettagliata dell'azienda che fornisce il servizio e del servizio stesso.

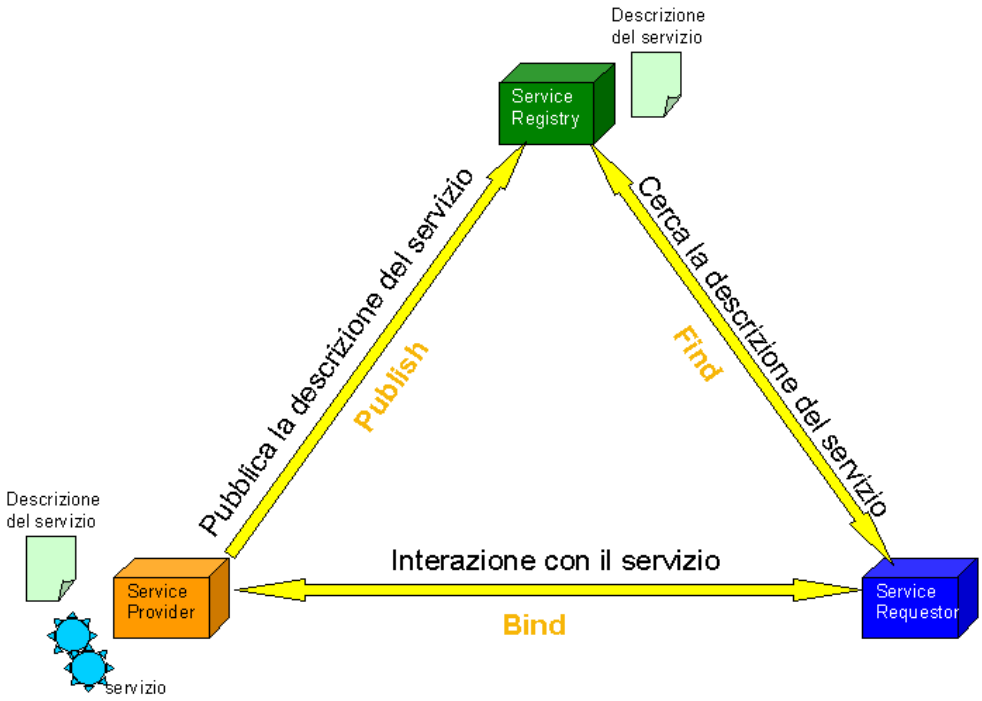


Figura 1.1 - *Ruoli e interazione nel modello concettuale dei Web Services.*

Le operazioni che consentono l'interazione tra i tre componenti del modello sono:

- **Publish** – Il service provider invia la descrizione del servizio, che ospita, al service registry. Questo classifica il servizio e lo pubblica in maniera tale che un service requestor possa trovarlo.
- **Find** – Il service requestor interroga il service registry per ottenere la descrizione del servizio richiesto.
- **Bind** – Il service requestor utilizza i dettagli presente nella descrizione del servizio per contattare il service provider mediante il quale interagisce con l'implementazione del servizio.

Attualmente i Web Service, sono semplici e statici. Non esistono ancora tecnologie standard che consentono un'interazione complessa

tra i servizi, o la loro scoperta dinamica (binding dinamico). Lo sviluppatore deve ricercare il Web Service desiderato presso un registro, ad esempio usando un comune browser HTML, e una volta trovato, deve effettuare il download della descrizione del servizio che gli permetterà di effettuarne l'accesso. Dunque, l'integrazione del servizio nell'applicazione deve avvenire al tempo di sviluppo e successivamente, a runtime, avverrà il binding sull'implementazione del servizio. Questo tipo di meccanismo viene definito binding statico. Un binding dinamico, invece, consiste nella scoperta a runtime del servizio da parte dell'applicazione, e conseguentemente l'interazione con la sua implementazione.

In definitiva, la scoperta del Web Service, attualmente, è un meccanismo ancora allo stadio primordiale. Per ottenere una scoperta dinamica bisognerà aspettare che i Web Services prendano piede e che le interfacce dei servizi d'ogni genere di categoria saranno standardizzate.

3.3.1. Tecnologie abilitanti.

Lo standard che consente ai customer, fornitori, e partner commerciale d'invocare un Web Service è **SOAP** (Simple Object Access Protocol). Mediante questo protocollo una qualsiasi applicazione può invocare un servizio a run-time, ovunque esso si trovi nel Web. I parametri d'ingresso e uscita del metodo del componente remoto sono definiti in un documento XML che viene incapsulato in un messaggio HTTP e inviato al Web Application Server del privider del servizio. Un dispatch SOAP, ecodificato il messaggio XML, provvede a invocare il metodo richiesto e prepara

una risposta, codificata in XML, che invia al richiedente. L'intero processo è illustrato per sommi capi in figura 2.

Come già detto in precedenza, per integrare automaticamente un Web Service a livello di programmazione è necessario un documento che descriva tutti i dettagli tecnici relativi al servizio, come l'interfaccia (nomi dei metodi, i tipi degli argomenti e dei valori di ritorno), il protocollo utilizzato per effettuarne il binding, informazioni relative all'endpoint e altri dettagli associati all'implementazione. Il linguaggio usato per scrivere questo documento è chiamato **WSDL** (Web Services Description Language) e la sua semantica è stata definita utilizzando l'estendibilità del metalinguaggio XML. La descrizione del servizio risulta essere indipendente dall'implementazione del servizio.

Infine, un Web Service può essere pubblicato e scoperto utilizzando un registro di disponibilità, una specie di pagine gialle elettronoche. Le specifiche che definiscono le strutture dati che rappresentano le informazioni dell'azienda che fornisce il servizio e i dettagli sul servizio stesso, le API (Application Programming Interface) delle operazioni di pubblicazione e ricerca nonché la tassonomia dei servizi sono proposte dello standard **UDDI** (Universal Description, Discovery and Integration).

3.3.1.1. SOAP, un protocollo leggero.

SOAP definisce un modello per realizzare, mediante l'uso di semplici messaggi di richiesta e risposta, un protocollo di base per lo scambio di qualunque informazione in formato XML in ambiente distribuito. SOAP non definisce un modello di programmazione, non implementa una specifica semantica ed è indipendente dal protocollo di trasporto

sottostante, che può essere HTTP(s), SMTP, e FTP. Queste caratteristiche possono essere sfruttate per implementare, in una piattaforma neutrale, un meccanismo simile a chiamate a procedure remote (RPC) che utilizzi il protocollo HTTP a livello di trasporto. Il meccanismo RPC-like è utilizzato per invocare il componente che implementa il servizio remoto e per il relativo scambio dei parametri.

La chiamata di procedura remota è rappresentata, sostanzialmente, da un messaggio XML che contiene il nome e gli argomenti del metodo del componente remoto, a differenza di DCOM e CORBA che invece usano un formato binario per il payload del messaggio (NDR e CDR, rispettivamente) e perciò non indipendente dalle tipo di piattaforma utilizzata (requisiti simmetrici).

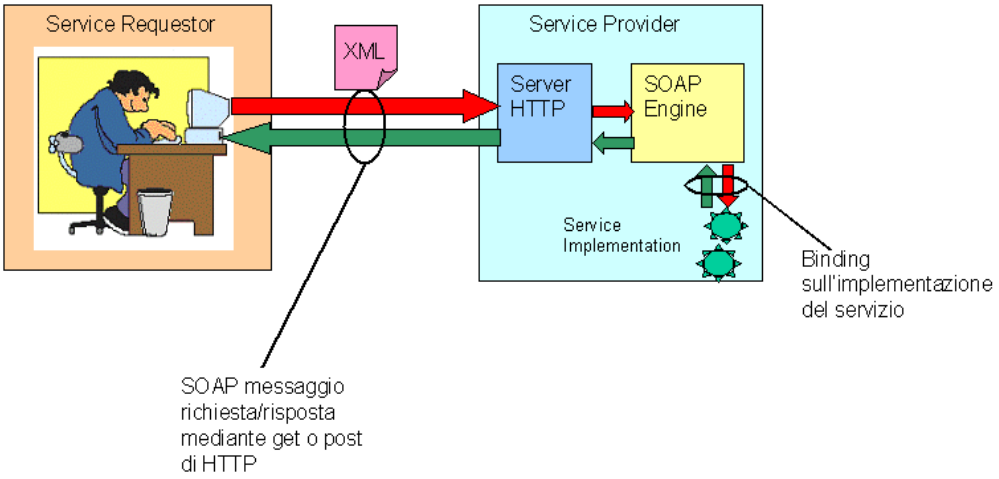


Figura 1.2 - *SOAP consente di scambiare messaggi XML mediante i metodi Get e Post del protocollo HTTP.*

Utilizzando SOAP-RPC, è possibile comunicare con qualsiasi componente software, di cui sia nota l'interfaccia, capace di interpretare un messaggio di richiesta e ritornare un'appropriata risposta. Il modulo software può essere un'applicazione legacy, un Java

Bean, un Enterprise Java Bean, un oggetto COM, un'applicazione scritta in C++, etc.

Nei listati 1.1 e 1.2 vengono riportati, rispettivamente, un messaggio SOAP di richiesta inviato al metodo *GetLastTradePrice* del servizio *StockQuote* e la relativa risposta.

Come si osserva da tali listati, il messaggio SOAP è costituito da tre pezzi principali:

- **Envelope** – E' un elemento obbligatorio, e fornisce un framework per descrivere cosa c'è nel messaggio e in che modo elaborarlo.
- **Header** – E' un elemento opzionale per aggiungere informazioni ausiliarie specifiche dell'applicazione, come per esempio la modalità di autenticazione, di gestione delle transazioni, etc.
- **Body** – Elemento obbligatorio che descrive il playload del messaggio, cioè, ciò che deve essere elaborato.

Gli elementi Header e Body devono essere inclusi nell'elemento Envelope.

Nel messaggio di richiesta è evidente la descrizione della chiamata al metodo *GetLastTradePrice*, al quale viene passato come parametro la stringa *DIS*. Il messaggio di risposta ritorna il valore *34.5* corrispondente all'ultimo prezzo commerciale del titolo identificato da *DIS*.

Request

```
POST /StockQuote HTTP/1.1                                   Messaggio
Host: www.stockquoteserver.com                              HTTP-Post
Content-Type: text/xml
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP:Envelope
xmlns:SOAP="urn:schemas.xmlsoap.org:soap.v1">
<SOAP:Header>
<t:Transaction xmlns:t="URI"
mustUnderstand="1">5</t:Transaction>
</SOAP:Header>                                              Messaggio
<SOAP:Body>                                                 SOAP
<m:GetLastTradePrice
xmlns:m="URI">
<symbol>DIS</symbol>
</m:GetLastTradePrice>
</SOAP:Body>
</SOAP:Envelope>
```

Listato 1.1 - Messaggio di risposta SOAP.

Inoltre, dai listati si evince che i messaggi SOAP sono trasportati come payload dei messaggi Post del protocollo HTTP, e per questo motivo essi possono attraversare tranquillamente la barriera dei firewall dell'Intranet aziendale, uscire su Internet e fare il percorso inverso. Diversamente da altri modelli a componenti distribuiti che invece utilizzano l'assegnamento dinamico delle porti per comunicare con gli oggetti e che per questo trovano difficoltà ad attraversare un firewall che lasci aperta la sola porta 80 per uscire dall'Intranet.

Response

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<SOAP:Envelope
xmlns:SOAP="urn:schemas.xmlsoap.org:soap.v1">
<SOAP:Header>
<t:Transaction xmlns:t=”URI”
xsi-type=”xsd:int”
mustUnderstand=”=”>5</t:Transaction>
</SOAP:Header>
<SOAP:Body>
<m:GetLastTradePriceResponse
xmlns:m="URI">
<return>34.5</return>
</m:GetLastTradePriceResponse>
</SOAP:Body>
</SOAP:Envelope>
```

Listato 1.2 – *Messaggio di risposta SOAP.*

In definitiva, SOAP ha successo laddove altre tecnologie falliscono. COM/DCOM è un protocollo di comunicazione solo tra piattaforme con sistemi operativi Windows. CORBA richiede la presenza di ORB compatibili su entrambi gli endpoint. RMI lavora solo tra applicazioni Java con librerie coordinate. Invece, SOAP, utilizzando il formato testuale XML, indipendente da qualsiasi piattaforma, può lavorare con qualunque implementazione, codice ereditato da sistemi legacy, oggetti COM, CORBA, Java, o un qualsiasi altro modulo che comprenda il protocollo SOAP [2].

Data l’importanza dell’argomento, riprenderò ed approfondirò queste osservazioni nel capitolo 3.

3.3.1.2. Rapida integrazione con WSDL.

Per accedere ad un Web Service, il potenziale utilizzatore deve conoscere l'interfaccia del componente che implementa il servizio, l'endpoint del servizio remoto e altri dettagli necessari per il binding. Tutte queste informazioni sono descritte mediante tag predefiniti del linguaggio di markup WSDL. Esso possiede elementi ed attributi per definire le operazioni messe a disposizione dal servizio, i tipi dei parametri di scambio e quelli dei parametri di ritorno, il tipo di protocollo utilizzato per comunicare con il componente ed altro.

Il documento WSDL separa la descrizione astratta dei dati da scambiare (Message) e delle operazioni realizzate dal servizio (PortType) con l'implementazione concreta (service) che consente di effettuare l'effettiva chiamata al servizio remoto. La parte concreta del documento, dunque, contiene un insieme di Port, in altre parole indirizzi URL e informazioni specifiche per effettuare il binding mediante il protocollo SOAP.

Un esempio di descrizione concreta del servizio StockQuote è mostrata nel listato 1.3. In essa è definito il metodo chiamato GetLastTridePrice, con le relative informazioni sul protocollo SOAP usato per trovare il componente, invocare il metodo, ed elaborare la risposta.

```
<binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
  <soap:bindingstyle="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetLastTradePrice">
        <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
        <input>
           <soap:body use="literal"/>
        </input>
        <output>
           <soap:body use="literal"/>
        </output>
      </operation>
</binding>
```

Listato 1.3 – *Esempio di documento WSDL.*

Nella figura 1.3 sono illustrate le varie componenti in un documento WSDL, suddivise per livelli di astrazione.

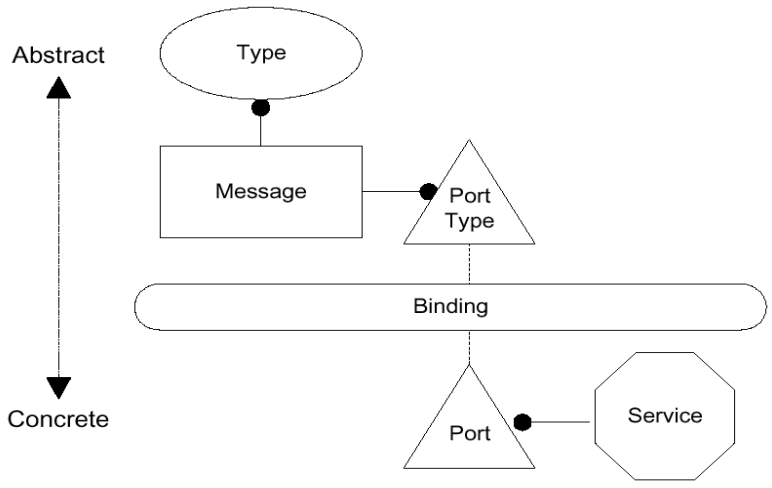


Figura 1.3 – *Elementi WSDL.*

Il service requestor può ottenere in vari modi il documento WSDL. Ad esempio, dopo aver effettuato una ricerca presso un catalogo di

Web Services, può scaricarlo dal sito del service provider, o anche riceverlo per posta elettronica.

Una volta ottenuta la descrizione del servizio, uno sviluppatore può utilizzarla, in fase di progettazione, per generare automaticamente, mediante un tool, il modulo client da incorporare nell'applicazione. Questo modulo, detto anche proxy, consente, a runtime, di comunicare, attraverso il protocollo SOAP, con il Web Service. L'intero processo è illustrato in figura 1.4.

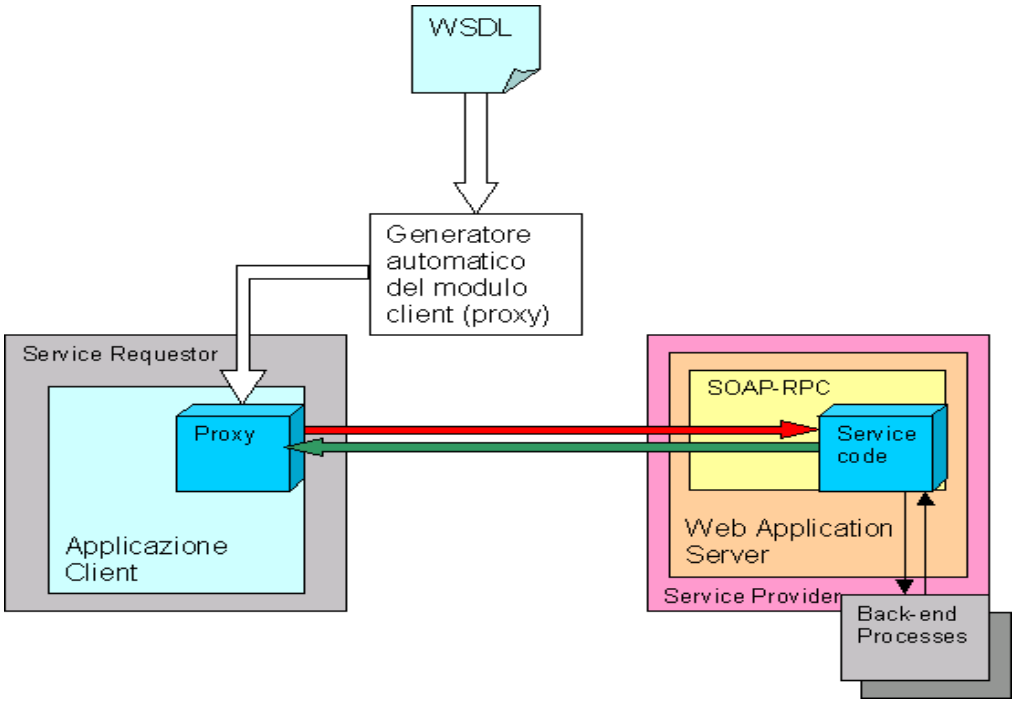


Figura 1.4 – *Integrazione di un Web Service in un'applicazione a partire dalla sua descrizione.*

3.3.1.3. UDDI, come le pagine gialle.

L'adozione di un Web Service framework e di un service registry consentono, ad utenti e fornitori d'ogni parte del mondo, di condividere le proprie informazioni, di accedere ai Web Service e nello stesso tempo di continuare ad usare la tecnologia che ciascuno ritiene più vantaggiosa.

In questo nuovo scenario, le informazioni da pubblicate sulla rete devono essere strutturate in modo da consentire il reperimento di partner in tempi brevi, da definire la modalità con cui collegarsi ai loro servizi, e soprattutto devono rispettare una specifica di pubblicazione e di ricerca che sia comunemente accettata.

Insomma, una specie di pagine gialle elettroniche cui si rivolgono i service requestor per cercare i Web Services desiderati, o anche solo le informazioni su una certa azienda, e i service provider per pubblicare il loro profilo e i loro servizi [4].

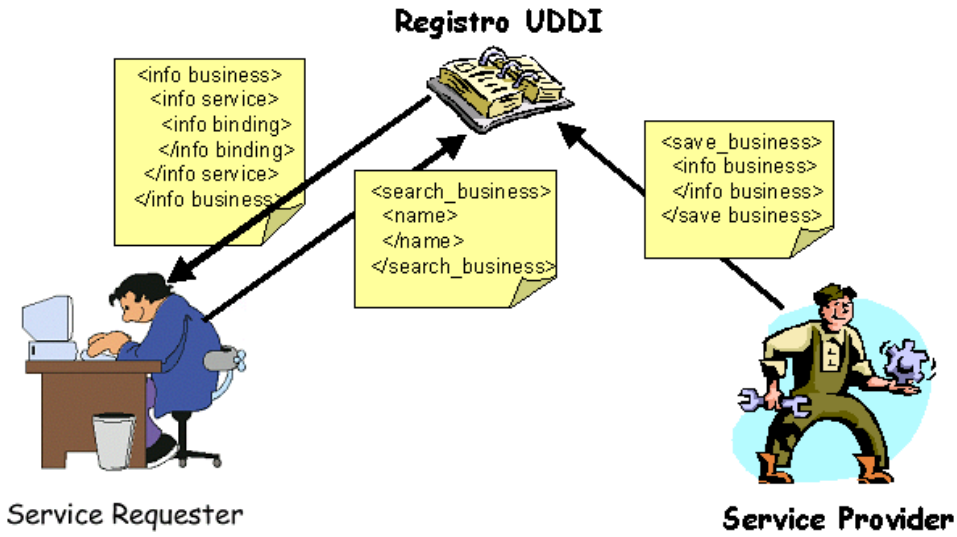


Figura 1.5 – *Le operazioni d'interrogazione e pubblicazione nonché le strutture dati sono tutti descritti mediante il formalismo XML.*

Per la comunicazione col registro, nell'ambito del modello dei Web Services, viene utilizzato il protocollo SOAP.

Tecnicamente, il service provider invia messaggi XML strutturati in maniera tale da descrivere i dati sulle proprie generalità, i dettagli relativi ai servizi e le operazioni di pubblicazione. Invece, il service requestor invia messaggi XML che descrivono le interrogazioni da effettuare, e ricevono messaggi XML con le informazioni cercate. Il meccanismo è illustrato in figura 1.5.

Nella seguente figura 1.6 sono mostrate le quattro strutture fondamentali dei dati descritte dalla specifica UDDI.

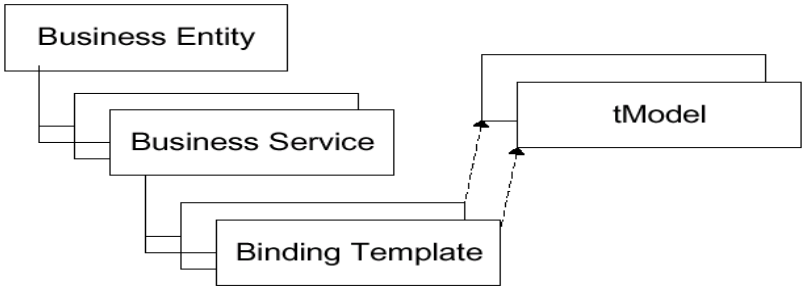


Figura 1.6 – Strutture dati UDDI.

Business Entity descrive le informazioni sull'azienda. Essa include il nome, la descrizione, i servizi offerti dall'azienda nonché le informazioni per essere contattata.

Business Service fornisce ulteriori dettagli per ciascun servizio offerto da una particolare azienda. Ogni servizio può avere più binding templates, ciascuno dei quali descrive un entry point per il servizio; per esempio, tramite posta elettronica, http, ftp, fax e comunicazione telefonica.

Infine, tModels descrive quale specifica o standard usa un certo servizio.

Con queste informazioni, un'azienda può localizzare altri servizi che sono compatibili con il proprio sistema.

In più, UDDI fornisce identificatori e categorie per contrassegnare ciascun'entità usando diversi sistemi di classificazione; per esempio, NAICS per i codici industriali, UNSPC per prodotti e servizi e ISO 3166 per aree geografiche.

La specifica UDDI definisce anche un insieme di API che consentono di interagire con il registro. Le API sono suddivise in due categorie:

- API Inquiry – Sono operazioni d'interrogazioni che consentono di ottenere le entità Business, Service, Binding, e tModel.
- API Publish – Sono operazioni di modifica che consentono di creare e cancellare le entità nel registro. Per utilizzare queste operazioni, è necessaria una valida autorizzazione da parte dell'operatore che gestisce il registro.

Le API UDDI sono messaggi SOAP trasportati su protocollo http, nel listato 4 è mostrato un esempio di tale messaggio.

Infine va detto che il registro UDDI può essere usato anche all'interno dell'azienda, come registro privato, per organizzare le funzionalità messe a disposizione da componenti sviluppati in ambito Intranet.

```
POST /get_bindingDetail HTTP/1.1
Host: http://www.someoperator.org
Content-type: text/xml: charset="utf-8"
Content-Length: nnnn

SOAPAction: ""
```

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<Envelope
xmlns=http://schemas.xmlsoap.org/soap/envelope/>
<Body>
<get_bidingDetail generic=”1.0” xmlns=”urn:uddi-org:api”>
. . .
</Body>
</Envelope>
```

Listato 1.4 – *Parte di un'operazione di interrogazione inviata al registro UDDI. get_bindingDetail consente di estrarre i dettagli di binding di un servizio.*

4. Web e Web Services.

Il Web diventa sempre più dinamico ed esigente. I portali, oggi, forniscono contenuti aggiornati minuto per minuto, quotazioni di titoli, news, contenuti personalizzati sui profili dell'utenza, servizi di motori di ricerca, etc. Questi portali sono di tipo *digital dashboard* [8] e raccolgono informazioni da diverse sorgenti per presentarle in una sola schermata, analogamente al cruscotto dell'auto che presenta una vista complessiva dello stato di vari sottosistemi, come la velocità, la temperatura, il livello del carburante, etc. Attualmente, le tecniche d'integrazione delle applicazione Web nei siti fanno uso di *link*, *frames*, *screen scraping* e *posting information* [8].

Il *link* è stato il primo metodo per muoversi tra le pagine Web e per richiamare applicazione Web. Tuttavia, è abbastanza grossolano, poiché costringe l'utente ad abbandonare la pagina Web corrente per visualizzare il nuovo contenuto puntato dal link.

I *frames* consentono un maggiore grado d'interoperabilità, potendo visualizzare su sezioni separate il contenuto di diversi siti Web,

utilizzando, ad esempio, particolari ambienti lato client, come applet Java, controlli ActiveX, etc. Comunque, la più grossa limitazione dei frames è dovuta alla frammentazione dell'aspetto del sito, dato che in ciascuna sezione compare il front-end grafico dell'applicazione Web integrata che appartiene, evidentemente, ad un altro sito. L'utente si trova davanti un puzzle di diversi formati di pubblicazioni, e ha l'impressione di consultare differenti siti Web nello stesso tempo, o un sito *cucito* con diversi pezzi d'aspetto grafico, layout e stili differenti.

L'approccio di tipo *screen-scraping* utilizza il Web server come client di un'applicazione Web, generalmente fornita da terze parti. Precisamente, il sito Web invia la richiesta dell'internauta all'applicazione remota. Questa dopo aver effettuato le opportune elaborazioni, racchiude il contenuto informativo in una pagina HTML e lo spedisce indietro come risultato. Il server estrae dalla pagina il contenuto informativo rilevante e lo pubblica in un formato consono allo stile del sito. In questo modo, l'utente, mentre consulta la risposta, non si accorge dell'intervento di un'applicazione Web esterna al sito. Questa tecnica è suscettibile di possibili errori dovuti a cambiamenti sensibili delle applicazioni Web integrate. Per evitare quest'inconveniente dovrebbe esistere una sorta di cooperazione tra il sito Web e l'applicazione utilizzata.

Il *posting information* si serve dei metodi Post e Get di HTTP per realizzare un protocollo di tipo request/response con l'applicazione client. Talvolta, il posting information viene identificato come la prima generazione dei Web Services, prima dell'avvento di XML, infatti, entrambi utilizzano il protocollo HTTP a livello di trasporto. Tuttavia,

nel posting information non esiste un formato standard per i messaggi, e tanto meno un meccanismo per effettuare chiamate a procedure remote come avviene invece in SOAP.

La prossima evoluzione nello sviluppo d'applicazioni Web è rappresentata dai Web Services, che offrono soluzioni più flessibili all'integrazione di portali digital dashboard. Un'applicazione Web richiama un servizio Web attraverso una funzione, come avviene per una normale applicazione, e acquisisce un risultato che può essere gestito come meglio si crede, pubblicarlo nel formato desiderato, o passarlo come argomento ad un altro servizio Web. Con questa tecnica, l'azienda può focalizzare le proprie competenze dentro la sua applicazione Web e integrare agevolmente altre capacità usando servizi Web forniti dai propri partner o da altre società.

Nel prossimo futuro, la banda larga sarà alla portata di un ampio numero di utenti e potrà trasportare contenuti più complessi e ancora più dinamici, integrati con streaming audio e video. Le capacità dei supporti di memorizzazioni (hard disk, DVD, CD-ROM, removibile-storage) continuano ad aumentare e diventano sempre più economici. Il Pervasive Computing apre nuovi canali di comunicazione, attraverso i quali i contenuti informativi raggiungono mobile phone, palmtop e pager.

I Web Services possono essere velocemente integrati nei portali, per offrire contenuti sempre nuovi e caratteristiche sufficienti da incoraggiare i visitatori ad azzardare oltre la homepage. Una compagnia può sviluppare Web Services che gestiscono contenuti integrati con streaming video e audio, o che effettuano complesse

elaborazioni che richiedono piattaforme hardware o software costosissime.

5. Ambienti di sviluppo.

Attualmente, esistono sul mercato parecchi toolkit che consentono agli sviluppatori di creare, esporre e testare Web Services. Inoltre, qualcuno di questi consente anche di trasformare componenti preesistenti, come oggetti COM e JavaBeans, in Web Services.

La soluzione Microsoft per creare, esporre e consumare Web Services è rappresentata dal framework .NET.

5.1. Microsoft .Net.

Nella visione .NET, le applicazioni sono costruite a partire da Web Services che lavorano insieme per fornire dati e servizi alle applicazioni. Questo concetto viene illustrato in figura 1.6.

Nel framework .Net è definito un linguaggio di runtime (Common Language Rubtime) che esegue codice scritto in parecchi moderni linguaggi di programmazione, e, in più, consente di utilizzare servizi che aiutano a semplificare lo sviluppo di codice e il deployment delle applicazioni.

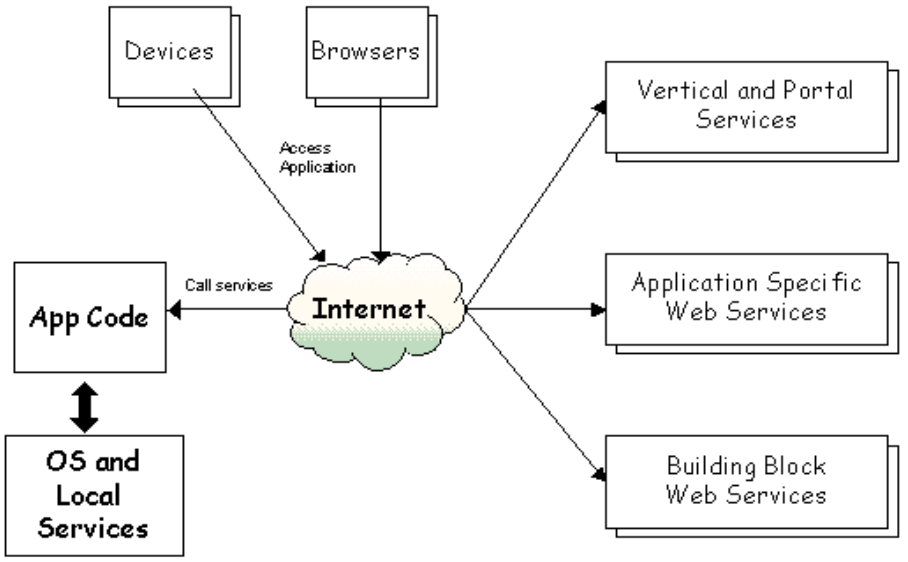


Figura 1.6 - Visione .NET di un'applicazione che usa servizi.

Il framework .NET include anche un set di librerie di classi che gli sviluppatori possono usare da un qualsiasi linguaggio di programmazione. Su queste librerie si collocano vari modelli di programmazione che forniscono componenti e servizi a più alto livello, ad uso, in special modo, degli sviluppatori di siti Web e Web Services.

Nella figura 1.7 viene mostrata l'architettura .NET. Un'ambiente di runtime, in pratica una virtual machine, carica ed esegue un particolare codice, detto codice gestito. Tale codice è scritto usando un sistema di tipi comuni, capace di esprimere la semantica di parecchi moderni linguaggi di programmazione. Il common language, dunque, definisce un set di tipi standard e alcune regole per creare nuovi tipi. Il runtime, poi, crea ed esegue questi tipi.

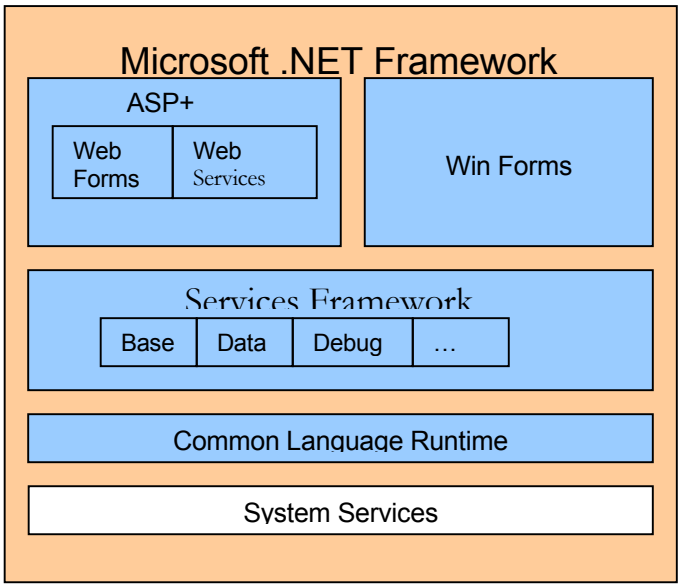


Figura 7 – *Mirosoft .NET Framework Architecture*

Utilizzando il common language e l'ambiente di runtime, è possibile integrare insieme classi scritte con differenti linguaggi di programmazione, fare in modo che codice scritto in un linguaggio possa ereditare l'implementazione da classi scritte in un altro linguaggio; gestire situazioni in cui le eccezioni possono essere lanciate da un codice scritto in un certo linguaggio e catturate da un codice scritto in linguaggio differente [6].

Il livello successivo al common language runtime è il services framework. Questo fornisce librerie di classi che possono essere chiamate dai moderni linguaggi di programmazione (Visual basic, C++, J++, etc.). L'insieme delle librerie è suddiviso in un sottoinsieme di base che è costituito da librerie standard, costituite da classi quali le collezioni, per la gestione input/output, stringhe, e classi numeriche. In aggiunta, troviamo le classi che consentono di accedere ai servizi del sistema operativo come per esempio classi per la gestione della grafica,

del networking, dei thread e cryptography. Un secondo sottoinsieme di librerie include classi per accesso ai dati e altre per il debugging, tracing, configurazione, installazione delle applicazioni. In particolare, per l'accesso ai dati il services framework mette a disposizione la libreria di classi ActiveX® Data Objects+ (ADO+). Le classi ADO+, che sono un'evoluzione di ADO, sono progettate per fornire l'accesso ai dati da parte di applicazioni Web-based e servizi.

Sul livello services framework poggiano due modelli d'applicazione: il modello Windows application e il modello Web application. Il primo modello può essere usato per sviluppare tradizionali applicazioni Windows-based, che ad esempio usano Web Services. Il secondo modello invece consente di realizzare applicazioni Web e Web Services, e viene comunemente chiamato modello Active Server Pages+ (ASP+). Il modello di applicazioni Web viene illustrato in figura 8.

ASP+ è un'evoluzione di Active Server Page che si avvantaggia del common language runtime e services framework per fornire un ambiente scalabile, robusto ed affidabile in cui eseguire applicazioni Web-based. Il componente principale di ASP+ è HTTP runtime che assolve il compito di elaborare le richieste HTTP. Questo componente è codice gestito ed è avviato all'interno di un processo non gestito, come IIS (Internet Information Server) su una macchina server o Internet Explorer su una macchina client. HTTP.

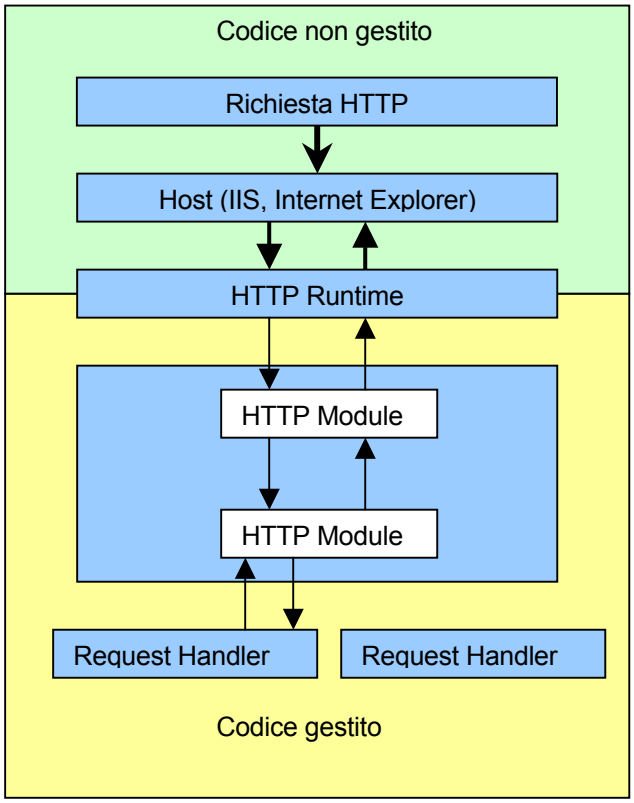


Figura 1.8 – *Modello d'applicazione ASP+.*

Esso elabora tutte le richieste HTTP, risolve l'URL d'ogni richiesta e le instrada verso i servizi target. HTTP. E' multithread e processa le richieste in modo asincrono, impedendone il blocco qualora un'applicazione dovesse provocare un malfunzionamento.

Le richieste HTTP sono instradate attraverso una pipeline di moduli HTTP, e giungono infine ai moduli di gestione delle richieste che trasformano la richiesta in una chiamata al metodo dell'oggetto che implementa il servizio cercato. Sia i moduli HTTP sia quelli di gestione delle richieste sono semplicemente delle classi gestite che implementano specifiche interfacce definite mediante ASP+. Per aggiungere un servizio ad un'applicazione Web è sufficiente fornire un modulo HTTP. Invece, per servizi a più alto livello, come Web Services o Web Form (form Visual Basic-based che possono essere

integrati in un'applicazione Web), bisogna implementare un modulo di gestione delle richieste.

Il modello di programmazione ASP+ permette di realizzare anche Web Services. Il meccanismo di creazione richiede, dopo lo sviluppo del componente che implementa il servizio, la preparazione di un file con estensione asmx e successivamente il suo deploy come un parte di una Web Application. Il file ASMX può contenere o un riferimento ad una classe gestita, definita altrove, oppure la definizione della classe stessa. Va peraltro notato che la classe deve essere derivata dalla classe WebService fornita da ASP+. I metodi pubblici della classe sono esposti come metodi del Web Service. Infine, questi metodi possono essere invocati, inviando richieste HTTP all'URL del file ASMX. ASP+ mette a disposizione un'utility che ispeziona l'interfaccia della classe e genera automaticamente un file SCL.

SCL (Service Contract Language) è un documento XML che descrive un contratto per il Web Service, cioè descrive i servizi in termini di messaggi accettati e generati e non come un certo servizio è implementato.

L'applicazione client che fa uso di Web Services, può utilizzare SOAP, HTTP Get e HTTP POST per sottomettere le richieste ai servizi remoti.

ASP+ fornisce anche un tool per la generazione automatica di proxy per qualunque Web Services descritto mediante il documento SCL. Il generatore mappa i messaggi descritti nel file SCL in metodi delle classi che realizzano il servizio remoto, nascondendo i dettagli per la serializzazione /deserializzazione SOAP dei dati.

Tutti i prodotti che consentono di implementare ed esporre Web Services e Web Application sono inclusi in .NET Enterprise Servers. Tra questi prodotti, va segnalato BizTalk Server 2000, che consente di gestire processi business e di scambiare informazioni con i partner. Questo fornisce il supporto per XML e messaggi SOAP-based, che trasmette su vari protocolli, inclusi HTTP, SSL, e SMTP.

In definitiva, la Microsoft consente agli sviluppatori di Web Service di scegliere quattro tipi di soluzioni:

Implementare Web Services, usando MSXML, ASP, oppure ISAPI, e testare il tutto senza l'ausilio di un ambiente di sviluppo dedicato.

Usare la piattaforma .NET e il modello ASP+ per Web Services.

Usare il SOAP ToolKit per Visual Basic 6.0 per creare Web Services da oggetti COM.

Usare Microsoft SOAP Toolkit versione 2.0 che aggiunge alla soluzione precedente il supporto a WSDL e la possibilità di registrare il servizio presso un registro UDDI, seppure manualmente.

Dunque, solo l'ultima soluzione realizza, in parte, il modello concettuale dei Web Services introdotto nel paragrafo 1.3.3.

5.2. Microsoft SOAP toolkit 2.0.

Questo toolkit consente di creare automaticamente documenti WSDL da oggetti COM, di pubblicare il servizio (deployment del servizio) presso il Web server ed infine d'invocarlo, mediante messaggi SOAP, da un client ASP, per la fase di testing.

Il componente client utilizza il SOAP client, che è un componente COM in grado di scambiare messaggi con il SOAP server, utilizzando le informazioni presenti nel documento WSDL, per effettuare la richiesta al servizio remoto. Una volta processata la richiesta, il server Soap la trasformerà in una chiamata al metodo del componente che implementa il Web Services e infine fornirà la risposta in formato XML per il client SOAP.

In fase di test, può essere chiamato direttamente il server SOAP, attraverso una pagina ASP, indirizzando direttamente il servizio e usandone i metodi disponibili.

Una volta creato l'oggetto COM, ad esempio con Visual Basic 6.0, si deve utilizzare l'utility WSDL generator per la creazione del file Wsdl. In questa fase verrà richiesto il nome del servizio e il percorso dell'oggetto COM creato. Nella fase successiva, devono essere specificati i metodi che si vogliono esporre sul Web. L'utility, ad ogni modo, scarterà tutti quei metodi che utilizzano tipi per i quali non riesce a generare automaticamente i serializzatori/deserializzatori necessari per lo scambio dei dati. Proseguendo, il wizard chiederà il percorso URL in cui cercare il SOAP listener incaricato di smaltire le richieste SOAP. Al termine della procedura guidata, verranno creati il file Wsdl e i file necessari a consumare il Web Service.

5.3. IBM WSDE.

IBM fornisce un ambiente di sviluppo che automatizza vari aspetti dello sviluppo dei Web Services, semplificando la fase di progettazione, il deployment e l'integrazione. Inoltre, fornisce alcuni tool che consentono di "wrappare" le applicazioni preesistenti,

generare la descrizione del servizio, e creare componenti proxy da questa.

L'ambiente WSDE (Web Services Develpment Enviroment) richiede la presenza di Web Application Server, come WebSphere o Tomcat, per eseguire il server SOAP (in pratica una servlet che gestisce messaggi SOAP) nonché le JSP (Java Server Page) per testare i Web Services.

L'applicazione WSDE permette di importare classi Java e di generare, a partire da esse, il documento WSDL e il deployment descriptor, un documento XML che viene usato per esporre il servizio presso l'engine SOAP-APACHE. Il file WSDL può essere anche modificato, usando l'editor WSDL.

Dalla descrizione del servizio e dalle classi che lo implementa, si può passare o alla fase di testing o a quella di costruzione del Web Service.

Nella figura 1.9 viene mostrato lo schema di testing seguito da WSDE.

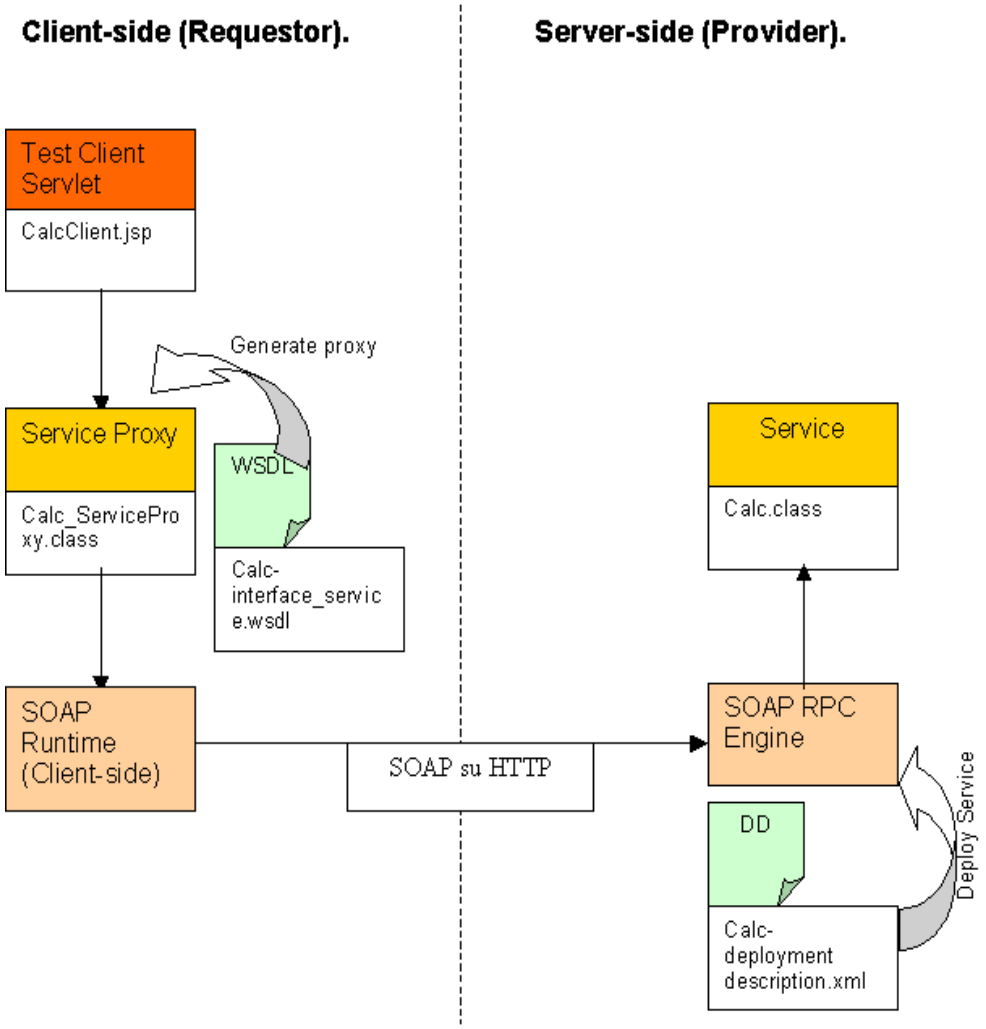


Figura 1.9 - *Schema di testing del toolbox IBM WSDE.*

Per effettuare il testing, l'applicazione pubblica il servizio presso il server SOAP, usando il deployment descriptor (DD), e genera i seguenti file temporanei:

- un proxy client (codice client che consente di comunicare col servizio remoto) che utilizza le classi del pacchetto SOAP di Apache versione 2.0.
- una Java Server Page che utilizza il proxy e un'interfaccia utente per provare il servizio.

Oltre alla creazione, esposizione e testing remoto o locale del Web Service, l'ambiente di sviluppo WSDE consente, anche, di pubblicarlo e scoprirlo presso un registro UDDI pubblico o privato. Nel caso di un registro privato è richiesta la presenza di RDBMS DB2.

Capitolo II

WSOA
(WEB SERVICES-ORIENTED ARCHITECTURE).

1. Introduzione.

WSOA è un'architettura aperta e indipendente dalla piattaforma in cui servizi distribuiti e lascamente accoppiati possono essere pubblicati e consumati sul Web. I blocchi principali di quest'architettura, mostrati nel diagramma 2.1, sono associati ai seguenti concetti relativi al servizio:

- Invocazione.
- Descrizione.
- Scoperta (discovery).

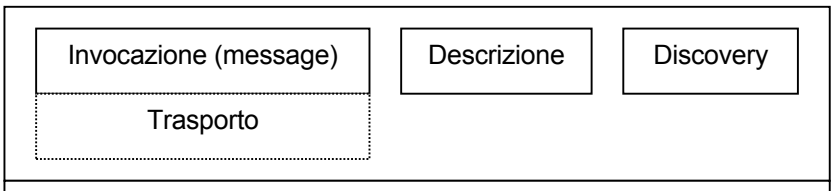


Figura 2.1 – *Blocchi principali dell'architettura orientata ai servizi Web.*

Queste tre entità sono coinvolte nelle interazioni tra i principali attori che pubblicano e consumano i servizi. La figura 2.1 illustra i tre ruoli fondamentali e le relative interazioni:

- Service provider, chi pubblica (publish) i servizi disponibili.
- Service requestor, chi scopre (find) e invoca (bind) i servizi desiderati.
- Service registry o broker, che consente ai service provider di pubblicare le descrizioni dei servizi e ai service requestor di trovare i servizi desiderati.

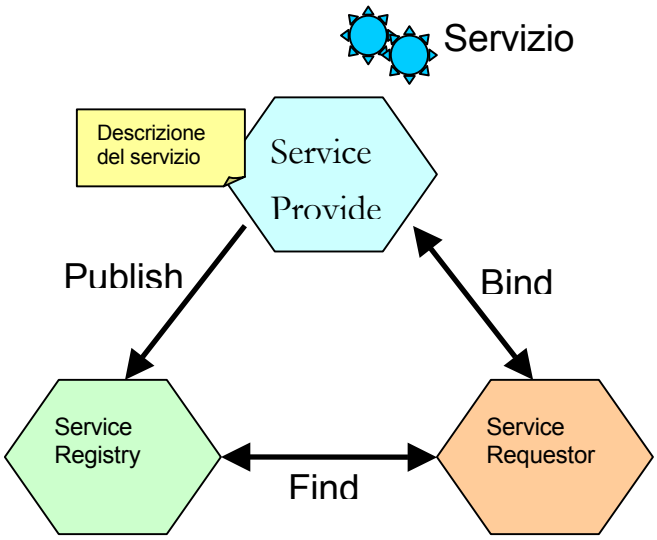


Figura 2.2 – *I ruoli e le interazioni fondamentali nell'architettura orientata ai servizi Web.*

Dall'implementazione dell'architettura WSOA si possono ottenere efficienti e dinamiche soluzioni e-business, mediante le quali le organizzazioni possono interagire agilmente con i loro customer e partner, creando nuovi canali per la fornitura di servizi e nuove forme di collaborazione. D'altra parte l'architettura orientata ai servizi non è un concetto nuovo. Circa un anno e mezzo fa, il prodotto e-Speack [7] di HP fu presentato al mercato come una piattaforma aperta per sviluppare e pubblicare e-Services. Un e-Service è un servizio che è

disponibile attraverso internet e realizza un dato compito o transazione, un po' come un Web Service. Esso può essere scoperto dinamicamente, invocato e composto da altri e-Services. E-Speak, sviluppato completamente in Java, ha un solido modello di sicurezza, basato sullo standard SPKI (Simple Public Key Infrastructure, http://www.ietf.org/html.charters/spki-charter.html).

Dovuta in parte ai suoi requisiti proprietari, e-speack non ha avuto un grosso successo.

Nel febbraio 2001, HP ha rinnovato la sua strategia software abbracciando la nuova tecnologia SOAP per l'accoppiamento dei componenti distribuiti, e si è mossa nella direzione di Microsoft e IBM realizzando una piattaforma basata sui Web Services.

I due grossi produttori di software, concordi sul fatto che il successo della realizzazione di un modello concettuale basato sui servizi Web può essere ottenuto utilizzando esclusivamente soluzioni software basate su open standard, hanno rinunciando al controllo sulle specifiche delle tecnologie, implementanti il modello, delegando al consorzio W3C il compito della standardizzazione. Microsoft, IBM, HP, ARIBA ed altri marchi illustri dell'industria software, tuttora, collaborano (stravaganza della competizione) per lo sviluppo di un'infrastruttura basata sui Web Services.

Questa cooperazione aperta gioca un ruolo chiave nell'assicurare l'interoperabilità delle implementazioni dei servizi. I primi frutti tangibili di questi sforzi comuni sono le specifiche SOAP, WSDL e UDDI, costruite tutte sul top di XML.

2. *Stack tecnologico dei Web Services.*

Per realizzare le tre operazioni del modello concettuale, publish, find e bind, in maniera interoperabile, lo stack deve essere costruito sulla base di standard aperti. Nella figura 2.3 sono mostrati gli strati principali della piattaforma Web Services proposta da Microsoft e IBM.

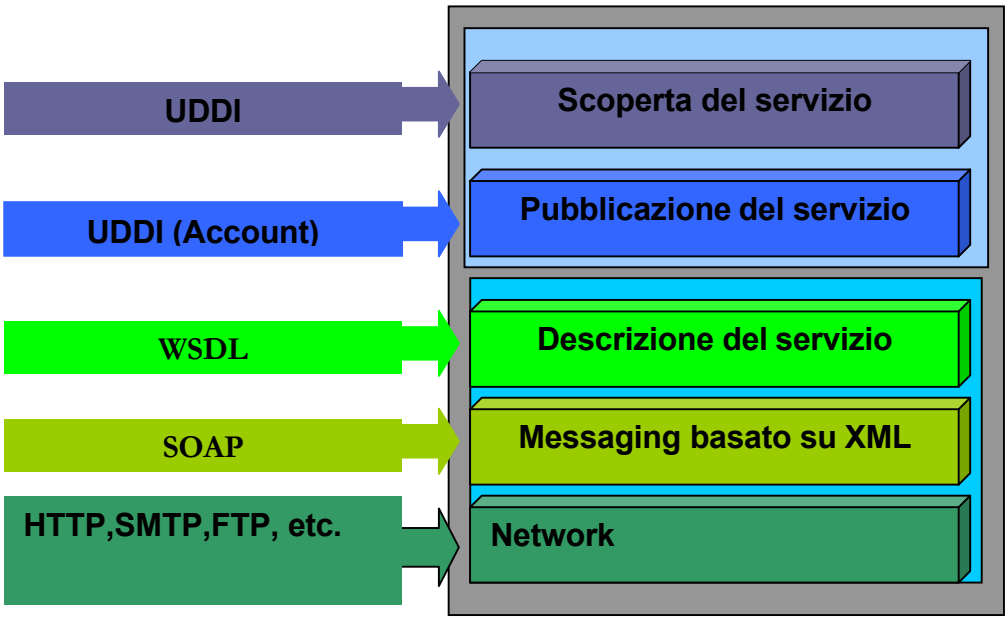


Figura 2.3 – *Stack tecnologico dei Web Services*

Ciascuno strato superiore è costruito sulle funzionalità fornite dai livelli inferiori. A sinistra della figura sono riportate le tecnologie che si applicano al livello indicato dalla freccia.

Il livello fondamentale su cui sono costruiti i Web Services è il livello network, che ne consente l'accesso da remoto. Tra i vari protocolli di trasporto che si possono usare a questo livello, verrà scelto HTTP (HyperText Transfer Protocol), principalmente perché è uno standard

de facto, oramai onnipresente e poi consente l'accesso ai servizi mediante Internet.

Il prossimo livello, XML-based messaging, si riferisce all'uso di XML come formato base dei messaggi per il protocollo di comunicazione. La scelta è SOAP, per le seguenti ragioni:

- È un protocollo di comunicazione sottoposto alla raccomandazione del consorzio W3C, attualmente disponibile in versione 1.2.
- E' molto leggero (se usa HTTP come protocollo di trasporto, richiede solo la capacità di inviare e ricevere messaggi HTTP, e di analizzare ed elaborare documenti XML).
- E' molto scalabile (su HTTP lo è in maniera nativa).
- Non è vincolato alla natura del protocollo di trasporto.
- Non dipende dal sistema operativo e dall'ambiente di programmazione.

Al livello della descrizione del servizio viene utilizzata la tecnologia WSDL che consente di definisce l'interfaccia e i meccanismi d'interazione dei servizi.

Un Web Service è essenzialmente un servizio accessibile attraverso il protocollo SOAP e rappresentato da una descrizione, per cui i primi tre livelli dello stack sono indispensabili per fornire o usare qualsiasi Web Service. Tale stack di base che garantisce l'interoperabilità dei

servizi Web, facendo leva sull'infrastruttura Internet esistente è illustrato in figura 2.4.

Figura 2.4 - *Stack di base che garantisce l'interoperabilità dei Web Services*

Mentre i tre livelli inferiori dello stack di figura 2.3 identificano tecnologie per conformità e interoperabilità, i prossimi due livelli di pubblicazione dei servizi e scoperta degli stessi ammettono un ampio range di scelte, in funzione della modalità di pubblicazione e scoperta del servizio.

2.1. Network.

Questo livello può essere rappresentato da un certo numero di protocolli di trasporto, come HTTP, FTP (File Trasfer Protocol), SMTP (Simple Mail Trasfer Protocol), Message Queuing (MQ), RMI (Remote Method Invocation) su Internet Inter ORB Protocol (IIOP). Il protocollo di rete usato in una qualsiasi data situazione dipende dai requisiti dell'applicazione.

Per Web Services accessibili da Internet, la scelta della tecnologia di rete è ricaduta sull'onnipresente protocollo HTTP. Per i Web Service che invece vengono forniti e consumati dentro un'Intranet, non si esclude l'opportunità di accordarsi sull'uso di tecnologie di rete

alternative. La tecnologia di rete può essere scelta basandosi su altri requisiti, come la sicurezza, disponibilità, performance e affidabilità. Questo permette ai Web Service di trarre vantaggio dalle esistenti infrastrutture per il networking.

Infine, si può sempre pensare di usare HTTP come bridge dentro un'azienda con molteplici tipi d'infrastrutture di rete.

2.2. Invocazione.

Il corrente standard industriale per XML messaging è SOAP. Microsoft®, IBM® e altri hanno sottomesso SOAP al W3C come base del XML Protocol Working Group. SOAP è un semplice e leggero meccanismo basato su XML per lo scambio di dati strutturati tra applicazioni distribuite. Come gia detto nel capitolo I, SOAP è costituito da tre parti fondamentali:

- Un envelope che definisce un framework per descrivere cosa c'è nel messaggio SOAP.
- Un set di regole di codifica per esprimere le operazione e i parametri di scambio dei componenti remoti
- Una convenzione per rappresentare chiamate simile a RPC

Nel capitolo III sarà approfondito ciascuno di quest'aspetto e illustrata un'implementazione di tale protocollo, invece in questo paragrafo presenterò, ponendomi ad un livello d'astrazione maggiore, la dinamica dello scambio dei messaggi tra il ruolo del service requestor e quello del service provider, in ambiente distribuito con SOAP come wire protocol.

Tipicamente, i requisiti di base per un nodo di rete che gioca il ruolo del service requestor, nell'architettura distribuita con SOAP come protocollo per il messaging e HTTP al livello di trasporto, sono componenti necessari per la comunicazione su Internet e un parser XML. Invece, per il service provider è comune utilizzare un server SOAP in esecuzione su una piattaforma con Web Application Server, con l'estensione per la gestione di documenti XML. Queste risorse menzionate sono, oggi giorno, già installate ed operative in molte realtà aziendali, che, quindi, possono trasformarsi senza traumi e agevolmente in service provider. Per quanto riguarda il service requestor, data la facile reperibilità d'implementazioni gratuite di client SOAP, non è difficile trasformare anche un semplice PC di casa in un client SOAP.

Nella figura 2.5 è mostrato un caso d'interazione tra un consumatore e un fornitore di servizi Web.

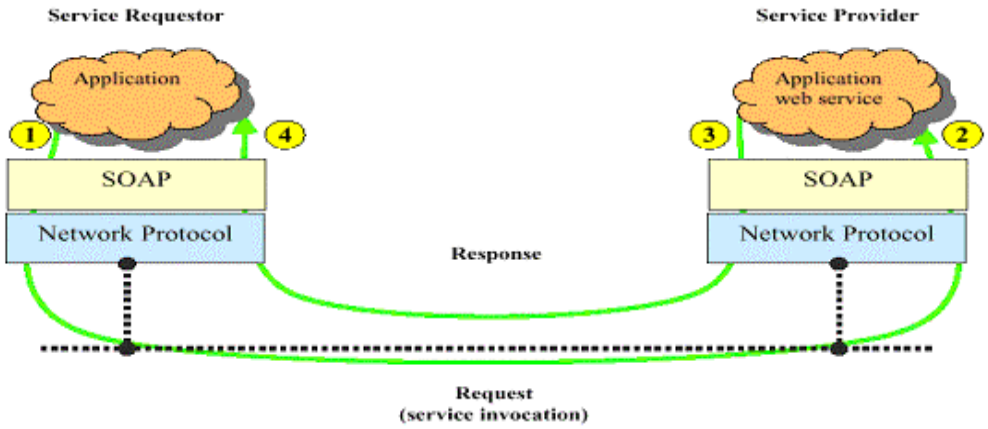


Figura 2.5 – *XML messaging usando SOAP*

L'applicazione in esecuzione sulla piattaforma del requestor integra un Web Services fornito dal service provider, l'esecuzione si svolge attraverso quattro step principali (numerate in figura 2.4):

1. L'applicazione del service requestor confezione un messaggio SOAP nel quale specifica il tipo di richiesta, ad esempio che si tratta di una richiesta SOAP RPC. Questo messaggio, insieme con l'indirizzo di rete del service provider, viene presentato all'infrastruttura SOAP sottostante (ad esempio un SOAP client runtime), che a sua volta interagisce con il protocollo di trasporto sottostante, HTTP, che immette il messaggio SOAP sulla rete.
2. L'infrastruttura di rete consegna il messaggio al server SOAP, in esecuzione sulla piattaforma del service provider, che estrae il documento XML, deserializza i tipi secondo il linguaggio di programmazione con cui è implementato il servizio remoto, e in base ad ulteriori informazioni, instrada la richiesta al servizio (per i dettagli di queste operazioni si rimanda al capitolo III).
3. Il Web Service processa la richiesta e formula una risposta che viene trasformata in un messaggio SOAP che il server SOAP consegna al protocollo di trasporto. Quest'ultimo provvede a consegnare il messaggio al mittente.
4. Il messaggio è ricevuto dal SOAP client runtime del service provider che lo trasforma in un oggetto del linguaggio di programmazione dell'applicazione client.

2.3. Descrizione del servizio.

La descrizione del servizio è un documento che contiene i dettagli relativi all'interfaccia e all'implementazione del servizio. La descrizione include le operazioni, i parametri scambiati, le informazioni per il binding (collegamento all'implementazione del servizio) e la localizzazione remota del servizio. Essa può includere anche altre informazioni ausiliarie che rendono più veloce la ricerca dei servizi sul Web e la loro classificazione.

E' attraverso la descrizione del servizio che i service provider comunicano tutte le informazioni necessarie al service requestor per invocare il Web Service. La descrizione del servizio è la chiave per rendere l'architettura dei Web Service lascamente accoppiata, specificando in essa tutta la conoscenza necessaria per la condivisione e l'integrazione dei servizi nella piattaforma di programmazione del consumatore. La descrizione, combinata con il protocollo SOAP, realizza una un'infrastruttura indipendente dalle piattaforme del service requestor e del service provider.

2.3.1. Descrizione di base.

L'architettura Web Service proposta da Microsoft e IBM usa WSDL per la descrizione base del servizio. WSDL è stato sottomesso alla standardizzazione del W3C ed è un documento XML per descrivere un Web Service attraverso un insieme di elementi astratti codificati mediante XML schema. Le operazioni e i messaggi sono descritti in

maniera astratta e successivamente delimitati da un concreto protocollo di rete come HTTP e di un protocollo di messaging SOAP.

L'uso di WSDL nell'architettura Web Service convenzionalmente divide la descrizione di base in due parti:

- la definizione dell'interfaccia del servizio
- la definizione dell'implementazione del servizio

Questo permette di definire separatamente e indipendentemente ciascuna parte, com'è illustrato in figura 2.6.

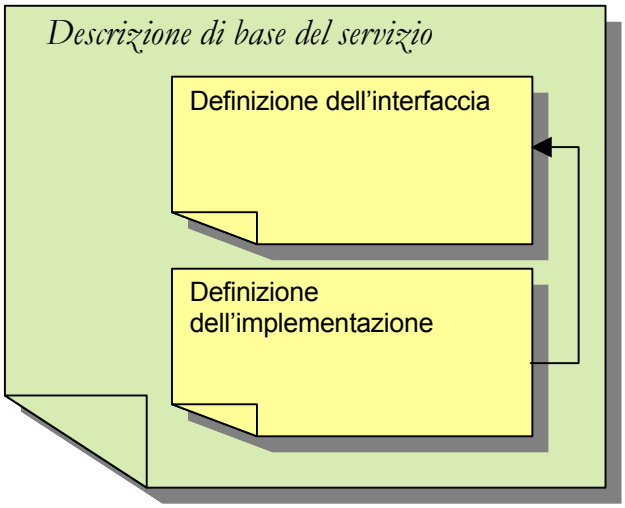


Figura 2.5 – *Descrizione di base del servizio.*

L'interfaccia del servizio contiene tutte le informazioni necessarie per implementare il servizio. Rappresenta una definizione astratta del servizio Web, ed è usata per descrivere uno specifico tipo di servizio.

La definizione dell'implementazione, invece, contiene una descrizione di un servizio che implementa l'interfaccia del servizio Web. Conseguentemente, il documento che definisce l'implementazione

deve referenziare quello che contiene la definizione dell'interfaccia del servizio.

Il documento di definizione dell'interfaccia è sviluppato dal service interface provider. Invece, il documento dell'implementazione del servizio è creato e pubblicato dal service provider. I ruoli di questi due attori sono logicamente separati, ma nella pratica possono essere la stessa entità, ad esempio la stessa azienda.

Nel terzo capitolo sarà presentata la tecnologia WSDL che consente di sviluppare sia la definizione dell'interfaccia sia quella dell'implementazione del servizio.

2.3.2. Descrizione completa.

La descrizione completa del servizio Web copre tutte le informazioni che sono necessarie per pubblicare il servizio in un registro UDDI. Le informazioni sul servizio come il nome, il tipo di specifica per la negoziazione e gli endpoint relativi alle varie implementazioni, possono essere ricavate direttamente dal documento della descrizione di base. Invece, i dati relativi al fornitore del servizio come il nome, le informazioni che ne consentono il contatto (indirizzo, telefono, email, etc.), il settore in cui opera, il tipo di servizio offerto, e altre descrizioni, devono essere immesse mediante l'interfaccia utente del registro.

Nel capitolo 5 verrà approfondita la tecnologia UDDI e presentato un meccanismo che consente di *mappare* alcune porzioni della descrizione di base del servizio nelle strutture dati del registro UDDI. Sempre nello stesso capitolo sarà mostrato un espediente per associare al

servizio pubblicato sul registro il suo documento di base, indispensabile per essere consumato.

2.4. Discovery.

Una volta che sappiamo come invocare un Web Service e come descriverlo, ci occorre un meccanismo per scoprirlo. Analogamente ad un motore di ricerca, il registro UDDI consente ad un fornitore di rendere pubblica l'esistenza del suo servizio, registrandolo in una directory e ad un consumatore di cercare il servizio di cui ha bisogno. Per facilitare la ricerca, il servizio di directory mette a disposizione delle aziende diverse classificazioni, come NAICS (North American Industry Classification) per le società fornitrici, UNSPSC (Universal Standard Products and Services Classification) per i prodotti e servizi., ed altre.

Una volta che il servizio è stato trovato, la fase successiva consiste nell'ispezione dei dettagli pubblicati per cercare di localizzare la descrizione di base del servizio.

Nella figura 2.6 è mostrato il blocco relativo alla scoperta del servizio Web.

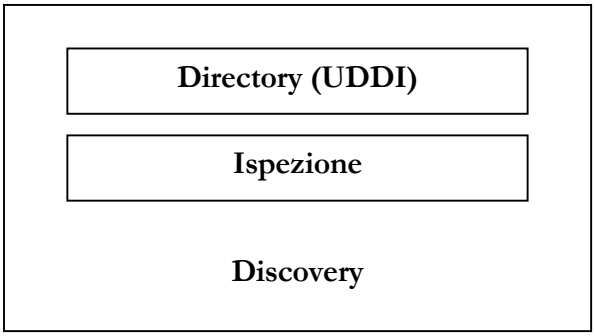


Figura 2.6 – *Blocco relativo alla ricerca del servizio.*

Un service requestor può cercare i servizi Web durante due differenti fasi del ciclo di vita della sua applicazione: nella fase di progettazione e a runtime. Generalmente, al tempo di progettazione si ricerca un servizio che è conforme ad una data interfaccia. Invece al tempo d'esecuzione si ricerca un servizio principalmente in base alla sua qualità.

Il più semplice esempio di scoperta è la quella statica (static discovery) in cui il service requestor recupera un documento WSDL da un file locale (ottenuto, ad esempio, da una precedente ricerca o in seguito a una pubblicazione diretta, in cui il fornitore invia direttamente la descrizione al richiedente), o da una repository locale delle descrizioni dei servizi. Nei casi più complessi, il servizio può essere scoperto dinamicamente al tempo di progetto oppure a runtime, utilizzando un tool che consente di interrogare il registro UDDI. In questo caso, il registro deve implementare un meccanismo di look-up per supportare la ricerca in base al tipo d'interfaccia, alle informazioni per agganciare il servizio (cioè, il protocollo di comunicazione), alle proprietà del servizio (parametri di qualità, QoS), allla categoria a cui appartiene il servizio o l'azienda fornitrice, etc.

3. Gli attori in WSOA.

Il requisito fondamentale dell'architettura WSOA è la facilità delle interazioni tra le implementazione dei servizi delle varie organizzazioni, che logicamente sono realizzate per essere eseguite su differenti piattaforme.

Per rispettare questa caratteristica, le interazioni tra i servizi devono essere basate su un protocollo di messanging che sia standard e accettato in maniera convenuta da tutti gli attori in scena.

3.1. Service Provider.

Il service provider è nella pratica un nodo di rete che fornisce un'interfaccia di un servizio a qualunque applicazione che ne faccia richiesta per integrarne le funzionalità offerte. Il servizio remoto può trovarsi presso la piattaforma del service provider, in tal caso il fornitore del servizio n'è anche il proprietario, o essere una risorsa software disponibile presso un'altra intranet aziendale.

Inoltre, all'interno della stessa intranet, la presenza di un service provider può consentire l'accesso ai componenti software aziendali alle varie unità organizzative, incentivando, in questo modo, il riuso del codice.

In generale, i service provider espongono le risorse software di una certa azienda affinché possano essere consumate su Internet. La natura concettuale di WSOA non impone la scelta di un particolare protocollo di trasporto e il meccanismo per l'accesso ai servizi. Comunque, per assicurare l'interoperabilità tra i servizi offerti dai diversi provider, ciascuno con la propria infrastruttura di rete, le risorse software devono essere Web Service, che come gia detto nel capitolo I, sono componenti software accessibili attraverso Internet e che possono essere usati per realizzare applicazioni distribuite su tale rete. Un Web Service può accettare le richieste per realizzare uno specifico set di task e rispondere a tali richieste usando un protocollo standard

per lo scambio dei messaggi. Inoltre, i Web Service possono essere aggregati insieme per realizzare un altri Web Service.

Il service provider espone il servizio all'ambiente runtime (ambiente che consente di invocare i servizi Web) per renderlo accessibile da una qualunque postazione remota.

Da una prospettiva aziendale identificarsi con questo ruolo WSOA, significa essere in grado di realizzare e manutenere servizi elettronici di qualità o, anche, di esporre come Web Services applicazioni esistenti, ad esempio facenti parte del proprio patrimonio aziendale, che si ritengono sufficientemente collaudate e mature.

Ad esempio, se una banca crede che i propri processi business per l'elaborazione dei prestiti sono abbastanza maturi da essere resi pubblicamente disponibili e sono pronti a supportarli come offerta commerciale, allora questa banca potrebbe rivestire il ruolo di provider di servizi per l'elaborazione di prestiti bancari.

3.2. Service Requestor.

Il service requestor può essere un'azienda che cerca servizi per realizzare una certa soluzione business, oppure un'applicazione software che invoca e avvia un'interazione con un servizio. Nel primo caso il service requestor utilizza il servizio di ricerca messo a disposizione dal service registry, e nel secondo caso le funzionalità del servizio fornito dal service provider.

Da un punto di vista aziendale, il ruolo del service requestor può essere ricoperto da due attività business:

- Content Aggregation è un'attività, dove un'entità business interagisce con un certo numero di fornitori di contenuti informativi, che poi elabora o pubblica in un formato di presentazione desiderato dal cliente. Esempi di tali entità business possono essere qualunque portale internet che integra, ad esempio, servizi di news, motori di ricerca, etc.

- Service Aggregation è un'attività dove un'entità business interagisce con fornitori di servizi per realizzare applicazioni integrate su commessa dei loro customer.

3.3. Service Registry.

Un service registry consente ai service provider di pubblicare (registrare e classificare) i propri dati personali e le informazioni di dettaglio inerenti ai servizi offerti. Inoltre, esso mette a disposizione meccanismi per cercare le entità fornitrici di servizi e di localizzare i servizi offerti.

In altre parole, il service registry è un registro di ricerca che consente ai service requestor di scoprire il servizio desiderato e, conseguentemente, di localizzarne la descrizione. Successivamente, mediante le informazioni contenute in questa, il consumatore del servizio comunica col service provider per accedere alle funzionalità offerte dal servizio.

Il service registry può trovare corrispondenza in tre tipi d'attività aziendali:

- Registry. Se un'entità business si trova già a raccogliere e catalogare dati d'altre aziende, può identificarsi in questo ruolo. Usualmente, un registro dovrebbe collezionare dati come il nome dell'azienda, descrizioni e informazioni di contatto.

- Broker. Costruito sul concetto di registro, usualmente n'estende il valore offrendo ricerca intelligente e classificazioni delle aziende o tassonomia dei dati.

- Aggregator/Gateway. Qualunque entità business che oltre alla scoperta dei servizi, fornisce ai service requestor tutte quelle risorse necessarie per effettuare il collegamento al servizio (dettagli per il binding) e interagire con questo da remoto.

4. Sviluppo di Web Service.

Come descritto in precedenza, nell'architettura orientata ai Web Services esistono tre ruoli chiave: service registry, service provider e service requestor. Ciascuno di questo ha specifici requisisti per ogni fase del ciclo di sviluppo dei servizi Web. Per quanto riguarda il service registry, questo ha un ruolo passivo, ed è sufficiente che sia operativo quando si pubblicano o cercano i servizi sul Web.

Il ciclo di sviluppo di un servizio Web è costituito dalle tre fasi principali, di seguito descritte [2].

Build.

Questa fase include le seguenti attività:

- Costruzione e testing dell'implementazione del servizio Web.
- Definizione della descrizione dell'interfaccia, o eventualmente localizzazione di una esistente.
- Definizione della descrizione dell'implementazione.

In generale, un Web Service può essere realizzato creando un nuovo servizio, trasformando uno o più componenti esistenti in un Web Service, e componendo in modo opportuno Web Services e applicazioni esistenti. Nel primo caso la fase di build comprende un normale ciclo di sviluppo del prodotto software, e quindi analisi, progettazione, implementazione e testing dell'applicazione che realizzerà il servizio Web. Nel caso della trasformazioni di applicazioni esistenti in Web Services, si deve produrre la descrizione dell'interfaccia del servizio e *wrappare* l'applicazione in modo da esporre solo le funzionalità rilevanti. Invece, la composizione di nuovi servizi Web da quelli esistenti richiede l'uso di una notazione per la descrizione della sequenza di messaggi scambiati tra i componenti interni ed esterni al contesto aziendale. Ad ogni modo, le metodologie di progettazione object-oriented (OOP) possono essere applicate allo sviluppo di servizi Web, anche se ciò non è richiesto.

Deploy

Questa fase è costituita da due sottofasi fondamentali, la prima è inerente alla pubblicazione del servizio Web in un registro pubblico o privato e consta dei seguenti task:

- Pubblicazione della descrizione dell'interfaccia del servizio.
- Pubblicazione della descrizione dell'implementazione del servizio.

La seconda invece espone l'implementazione del Web Services all'ambiente d'invocazione (ad esempio a un SOAP Engine), allocando tutte le risorse necessarie per localizzarla ed eseguirla

Run.

Quando un servizio Web è operativo (realizzato e pubblicato), un service requestor può trovare la definizione del servizio e invocane le operazioni implementate.

Il service provider ha a sua disposizione quattro differenti approcci allo sviluppo dei Web Services, riportati nella tabella 2.1 e descritti in dettaglio in seguito [2].

	La definizione dell'interfaccia esiste.	*La definizionee dell'interfaccia non esiste.*
Nuovo Web Service	**Green Field**	**Top-Down**
Componente esistente	**Bottom-Up**	**Meet-in-the-Middle**

Tabella 2.1 – *Metodi per lo sviluppo di Web Services.*

4.1. Green Field (tabula rasa).

Questo metodo descrive come costruire una nuova descrizione dell'interfaccia per un nuovo servizio Web. Per prima cosa viene realizzato il servizio Web (seguendo un normale ciclo di sviluppo), e da questo, in un secondo momento, si genera la descrizione dell'interfaccia.

Il primo step della fase *build* riguarda la progettazione, implementazione e testing del componente che rappresenta il Web Service. Questo componente deve esibire, attraverso la sua interfaccia, poche e ben definite funzionalità di alto livello (funzioni al livello della logica aziendale), che portano a termine specifici compiti. In pratica, deve essere un servizio prima di diventare un Web Service.

Il secondo step di questa fase è la creazione della definizione dell'interfaccia dall'implementazione del nuovo servizio Web.

Nella fase di *deploy* viene pubblicata la definizione dell'interfaccia in un registro, ed effettuato il deploy del codice runtime notificandone la presenza a un SOAP Engine, che si incarica di instradare le richieste ed invocare il servizio. Dopo la notifica, noto l'endpoint del servizio Web, si genera la definizione dell'implementazione che viene pubblicata nel registro.

Quando il service requestor utilizzerà il servizio Web, questo sarà eseguito nell'ambiente runtime della piattaforma del provider su cui è stato esposto.

4.2. Top-Down.

Con questo metodo è possibile sviluppare servizi Web conformi ad una specifica definizione dell'interfaccia pubblicata in un registro, ad

esempio, da un'industria che fornisce interfacce standard. Questa definizione dell'interfaccia standard può essere cercata nel registro ed implementata da diversi fornitori di servizi.

Nella fase di *build* il service provider cerca nel registro l'interfaccia del servizio, la localizza e la usa come template per implementare il servizio Web. Il template contiene tutti i metodi e parametri che devono essere realizzati e utilizzati per rendere il servizio conforme alla sua interfaccia. La realizzazione del componente che rappresenta il Web Service segue il tradizionale ciclo di sviluppo, come, per esempio, quello ad assemblaggio di componenti.

La fase di *deploy* ha gli stessi task visti per quella del metodo *green field* ad eccezione della pubblicazione della definizione dell'interfaccia, in quanto già pubblicata.

4.3. Bottom-up.

Questo metodo è utile per realizzare una nuova definizione dell'interfaccia da una o più applicazioni esistenti, implementate in Java, in C++, in C, o mediante oggetti COM (Component Object Model).

Nella fase di *build* si deve analizzare attentamente l'applicazione ed estrarre da queste solo quelle funzioni in grado di fornire un servizio. Tali funzioni, in generale, devono portare a termine uno specifico compito e possibilmente in una sola invocazione.

La fase di *deploy* consiste nella pubblicazione della definizione dell'interfaccia del servizio Web, nell'esposizione delle singole

funzionalità del servizio ad un SOAP Engine, poi si prosegue con la creazione e pubblicazione della definizione dell'implementazione.

4.4. Meet-in-the-Middle.

Come mostrato in tabella 2.1, questo metodo è usato quando esiste sia la definizione dell'interfaccia sia l'applicazione che implementa il Web Service.

Il primo passo della fase di *build* consiste nel trovare la definizione dell'interfaccia, che deve essere usata per fornire un contratto al servizio. Una volta che questa è stata trovata, la si usa per generare il template del servizio, che verrà utilizzato per progettare e implementare un wrapper (codice che incapsula un'applicazione per esporne tutte o alcune funzioni, generalmente, in un linguaggio di programmazione diverso da quello usato per implementare l'applicazione d'interesse) capace di mappare la definizione dell'interfaccia sull'interfaccia dell'applicazione.

Il *deploy* consta dell'allocazione all'ambiente di runtime di tutte le risorse necessarie per eseguire il servizio, della creazione e della pubblicazione della definizione dell'implementazione.

5. Utilizzo dei Web Services.

L'utilizzo di un servizio Web in un'applicazione richiede la presenza di un proxy, ovvero un modulo software che contiene tutti i requisiti per accedere ed invocare il servizio remoto. Questo può essere generato automaticamente dal documento della definizione dell'interfaccia o

manualmente. In entrambi i casi il proxy deve essere implementato nello stesso linguaggio usato per implementare l'applicazione che ne richiede l'uso.

Ci sono, sostanzialmente, due metodi per agganciare (binding) uno specifico servizio. Il metodo *static binding*, usato solo al tempo di sviluppo, e *dynamic binding*, usato al tempo di sviluppo o anche al tempo di esecuzione.

5.1. Binding statico.

In generale, un binding di tipo statico viene usato nella fase di sviluppo quando esiste una sola implementazione del servizio, la stessa che sarà usata a runtime dall'applicazione. Per prima cosa lo sviluppatore dovrà cercare la definizione dell'implementazione nel registro. Questa contiene un riferimento alla definizione dell'interfaccia e all'endpoint che consente l'accesso al servizio Web. Successivamente, secondo queste due informazioni viene generato un proxy. Prima di integrarlo nell'applicazione o di esporlo come servizio Web, il proxy deve essere testato al fine di verificare il corretto funzionamento nell'interazione col servizio remoto. Nella figura 2.7 sono mostrate le interazioni tra gli attori dell'architettura orientata ai Web Services necessarie per realizzare uno scenario completo di binding statico.

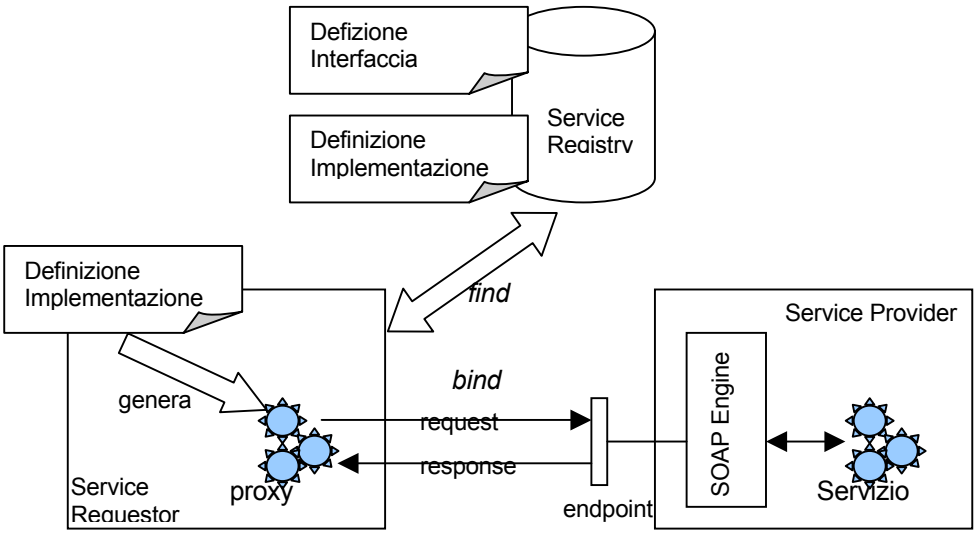


Figura 2.7 – *Binding statico.*

5.2. Binding dinamico in fase di sviluppo.

Se l'endpoint dello specifico servizio Web, che un requestor desidera usare, non è noto al tempo di sviluppo o se questo può cambiare dinamicamente a runtime, allora conviene usare il binding dinamico.

Il primo passo da compiere consiste nel cercare la definizione dell'interfaccia relativa al tipo di servizio che si vuole usare. Poiché questa contiene solo le definizioni astratte delle funzionalità offerte dal servizio, da essa si può generare un proxy generico che consente di accedere ad ogni implementazione del servizio Web. Infatti, il proxy generico, a differenza del proxy generato nel binding statico, non incorpora l'informazione inerante ad una specifica implementazione del servizio Web. Ovviamente, il proxy generico deve contenere codice aggiuntivo per realizzare un meccanismo che consenta a runtime di cercare nel registro una definizione dell'implementazione del servizio e, dopo averla localizzata e scaricata, di invocarla.

5.3. Binding dinamico a runtime.

In questo caso la definizione dell'interfaccia del servizio è trovata e scaricata dinamicamente a runtime, così come la definizione dell'implementazione [2]. Per realizzare questo meccanismo, l'applicazione del service provider deve integrare un proxy evoluto la cui interfaccia a livello di programmazione combacia con quella del servizio desiderato. A runtime il proxy cerca nel registro le descrizioni delle interfacce dei servizi in base a particolari parametri di ricerca, ad esempio, localizzando prima l'azienda o il tipo di settore in cui essa opera, e successivamente trovare i servizi conformi all'interfaccia desiderata. Se la ricerca va a buon fine, genera, compila e invoca un proxy generico di supporto capace di agganciare dinamicamente una specifica implementazione (come nel caso del binding dinamico a tempo doi sviluppo). Per trovare la definizione dell'implementazione invece si deve cercare tra i *binding template* (specifiche tecniche dei servizi, vedi capitolo 5) del servizio.

Questo tipo di binding dovrebbe essere implementato utilizzando un'interfaccia utente che guidi l'utente nella ricerca della definizione dell'interfaccia del servizio, un'interazione automatica machine-to-machine tra il service requestor e il service registry potrebbe essere molto complessa se non impossibile da realizzare.

Capitolo III

SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

1. Introduzione.

SOAP è un protocollo leggero, estendibile, e XML-based che consente la comunicazione, sul Web, tra diverse entità software, in generale oggetti, ciascuna delle quali non possiede alcuna precognizione delle piattaforme delle altre e del linguaggio con cui queste sono state implementate.

SOAP, fondamentalmente, è un meccanismo per scambiare informazioni testuali e strutturate in ambiente Internet. Le informazioni sono codificate nel formato XML, mediante specifiche regole di codifica ed elaborazione. Attualmente, la trasmissione di messaggi SOAP è gestita al livello del trasporto dal protocollo HTTP, servito da un Web Server. In questo senso, SOAP è un protocollo leggero, di fatti si richiede che un'eventuale sua implementazione sia capace di inviare e ricevere pacchetti HTTP e analizzare ed elaborare dati in formato XML. D'altronde, HTTP è una tecnologia che oramai si trova ovunque, non si riesce ad immaginare azienda che non gestisca questo protocollo, o che non interagisca con Internet in maniera significativa. Tuttavia, l'uso di HTTP come protocollo di trasporto,

impone, in fase di progettazione dei componenti distribuiti, il riferimento al modello Internet Distribution Model. In questo modello, gli oggetti sono senza stato (stateless), cioè il loro stato è perso tra una chiamata e la successiva. Per questo motivo, qualsiasi stato significativo deve essere memorizzato in maniera permanente, per esempio mediante il supporto di un database o l'uso di un cookie. Quest'accorgimento consente a chiamate successive di riutilizzare lo stato precedente.

La scelta di HTTP come protocollo di trasporto consente alle applicazioni, che usano il protocollo SOAP RPC per le chiamate ai metodi dei componenti remoti, di accedere ad Internet attraverso i firewall, giacché la porta usata da HTTP è lasciata generalmente aperta (una tipica configurazione aziendale limita il traffico alla porta 80). Diversamente accade, per esempio, alla porta 135 (RPC EndPoint Mapper) che è tipicamente chiusa per ragioni di sicurezza. Conseguentemente, qualunque tecnologia ad oggetti distribuiti basata su RPC, che utilizza la suddetta porta per comunicare, fallisce miseramente se tenta di accedere ad oggetti oltre il firewall.

Sebbene, attualmente, la quasi totalità delle implementazioni di SOAP usano HTTP, la versione 1.1 della specifica SOAP non impone questa scelta, estendendo la possibilità di utilizzare anche SMTP e FTP. Tuttavia, le combinazioni tra questi ultimi e SOAP non vengono descritte nel testo della specifica.

Infine, il protocollo SOAP è estendibile (proprietà questa di tutte le tecnologie costruite su XML), nel senso che qualora esso dovesse

evolvere, le applicazioni esistenti, che ne fanno uso, continueranno a funzionare correttamente.

2. Obiettivi e vantaggi.

"L'obiettivo principale di SOAP è la semplicità e l'estendibilità" [9]. Per il raggiungimento di questo scopo, la specifica SOAP (versione 1.1) evita completamente di affrontare aspetti comuni alle architetture ad oggetti distribuiti. D'altronde, SOAP non si vuole proporre come un'architettura distribuita, ma semplicemente come un wire protocol, della quale n'è solo un componente.

Tali aspetti sono:

- Distributed Garbage Collection (gestione degli oggetti orfani).
- Comunicazioni HTTP bidirezionali (gestione dei callback).
- Object-by-reference (che richiede la Distributed Garbage Collection).
- Remote Object Activation (modulo d'attivazione degli oggetti remoti)

Molti di questi aspetti dovrebbero essere risolti all'atto dell'implementazione di un'architettura distribuita con SOAP come wire protocol, e perciò non riguardano la specifica SOAP. Ad esempio, per gestire oggetti attivati e non più utilizzati dal client, si potrebbe prendere in considerazione un approccio di tipo timeout. Se il client

non esegue almeno una chiamata entro un certo intervallo di tempo prefissato, l'istanza dell'oggetto viene distrutta.

Di seguito riporto i vantaggi che un'applicazione distribuita può trarne dall'uso del protocollo SOAP:

- SOAP è costruito su tecnologie aperte (indipendenti da produttori), e facilita l'interoperabilità distribuita. Nessun singolo produttore domina il mercato SOAP, almeno fino a questo momento.
- Il protocollo SOAP, quando realizzato sul trasporto HTTP, consente alle applicazioni distribuite di passare attraverso il firewall aziendale, senza minarne la sicurezza.
- SOAP consente di realizzare applicazioni distribuite loosely-coupled.
- I cambiamenti apportati all'infrastruttura SOAP non dovrebbero coinvolgere le applicazioni che usano il protocollo (sempre che non si tratti di cambiamenti radicali apportati ad aspetti fondamentali come, per esempio, la serializzazione e deserializzazione).

3. Uno sguardo a XML.

A questo punto è d'obbligo una premessa: poiché SOAP (così come le altre tecnologie utilizzate in questo studio, WSDL e UDDI) si appoggia essenzialmente sui concetti della tecnologia XML, per

comprenderne le sue capacità si dovrebbe possedere una buona conoscenza di XML. Fornire una guida completa e concisa alla sintassi XML va oltre lo scopo di questo lavoro di studio. Tuttavia, nei prossimi paragrafi presenterò alcuni concetti base di questa tecnologia che sono ampiamente utilizzati da SOAP.

XML è l'acronimo di eXtensible Markup Language ed è un linguaggio di markup completamente estendibile. Attraverso il linguaggio di markup il programmatore può definire dei tag per descrivere gli elementi del suo documento strutturandolo in base alle informazioni che dovrà contenere [10]. I tag del documento definiscono i marcatori che verranno considerati significativi durante l'analisi (parsing) dello stesso e in corrispondenza dei quali si trovano le informazioni da elaborare. Di seguito riporto un esempio di definizione dell'informazione usando un linguaggio di markup ben noto, HTML:

```
<HTML>
    <HEAD>
      <TITLE>
          HTML e il linguaggio di markup
      </TITLE>
    </HEAD>
    <BODY>
     HTML è un linguaggio di markup
    </BODY>
</HTML>
```

Listato 3.1 – *Un esempio di linguaggio di markup.*

Come si osserva dal listato, ogni elemento, componente base del linguaggio di markup, ha un nome e un contenuto. Il contenuto è

delimitato dal tag iniziale che rappresenta il nome dell'elemento fra parentesi angolari, e dal tag finale a cui si aggiunge un carattere (barra obbliga) prima del nome. Ad esempio, l'elemento TITLE è marcato dal tag <TITLE> e il suo contenuto è il titolo vero e proprio del documento HTML.

3.1. Nessun elemento predefinito.

XML, così come HTML, deriva dalla semplificazione dello standard SGML, che sta per Standard Generalized Markup Language, utilizzato per descrivere la struttura dei documenti. XML non è solo un linguaggio di markup, esso è precisamente un linguaggio di markup completamente estendibile, questo significa che il programmatore può definire un numero illimitato di tag per definire gli elementi che descrivono le informazioni di suo interesse. In altre parole XML è un metalinguaggio, un linguaggio che può essere usato per definire altri linguaggi sempre basati sull'uso dei tag (HTML compreso). In tal senso, il programmatore potrà usare XML per definire il linguaggio che più si adegua alla descrizione del contenuto informativo del suo documento.

Diversamente da XML, HTML possiede un set di tag precostituiti e limitato che sono stati definiti insieme al linguaggio stesso, tant'è che per estendere tale insieme di tag è necessario passare dal processo di standardizzazione del W3C. Le eventuali estensioni di HTML non soggette a raccomandazione rimangono caratteristiche proprietarie del browser che le supporta, generando in questo modo i noti problemi d'incompatibilità.

3.2. Il documento XML.

Nel seguente esempio utilizzo il linguaggio XML per descrivere una semplice rubrica elettronica:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE auto SYSTEM "auto.dtd">
<!--Fine intestazione-->
<!--Inizio contenuto-->
<address-book>
    <entry>
      <identita>
        <nome>Mario</nome>
        <cognome>Rossi</cognome>
        <codicefiscale>RSSMRR1A6M6OF634</codicefiscale>
      </identita>
      <indirizzo>
        <via>via Tal dei Tali</via>
        <localita>Napoli</localita>
        <provincia>NA</provincia>
        <cap>80125</cap>
        <nazione>Italy</nazione>
      </indirizzo>
      <tel>081-000000</tel>
      <tel preferred="false">081-888888</tel>
      <email href="mailto:avale@domain.it"/>
   </entry>
</address-book>
<!—Fine contenuto-->
```

Listato 3.2 – *Un esempio di documento XML che descrive una semplice agenda elettronica.*

Chi possiede una certa familiarità con HTML troverà questo documento abbastanza strano; infatti, noterà che i tag e la grammatica ad essi associata sono completamente personalizzati. Il tag <entry>, usato in un documento HTML, sarà ignorato dalla maggior parte dei

browser poiché non è riconosciuto come un tag al quale è possibile associare un ruolo specifico. Dall'analisi del file presentato nel listato x.2, si osserva che l'estendibilità di XML non è riferita solo alla definizione dei tag, ma anche alla grammatica che ne descrive l'uso corretto. Consideriamo ancora il linguaggio HTML, in esso il tag precostituito <TABLE>, che consente di costruire una tabella, possiede un insieme di attributi per impostare le proprietà della tabella, quali il colore dello sfondo, l'ampiezza, etc., ma la grammatica assegnata a questo tag non consente di impostare, ad esempio, l'attributo Type, in quanto non incluso nell'insieme precostituito di attributi accettato per il tag <TABLE>. Mediante XML l'insieme degli attributi per ciascun tag può essere deciso dal programmatore, così come i valori di default e i vincoli cui tali attributi sono soggetti, ovviamente nel rispetto della struttura e delle proprietà globali richieste dal linguaggio.

In definitiva, la flessibilità con cui XML permette di definire il contenuto, e non l'aspetto, dei dati è il presupposto alla portabilità degli stessi.

Ovviamente il fatto che il programmatore possa specificare qualsivoglia tag può creare dei problemi di comprensibilità da parte delle applicazioni XML. Per questo motivo lo standard XML suggerisce di affiancare al documento che definisce il contenuto delle informazioni, dati da pubblicare o da elaborare, un altro documento che sia in grado di definire il "linguaggio" utilizzato dal programmatore per descrivere codeste informazioni (ricordo che XML è un

metalinguaggio), tale documento si chiama DTD e sarà trattato in maniera più dettagliata nel corso di questo studio.

3.3. Struttura del documento XML.

3.3.1. Intestazione.

Un documento XML inizia con un'*intestazione* (header) che procura al parser e alle applicazioni XML le informazioni necessarie per il trattamento del documento stesso. Il concetto di *header* non è una definizione formale della specifica XML, ma è comunemente usato. Ad esempio, prendiamo in considerazione il listato 3.2, le prime due righe fanno parte dell'intestazione del documento, in particolare:

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

è un'istruzione XML destinata esclusivamente al parser al quale fornisce la versione XML utilizzata, e il sistema di codifica per i caratteri, standard Unicode a 8 bit. L'header fornisce anche altre informazioni per la cui trattazione si rimanda alla specifica.

3.3.2. Istruzioni, riferimento a DTD e commenti.

Tutte le istruzioni iniziano col carattere ? e hanno la seguente forma:

<?target istruzione?>.

Esse possono essere suddivise in due insiemi:

- L'insieme delle istruzioni di elaborazione PI (Processing Instruction) che vengono passate dal parser all'applicazione che elabora il documento XML.

- L'insieme delle istruzioni con target "xml" che invece vengono processate dal parser.

Dopo l'istruzione xml iniziale, spesso, compare una dichiarazione del seguente tipo:

```
<!DOCTYPE address-book SYSTEM "address-book.dtd">
```

che possiede una propria sintassi e il cui scopo è quello di informare al parser che il documento XML è supportato dal documento DTD (Document Type Definition, è un documento, non di formato XML, che contiene regole per valicare i documenti XML, per maggiori chiarimenti si veda [10]) rappresentato dal file *address-book.dtd*, situato nel file system locale (SYSTEM) e che inoltre l'elemento radice del documento XML (l'elemento posto al livello più alto in tale documento) è *address-book*.

La versione 1.1 di SOAP non consente l'uso di DTD e di Processing Instruction.

La terza riga del listato riporta un esempio di commento:

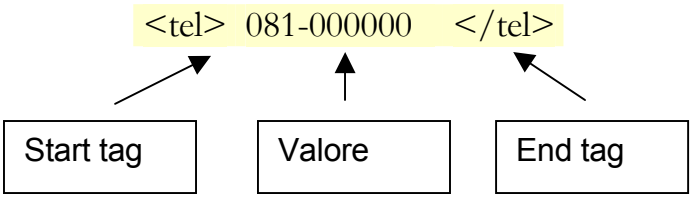
<!--Fine intestazione-->

tutto ciò che si trova tra i marcatori <!-- e --> viene considerato dal parser come commento.

3.3.3. Elementi e attributi.

La seconda parte del documento è costituito dal contenuto vero e proprio, rappresentato da costrutti in cui sono presenti gli elementi, gli attributi e i dati di testo visti come valori o contenuti degli elementi.

Un *elemento* consiste di un tag iniziale (start tag), di un valore e di un tag finale (end tag), come mostrato di seguito:



Un elemento genitore può contenere un numero infinito di elementi figli e ciascun figlio può apparire molteplice volte sotto l'elemento genitore.

Il primo elemento di ciascun documento XML, definito *elemento radice*, rappresenta il primo e unico tag di apertura e l'ultimo tag di chiusura del documento e quindi deve includere tutti gli altri elementi presenti nel documento.

Un *attributo* aggiunge informazioni più specifiche a un particolare elemento. In generale, esso viene usato per denotare una proprietà intrinseca di un elemento. Ad esempio, href dell'elemento email è una proprietà che si presenta sempre ad ogni occorrenza di questo, e quindi è stata 'promossa' ad attributo dell'elemento stesso.

3.3.4. Riferimenti ad entità e sezioni carattere (CDATA).

XML distingue tra due diversi tipi di dati:

- Parsed character data, cioè caratteri che vengono analizzati dal parser.

- Character data, sono dati che il parser non analizza, ma si limita a consegnarli all'applicazione che elabora il documento.

Tutti i caratteri di markup, come '<', '>', '"', etc., sono parsed character data e perciò non possono trovarsi nei contenuti degli elementi e degli attributi. Ad esempio, il seguente frammento è sintatticamente scorretto:

```
<via>"Tal dei Tali"</via>
```

Per istruire il parser a trattare i parsed character data come character data, è necessario utilizzare i riferimenti ad entità, che sono utilizzati per le operazioni di escape sui caratteri. Ad esempio, in luogo di '<' si deve utilizzare <, in cui &…; è il riferimento e lt invece è l'entità. Nella tabella 4.1 vengono riportati i caratteri di escape.

Caratteri di markup	Riferimenti ad entità
"…"	"…;
<	&lt;
>	&gt;
&	&amp
'	&apos ;

Tabella 4.1 – *Tabella di corrispondenza tra i caratteri di markup e i riferimenti ad entità*

La definizione di un'entità avviene nel seguente modo:

```
<!ENTITY copy "&#169;">
```

che definisce l'entità copy che rappresenta il simbolo del copyright nel set di caratteri Unicode;

CDATA è un'altra forma di markup che istruisce il parser ad interpretare un intero testo come una sequenza di character data, ad esempio:

<via><![CDATA[“via Tal dei Tali”]]></via>

è sintatticamente corretto e il contenuto dell'elemento *via* è racchiuso tra i doppi apici.

Le sezioni CDATA vengono usate per isolare quei testi in cui i caratteri di markup abbondano. Conseguentemente, una sezione CDATA può essere usata per includere nel documento un intero frammento XML.

Il linguaggio XML ha una sintassi rigorosa, per questo motivo il programmatore è obbligato a scrivere documenti XML corretti o come si dice ben-formati (well-formed). Un documento XML è ben formato se e solo se è conforme alle specifiche generali dello standard XML, in particolare devono essere rispettate le seguenti regole generali:

- Deve esistere un solo elemento radice.
- Ogni tag aperto deve essere chiuso (XML non ha tag predefiniti)
- L'annidamento dei tag deve essere effettuato in modo corretto come indicata dalla specifica. Ad esempio, non è possibile sovrapporre i tag, il seguente frammento:
- nome><cognome>Mario</nome>Rossi</cognome>
- non è valido.

- I valori degli attributi devono essere racchiusi tra " ".
- I caratteri usati per marcare non possono essere usati all'interno dei dati di testo in cui devono essere rimpiazzati da caratteri speciali, ad esempio al posto di < bisogna usare &lt.
- Un documento XML viene elaborato da un'applicazione, e in primo luogo dal parser, solo se risulta essere ben-formato, perciò è importante che il programmatore conosca bene la sintassi XML.
- Un'altra proprietà opzionale del documento XML, peraltro già accennata in precedenza, è la sua validità. Un documento XML è valido se conforme al suo DTD, in altre parole, se scritto in modo conforme alle regole specificate nel DTD allegato (per i dettagli sul DTD si veda [10]).

3.3.5. URI, URL, URN e Namespace.

URI (Uniform Resource Identifier) [13] è un termine generico usato per denotare univocamente una risorsa sul Web, mediante l'uso di una stringa. Differenti tipi di risorse sul Web richiedono diverse forme di URI. Gli URL (Uniform Resource Locator) [14] e URN (Uniform Resource Name) [15] sono entrambi forme di URI ciascuna con la propria sintassi e semantica. Nella figura 3.1 è riportata la loro relazione insiemistica.

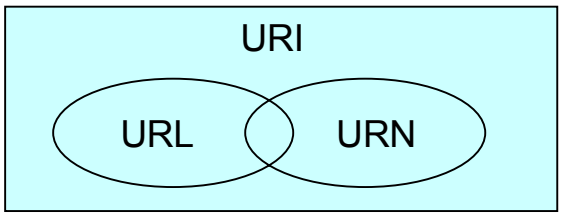


Figura 3.1 – *Relazione tra URI, URL e URN*

L'URL è usato per localizzare univocamente una particolare risorsa sul Web, attraverso un identificatore astratto (un indirizzo unico). In generale, gli URL hanno la seguente forma:

<schema>:<schema-specific-part>

in cui <schema> rappresenta il nome di un particolare schema (ad esempio http, che denota il protocollo di trasporto), ed è una sequenza di caratteri scelti tra "a"…"z", '+', '.', e '-'. Generalmente, i programmi che usano stringhe URL trattano i caratteri minuscoli e maiuscoli in modo equivalente. L'elemento <schema-specifica-part>, invece, dipende dal tipo di schema usato. Ad esempio, se lo schema è http, utilizzato per indirizzare univocamente risorse su Internet, si ha la seguente forma:

http:<host>:<port>/<path>?<searchpart>

dove:

http: è il nome dello schema.

<host>: è un qualificatore completo del nome del dominio della rete host, o il suo indirizzo IP, ad esempio:

www.webservices.org oppure 194.201.48.31

- <port>: è il numero di porta della macchina host per la connessione (per default è la porta 80)
- <path>: è un percorso http
- <searchpart>: è una stringa d'interrogazione del tipo nome="valore".

Un esempio di URL sullo schema HTTP:

http://www.google.it/search?q=axis&hl=it&start=10

L'URN fornisce un identificatore globalmente unico e persistente per il riconoscimento e l'accesso a caratteristiche della risorsa o alla risorsa stessa. In altre parole, l'URN è il nome univoco e non modificabile (persistente) che identifica globalmente una risorsa sul Web. In particolare, gli URN non sono indirizzi e, perciò, non viene generato nessun errore se la risorsa che loro identificano viene spostata. Un esempio di URL potrebbe essere l'identificazione ISBN di un libro.

La forma di un URL è la seguente:

urn:<Nomespace Identifier>:<Nomespace Specific String>

che nel caso dell'identificazione di un libro può essere del tipo:

urn:ISBN:0-7897-2242-9

Un XML namespace [12] è rappresentato da un URI e identifica univocamente i nomi degli elementi e degli attributi, usati all'interno di un documento XML, al fine di evitare eventuali situazioni contraddittorie. L'uso dei namespace è diretta conseguenza della proprietà di estendibilità di XML.

Per dichiarare un namespace si utilizza la seguente forma:

xmlns:<Namespace Prefix>=<URI>

Il prefisso <Namespace Prefix> viene utilizzato per associare elementi e attributi con l'URI specificato. In realtà, quest'ultimo non deve necessariamente indirizzare qualcosa, ma il suo uso ha solo lo scopo di rendere unico il namespace.

Un esempio di utilizzo di namespace è mostrato nel listato 3.3, in cui viene mostrata una chiamata SOAP al metodo print di un oggetto Calc.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV=”http://schemas.xmlsoap.org/soap/envelope”> SOAP-
ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding”>
  <SOAP-ENV:Body>
       <m:Print xmlns:m=”http://www.mycalc.com/calc”>
            <Message>Hello World !</Message>
       </m:Print>
  </SAOP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listato 3.3 – *Esempio d'uso di namespace in un documento XML*

3.4. Parser XML.

Le applicazione che analizzano il documento XML e restituiscono i contenuti degli elementi, degli attributi e delle istruzioni d'elaborazione vengono definite processori o parser XML.

I parser XML vengono classificati in convalidanti e non convalidanti. I parser validanti, oltre al controllo sintattico, controllano la correttezza semantica del documento XML rispetto alle regole definite nel proprio DTD o nello XML Schema. I parser non valicanti si limitano a controllare se un documento XML è ben-formato, cioè se rispetta la sintassi generale del metalinguaggio XML.

Attualmente, esistono due note API (Application Programming Interface) disponibili per il parsing dei documenti XML:

- Simple API per XML (SAX) [17]
- Document Object Model (DOM) [18].

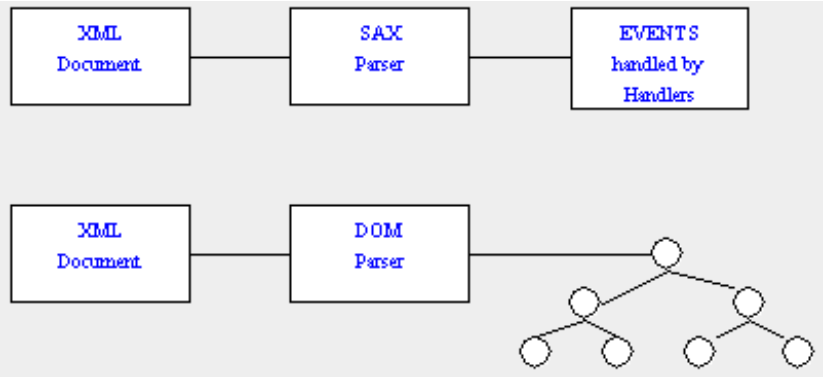


Figura 3.2 – *Parser XML.*

3.4.1. SAX.

Sax è un parser event-driven che opera in maniera simile ad una GUI (Graphical User Interface). Durante l'analisi del documento, in corrispondenza di uno start tag, è generato un evento che esegue una funzione callback, o event handler, che il programmatore ha predisposto per gestire l'evento.

Tutti i metodi di callback si trovano in una classe handler che si mette in ascolto (listener) d'eventuali eventi 'triggerati'. Nell'handler è consentito l'accostamento di codice dell'applicazione utente agli eventi che vengono generati dal parser durante l'analisi del documento. Dunque, nei metodi di callback l'utente può implementare una propria struttura dati in cui memorizzare il contenuto del documento. Occorre considerare che tutti gli eventi vengono generati durante il parsing e non termine di questo. Questa caratteristica sequenziale dell'analisi

svincola l'interfaccia SAX a caricare in memoria l'intero documento XML [10].

Nel listato 3.4 è mostrata una classe Java che implementa l'interfaccia SAX ContentHandler (versione 2.0) [17] per la gestione degli eventi. Il callback startDocument() è associato all'evento che si genera quando si incontra l'inizio del documento XML (una sola volta).

```
class MyContentHandler implements ContentHandler{
  private Locator locator=null;

public void setDocumentLocator(Locator locator){
// codice utente }
public void startDocument() throws SAXException{
// codice utente}
public void endDocument() throws SAXException{
// codice utente}
public void processingInstruction(String target, String data) throws
SAXException {
//target: è l'applicazione a cui è rivolta la PI (istruzione d'elaborazione)
//data: generalmente sono coppie attributo=valore
}
public void startPrefixMapping(String prefix, String url){
//evento generato all'inizio di un prefisso di namespace.
}
```

```
public void end PrefixMapping(String prefix){
//evento che si genera alla fine del prefisso di namespace.
}
public void startElement(String namespaceURI, Strong localName, String
rawName, Attributes atts) throws SAXException {
// evento generato quando si incontra uno start tag
}
public void endElement(String namespaceURI, Strong localName, String
rawName) throws SAXException {
// evento generato quando si incontra uno end tag
}
public void characters(char[] ch, int start, int length) throws SAXException{
// riporta i dati carattere di un elemento
public void ignorableWhitespace(char[] ch, int start, int length) throws
SAXException{
// riporta gli spazi bianchi che possono essere ignorati nel documento.
}
public void skippedEntity(String name) throws SAXException(){
//riporta le entità saltate durante il parsing senza convalida.
}
}
```

Listato 3.4 – *Esempio di handler per la gestione degli eventi.*

3.4.2. DOM

Il parser DOM analizza il documento XML e restituisce un albero d'oggetti che contiene tutte le entità di questo documento. Tale albero è mostrato in figura 3.2.

Un documento XML consiste di un oggetto *documento* che contiene un singolo oggetto *nodo*. Quest'oggetto *nodo* usa una lista di nodi (node list) per referenziare uno o più nodi figli (child node). Ciascun elemento XML può essere considerato un nodo. In particolare, l'elemento radice è il nodo figlio della radice dell'albero. Il nodo è il

tipo fondamentale (interfaccia principale) di tutti gli oggetti del modello.

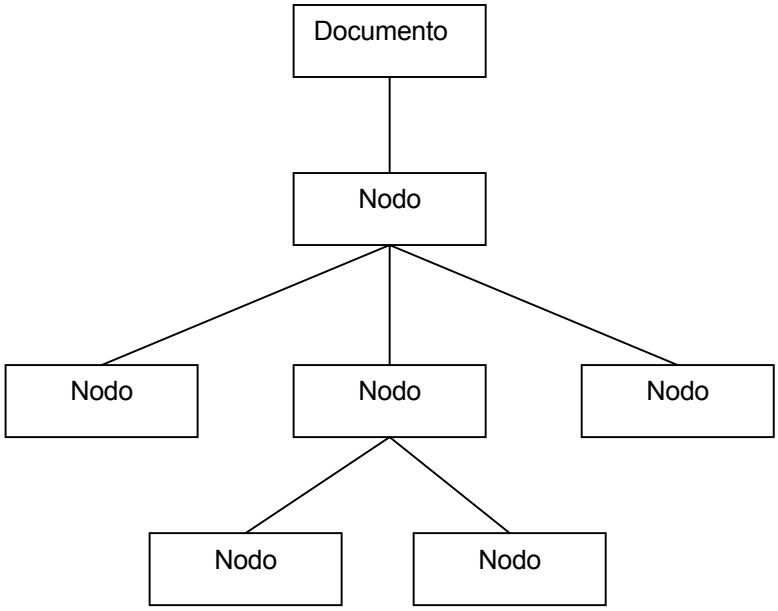


Figura 3.2 – *Modello ad oggetti di un documento.*

Le specifiche DOM sono organizzate in livelli [16]. DOM level 1 è una raccomandazione accettata del W3C [18], e definisce le interfacce principali per la manipolazione (navigazione) di un documento (HTML e XML). DOM level 2, al momento, è allo stato di Candidate Recommendation W3C e apporta aggiunte al livello 1, fornendo moduli per modelli di contenuti specifici, come XML, HTML, e CSS (Cascading Style Sheets) [18]. Nella figura 3.3 sono mostrate le interfacce in UML di DOM livello 2.

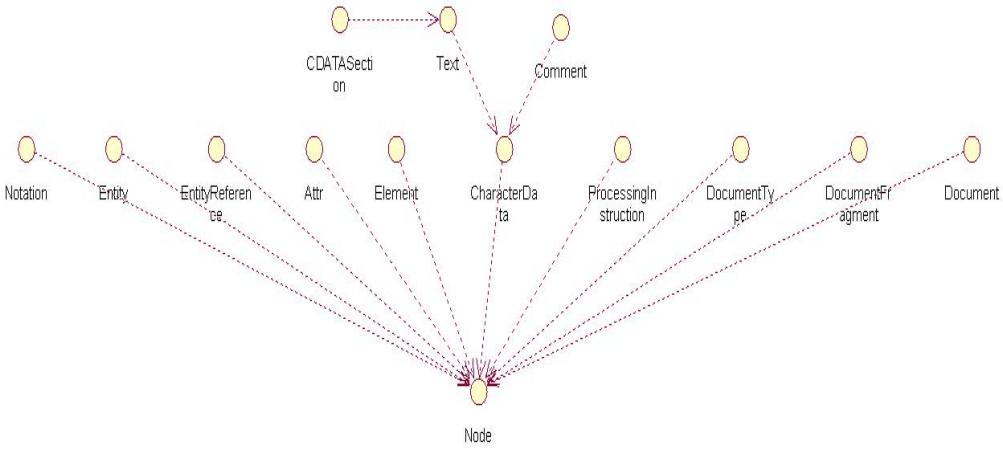


Figura 3.3 – *Interfacce DOM level 2.*

L'albero degli oggetti si trova in memoria, anche dopo che il parser ha completato l'analisi, e perciò è possibile elaborare più volte la stessa struttura.

3.4.3. Differenza tra SAX e DOM.

La principale differenza tra i due modelli consiste nell'ammontare di memoria richiesta da ciascuno per effettuare il parsing. DOM, per l'elaborazione, richiede che l'intero documento sia caricato in memoria, invece in SAX l'elaborazione è di tipo sequenziale e avviene nel corso del parsing. Per questo motivo, per documenti estremamente grandi in dimensione conviene usare SAX per evitare ripercussioni sulla perforance delle applicazioni. Invece, qualora si richieda un'attività d'analisi e d'elaborazione più efficiente e intensiva (ad esempio si ha l'esigenza di navigare più volte il documento), su

documenti di grandezza contenuta, conviene preferire per un parser DOM.

4. XML Schema.

Lo Schema XML è un linguaggio di definizione XML usato per vincolare la struttura dei documenti XML. Esso 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 [11]). Ad esempio, si può utilizzare lo schema per decidere l'ordine che gli elementi devono seguire nel documento XML, oppure il tipo di dato d'ogni elemento, consentendo al parser di verificarne la giusta corrispondenza. In altre parole, uno schema di un documento XML contiene metadati che descrivono i dati contenuti nel documento. Il DTD (Document Type Definition) è un altro documento di regole formali, la cui sintassi, però, non segue quella del linguaggio XML, che consentono di definire il linguaggio utilizzato per descrivere il contenuto informativo del documento ed indica al parser il modo in cui tale contenuto deve essere analizzato e convalidato. Sebbene, il DTD è utilizzato in molte applicazioni XML, esso non possiede le caratteristiche necessarie per definire tipi di dati complessi e per e applicare costrutto come l'ereditarietà. Per sopperire a queste limitazioni, da una bozza della Microsoft è nato XML Schema.

La specifica XML Schema è suddivisa in due parti [12]:

- 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.

Nel listato 3.5 è riportato un esempio di documento XML Schema.

Da questo si evince che un documento XML Schema, o più semplicemente schema, inizia sempre con l'elemento <schema> e contiene, in generale, un certo numero di componenti, come complextType, element, etc. Inoltre, ogni elemento nello schema è preceduto dal prefisso *xsd:* che è associato con il namespace xmlns:xsd="http://www.w3.org/2001/XMLSchema ", che appare nell'elemento <schema>. In luogo di *xsd:* può essere usato un qualsiasi altro prefisso, o nessun prefisso.

```
<?xml version="1.0"?>
<xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<element name="Libro" type="tipoLibro"/>
<complexTipe name="tipoLibro">
    <element name="Titolo" type="xsd:string"/>
    <element name="Contenuti" type="tipoContenuti"/>
    <element name="Copyright" type="xsd:string"/>
</complexType>
<complexType name="tipoContenuto">
   <element name="Capitolo" maxOccurs="*">
    <complexType>
        <element name="Testata" type="xsd:string"
                                              minOccurs="0"/>
        <element name="Argomento" maxOccurs="*">
          <complexType content="xsd:string">
          <attribute name="sottoSezione" type="xsd:integer"/>
          </complexType>
        </element>
        <attribute name="focus" default="XML Schema">
            <simpleType base="xsd:string">
              <enumeration value="XML"/>
              <enumeration value="SOAP"/>
            </simpleType>
        </attribute>
    </complexType>
  </element>
  <element name="interruzioneSezione" minOcccurs="0"
                                              maxOccurs="*">
      <complexType content="empty"/>
  </element>
</complexType>
<schema>
```

Listato 3.5 – *Esempio di documento XML Schema.*

4.1. *Dichiarazione degli elementi.*

Per specificare un elemento viene utilizzata la seguente dichiarazione:

<element name=”[nome elemento]” type=”[tipo elemento]” [opzioni…]>

dove [nome elemento] è il nome dell'elemento utilizzato nel documento XML a cui si vogliono associare dei vincoli; [tipo elemento] è il tipo dell'elemento, che può essere un tipo built-in di XML Schema o un tipo definito dall'utente; [opzioni …] viene utilizzato per esprimere alcuni vincoli come il numero minimo e massimo di occorrenze dell'elemento all'interno del documento XML.

Ad esempio, il seguente frammento:

```
<element name="Titolo" type="xsd:string" minOccurs="0"/>
```

dichiara un elemento Titolo di tipo stringa e la cui presenza all'interno del documento XML è opzionale.

4.2. *Dichiarazione degli attributi.*

Per la dichiarazione degli attributi viene utilizzato l'elemento XML attribute con i seguenti attributi:

<attribute name=”[nome attributo]” type=”[tipo attributo]” [opzioni]>

[nome attributo] e [tipo attributo] sono rispettivamente il nome e il tipo dell'attributo che compare in un elemento del documento XML; [opzioni] sono ulteriori vincoli sull'attributo.

L'attributo di un elemento viene dichiarato all'interno della definizione del tipo dell'elemento stesso.

4.3. Tipi definiti dall'utente.

L'utente può definire tipi di dati semplici e tipi di dati complessi. I tipi di dati semplici (simpleType) possono essere derivati dai tipi semplici messi a disposizione da XML Schema (simpleType built-in [12]), mediante il meccanismo della restrizione.

Il listato 3.6 è un esempio di definizione di un tipo semplice ottenuto dalla restrizione del tipo base intero (circoscrivendo i valori assunti da questo tipo).

```
<simpleType name="HourType">
    <restriction base='integer'>
        <minInclusive value="1"/>
        <maxInclusive value="12"/>
    </restrinction>
</simpleType>
```

Listato 3.6 – *Definizione del tipo semplice HourType mediante la restrizione di un tipo di sistema (integer) di Schema.*

In questo caso, ***HourType*** è un tipo intero i cui valori sono compresi tra 1 e 12. Questo nuovo tipo potrà essere usato nella definizione degli elementi di un ipotetico documento XML, come nel seguente caso:

```
<element name="Time">
    <attribute name="Hour" type="HourType"/>
    <attribute name="Minute" type="MinuteType"/>
</element>
```

In tabella 3.2 sono riportati alcuni tipi primitivi e i rispettivi derivati (mediante restrizione) supportati dalla versione corrente di XML

Schema, nella quale, precisamente, è definito un insieme di tipi built-in, suddivisi in tipi primitivi (built-in primitive type) e tipi derivati (built-in derived type).

Tipo	*Sottotipi derivati*	*Significato*
string	NMTOKEN [10]	Stringhe di caratteri
boolean	nessuno	Valore logico booleano
float	nessuno	Numero a virgola mobile a 32 bit
double	nessuno	Numero a virgola mobile a 64 bit
decimal	integer	Numero in notazione decimale standard, positivo o negativo
dateTime	nessuno	Combinazione di data e ora che rappresenta un singolo istante di tempo.
duration	nessuno	Intervallo di tempo
time	nessuno	Istante di tempo che ricorre ogni giorno (tipo 12:00:20)
date	nessuno	Rappresenta la data del calendario Gregoriano nella forma 2001-10-11
hexBinary	nessuno	Dati binari codificati in esadecibale
base64Binary	nessuno	Rappresentazione di un dato binario in Base64 19]
anyURI	nessuno	URI (Uniform Resourse Identifier)

Tabella 3.2 – *Alcuni tipi primitivi di Schema XML.*

Per la definizione di un tipo di dato complesso viene utilizzata la dichiarazione complexType che, generalmente, contiene dichiarazione di elementi, di riferimenti e di attributi.

Nel listato 3.5 è riportata la dichiarazione del tipo complesso tipoLibro che è costituito dagli elementi Titolo, Contenuto e Copyright (nel linguaggio C questo può essere mediante il tipo struct).

Una definizione di tipo può essere esplicita o implicita. Un tipo definito attraverso un nome può essere esplicitamente usato da un qualsiasi elemento posto in una qualunque sezione dello schema. Tuttavia, ci sono dei casi in cui un tipo è specifico solo per un certo elemento. Allora, il tipo può essere implicitamente definito all'interno della dichiarazione dell'elemento stesso. Nel listato 3.5 l'elemento Capitolo, usato per la definizione del tipo Contenuto, definisce un tipo implicito costituito dai campi Testata e Argomento. A sua volta, l'Argomento viene definito come una stringa con un attributo sottoSezione di tipo intero, e con un numero illimitato di occorrenze.

4.3.1. Vincoli d'occorrenza.

Il numero di occorrenze per un certo elemento viene determinato dai vincoli *minOccurs* e *maxOccurs*. In generale, un elemento è optionale se *minOccurs*=0; è, invece, obbligatorio se *minOccurs*.≥1. Il massimo numero di volte che un elemento può apparire in un documento XML è determinato dal valore del suo attributo *maxOccurs*. In particolare, se *maxOccurs*=unbounded allora il numero d'occorrenze dell'elemento è infinito. Per default i valori degli attributi d'occorrenza sono entrambi

unitari. Da ciò consegue che, dovendo essere minOccurs<= maxOccurs, se nella dichiarazione di un elemento, con un certo valore di minOccurs, viene omesso l'attributo maxOccurs, allora, quell'elemento può apparire al massimo una volta. Similmente, se si specifica il valore per i solo maxOccurs, allora questo deve essere maggiore di 1. Un elemento senza attributi d'occorrenza può comparire una e una sola volta. La specifica consente l'uso dell'asterisco per indicare un numero illimitato di occorrenze. I vincoli d'occorrenze possono essere usati all'interno della dichiarazione di elementi e attributi.

4.4. Tipi derivati.

XML Schema consente la derivazione dei tipi in maniera simile a come avviene dei linguaggi di programmazione ad oggetti. Un tipo si può derivare da uno esistente mediante il meccanismo della restrizione (derivation by restriction), dall'estensione (derivation by extension), derivazione mediante lista (derivation by list) e derivazione mediante unione (derivation by union). In particolare, la restrizione di un qualunque tipo semplice è ancora un tipo semplice, mentre l'estensione di un qualunque tipo (semplice o complesso) da sempre origine ad un tipo complesso. Dunque, un tipo complesso si può ottenere dalla restrizione di un altro tipo complesso (tipo base), o dall'estensione di un tipo semplice o di un tipo complesso [12].

Nell'esempio 3.es1 è mostrato un caso di derivazione mediante restrizione del tipo primitivo string.

```
<simpleType name="Targa'>
 <restriction base='string'>
     <pattern value='[A-Z]{2}\d{3}[A-Z]{2}'/>
 </restriction>
</simpleType>
```

Il tipo Targa deriva dal tipo string e il suo spazio di valori comprende stringhe che combaciano con il pattern: due caratteri maiuscoli seguiti da 3 cifre seguite da altri due caratteri maiuscoli (esempio: AL251SL).

5. SOAP e XML Schema.

XML Schema può essere utilizzato come linguaggio neutrale per definire tipi di dati e interfacce di componenti software. Inizialmente, i creatori di SOAP pensarono di definire un linguaggio che potesse descrivere il loro protocollo. Questo linguaggio fu chiamato CDL (Component Description Language) ed era basato su XML. CDL è stato abbandonato per seguire l'iniziativa di XML Schema, e riusare questa tecnologia che, attualmente, è una raccomandazione proposta al consorzio W3C.

Ad ogni modo, SOAP non richiede l'uso di XML Schema. Tuttavia, utilizza i tipi di dati built-in di XML Schema per le definizioni dei tipi che possono essere usati per realizzare il mapping delle chiamate e risposte RPC (serializzazione/deserializzazione). Inoltre, si potrebbe usare uno schema per convalidare il messaggio SOAP. In esso vanno dichiarati tutti gli elementi usati nel messaggio: envelope, header e

body, i relativi attributi, i tipi corrispondenti, il numero delle occorrenze e altro.

6. Formato del messaggio SOAP.

Un messaggio SOAP, mostrato nel listato 3.7, è un documento XML con i seguenti elementi:

- Envelope
- Header
- Body

e con gli identificatori di namespace:

- *http://schemas.xmlsoap.org/soap/envelope/* per l'elemento envelope
- *http://schemas.xml.soap.org/soap/encoding/* per la serializzazione

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Header>
      <!—elementi header -->
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <!—elementi body -->
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listato 3.7 – *Struttura del messaggio SOAP.*

L'***envelope*** (obbligatorio) è l'elemento radice del documento XML e rappresenta il messaggio SOAP. Esso contiene un header (opzionale) e un body (obbligatorio). Il ***body*** è inteso come il recipiente del contenuto informativo del messaggio (payload). Se SOAP viene utilizzato come RPC-like, allora il body deve contenere informazione sui metodi serializzati, i risultati delle chiamate ai metodi, e le informazioni di fallimento (fault).

In generale, si possono verificare due tipologie d'errori nella comunicazione con un server che riceve richieste SOAP. Il server non è raggiungibile, in questo caso il client riceverà un errore HTTP di timeout. Se invece il server è raggiungibile, ma questo riscontra problemi nella deserializzazione dei pacchetti SOAP, quando chiama un metodo remoto, o, in generale, non comprende il contenuto del messaggio, allora il client riceverà un'opportuna eccezione SOAP (per i fault code si veda [9]) racchiusa nel payload della risposta, come contenuto dell'elemento <SOAP-ENV:Fault>.

L'***header*** contiene informazioni ausiliari che estendono il messaggio, come l'autenticazione, informazioni per l'elaborazione intermedia del messaggio, etc. Poiché l'header è ortogonale al payload del body, le informazioni in esso contenute non hanno alcun effetto sull'elaborazione di quest'ultimo (per maggiori dettagli sugli elementi del messaggio SOAP si veda [9]).

Si badi che nell'utilizzare gli elementi bisogna osservare specifiche regole grammaticali che sono riportate nella specifica [9]. Ad esempio, l'elemento header, se presente, deve necessariamente seguire l'elemento envelope.

Per quanto riguarda i namespace, si raccomanda di utilizzarli su tutti gli elementi e attributi del messaggio. Un'applicazione client, che elabora i messaggi SOAP, dovrebbe rifiutare quelli privi di namespace. Un ipotetico server SOAP, invece, può decidere di rifiutarli, oppure, di rispondere con un messaggio di fallimento.

L'attributo SOAP-ENV:encodingStyle è utile qualora si voglia utilizzare una metodologia di serializzazione dei dati differente da quella corrente di SOAP (identificata dal valore *http://schemas.xmlsoap.org/soap/encoding/*). Ad esempio, se si utilizza XML Schema si possono aggiungere altre regole per l'interpretazione dei dati e derivare nuovi tipi dall'estensione e dalla restrizione di tipi presenti nello schema SOAP.

7. SOAP-RPC.

I messaggi SOAP sono, fondamentalmente, trasmissioni del tipo one-way, cioè a semplice invio da un nodo mittente (sender) ad un nodo ricevente (receive). Tuttavia, possono essere opportunamente combinati per implementare meccanismi di trasmissione del tipo request/response (sincroni) [19].

La specifica SOAP stabilisce un insieme di regole per definire sia un modello per lo scambio dei messaggi (Message Exchange Model) sia un meccanismo per effettuare chiamate a procedure remote. Queste regole riguardano la codifica dei tipi di dati e la serializzazione della

chiamata al metodo remoto e degli argomenti passati dall'applicazione utente.

Il concetto di Remore Procedure Call non è certamente nuovo, ma risale ad almeno venti anni fa, allora noto come modello client/server. Nel corso degli anni ha subito vari cambiamenti in funzione prevalentemente dello sviluppo dei sistemi distribuiti.

Inizialmente, la rete aveva una larghezza di banda molto limitata, era incline a molti errori, e non si estendeva per lunhe distanze. Nel tentativo di migliorare la performance e la stabilità complessiva della rete nacquero i protocolli di rete come TCP/IP, IPX e NetBios, che ebbero un grande successo e tuttora sono ampiamente utilizzati.

Successivamente, la rete diventò più affidabile e le distanze tra computer remoti s'incrementavano. Gli sviluppatori, allora, cercarono soluzioni per distribuire il carico elaborativo su più sistemi remoti. Inizialmente, svilppavano il codice per la gestione della comunicazione tra i sistemi, direttamente dentro le loro applicazioni. Conseguentemente, un programma era responsabile di aprire la connessione remota, trasmettere i dati opportuni per l'elaborazione, e prelevare i risultati. Questo durò finchè non si pensò di costruire un livello applicativo sul top dell'infrastruttura di rete, progettando librerie che rendessero le interazioni con la rete sufficientemente trasparenti. Allora, la realizzazione d'applicazioni distribuite diventò più semplice.

In termini di linguaggio di programmazione, un RPC è molto simile alle chiamate di funzioni in C, ovvero il chiamante inserisce (push) i parametri nello stack della chiamata, trasferisce il controllo

d'esecuzione alla funzione, e aspetta che questa abbia terminato il suo lavoro. Subito dopo, la funzione preleva (pop) i parametri dallo stack, esegue qualche compito, inserisce il risultato sullo stack, e ritorna il controllo al chiamante. Questo allora recupera il risultato e continua nella sua esecuzione.

Ora nel caso d'interazione remota, la chiamata di una funzione che si trova su una macchina B da parte di un programma che risiede su una macchina A, viene prima serializzata e poi trasferita in rete alla macchina B. Dall'altra parte i parametri vengono deserializzati e inseriti nello steck della chiamata, logicamente dopo che questa è stata invocata. Infine, il valore di ritorno viene serializzato e spedito in rete alla macchina A. L'applicazione allora estrae il contenuto significativo dal messaggio lo deserializza, lo inserisce nello stack di chiamata e lo usa. Da quanto detto si intuisce che tra l'applicazione e la funzione remota ci devono essere, oltre che un protocollo di comunicazione convenuto da entrambe le parti, due processi in ascolto che realizzano lo scambio sincrono dei messaggi.

su gli argomenti vengno serializzati e da una parte e deserializzati dall'altra, in modo che possano essere trasmessi in rete. Quindi, il programma chiamante serializza gli argomenti

Nella figura 3.4 sono mostrate le fasi di un ipotetico scenario dove un'applicazione client invoca il metodo di un oggetto remoto usando SOAP:

1. L'applicazione utente invoca il metodo remoto chiamando un metodo del client SOAP (un proxy con librerie SOAP) che

consente di realizzare il binding sull'oggetto remoto. Supponiamo, ad esempio, che all'oggetto remoto venga passato, attraverso il metodo, una struttura mediante riferimento. Allora, all'atto dell'invocazione viene creato lo stack della chiamata e l'oggetto proxy (rappresentanza locale dell'oggetto remoto) accede alle informazioni della struttura. Successivamente, i dati della struttura vengono serializzate in un messaggio SOAP in formato XML.

2. A partire dal messaggio SOAP e dall'indirizzo del server che gestisce la richiesta SOAP per conto del metodo remoto, viene confezionato un messaggio HTTP e spedito attraverso Internet.

3. Il server HTTP acquisisce la richiesta e la instrada ad un gestore SOAP.

4. Il server SOAP, mediante un parser XML, estrae i contenuti degli elementi e degli attributi del payload. Questi dati, che rappresentano i valori testuali degli argomenti del metodo invocato, vengono decodificati, in relazione al tipo di linguaggio in cui è stato implementato l'oggetto remoto, e con essi è ricreata la struttura dati in memoria. Infine, viene creato l'oggetto e invocato il metodo al quale viene passato il riferimento della struttura. Il metodo effettua le elaborazioni richieste sulle informazioni della struttura e ritorna il risultato al server SOAP che si occuperà di rispedirlo al mittente.

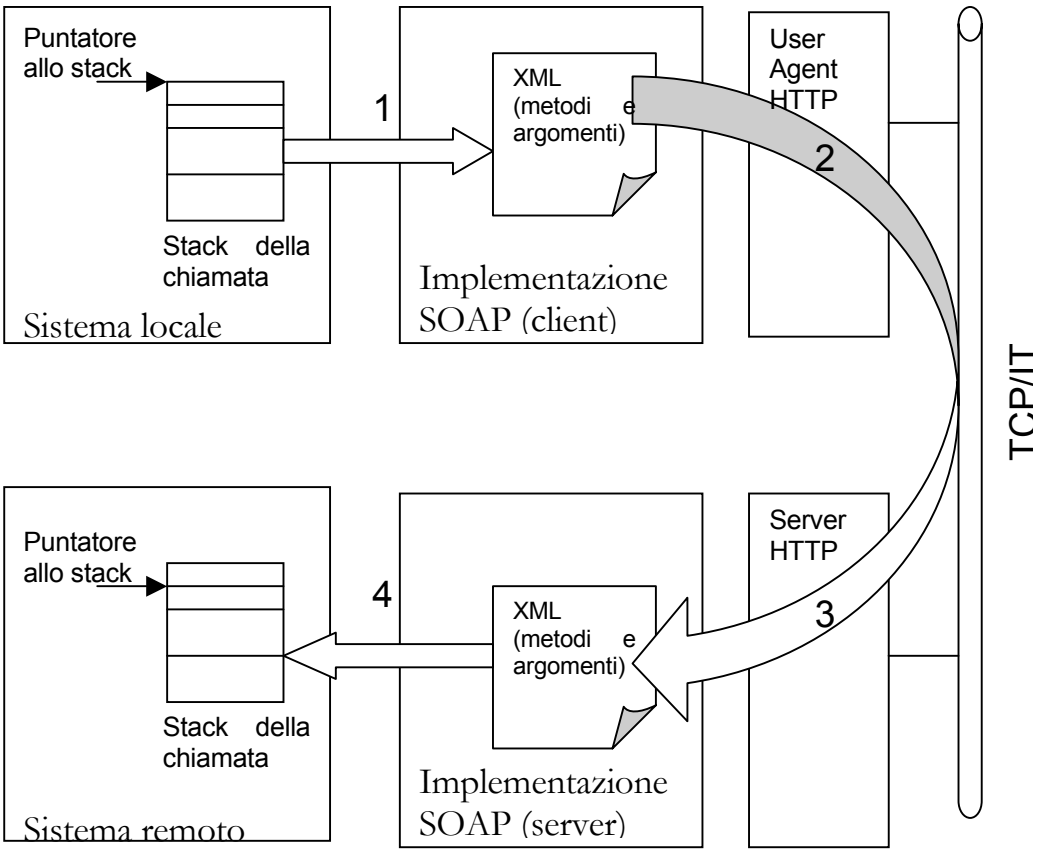


Figura 3.4 – *Invocazione di un metodi remoto usando SOAP.*

Il vantaggio di usare SOAP nell'invocazione dei metodi remoti è evidente. Invece di serializzare lo stack della chiamata in un formato di serializzazione proprietario, o specifico per una determinata architettura, le regole di serializzazione dettate dalla specifica SOAP consentono di trasformarlo in un documento formato XML.

7.1. Tipi di dati SOAP.

I tipi di dati in SOAP si suddividono in tipi semplici (simple type) e tipi composti (compound type) [9].

Un tipo semplice è un qualunque tipo built-in (primitivo e derivato) definito nella specifica di XML Schema. Alcuni esempi di tipi semplici sono rappresentati nella tabella 3.2.

I tipi composti, invece, rappresentano strutture di dati complesse, come array, unioni, e classi.

Un parametro di un metodo o un componente del tipo composto qualificato da un nome è detto accessor (in XML Schema è rappresentato da un elemento di un complextType esplicitamente nominato, paragrafo 4.5.3).

Uno struct è una collezione di accessor distinti; in altre parole, ciascun membro di uno struct deve avere un nome unico. Ovviamente anche uno struct può essere un accessor, ad esempio, quando contenuto in un altro struct.

Un elemento che appare al massimo livello della serializzazione (figli dell'elemento body) viene detto elemento indipendente, e rappresenta un'istanza di tipo. Tutti gli altri elementi che non sono indipendenti vengono detti embedded e rappresentano gli accessor.

Un elemento che può essere referenziato da un solo accessor viene detto single-reference.

Un elemento che può essere referenziato da più accessor è definito multi-reference.

In particolare, SOAP mette a disposizione la coppia id/href per realizzare i riferimenti. 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 '#'.

Come riferito nel precedente paragrafo 4.5, l'elemento di un documento XML viene tipizzato attraverso la dichiarazione di un opportuno elemento di XML Schema (che è un elemento XML avente specifici attributi). Supponiamo che <auto>Fiat Punto</auto> sia un elemento del documento XML, allora questo viene vincolato al suo tipo mediante la dichiarazione <element name="auto" type="xsd:string"/> dell'elemento XML Schema. Ora, se nel messaggio SOAP si utilizzassero questi due elementi per descrivere il valore di un argomento di un metodo e per tipizzarlo, si dovrebbe disporre di un parser convalidante abilitato all'uso del documento XML Schema. Tuttavia, questi non sono, attualmente, riconosciuti dallo standard XML, che, invece, continua ad usare i DTD. Per porre rimedio a questo problema, la specifica SOAP usa l'elemento di XML Schema esplicitamente nominato, detto appunto accessor, anche per contenere o riferire il valore. In questo modo si ottiene la seguente forma <auto type="xsd:string">Fiat Punto</auto>, in cui l'accessor auto contiene il valore 'Fiat Uno'.

Ciascun accessor del payload SOAP deve mappare un argomento passato al metodo all'atto della sua invocazione, e perciò oltre ad avere un nome e un tipo, deve anche referenziarne il valore. In realtà, la specifica non vincola l'implementazione a questa caratteristica, consentendo di rappresentare il valore di un argomento senza indicarne il tipo (in questo caso non è un elemento di XML Schema, ma più in generale di XML). In questo caso, è necessaria una descrizione dell'oggetto remoto (attraverso ad esempio dei metadati) che riporti per ciascun parametro del metodo invocato il tipo

corrispondente. La convenienza di non rappresentare il tipo dell'elemento è evidente. La dimensione del payload si riduce e di conseguenza anche quella del messaggio http, e in più lo stesso playload viene analizzato più velocemente. Il rischio, però, è altrettanto chiaro. La mancanza dell'informazione sul tipo rende impossibile l'estrazione dei metadati delle firme di metodi in overloading e conseguentemente non sarà possibile determinare quale metodo è stato invocato.

Un valore nel payload (argomento del metodo) è referenziato singolarmente (value single-reference) se è referenziato solo dall'accessor circostante. Invece, è molteplicemente referenziato (value multi-reference) se è referenziato da più di un accessor. In figura 3.5 è mostrato un esempio di sigle-reference (a) e uno di multi-reference (b).

Nel caso a) il valore Ferrari è referenziato dai rispettivi elementi circostanti, che in pratica lo contengono. Questo accade quando si serializzano argomenti che sono passati per valori e sono tutti diversi tra loro.

Nell'esempio b) il valore Ferrari è referenziato da due accessor, scuderia e auto.

a. Valore single-reference

```
<scuderia>Ferrari</scuderia>
<auto>Ferrari</auto>
```

b. Valore multi-reference

```
<scuderia href="#ferraricar"/>
<auto id="ferraricar">Ferrari</auto>
```

Figura 3.5 Esempi di riferimento a valori

Questa situazione accade quando si serializzano argomenti che sono passati per riferimento o nel caso in cui esistono argomenti uguali.

7.2. Codifica del payload.

Il body del messaggio SOAP è il contenitore in cui vengono inseriti i dati in uscita dalla serializzazione delle chiamate ai metodi remoti e dalle relative risposte. La risposta può essere un valore valido oppure un codice di fallimento. Di conseguenza, ci sono tre tipi di elementi fondamentali nel body:

- Call (informazioni per invocare il metodo)
- Response (informazioni per il valore o i valori ritornati dal metodo)
- Fault (la chiamata non è andata a buon fine)

Logicamente, la presenza di questi elementi nel body è mutuamente esclusiva.

Le chiamate e le risposte dei metodi remoti, trasportate dal body, devono essere documenti XML e quindi presentano una particolare struttura. La specifica definisce una rappresentazione sia per la chiamata sia per la risposta.

Rappresentazione della chiamata al metodo remoto:

- Una chiamata ad un metodo remoto deve essere modellata come uno struct contente un *accessor* per ogni parametro in input o input/output del metodo. L'elemento più esterno deve essere nominato col nome del metodo.

- Ciascun accessor deve rappresentare un parametro in input o input/output, ed avere nome e tipo corrispondenti a quelli del parametro (il nome può essere anche diverso). Gli accessor devono apparire nello stesso ordine della firma del metodo invocato.

Rappresentazione della risposta ritornata dal metodo remoto:

- La risposta del metodo è modellata come uno struct, avente un accessor per il valore ritornato e, opzionalmente, per ogni parametro d'input o input/output. Il primo accessor è il valore ritornato seguito dai parametri nello stesso ordine della firma del metodo.

- Ogni accessor di un parametro ha il nome corrispondente al nome del parametro e il tipo corrispondente al tipo del parametro. Il nome dell'accessor del valore ritornato non è

significativo. Anche il nome dello struct è arbitrario, purché lo si faccia seguire da una stringa "Response".

Rappresentazione della risposta in caso d'errore:

- Il metodo fault viene codificato utilizzando l'elemento SOAP Fault seguito dai figli faultcode e faultstring. Nella tabella 3.3 sono riportati i codici di errore.

Nome	***Significato***
VersionMismatch	Il modulo di elaborazione ha trovato un namespace invalido per l'Envelope
Client	Il messaggio non può essere elaborato o perché non valido (non conforme al messaggio SOAP) o perché non contiene informazioni indispensabili per l'elaborazione (ad esempio, l'autenticazione). Ad ogni modo la causa è imputabile al client.
Server	Indica che il messaggio non può essere elaborato per alcune ragioni attribuibili al server e non al contenuto del messaggio.

Tabella 3.3 – *Codici di fault definiti da SOAP.*

Nel figura 3.6 sono mostrati esempi dei tre tipi di messaggi, a) richiesta, b) risposta e c) risposta d'errore.

Il metodo remoto invocato è getCAP che restituisce il C.A.P. a partire dal nome di un comune d'Italia. getCAP ha la seguente semplice interfaccia in Java:

String getCAP(String comune)

Quando il metodo viene localmente invocato dall'applicazione utente:

getCAP("Avellino");

la serializzazione SOAP genera uno struct con lo stesso nome del metodo, getCAP appunto, e un accessor con il nome 'arg', il tipo string che referenzia singolarmente (single-reference) il valore 'Avellino'.

Nell'esempio b), il valore ritornato viene referenziato dall'accessor <result> il cui tipo è una stringa (stringa di XML Schema).

Infine, nel terzo esempio troviamo l'elemento indipendente <SOAP-ENV:Fault> con gli elementi <faultcode> e <faultstring>.

L'elemento <faultcode> è destinato per riportare codici di fallimento, alcuni di quelli definiti dalla specifica sono riportati in tabella 3.3. Invece, l'elemento <faultstring> contiene una spiegazione plausibile del fallimento (tipo reason-phrase di http).

a. Messaggio di richiesta.

```
<?xml version="1.0" encoding="UTF-8"?>
SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
     <ns1:getCAP xmlns:ns1="http://ejbean:8080/soap/">
        <arg xsi:type="xsd:string">Avellino</arg>
     </ns1:getCAP>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

b. Messaggio di risposta.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:getCAPResponse xmlns:ns1="
                              http://ejbean:8080/soap/">
          <result xsi:type="xsd:string">Hello!</result>
    </ns1:getCAPResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

c. Messaggio di risposta con errore.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
          <faultcode>SOAP-ENV:MustUnderstand</faultcode>
          <faultstring>SOAP Must Understand Error</faultstring>
      </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 3.6 – *Esempi di messaggi SOAP request/response.*

Infine, nei messaggi di richiesta e risposta vengono usati quattro prefissi. SOAP-ENV è usato per gli elementi fondamentali del messaggio, envelope, body, header e fault. Il prefisso xsd contraddistingue i tipi di sistema di XML Schema e xsi la loro istanza all'interno degli accessor. Per ultimo, ns1 è associato con un nomespace rappresentativo dell'oggetto remoto, che, in generale, viene utilizzato per distinguere i metodi degli oggetti che implementano una

stessa interfaccia. Un caso abbastanza comune consiste nell'usare, come namespace, l'identificativo URN dell'oggetto.

7.2.1. Alcuni esempi di serializzazione.

Le regole di serializzazione di SOAP consentono di codificare tutti i tipi di dati semplici e composti, compresi quelli generici (generic compound type), definiti dalla specifica. In generale, un tipo composto generico è un'istanza del tipo struct in cui i membri sono nominati e tipizzati a runtime. Questo tipo composto può essere usato, per esempio, per rappresentare le strutture dati ritornate dalle interrogazioni effettuate ad un database, o, in generale, in tutti quei casi in cui il formato delle informazioni non è conosciuto a priori.

I tipi semplici dei parametri dei metodi invocati possono essere facilmente serializzati. A tale proposito è sufficiente rappresentare ogni singolo parametro con un accessor con lo stesso tipo del paramentro.

Riporto solo alcuni casi di serializzazione, a titolo esemplificativo. Le interfacce dei metodi remoti vengono descritte mediante il linguaggio IDL (Interface Description Language) per il quale le notazioni [in], [out] e [in/out] significano rispettivamente che le informazioni sono trasmesse al metodo remoto, le informazioni sono ritornate dal metodo e le informazioni sono trasmesse e ricevute usando lo stesso argomento.

Nell'invocazione del metodo remoto, qualora la corrispondenza dei parametri effettivi e quelli formali è per valore, la serializzazione utilizzata è del tipo single-reference. In seguito è riportato un esempio in cui i valori degli argomenti vengono passati per valore.

```
void myMethod([in] int arg1, [in] int arg2); //firma del metodo
remoto.
…
i=10;
j=20;
myMethod(i,j); //chiama il metodo remoto
```

In questo caso i due argomenti possono essere rappresentati da due accessori single-reference, e cioè da due elementi embedded (non figli diretti dell'elemento body) che contengono i valori degli argomenti. Il messaggio d'invocazione remota è rappresentato dal listato 3.8.

```
<?xml version="1.0" encoding="UTF-8"?>
SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
     <ns1:myMethod xmlns:ns1="http://ejbean:8080/soap/">
        <arg1 xsi:type="xsd:int">10</arg1>
        <arg2 xsi :type="xsd:int ">20</arg2>
     </ns1:myMethod>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listato 3.8 – *Esempio di serializzazione single-reference.*

Diversamente dal caso precedente, quando gli argomenti vengono passati per riferimento è necessario codificarli usando la multi-reference. In pratica, bisogna trovare una forma di codifica alternativa, che avvisi il modulo di attivazione del metodo remoto, che l'indirizzo dell'argomento passato deve essere ricopiato nel corrispondente parametro formale. L'esempio che segue illustra questo caso. Il primo

argomento viene passato per riferimento, mentre il secondo per valore. Il listato 3.9 rappresenta la codifica dell'invocazione del metodo.

```
void myMethod([in,out] int arg1, [in] int arg2); //firma del metodo
remoto.
…
i=10;
j=20;
myMethod(&i,20); //chiama il metodo remoto
```

```
<?xml version="1.0" encoding="UTF-8"?>
SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
      <ns1:myMethod xmlns:ns1="http://ejbean:8080/soap/">
         <i href="#arg1"/>
         <j xsi :type="xsd:int ">20</j>
      </ns1:myMethod>
     <ns1:i id="arg1" xsi:type="int">
       10
     </ns1:i>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listato 3.9 – *Esempio di serializzazione multi-reference.*

Il seguente esempio mostra la serializzazione di un array d'interi. L'array è un tipo composto SOAP:ENC:Array i cui membri non sono referenziati da un nome, bensì da un indice (posizione ordinale).

```
int myMethod(int[] param); //firma del metodo remoto.
…
int intArray[3]={0,1,2};
int errorCode=myMethod(intArray);
```

Il metodo remoto riceve un array d'interi e restituisce un codice d'errore. Il messaggio SOAP con i dati codificati è il listato 3.9.

```
<?xml version="1.0" encoding="UTF-8"?>
SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
      <ns1:myMethod xmlns:ns1="http://ejbean:8080/soap/">
         <param href="#array"/>
      </ns1:myMethod>
     <ns1:intArray id="arg1" SOAP-ENC:arrayType="xsi:int[5]">
      <int>0</int>
      <int>1</int>
      <int>2</int>
     </ns1:intArray>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listato 3.9 – *Esempio di serializzazione multi-reference.*

7.3. SOAP e HTTP.

Attualmente il protocollo HTTP, su cui è basata pesantemente l'infrastruttura Internet, è la base di trasporto utilizzata dal payload SOAP. Scelta che consente ai documenti XML di SOAP di attraversare i firewall, poiché in una tipica configurazione aziendale la porta associata al protocollo HTTP è lasciata sempre aperta, consentendo lo scambio d'informazioni attraverso le più disparate piattaforme. Inoltre, i Web server e Web Application server mettono a disposizione molte funzionalità estensibili che consentono di realizzare facilmente server SOAP, ad esempio utilizzando servlet o a asp (active

server pages), a cui inviare le richieste, e nel contempo assicurano alti gradi d'affidabilità e scalabilità.

Comunque, la semplicità e la compatibilità di usare un protocollo text-based, seppure meno performante di uno binario, è la ragione fondamentale che ha spinto SOAP ad usare HTTP.

Ovviamente, poiché SOAP utilizza HTTP per le richieste e risposte, esso n'eredita tutta l'architettura sottostante. Il modello request/response usato da Internet è un modello client/server. In generale, un client si connette al server per utilizzare una risorsa, e questo, se ritiene valida la richiesta, fornisce le informazioni cercate. Dopo la risposta, il server cancella lo stato relativo alla richiesta, chiude la connessione e passa a processare la successiva richiesta. Questo semplice meccanismo comporta un basso overhead fornendo un ambiente scalabile per i Web server che accettano richieste da un numero considerevole di client [19].

HTTP 1.1 complica leggermente il modello, fornendo una caratteristica, detta *keep-alive* [20], che consente ai client di effettuare richieste multiple su una singola connessione TCP, senza essere scollegati dal server, soluzione questa che comporta meno sovraccarico del server http ed evita la congestione su Internet, non dovendo ogni volta rinegoziare per una nuova connessione. Questo tipo di connessione è molto utile per scaricare pagine Web, poiché tipicamente richiedono grosse quantità di risorse. Tuttavia, nel caso di SOAP, e in generale per un sistema distribuito, non conviene accettare frequenti e consecutive richieste dallo stesso client per evitare di monipolarizzare le risorse condivise.

Un messaggio di richiesta HTTP è costituito da tre parti:

- Un metodo di richiesta (obbligatorio)
- Un insieme di header field.
- Contenuto della richiesta.

e termina sempre con i caratteri {CR}{LF}.

I metodi di richiesta usati per trasportare informazioni sono GET e POST. SOAP utilizza quest'ultimo metodo.

Un request header indendifica informazioni aggiuntivi (come i tipi di dati accettati dell'user agent [20], il tipo di client software che effettua la richiesta, l'host che serve la richiesta, etc.) attraverso coppie del tipo nome:valore.

Sotto gli header field compare il payload del messaggio, che nostro caso è il messaggio SOAP.

Quando la richiesta è processata dal server, esso individua la risorsa, effettua le opportune elaborazioni e invia al client un messaggio di risposta. Il messaggio di risposta è formato da una status line, da un insieme di coppie nome:valore e dall'informazione richiesta. La linea di stato contiene tre testi separati da spazi. Il primo è la vesione del protocollo HTTP supportata dal server, seguito da un codice di stato (status code) e infine da una stringa (reason phrase) che spiega lo stato del messaggio di ritorno.

7.3.1. SOAP e il metodo POST.

POST è un metodo che consente di trasmettere il contenuto della richiesta nel body del messaggio HTTP (a differenza di GET che invece invia le informazioni attraverso l'URL).

Per referenziare un particolare componente remoto sono necessarie le seguenti informazioni [19]:

- Protocollo di trasporto (HTTP in questo caso)
- Numero di porta (per default 80).
- Indirizzo IP
- Identificatore del componente o dell'oggetto.
- Identificatore della funzione e dell'interfaccia.

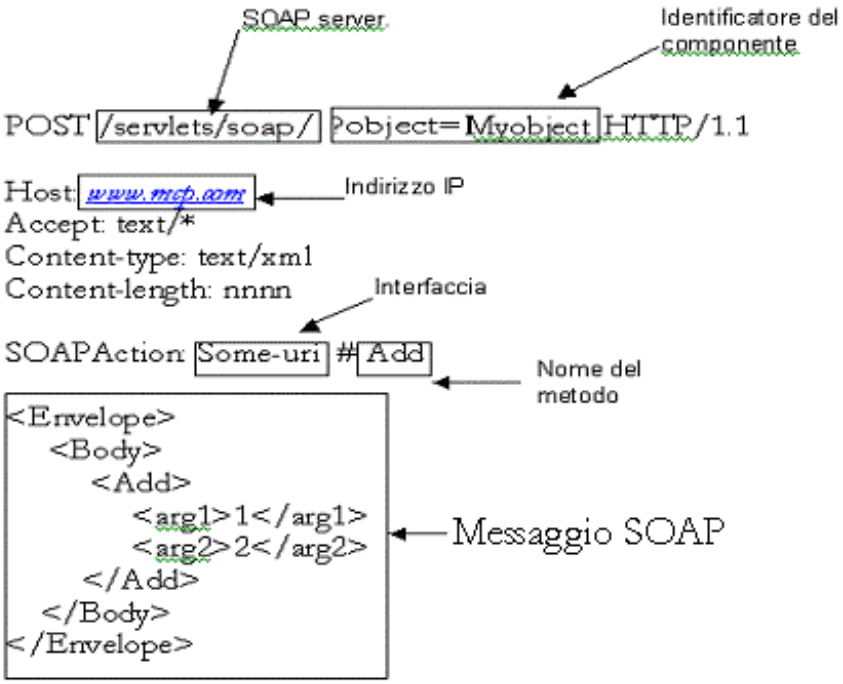


Figura 3.5 – *Richiesta SOAP (nel messaggio SOAP sono stati omessi i namispace per concisione).*

- Tutte questi dati sono presenti nella richiesta SOAP, come mostrato in figura 3.5.

- Tipicamente il protocollo di trasporto HTTP è assegnato al numero di porta 80 ed è servito da un Web server (che è in ascolto su questa porta).

- Il nome dell'host che identifica univocamente la macchina sulla rete, che serve la richiesta, è specificato dal campo host (l'hostname viene risolto in un indirizzo IP).

- Quando il Web server riceve la richiesta esegue un'applicazione identificata da un path relativo (/servlets/soap). In questo specifico caso, il Web server esegue una servlet Java [22] che dopo aver analizzato ed elaborato il messaggio SOAP (payload SOAP) attiva il componente Myobject ed invoca il metodo Add. La servlet implementa un server SOAP che accetta richieste da più client e le instrada a un modulo d'attivazione (detto anche provider) che s'incarica d'istanziare il componente, di invocarne il metodo e di restituire al mittente l'eventuale risultato. Ovviamente, il path relativo può indicare anche uno script Perl, un'applicazione cgi, un asp (active server pages) o qualunque altra applicazione del livello intermedio di un'architettura Web-based, capace di processare i messagi SOAP. Ad esempio nel caso di uno script Perl si può utilizzare il seguente endpoint (unione del path relativo e il riferimento al componente)

/creator.pl?object=Myobject per eseguire il componente Myobject.

- Nel campo SOAPAction (definito dalla specifica SOAP [9]) può essere messa l'identifcativo dell'interfaccia del componente, qualora, ad esempio, questo fosse un oggetto. In generale, questo campo va utilizzato per indicare che il messaggio HTTP POST contiene un messaggio SOAP e nello stesso tempo per comunicare l'intento del messaggio SOAP (tipicamente una chiamata ad un metodo remoto), in modo tale che un firewall possa processare l'header HTTP e assicurarsi che la chiamata abbia le autorizzazioni per proseguire.

- La versione 1.1 di SOAP specifica che il campo SOAPAction deve essere presente nel messaggio HTTP, sebbene possa essere vuoto. In generale, quando questo campo è vuoto allora significa che l'intento del messaggio può essere dedotto dall'URL indicato nel metodo POST, che indica dove il messaggio è diretto.

Il campo contet-Type indica il type MIME (Multipurpose Internet Mail Extensions [21]) del contenuto del messaggio POST, mentre Content-Length indica la sua lunghezza in byte. Accept invece indica la lista di tipi di MIME accetti dal client.

Capitolo IV

WSDL
(WEB SERVICES DESCRIPTION LANGUAGE)

1. Introduzione.

Web Service Desription Language (WSDL) è un linguaggio XML-based per la descrizione delle interfacce dei servizi remotamente accessibili. Esso è descritto formalmente nella specifica 1.1 (cui si fa riferimento in questo studio [23]), che è allo stato di sottomissione del gruppo di standardizzazione W3C.

I Web Services usano i documenti WSDL per descrivere la loro interfaccia e le specifiche per l'accesso e l'invocazione. L'uso di tale descrizione consente ai client di chiamare i servizi Web senza conoscere i dettagli sulla loro implementazione, o su quale piattaforma o sistema operativo essi sono in esecuzione.

WSDL è costrutito sul top di standard aperti, come XML, XML Schema, SOAP e MIME.

Il 25 settembre del 2000, IBM, Ariba, e Microsoft presentarono la versione 1.0 della specifica WSDL, in cui fu illustrato come la tecnologia XML potesse essere usata per descrivere i dettagli dei servizi disponibili in rete.

Tuttavia, l'approccio all'uso di XML per la descrizione dei servizi risale a periodi precedenti. Web Interface Definition Language (WIDL) [23] di Web Method è stato uno dei primi documenti in formato XML per la descrizione di funzionalità accessibili su macchine remote, ad uso, principalmente, di tecnologie basate su chiamate a procedure remote (RPC).

Insieme al protocollo SOAP, Microsoft sviluppò, all'inizio del 2000, il linguaggio SCL (SOAP Contract Language) [8] nel tentativo di fornire un sistema per la descrizione di servizi on-line raggiungibili mediante l'uso del suddetto protocollo di handshake. Questo lavoro, congiuntamente agli sforzi di altri produttori software, come IBM e Ariba, è confluito nel progetto WSDL. In aggiunta a SCL, Microsoft fornisce Discovery of Web Services (DISCO), un sistema per la ricerca delle descrizioni SCL dei servizi che verificano particolari requisiti.

Parallelamente a SCL di Microsoft, l'IBM sviluppò Network Accesible Service Specification Language (NASSL) insieme ad un meccanismo di discovery di servizi chiamato Advertisement and Discovery of Services (ADS).

Appena prima dell'emergente WSDL, un consorzio di 36 compagnie, incluse IBM, Microsoft e Ariba, lanciarono l'Universal Description, Discovery and Integration (UDDI). L'intento era, ed è tuttora, quello di fornire strutture dati e API standard per la pubblicazione e ricerca di servizi on-line forniti dalle aziende. Nel capitolo successivo sarà mostrato un possibile impiego di WSDL per estendere le informazioni pubblicate nel registro UDDI.

Allo stato attuale della specifica, il documento WSDL può essere associato solo ai servizi remoti accessibili mediante il protocollo SOAP su HTTP, appunto i Web Services.

2. Documento WSDL.

WSDL definisce una grammatica XML per la descrizione dei servizi distribuiti. Questi vengono visti, in maniera astratta, come degli endpoint di rete capaci di scambiare messaggi SOAP reciprocamente o con un'applicazione.

Un documento WSDL è costituito dall'insieme degli elementi mostrati in tabella 4.1.

Elementi	*Descrizione*
types	un set di definizioni di tipi di dati corrispondenti a un dato sistema d tipi (ad esempio quello usato da XML Schema).
message	una definizione astratta dei dati che vengono scambiati con gli endpoint.
operation	una definizione astratta delle operazioni supportate dall'endpoint.
port type	un set astratto di operazioni supportate da uno o più endpoint.
binding	consente di mappare le operazioni e i messaggi, descritti in maniera astratta, in un concreto protocollo e specifico formato di dati.
port	associa un indirizzo URL a un elemento binding.
service	un insieme di endpoint.

Tabella 4.1 – *Elementi XML usati per la definizione di servizi remoti.*

Tutti questi elementi devono essere inclusi nell'elemento ***definitions*** che rappresenta la radice del documento WSDL.

In figura 4.1 è mostrata la struttura del documento mediante un diagramma concettuale di classe secondo la notazione UML.

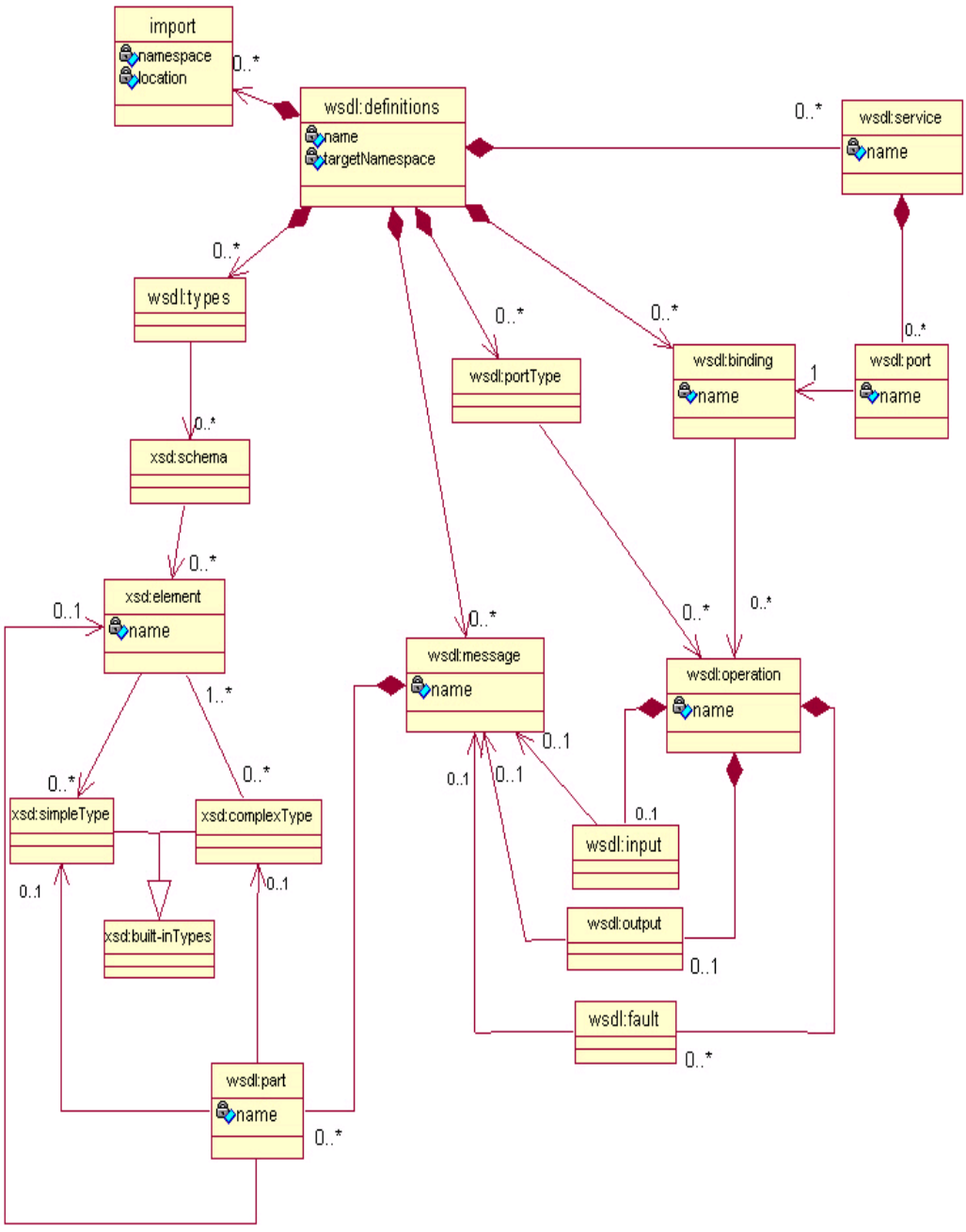


Figura 4.1 – Struttura del documento WDSL.

Per importare un documento WSDL può essere usato l'elemento ***import*** nel quale va specificato l'URI che consente la sua localizzazione, e lo stesso namespace usato dal documento contenitore.

L'elemento ***wsdl:types*** contiene le definizioni dei tipi di dati che sono utilizzate per tipizzare, ad esempio, i parametri di un eventuale metodo remoto. Il sistema di tipi usato da WSDL è fondamentalmente quello definito in XML Schema (vedi paragrafo 3.4.5.3), al quale viene aggiunto un elemento di estensione che consente di introdurre nuovi tipi definiti dall'utente. Il meccanismo d'estensione del WSDL è ereditato dal linguaggio XML e consente di definire nuovi elementi per personalizzare i documenti.

Gli elementi ***wsdl:message*** rappresentano i messaggi astratti, e, di conseguenza, indipendenti dai protocolli usati per inviarli. Contengono uno o più parti logiche, rappresentate dagli elementi ***wsdl:part***. Ciascuna di questa è associata o con un elemento definito in xsd:schema o con un tipo semplice o complesso definito sempre in xsd:schema. Il nome del messaggio deve essere un identificativo unico all'interno dell'intero documento WSDL (comprensivo d'eventuali documenti importati).

Gli elementi *part* di *message* sono usati per descrivere il contenuto astratto dei messaggi. Con riferimento ad una chiamata RPC-like, i *wsdl:part* rappresentano i parametri logici in, out e in/out della firma della chiamata remota. Il nome del messaggio può essere scelto in modo da suddividere gli elementi *part* in liste che rispecchiano le direzioni del flusso dei dati. Ad esempio, un messaggio nominato

In*some-method*Request può contenere una lista di elementi *wsdl:part* corrispondenti ai parametri d'ingresso del metodo *some-method*. In particolare, i tipi astratti e l'ordine dei *wsdl:part* corrispondono, rispettivamente, ai tipi concreti e all'ordine dei parametri nella RCP call. Lo stesso procedimento può essere applicato nei riguardi del valore di ritorno. Un *wsdl:part* che compare sia nel messaggio d'ingresso sia in quello d'uscita riflette un parametro in/out.

Un meccanismo di binding trasformerà il messaggio astratto nel formato usato dal protocollo di messaging per negoziare con il servizio remoto. Tuttavia, se come protocollo di messaging si utilizza SOAP, la definizione astratta corrisponde esattamente con quella concreta, dato che SOAP utilizza XML Schema come sistema di riferimento per la serializzazione dei tipi di dati.

Un ***wsdl:portType*** incapsula un insieme di operazioni astratte con i relativi parametri d'input e output che puntano ai messaggi logici. Le operazioni sono rappresentate dagli elementi ***wsdl:operation***. Ciascuna ha come attributo un nome univoco nell'ambito del documento WSDL, che può essere utilizzato come riferimento.

Ci sono quattro forme d'operazioni che sono implicitamente supportate da WSDL:

- Trasmissione one-way. L'endpoint riceve un messaggio, ma non invia una risposta. Un esempio, potrebbe essere una sottomissione di un ordine ad un sistema di vendita on-line.
- Trasmissione request/response. L'endpoint riceve un messaggio, elabora le informazioni, e invia un messaggio

correlato. In un contesto RPC-style è simile alla chiamata di una funzione, che accetta una lista di parameti e ritorna un valore.

- Trasmissione solicit/response. L'endpoint invia un messaggio, e riceve una risposta correlata. Quest'operazione è opposta a quella request/response, in quanto è il server a sollecitare il client e quindi ad iniziare la comunicazione. Tuttavia, WSDL 1.1 non definisce un binding per questo tipo di operazione, per cui attualmente è solo una nozione astratta di una primitiva di trasporto [9].

- Notification. L'endpoint invia un messaggio, ma non riceve una risposta. Anche questo tipo di operazione è astratta poiché non è definito un binding per essa.

Le forme one-way e notification sono trasmissioni unidirezionali, mentre le altre due sono bidirezionali. Sebbene le primitive di trasmissione bidirezionali possano essere modellate usando quelle di one-way e notification, si preferisce mantenere questo modello, perché è abbastanza comune (sono le stesse primitive usate da SOAP), facilita la realizzazione di un algoritmo per ricavare il flusso dei dati, e consente di definire sequenze di comunicazioni più complesse.

Le direzioni del flusso dei dati vengono rappresentate attraverso gli elementi ***wsdl:input*** e ***wsdl:output***, rispettivamente per il flusso in ingresso e in uscita rispetto all'endpoint. Essi hanno un nome (opzionale) unico all'interno di un *portType* e referenziano il nome del

messaggio astratto, ***wsdl:message***, che contiene la lista dei parametri logici.

In un'operazione one-way vengono definiti solo messaggi d'ingresso, ma non messaggi d'uscita, esattamente l'opposto dell'opeazione notification.

L'elemento ***wsdl:fault***, usato solo nelle trasmissioni bidirezionali, definisce il formato astratto di un eventuale messaggio di errore che può verificarsi durante l'elaborazione della richiesta. Nel listato 4.1 sono riportati i formati delle operazioni request/response e solicit/response.

```
<wsdl:portType>
    <wsdl:operation name="some-request-response">
        <wsdl:input message=" some-message-input">
        <wsdl:output message="some-message-output">
        <wsdl:fault message="some-message-fault">
    </wsdl:operation>
    <wsdl:operation name="some-solicit-response">
       <wsdl:output message=" some-message-output">
       <wsdl:input message="some-message-input">
       <wsdl:fault message="some-message-fault">
    </wsdl:operation>
</wsdl:portType>
```

Listato 4.1 – Esempio di operazioni WSDL.

Un particolare *binding* può essere utilizzato per determinare come i messaggi devono essere concretamente inviati: attraverso una singola comunicazione (ad esempio attraverso una comunicazione request/response HTTP) o attraverso due comunicazioni indipendenti (come due richieste HTTP).

Nel caso di una chiamata RPC-like, il nome dell'operazione deve essere uguale al nome del metodo remoto che si vuole invocare, l'elemento *wsdl:input* deve referenziare un messaggio che logicamente rappresenta la lista dei paramentri d'input, e l'elemento wsdl:output quello della lista dei parametri d'output.

L'elemento ***wsdl:binding***, come anticipato, definisce i dettagli sul formato del messaggio e sul protocollo per le operazioni e messaggi definiti in maniera astratta dagli elementi *wsdl:message* e *wsdl:portType*.

Un elemento *wsdl:operation* all'interno del *binding* specifica le informazioni che consentono di negoziare con il metodo remoto rappresentato logicamente dall'operazione omonima presente nell'insieme *wsdl:portType*. WSDL consente di usare gli elementi d'estensione per specificare la concreta grammatica dei messaggi d'input, output e fault, e delle operazioni logiche, inerente a uno specifico protocollo di messaging. Tuttavia, nell'elemento *binding* non devono essere specificate informazioni inerenti agli indirizzi di rete coinvolti.

Nell'elemento ***wsdl:port*** viene definito l'endpoint specificando un singolo indirizzo per una determinata negoziazione. La locazione fisica dell'endpoint è il valore di un appropriato attributo dell'elemento d'estensione inerente a *wsdl:port*. Essa, in generale, coincide con l'indirizzo del server SOAP del service provider che consente di negoziare (binding) con un'implementazione del servizio.

Il nome del *port* deve essere un identificatore unico nell'ambito dell'intero documento WSDL. Il riferimento allo specifico *wsdl:binding* avviene mediante l'attributo binding.

I *wsdl:port* vengono raggruppati nell'elemento ***wsdl:service*** che rappresenta il servizio. Ad esempio, supponiamo di aver lanciato un servizio su una data interfaccia di connessione (*port*). Ad un certo punto ci accorgiamo che le performance del servizio calano, probabilmente perché il server SOAP che fornisce l'implementazione del servizio è intasato. Allora, possiamo decidere di cambiare server, ad esempio utilizzare un mirror che si trova in qualche altro posto. A tale proposito, è sufficiente cambiare semplicemente l'interfaccia di connessione. Infatti l'elemento *wsdl:service* garantisce che tutti i *port* che contiene corrispondono al medesimo servizio. Ciò che cambia può essere l'implementazione del servizio, il tipo di negoziazione, ed eventualmente il provider del servizio, ma non le funzionalità del servizio.

In altre parole, i *port* consentono di scegliere una particolare implementazione del servizio (accessibile mediante l'endpoint) con riferimento ad uno specifico protocollo di rete, usato per negoziare con l'implementazione del servizio.

Infine, si può utilizzare ***wsdl:documentation*** (non riportato in figura 4.1 per non complicare il layout del diagramma) in tutti gli elementi WSDL per specificare una nota o un'osservazione in riferimento al particolare contesto.

3. SOAP Binding.

Come descritto in precedenza, l'elemento <wsdl:binding> viene utilizzato per trasformare i tipi di dati, messaggi e operazioni astratti in una rappresentazione concreta di messaggio di rete. Quest'elemento è significativamente legato al formato di protocollo utilizzato. Nel caso di SOAP bisognerà trasformare le informazioni astratte in un messaggio SOAP adeguato a negoziare con l'implementazione remota del servizio.

WSDL è stato progettato per essere altamente estensibile, e in teoria qualunque protocollo può essere usato per eseguire il binding dei messaggi astratti. Tuttavia, oltre al protocollo SOAP, attualmente, sono definiti solo gli elementi di binding per HTTP GET/POST e MIME, che non sono stati discussi in questo lavoro.

WSDL fornisce un meccanismo per specificare il binding per SOAP. Consente di specificare un indirizzo per l'endpoint SOAP, l'URI per il campo SOAPAction dell'header del messaggio HTTP, e una lista di elementi header da trasmettere come parte dell'envelope del messaggio SOAP.

Gli elementi che consentono di effettuare il binding SOAP sono gli elementi WSDL estensibili.

Nel listato 4.2 è mostrato l'elemento ***soap:binding*** (obbligatorio) e i relativi attributi (il punto interrogativo indica che l'attributo, o l'elemento sono opzionali e non ripetibili, molteplicità 0..1).

```
<wsdl:binding …>
    <soap:binding transport=”uri”? style=”rpc”?>
</wsdl:binding>
```

Listato 4.2 – *Definizione dell'elemento soap:binding.*

Il valore dell'attributo *style* descrive il contenuto del messaggio. Ad esempio, il valore *"rpc"* indica che il messaggio contiene un'invocazione ad un metodo remoto, mentre il valore *"document"* (valore di default per l'attributo style) segnala che il contenuto del messaggio è un documento.

Il valore dell'attributo *transport* è un URI che deve indicare il protocollo di trasporto utilizzato da SOAP, ad esempio, nel caso di HTTP può essere usato il namespace "http://schemas.xmlsoap.org/soap/http".

Per specificare come le operazioni devono apparire nel messaggio SOAP viene utilizzato l'elemento ***soap:operation*** (opzionale), illustrato nel listato 4.3.

```
<wsdl:operation …>
    <soap:operation soapAction=”uri”? style=”rpc|document”?>?
</wsdl:operation>
```

Listato 4.3 – Definizione dell'elemento soap:operation.

L'attributo *style* indica il tipo d'operazione, ad esempio RPC-oriented (il messaggio contiene parametri e ritorna valori) o document-oriented (il messaggio contiene documenti).

L'attributo *soapAction* specifica il valore del campo SOAPAction nell'header del messaggio SOAP-HTTP per una determinata

operazione. Questo valore può essere usato per indicare l'urn del servizio.

Le parti di un messaggio del documento WSDL possono essere viste o come definizioni astratte di tipi di dati oppure come definizioni concrete. Nel primo caso i tipi astratti devono essere codificati mediante un sistema di regole definite da uno specifico stile di codifica. Analogamente alla specifica SOAP, la specifica WSDL utilizza una lista di URI per identificare un certo stile di codifica. In particolare, se lo stile di codifica consente delle variazioni nel formato del messaggio per un dato insieme di tipi astratti (come quello di SOAP), allora questi possono essere utilizzati senza subire alcuna trasformazione. L'eventuale interpretazione delle variazioni è a cura del destinatario del messaggio. Invece, i messaggi WSDL dovranno conformarsi esattamente allo schema imposto dallo stile di codifica, se questo non consente variazioni. In questo caso, chi scrive il documento WSDL deve definire messaggi concreti.

L'elemento ***soap:body***, mostrato nel listato 4.4, fornisce le informazioni necessarie per assemblare le differenti parti dei messaggi WSDL dentro il body del messaggio SOAP. Il modo in cui ciò avviene dipende essenzialmente dal tipo di operazione. Con riferimento ad un'invocazione di un metodo remoto, ogni parte (*wsdl:part*) è vista come un parametro o un valore di ritorno e appare nel payload del messaggio SOAP sotto un elemento indipendente (o wrapper, figli diretti dell'elemento body, vedi paragrafo 3.6.2).

```
<wsdl:operation …>
    <wsdl:input>
        <soap:body parts=”nomePart”? use=”literal|encoded”?
                                    encodingStyle=”uri-list”?
namespace=”uri”?>
    </wsdl:input>
    <wsdl:output>
        <soap:body parts=”nomePart”? use=”literal|encoded”?
                                    encodingStyle=”uri-list”?
namespace=”uri”?>
    </wsdl:output>
</wsdl:operazion>
```

Listato 4.4 – *Definizione dell'elemento soap:body.*

L'elemento wrapper deve avere lo stesso nome dell'operazione e il suo namespace deve corrispondere a quello indicato dall'attributo namespace dell'elemento *soap:body*.

Se invece lo stile dell'operazione è document-oriented, allora le parti del messaggio vengono inserite direttamente sotto l'elemento body del messaggio SOAP come elementi indipendenti.

Per indicare che i *wsdl:part* dei massaggi sono codificati usando specifiche regole di codifica si usa l'attributo *use* col valore "encoded". In questo caso le parti del messaggio referenziano tipi astratti (ad esempio quelli di XML Schema) attraverso l'attributo *type*. Questi tipi di dati astratti sono usati per produrre un concreto messaggio applicando lo stile di codifica indicato dal valore dell'attributo *encodingStyle* (le variazione nella codifica sono particolarmente significatice per tipi di dati complessi). Per effettuare la codifica sono

necessarie le informazioni inerenti agli attributi *name* e *type* degli elementi *wdl:part* dei messaggi astratti.

Per indicare, invece, che i *wsdl:part* definiscono un concreto schema di messaggio si usa il valore "literal", in questo caso le parti devono referenziare un schema (wsdl:schema) concreto usando i riferimenti *element* o *type*. Gli elementi riferiti appaiono direttamente nel messaggio SOAP, mentre i tipi riferiti appaiono come attributi *type* degli elementi *accessor* (vedi paragrafo 3.6.1) nel corpo di uno *struct* del body SOAP.

Infine, per fornire un indirizzo all'elemento *wsdl:port* viene usato l'elemento ***soap:address***, come mostrato nel listato 4.5.

```
<wsdl:service …>
  <wsdl:port…>
     <soap:address location="uri"/>
  </wsdl:port>
</wsdl:service>
```

Listato 4.5 – *Definizione dell'elemento soap:address.*

La specifica WSDL definisce un meccanismo di binding anche per i metodi GET e POST di HTTP. Questo consente di descrivere l'interazione tra un Web browser e un Web Site. Tuttavia, la trattazione di questa parte della specifica esula dagli obiettivi di questo lavoro di studio.

4. *Esempio di descrizione di base di un Web Service.*

Di seguito è riportato un esempio di descrizione dell'interfaccia del servizio Web (vedi paragrafo 2.2.3) Calc. Questo è un una semplice calcolatrice che esegue le due operazioni fondamentali di somma e moltiplicazione su interi e restituisce il risultato. La sua interfaccia nel linguaggio di programmazione java [25]è:

```
interface Calc{
      int getSum(int num1,int num2);
      int getMul(int num1,int num2);
}
```

Listato 4.1 – *Interfaccia del servizio Calc espressa mediante linguaggio Java.*

Per invocare le due funzionalità del servizio viene utilizzato il protocollo SOAP.

Il documento WSDL, mostrato nel listato 4.2, è stato generato automaticamente dalla classe *Cal.class* (classe Java) mediante il tool *genWSDL* del prodotto WSTK 2.23 dell'IBM,

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Calc"
 targetNamespace="http://www.Calc.wsdl.com/wrapperedService"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:cal="http://www.Calc.wsdl.com/wrapperedService"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<message name="IngetMulRequest">
 <part name="meth1_inType1" type="xsd:int"/>
 <part name="meth1_inType2" type="xsd:int"/>
</message>

<message name="OutgetMulResponse">
 <part name="meth1_outType" type="xsd:int"/>
</message>

<message name="IngetSumRequest">
 <part name="meth2_inType1" type="xsd:int"/>
 <part name="meth2_inType2" type="xsd:int"/>
</message>

<message name="OutgetSumResponse">
 <part name="meth2_outType" type="xsd:int"/>
</message>

<portType name="Calc">
 <operation name="getMul">
  <input message="IngetMulRequest"/>
  <output message="OutgetMulResponse"/>
 </operation>

 <operation name="getSum">
  <input message="IngetSumRequest"/>
  <output message="OutgetSumResponse"/>
 </operation>
</portType>

<binding name="CalcBinding" type="Calc">
```

```
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getMul">
          <soap:operation soapAction="urn:Calc"/>
          <input>
              <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                          namespace="urn:Calc" use="encoded"/>
          </input>
          <output>
              <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                          namespace="urn:Calc" use="encoded"/>
          </output>
    </operation>
    <operation name="getSum">
          <soap:operation soapAction="urn:Calc"/>
          <input>
              <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                          namespace="urn:Calc" use="encoded"/>
          </input>
          <output>
              <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                          namespace="urn:Calc" use="encoded"/>
        </output>
     </operation>
</binding>

</definitions>
```

Listato 4.2 – *Rappresentazione dell'interfaccia del servizio Calc mediante elementi del linguaggio WSDL.*

In questo documento non è presente la sezione per la definizione dei tipi astratti in quanto i parametri e i valori di ritorno delle due funzioni

del componente Calc sono tipi predefiniti di XML Schema, ovvero xsd:int.

Gli elementi *wsdl:message* specificano, attraverso gli elementi *wsdl:part*, i tipi dei dati che vengono scambiati nelle chiamate ai metodi remoti e perciò sono usati per definire i parametri d'input ed output delle operazione *Sum* e *Mul*. I nomi dei messaggi vengono usati come riferimenti dagli elementi *wsdl:operation*. Questi elementi consentono di rappresentare le operazioni del tipo request/respose, utilizzando in maniera opportuna i sottolementi *wsdl:input* e *wsdl:utput* associati ai nomi dei messaggi. I nomi delle operazioni devono coincidere con i nomi dei metodi messi a disposizione dal componente Calc. Ad esempio, l'operazione *getSum* è un'operazione request/response che invia al servizio il messaggio *IngetSumRequest* e si mette in attesa del messaggio di risposta *OutgetSumResponse*. I flussi input e output sono con riferimento all'endpoint, ovvero al servizio.

Dal valore dell'attributo style dell'elemento *soap:binding* si evince che si tratta di una RPC. Negli sottoelementi input e output dell'elemento *soap:body* vengono definiti lo stile di codifica adottato dal protocollo SOAP, il namespace per la dichiarazione delle operazioni dentro il messaggio SOAP, e infine la modalità in cui vengono usati i messaggi, astratta o concreta.

Ad esempio, seguendo il binding descritto dal documento in relazione all'operazione *getSum* si ottiene il mossaggio SOAP mostrato nel listato 4.3 (si osservi che il messaggio non può essere ancora incluso in un POST HTTP perché si suppone non ancora nota la descrizione dell'interfaccia).

```
SOAPAction=”urn:Calc”

<SOAPENV:Envelope
xmlns:SOAPENV=”http://schemas.xmlsoap.org/soap/envelope/”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance “
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAPENV:Body>
        <m:getSum xmlns=”urn:Calc”>
          <m:meth2_inType1 xsi:type=”xsd:int”>
            1
          </m:meth2_inType1>
          <m:meth2_inType2 xsi:type=”xsd:int”>
            2
         </m:meth2_inType2>
      </m:getSum>
  </SOAP:ENV:Body>
</SOAP:ENV:Envelope>
```

Listato 4.3 – *Messaggio SOAP relativo all'operazione getSum.*

Nel listato 4.4 invece viene mostata la definizione dell'interfaccia del servizio Web Calc.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Calc"
 targetNamespace="http://www.Calc.wsdl.com/wrapperedService"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns=" http://www.Calc.wsdl.com/wrapperedService"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<import
   location="http://localhost:8080/wsdl/Calc_Service-
interface.wsdl"
   namespace="http://www.calcservice.com/Calc-interface">
</import>

<service name="Calc ">
 <documentation>IBM WSTK 2.0 generated service definition
```

```
    file</documentation>
  <port binding="CalcBinding" name="CalcPort">
    <soap:address
          location="http://localhost:8080/soap/servlet/rpcrouter"/>
  </port>
</service>

</definitions>
```

Listato 4.4 – Definizione dell'implementazione del servizio

Il primo elemento che si incontra è *import* che punta al documento della definizione dell'intefaccia del servizio.

Il valore *CalcBinding* dell'attributo *binding* dell'elemento *port* è un riferimento all'elemento *binding* della descrizione dell'interfaccia a cui viene associato l'endpoint *http://localhost:8080/soap/servlet/rpcrouter*. Questo URL corrisponde al server SOAP Apache (una servlet) [26] che processa la richiesta SOAP e attiva il servizio richiesto. Per conoscere di quale servizio si tratta il server SOAP utilizza il valore *urn:Calc* definito nel namespace dell'operazione nella descrizione dell'interfaccia. Con questo valore entra in una tabella dei descrittori dei servizi (deployment description) ed estrae il nome del componente che implementa il servizio, ammesso che questo esista.

In generale, per ogni interfaccia di un servizio esistono più definizione dell'implementazione.

Ad esempio, si supponga che un'industria per la standirdazizone delle interfacce ha proposto una data definizione dell'interfaccia (in WSDL) di un servizio per il controllo del numero della carda di credito. Un

service provider può scegliere di sviluppare un Web Service che implementa tale interfaccia, anziché progettarne una nuova. Una volta sviluppato il servizio, un certo provider, diciamo P1, mette a disposizione, ad un eventuale richiedente, un documento WSDL per la descrizione di tale implementazione, definendo in esso un dato endpoint. Il richiedente, un service requestor, in possesso della descrizione dell'interfaccia, può usare la definizione dell'implementazione per utilizzare il servizio del provider P1. Tuttavia, se la performance di tale servizio non sono soddisfacenti, allora il service requestor può decidere di cambiare implementazione del servizio rivolgendosi ad un altro provider P2 a cui richiede solo l'implementazione del servizio, in quanto è già in possesso della definizione dell'interfaccia.

Ovviamente, il provider P2 deve fornire un servizio la cui interfaccia è conforme a quella standard, fornita dall'industria.

La definizione dell'interfaccia del servizio insieme con la definizione dell'implementazione del servizio costituisce una completa definizione WSDL del servizio (vedi 2.2.3). Questa coppia di descrizioni contengono informazioni sufficienti fornire al service requestor tutta la conoscenza necessaria per invocare e per interagire con il Web Service, in altre parole, per creare il proxy.

Capitolo V

UDDI
(UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION)

1. Introduzione.

Negli ultimi anni si è assistito ad una forte crescita del B2B (Business-to-Business). Internet ha offerto alle aziende l'opportunità di trovare nuovi clienti, di fornire nuovi tipi di servizi, e conseguentemente di incrementare i propri profitti e di estendere i propri orizzonti commerciali e competitivi.

Per aprirsi a questo nuovo mondo commerciale le organizzazioni di tutto il mondo hanno adottato soluzioni che consentissero l'integrazione delle proprie infrastrutture tecnologiche e quindi dei propri servizi. Eppure, nonostante questi sforzi, l'enorme potenziale del B2B è ancora soffocato dalla difficoltà di scoprire facilmente e velocemente nuovi partner, e di comunicare con questi indipendentemente dalle piattaforme su cui risiedono i propri servizi e dalle relative infrastrutture di rete. Questo è il risultato dell'adozione di soluzioni e approcci improntati su requisiti personali e proprietari per l'integrazione dei servizi, di cui le aziende si sono serviti in questi anni per portare i propri affari in rete [32]. D'altronde, ognuno ha il proprio

modo di lavorare, e non si può pretendere che lo cambi solo per interagire con un'altra azienda.

Se non è possibile uniformare processi e tecnologie, si può sempre cercare di realizzare un'infrastruttura comune a tutte le parti in gioco, una concezione globale della condivisione delle informazione relative alle aziende, ai loro prodotti, e dei canali necessari per collegarsi con i servizi dei partner commerciali. Il principio è quello delle pagine gialle, ovvero un grande elenco distribuito che permette ad ogni società di trovare fornitori, partner e clienti in modo semplice ed inequivocabile. La soluzione è UDDI, che si propone, essenzialmente, tre obiettivi: consentire a qualunque azienda di pubblicare una descrizione di se stessa e dei servizi che è in grado di fornire, in particolare Web Services; il secondo scopo è quello di facilitare il ritrovamento dei servizi desiderati e i relativi fornitori; il terzo è l'integrazione tra le società che operano nel settore B2B e l'interoperabilità fra i vari servizi offerti [30].

Per raggiungere questi obiettivi è richiesta un'architettura aperta, basata su registri che usano uno standard per la pubblicazione delle informazioni, per la scoperta delle stesse e in cui i servizi possono essere facilmente raggiunti e consumati da qualunque posizione. Tale infrastruttura tecnologica dovrà essere calata sull'esistente e farne da collante universale.

L'adozione di un Web Service framework e di un service public registry consentiranno a compratori e venditore d'ogni parte del mondo di condividere le proprie informazioni, di accedere ai Web Services e nello stesso tempo senza intaccare la differenziazione

tecnologica che esiste in ambito intranet, ciascuna azienda può continuare ad usare la tecnologia che ritiene più vantaggiosa.

In questo nuovo scenario, le informazioni da pubblicate sulla rete devono essere strutturate in modo da consentire il reperimento dei partner in tempi brevi, devono definire la modalità con cui collegarsi ai loro servizi, e soprattutto devono rispettare una specifica di pubblicazione e di ricerca che sia comunemente accettata.

UDDI è gestito dal consorzio UDDI (www.uddi.org) costituito da oltre 80 compagnie, inclusi business leader come Merrill Lynch (*http://askmerrill.ml.com/*) and Cargill (www.cargill.com/), technology leader come IBM, Microsoft, and Sun Microsystems; e compagnie che lavorano nell'innovativo settore del B2B (e-marketplace) come Ariba (www.ariba.com), CommerceOne (www.commerceone.com), e VerticalNet (www.verticalnet.com). La tecnologia UDDI è quindi proprietaria di questo consorzio, ed anche se si basa su tecnologie aperte, definite dal W3C (come XML e HTTP e SOAP), è una tecnologia attualmente non "propriamente aperta".

2. UDDI Business Registry.

Come impegno preso dalle maggiore industrie di software verso la definizione delle specifiche UDDI, è stato lanciato da Microsoft, IBM e Ariba, nell'ottobre del 2000, il primo registro pubblico operante sul Web, che implementa appunto tale specifiche. L'UDDI Business Registry è una directory Internet di informazioni su aziende e di

applicazioni che vengono esposte come Web Services ai partner commerciali e ai customer che le consumano. Ciascun' azienda che vuole pubblicare le proprie informazioni sull'UDDI Business Registry deve necessariamente registrarsi. Dopo la registrazione riceverà un identificativo globalmente unico.

Le informazioni sulle compagnie registrate saranno liberamente accessibili da potenziali consumatori.

L'UDDI Business Registry è un elenco logicamente centralizzato ma fisicamente distribuito su più nodi che forniscono le funzioni necessarie sia per la gestione sia per la consultazione dei dati in essi contenuti. Un' organizzazione che gestisce un nodo viene definita 'operatore'.

Le motivazioni che hanno portato alla creazione di un registro distribuito non sono solo tecniche, ma sono anche orientate a rafforzare la compatibilità tra i diversi produttori dimostrando un certo grado di apertura l'uno con l'altro, evitando in questo modo di legare UDDI alla tecnologia specifica di un singolo produttore.

Per interagire col nodo della Microsoft si può visitare il sito Web *http://uddi.microsoft.com/default.aspx*. La registrazione è completamente gratuita, e si possono cercare le interfacce di alcuni Web Services. I servizi sono classificati secondo gli standard NAICS [34], UNSPSC [35], ISO 3166 Geographic Taxonomy [36] ed altri.

3. Strutture dati.

Un Web Service è utile quando può essere facilmente scoperto, localizzato e utilizzato. I registri UDDI possono essere utilizzati per fornire un meccanismo comune per la scoperta dei servizi sul World Wide Web. Il framework UDDI utilizza la tecnologia cross-platform XML per descrivere i servizi esposti da una qualunque azienda o società che ospita servizi. In questo modo, la descrizione delle informazioni trovate in un registro UDDI sono fornite in un formato neutrale e indipendente da particolari requisiti di piattaforma e tecnologici.

Le informazioni registrate presso il registro consistono di cinque strutture dati: entità aziendale (businessEntity) [27], servizi (businessService), specifiche tecniche (bindingTemplate), modelli (tModel) e relazioni tra entità aziendali (publisherAssertion). Questa divisione strutturale consente di localizzare più facilmente le informazioni, e di comprendere meglio i dati che un'azienda pubblica al fine di rendere reperibili i propri servizi. Nella figura 5.1 vengono illustrati le cinque strutture dati fondamentali, definite nella specifica UDDI API versione 2.0.

Una particolare istanza di un fatto o di un insieme di fatti, quali la descrizione logica e tecnica del servizio, è espressa in formato XML e rappresenta un'entità separata all'interno del registro UDDI.

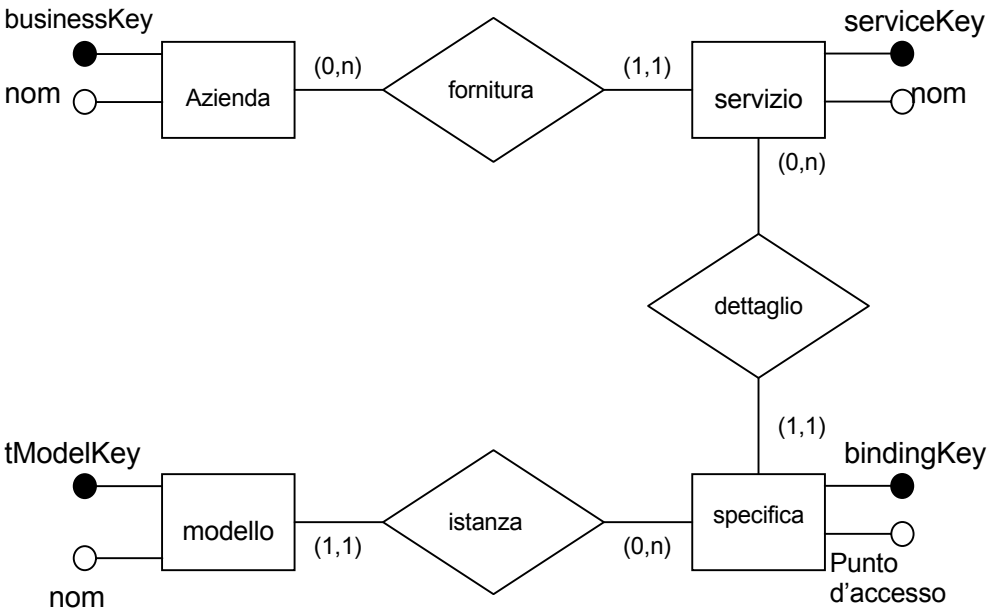


Figura 5.1 – *Diagramma Entità-Relazione relativo alle entità fondamentali del registro UDDI.*

Le informazioni di un'azienda, i suoi servizi e dati tecnici sono mantenuti separati e localizzati mediante una chiave globale. Un registro UDDI assegna questi identificatori unici quando le informazioni sono salvate per la prima volta, e usati successivamente come chiavi per accedere alle informazioni al momento della richiesta.

L'identificatore deve essere un indirizzo universalmente unico, chiamato Universally Unique ID (UUID), e in pratica è una stringa di 32 cifre esadecimali (128 bit) generata mediante un algoritmo[37] sufficientemente preciso da evitare di generare duplicati. L'UUID è raggruppato in cinque sequenze secondo il formato xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, ovvero 8-4-4-4-12.

L'uso del metalinguaggio XML per la definizione delle strutture dati, comporta, inevitabilmente, una strutturazione gerarchica ad albero delle informazioni gestite dal registro UDDI. Ad esempio, la struttura ***businessEntity*** contiene almeno una struttura ***businessService*** che a sua volta contiene istanze della struttura ***bindingTemplate*** [27]. Una struttura *bindingTemplate* invece referenzia specifiche istanze della struttura ***tModel*** [27].

L'elemento *businessEntity* racchiude le informazioni sull'azienda, come le informazioni principali (per esempio, il nome dell'azienda e le informazioni di contatto) e le informazioni di classificazione (per esempio, il tipo di settore in cui opera l'azienda). Inoltre, contiene una collezione di *businessService*, una per ciascun Web Service che l'azienda desidera pubblicare. A sua volta ciascun *businessService* contiene un set di informazioni tecniche e descrittive, *bindingTemplate*, di un particolare Web Service.

Un *bindingTemplate* contiene informazioni d'accesso (per esempio, l'endpoint che consente l'accesso al servizio) e descrive come il *businessService* usa varie specifiche tecniche. Una specifica tecnica, ad esempio relativa a una particolare implementazione del servizio, è interpretata mediante un *tModel*. Questo è un meta-modello in grado di modellare parecchi concetti, come, per esempio, un tipo di servizio, un tipo di tecnologia impiegata e anche una tassonomia.

3.1. Struttura businessEntity

Dal punto di vista XML, *businessEntity* è l'elemento di massimo livello (top-level element) del documento di definizione dei fatti inerenti

all'azienda o entità che pubblica. Il listato 5.1 mostra lo schema XML dell'elemento *businessEntity*, che può essere usato anche per convalidare tale struttura.

```
<element name = "businessEntity">
  <complexType>
      <sequence>
         <element ref = "discoveryURLs" minOccurs = "0"/>
         <element ref = "name" maxOccurs = "unbounded"/>
         <element ref = "description" minOccurs = "0" maxOccurs =
                                                    "unbounded"/>
         <element ref = "contacts" minOccurs = "0"/>
         <element ref = "businessServices" minOccurs = "0"/>
         <element ref = "identifierBag" minOccurs = "0"/>
         <element ref = "categoryBag" minOccurs = "0"/>
      </sequence>
      <attribute ref = "businessKey" use = "required"/>
      <attribute ref = "operator"/>
      <attribute ref = "authorizedName"/>
  </complexType>
</element>
```

Listato 5.1 – *Schema dell'elemento* ***businessEntity***.

Da questo schema si evince che l'elemento *buisinessEntity* contiene sette sottoelementi e tre attributi. Questi elementi descrivono altrettante strutture dati, le cui occorrenze all'interno dell'elemento radice sono fissate degli attributi minOccurs e maxOccurs (vedi paragrafo 2.5).

Nella tabella 5.1 sono riportate brevi descrizioni degli attributi e degli elementi impiegati. Tutti i campi in grassetto sono obbligatori.

Nome Campo	*Descrizione*	*Tipo Dato*
businessKey	Attributo obbligatorio che identifica in maniera univoca una struttura businessEntity	UUID
authorizedName	Attributo che indica il nome dell'individuo che gestisce la copia originale dell'informazione.	Stringa
operator	Attributo che indica il nome dell'operatore del registro UDDI.	Stringa
discoveryURLs	E' un elemento opzionale e modella una lista di URL che puntano a documenti che possono essere indirizzati da un motore di ricerca.	struttura
Name	Elemento richiesto e ripetibile. Rappresenta un nome umanamente leggibile dell'entità che pubblica.	Stringa
description	Elemento reiterabile, per brevi descrizioni sull'azienda.	Stringa
contacts	E' una struttura rappresentata da una lista di informazioni che consentono di contattare l'azienda.	Struttura
businessService	Si riferisce a una struttura businessService che descrive il servizio offerto dall'azienda dal punto di vista logico.	Struttura
identifierBag	E' una lista opzionale di coppie nome-valore che possono essere usati come identificativi all'interno delle informazioni relative all'entità.	Struttura
categoryBag	Rappresenta una lista di coppie nome-valore che possono essere usate per categorizzare entità, servizi e modelli	Struttura

Tabella 5.1 – Descrizione degli attributi ed elementi relativi all'entità aziendale.

Nella figura 5.2 è riportato una vista dello schema E-R concettuale del registro UDDI rispetto all'entità *businessEntity*. Lo schema è stato ottenuto eseguendo un reverse engineering sul database utilizzato da un'implementazione reale del registro UDDI, la stessa impiegata nel caso di studio. Il tool adoperato per il reverse engineering è DataArchitect 6.1 di Sybase. Il sistema utilizza la seguente notazione grafica: tutti gli attributi vengono rappresentati direttamente dentro le entità e le chiavi primarie sono sottolineate; le linee rappresentano relazioni e le coppie di numeri racchiusi dalle parentesi quadre esprimono i vincoli di cardinalità; un pallino su un lato della relazione indica che la partecipazione all'associazione da parte dell'entità sullo stesso lato è facoltativa, mentre una piccola linea ortogonale indica una partecipazione obbligatoria; una linea terminante con una specie di tridente indica una relazione uno a molti; un numero accanto al nome di un'entità indica che quell'entità è un sinonimo, riproducendo lo stesso oggetto in differenti posti nel modello ne migliora la leggibilità.

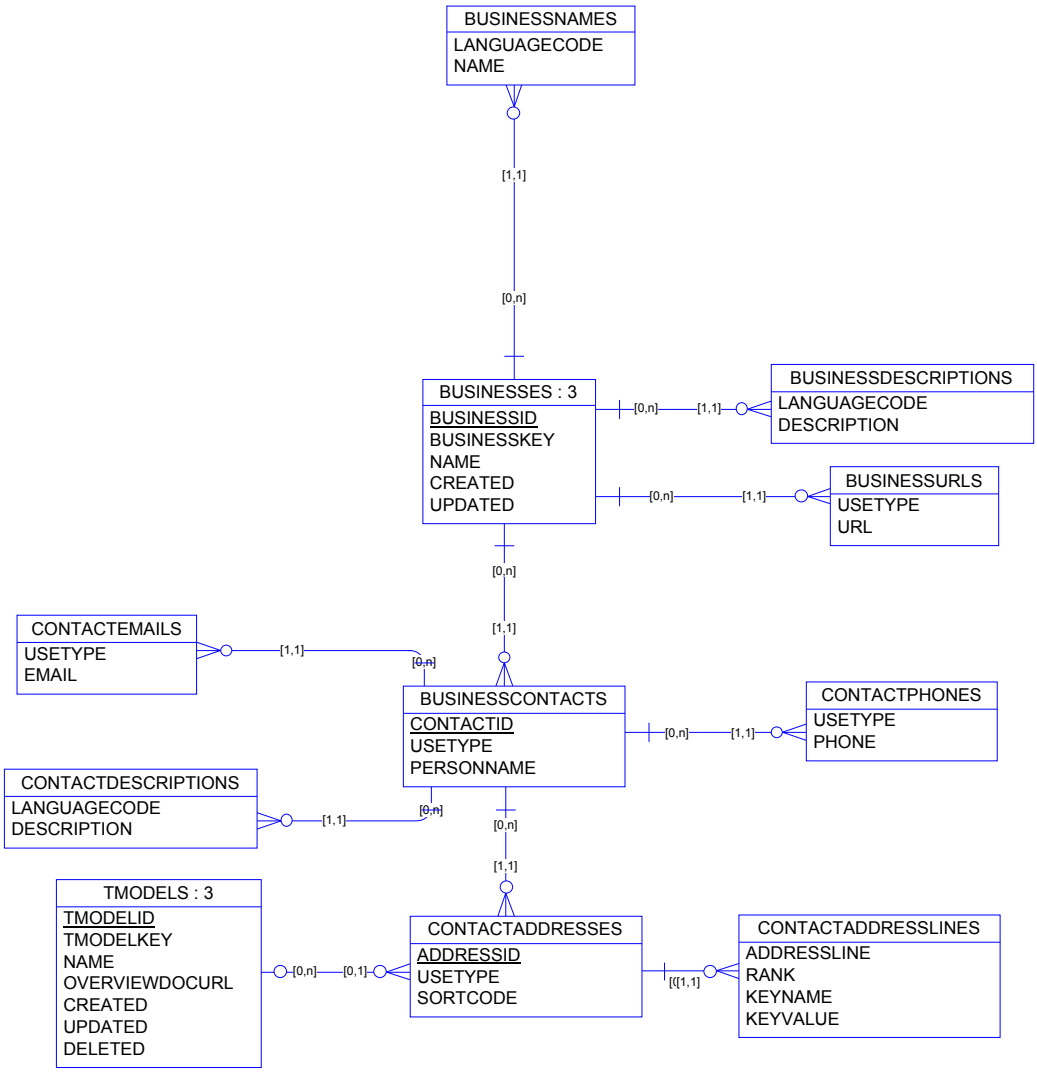


Figura 5.2 – *Diagramma E-R (notazione alternativa) in riferimento all'entità aziendale.*

Nella figura 5.3 è riportato il diagramma di classe concettuale della struttura *contacts*, secondo la notazione UML.

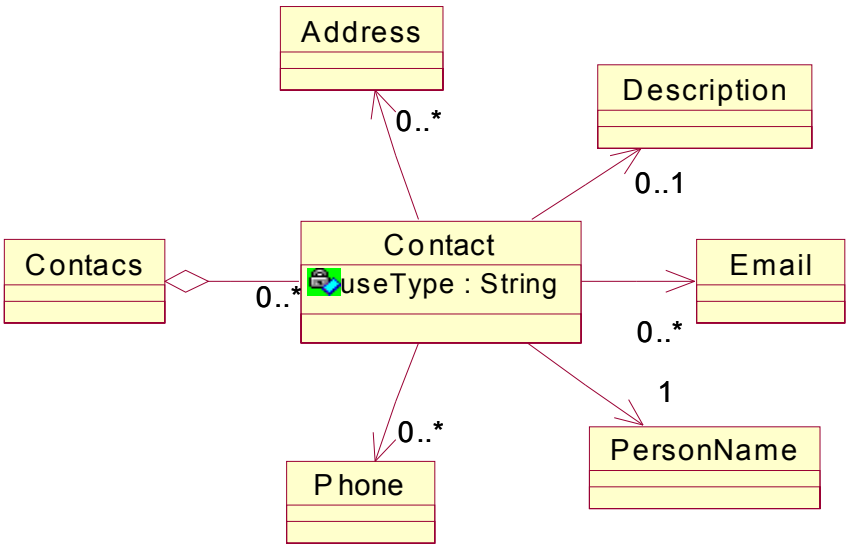


Figura 5.3 – *Diagramma concettuale di classe della struttura contacts, secondo la notazione grafica UML.*

dove useType è un attributo usato per descrivere il tipo di contatto, ad esempio "contatto tecnico", etc. Il diagramma concettuale della struttura address è riportato in figura 5.4.

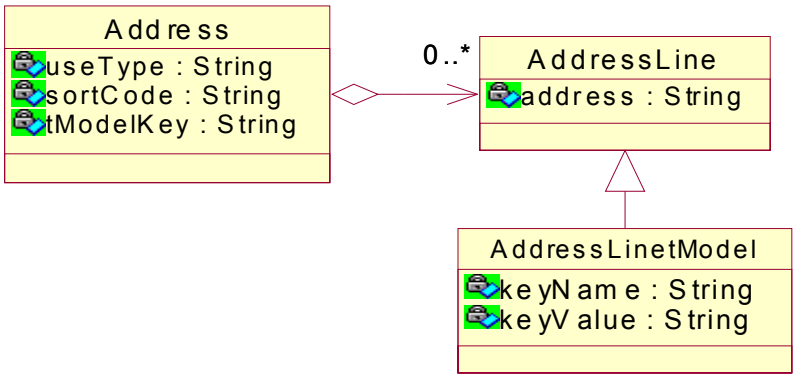


Figura 5.4 – *Digramma di classe dell'indirizzo.*

L'attributo *useType* definisce il tipo di indirizzo, mentre sortCode il tipo di ordinamento con cui si vuole visualizzare la lista delle linee indirizzi.

Le coppie nome-valore dell'*addressLine* possono essere utilizzare per modellare un particolare indirizzo, in tal caso deve essere obbligatoriamente presente nell'*address* l'identificativo del modello che consente di interpretare tale coppie. Per maggiori chiarimenti rimando al paragrafo 5.4, dove verrà definita la struttura *tModel*.

3.2. Struttura businessService.

Descrive il servizio pubblicato dall'azienda da una prospettiva logica. Ogni *businessService* è figlio di una singola struttura *businessEntity*. L'identità dell'azienda fornitrice è ottenuta esaminando il valore di *businessKey* del servizio che referenzia la relativa struttura *businessEntity*.

Il listato 5.2 mostra lo schema XML dell'elemento *businessService* [27].

```
<element name = "businessService">
 <complexType>
    <sequence>
        <element ref = "name" maxOccurs = "unbounded"/>
        <element ref = "description" minOccurs = "0" maxOccurs =
          "unbounded"/>
        <element ref = "bindingTemplates"/>
       <element ref = "categoryBag" minOccurs = "0"/>
    </sequence>
    <attribute ref = "serviceKey" use = "required"/>
    <attribute ref = "businessKey"/>
 </complexType>
</element>
```

Listato 5.2 – *Schema dell'elemento* ***businessService.***

L'attributo ***serviceKey*** è obbligatorio ed è, ovviamente, un UUID, invece, *businessKey* è un attributo opzionale qualora la struttura *businessService* fosse contenuta completamente in *businessEntity,* e obbligatorio se il servizio è descritto al di fuori della struttura relativa all'entità aziendale corrispondente. Ad esempio, quando si vuole pubblicare un nuovo servizio è necessaria la chiave che identifica il fornitore.

Il nome del servizio è obbligatorio e può essere replicato in diverse lingue. L'elemento ***bindingTemplates*** contiene una lista di specifiche tecniche relative al servizio offerto. I due elementi opzionali *description* e *categoryBag* vendono utilizzati per descrivere il servizio e associarlo a una specifica categoria.

Nella figura 5.5 è riportato una porzione di schema E-R (in notazione grafica alternativa, vedi nel paragrafo 3.1) del registro in riferimento alla struttura *businessService*.

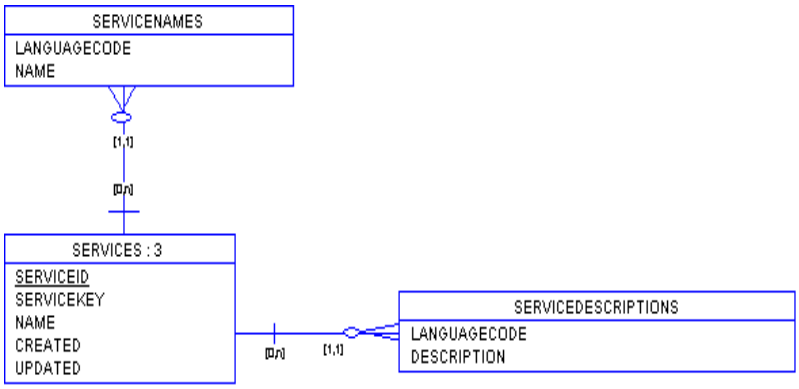


Figura 5.5 – *Diagramma E-R con riferimento ai servizi pubblicati da un'azienda.*

3.3. Struttura bindingTemplate.

Le strutture bindingTemplate forniscono un supporto alla determinazione dell'entry point che consente di raggiungere il Web Service e alla descrizione di caratteristiche tecniche uniche di una data implementazione. Il listato 5.3 è lo schema di tale struttura dati.

```
<element name = "bindingTemplate">
  <complexType>
      <sequence>
          <element ref = "description" minOccurs = "0" maxOccurs =
            "unbounded"/>
          <choice>
              <element ref = "accessPoint" minOccurs = "0"/>
              <element ref = "hostingRedirector" minOccurs = "0"/>
          </choice>
          <element ref = "tModelInstanceDetails"/>
      </sequence>
      <attribute ref = "bindingKey" use = "required"/>
      <attribute ref = "serviceKey"/>
 </complexType>
</element>
```

Listato 5.3 – *Schema dell'elemento* ***bindingTemplate.***

I campi obbligatori sono due elementi e un attributo. L'attributo è *bindingKey*, il solito identificatore UUID. Solo uno dei due elementi ***accessPoint*** e ***hostingRedirector*** è obbligatorio. Il primo viene utilizzato per definire, mediante una stringa, come accedere al servizio. Le modalità d'accesso sono sette: via posta elettronica (mailto:), via HTTP (http:) o HTTPS (https:), via FTP (ftp:), via fax (fax:) o telefono

(phone:), oppure attraverso un altro mezzo (other:) che dovrà essere specificato mediante un modello.

L'elemento *hostingRedirector* possiede un attributo *bindingKey* che viene utilizzato per referenziare un altro *bindingTemplate*. Questo elemento è particolarmente utile quando più servizi fanno riferimento allo stesso punto d'accesso.

Il secondo elemento obbligatorio è ***tModelInstanceDetails*** che, in generale, viene utilizzato per referenziare un particolare modello di negoziazione con il servizio. Questa informazione può essere usata, ad esempio, in fase di ricerca per individuare quali servizi sono compatibili tra loro, che sono, appunto, tutti quelli le cui specifiche tecniche fanno riferimento allo stesso modello di negoziazione [32].

L'attributo opzionale *serviceKey* ha lo stesso ruolo di *businessKey* in *businessService*. L'elemento opzionale *description* fornisce una breve descrizione della specifica tecnica.

In figura 5.6 è mostrato il diagramma E-R in riferimento alla specifica tecnica.

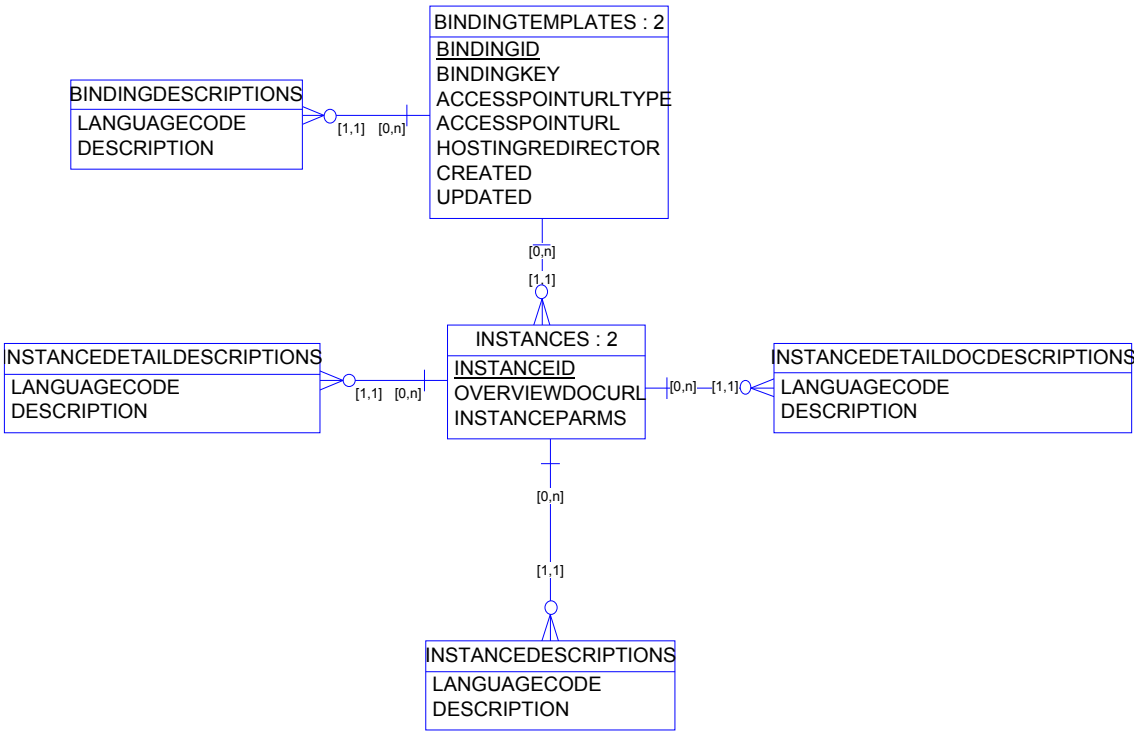


Figura 5.6 – Modello E-R concettuale con riferimento alla specifica tecnica dei servizi.

3.4. La struttura tModel.

L'azienda utilizza le strutture dati viste in precedenza per definire cosa offre e come. Se però, si vuole integrare i servizi con i propri partner è essenziale che non ci siano equivoci, e che ogni campo informativo sia estremamente chiaro. A tale proposito, il registro UDDI utilizza un approccio simile ai metadati, ovvero informazioni sui dati. Ad esempio, il seguente codice 39-347-4545-8 potrebbe essere un numero telefonico con prefisso internazionale +39, oppure il codice d'identificazione di un prodotto in un catalogo [32]. Non sapremo mai il vero significato di tale codice se non si associa ad esso un modello che indichi come utilizzarlo. Ad esempio, supponiamo che esso faccia

riferimento al modello isbn, allora tutto torna chiaro. Si tratta di un codice per identificare una pubblicazione, che consente di ordinare un libro via Internet, o telefonicamente senza dover specificare autore, titolo o casa editrice.

Il registro UDDI utilizza la struttura ***tModel*** per modellare informazioni descrittive del comportamento del servizio, tipi di specifiche utilizzate o standard a cui un servizio è conforme, etc. Queste informazioni non sono caratteristiche di una particolare azienda o entità che pubblica e tanto meno di un singolo servizio, ma generalmente interessano un insieme di servizi, indipendentemente dal tipo di azienda che li fornisce. Per questo motivo la struttura dati *tModel*, diversamente dalle altre strutture business, non è inclusa nell'elemento *businessEntity*.

Nella versione 2.0 della specifica della struttura dati UDDI [27], *tModel* segue solo tre convenzioni:

- Struttura che fornisce informazioni necessarie per determinare compatibilità tra servizi (riferita mediante *bindingTemplate*).
- Struttura per la definizione di tassonomie che consentono di classificare aziende e servizi (riferita mediante *categoryBag*).
- Struttura per contestualizzare un'informazione, mediante l'uso di namespace (rifeta mediante *identifierBag*)

Nel primo caso il *tModel* viene usato per specificare il tipo di protocollo di rete, formato di scambio dei messaggi, regole che definiscono la

sequenza di scambio, etc., da usare per interagire con un particolare servizio, o insieme di servizi.

Ad esempio supponiamo che due prodotti software comunicano tra loro attraverso un mezzo di comunicazione che aderisce a una certa specifica convenuta tra le due parti in gioco, ebbene il progettista di questa specifica può stabile un'unica identità tecnica e registrarla in un *tModel* presso un UDDI registry. Tutti quei servizi che si registrano successivamente e che ritengono di essere conformi a quella specifica possono includere nel loro *bindingTemplate* il riferimento al particolare modello rappresentativo dell'identità della specifica. A questo punto, un service requestor, che effettua una ricerca per scoprire servizi che sono conformi ad una particolare specifica, riceverà come risposta dal registro tutti quei servizi che fanno riferimento al modello che aderisce alla tipologia di richiesta effettuata.

In altre parole, il *tModel* registrato in questo modo diventa un *'fingerprinter'* (impronta digitale) di una data specifica [27].

Nel secondo caso, particolari strutture *tModel* di categoria possono essere *'linkate'* tra di loro per classificare aziende e servizi, formando strutture ad albero. Ad esempio, si può creare un modello di servizi di tipo *banking* e un modello di servizi per l'elaborazione del credito, che referenzia il primo. A sua volta, il secondo modello può essere puntato da un campo della sottostruttura *categoryBag* associata a un particolare servizio. Questo sarà automaticamente classificato come un servizio di banca per l'elaborazione del credito. Attualmente non esiste alcuna specifica UDDI che definisca formalmente un meccanismo di

classificazione dei servizi, che perciò rimane una soluzione individuale di ciascun operatore di registro.

Nel terzo caso, invece, il *tModel* può essere usato per definire un particolare spazio dei nomi, in cui alcune coppie nome-valore assumono un certo significato. Ad esempio, si potrebbe creare una lista di coppie nome-valore per definire le varie componenti di un indirizzo, come il destinatario, il CAP, la località e la provincia, in maniera tale che un sistema automatico possa interpretarlo ed elaborarlo correttamente. In particolare, il campo destinatario è molto variabile a dispetto degli altri che in generale possono essere definiti mediante una maschera. In generale, la varietà di tipologie di indirizzi porta a definire schemi complessi attraverso coppie nome-valore raggruppate sotto lo stesso modello, diciamo modello '*indirizzo'*. Per fare questo si deve utilizzare l'elemento *identifierBag*, che è una lista di coppie nome-valore definite mediante l'elemento *keyedReference.* Il listato 5.4 è un esempio di schema dell'indirizzo in questione. Una volta definita la struttura, l'elemento *identifierBag* può essere aggiunto al modello '*indirizzo*' [32]. Successivamente, questo può essere referenziato dalla sottostruttura *address* per interpretare in modo opportuno le linee d'indirizzo. Un esempio d'indirizzo costruito con le coppie nome-valore definite nell'elemento *identifierBag* è il listato 5.5.

```
<identifierBag>
    <keyedReference keyName=”cap” keyValue=”1001”/>
    <keyedReference keyName=”provincia” keyValue=”1002”/>
    <keyedReference keyName=”località” keyValue=”1003”/>
    <keyedReference keyName=”nro-civico” keyValue=”1004”/>
    <keyedReference keyName=”via” keyValue=”1005”/>
    <keyedReference keyName=”piazza” keyValue=”1006”/>
   <keyedReference keyName=”dest-persona” keyValue=”1007”/>
   <keyedReference keyName=”presso” keyValue=”1008”/>
    ...
   <keyedReference keyName=”dest-società” keyValue=”1005”/>
    ...
    </identifierBag>
```

Listato 5.4 – *Struttura di un ipotetico indirizzo.*

```
<address useType=”Ufficio vendite” tModelKey=”uuid del modello
       ‘indirizzo’”>
  <addressLine keyName=”dest-persona” KeyValue=”1007”>
     Mario Rossi
  </addressLine>
  <addressLine keyName=”via” KeyValue=”1005”>
     via Tal dei Tali
  </addressLine>
  <addressLine keyName=”nro-civico” KeyValue=”1004”>
   100
  </addressLine>
  <addressLine keyName=”cap” KeyValue=”1001”>
   83024
  </addressLine>
  <addressLine keyName=”località” KeyValue=”1003”>
   Monteforte Irpino
  </addressLine>
  <addressLine keyName=”provincia” KeyValue=”1007”>
   (AV)
  </addressLine>
```

```
</address>
```

Listato 5.5 – *Esempio d'indirizzo che segue la struttura definita in precedenza.*

Lo schema XML della struttura *tModeld* è rappresentato dal listato 5.6.

```
<element name = "tModel">
  <complexType>
      <sequence>
         <element ref = "name"/>
         <element ref = "description" minOccurs = "0" maxOccurs =
           "unbounded"/>
        <element ref = "overviewDoc" minOccurs = "0"/>
        <element ref = "identifierBag" minOccurs = "0"/>
        <element ref = "categoryBag" minOccurs = "0"/>
      </sequence>
      <attribute ref = "tModelKey" use = "required"/>
      <attribute ref = "operator"/>
      <attribute ref = "authorizedName"/>
  </complexType>
</element>
```

Listato 5.6 – *Schema della struttura* ***tModel.***

A parte il solito identificativo, il nome, la descrizione, il nome di chi pubblica il modello e quello dell'operatore del registro, un modello è caratterizzato da un elemento, *overviewDoc*, che referenzia un documento remoto (in formato XML, PDF, o un URL di un sito, etc.). Il cui scopo è quello di descrivere in maniera più o meno dettagliata l'informazione cui il modello si riferisce. Quest'elemento non è obbligatorio, infatti, è sufficiente fare riferimento al modello, senza necessariamente specificare dove esso sia descritto. Gli elementi

opzionali *identifierBag* e *categoryBag* hanno lo stesso significato visto per la struttura *businessEntity*.

A titolo esemplificativo si consideri un modello ipotetico per la certificazione ISO-9001 delle aziende, rappresentato dal listato 5.7.

```
<tModel tModelKey=”uuid:67AB9E12-5OFF-3A21-6E1C-
   78F200CA88VZ”>
  <name>iso-org:iso9001</nome>
  <description>Standard ISO9001</description>
</tModel>
```

Listato 5.7 – *Definizione di un possibile modello per la certificazione ISO9001 delle aziende.*

Se un’azienda è certificata secondo lo standard di qualità ISO9001 può referenziare, mediante *tModelInstanceDetail,* il modello in questione. In fase di ricerca un service requestor può farsi listare tutte le aziende certificate secondo tale standard, e scegliere di conseguenza.

In figura 5.7 riporto il diagramma concettuale E-R relativo al modello.

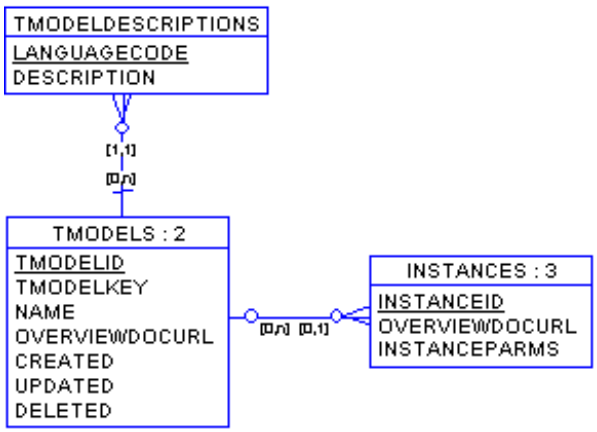


Figura 5.7 – Diagramma concettuale relativo al modello.

3.5. Struttura publisherAssertion

Questa struttura permette di creare delle relazioni tra differenti aziende registrate presso il registro UDDI, in modo da realizzare una specie di comunità. Ad esempio, un'azienda con più filiale che pubblicano presso un registro UDDI può decidere di definire una *publisherAssertion* sulle *businessEntity* che definiscono le proprie filiali. Ovviamente, affinché una relazione sia visibile nelle registrazioni, i due partecipanti devono essere d'accordo; invece, nel caso in cui l'entità che pubblica è responsabile per entrambe le parti che partecipano alla relazione, come nel caso delle filiali, allora questa diventa automaticamente visibile.

Il listato 5.8 è lo schema di questa semplice struttura.

```
<element name = "publisherAssertion">
  <complexType>
      <sequence>
          <element ref = "fromKey"/>
          <element ref = "toKey"/>
          <element ref = "keyedReference"/>
      </sequence>
 </complexType>
</element>
```

Figura 5.8 – *Schema della struttura* ***publisherAssertion.***

Tutti gli elementi della struttura sono obbligatori. **fromKey** e **toKey** sono due UUID che referenziano, rispettivamente, la prima e la seconda azienda che partecipano all'associazione. L'elemento

keyedReference mette a disposizione coppie *keyName-KeyValue* per definire il tipo di relazione stabilita tra le due *businessEntity* .

Il listato 5.9 è un esempio di associazione tra due società, riferite mediate i rispettivi UUID. In particolare, analizzando i valori delle due chiavi dell'elemento *keyedReference*, si desume che la prima è una società finanziaria di controllo (holding company) nei confronti della seconda. Il modello di riferimento per questo tipo d'associazione è puntato dal valore di *tModelKey*.

```
<publisherAssertion>
<fromKey>F5E65…</fromKey>
<toKey>A237B…</toKey>
<keyedReference tModelKey="uuid:34D5…" keyName=" Holding
Company" keyValue="parent-child"></keyedReference>
</publisherAssertion>
```

Listato 5.9 – *Esempio di struttura* ***publisherAssertion.***

Il diagramma concettuale E-R relativo alla struttura *publisherAssertion* è mostrato in figura 5.8.

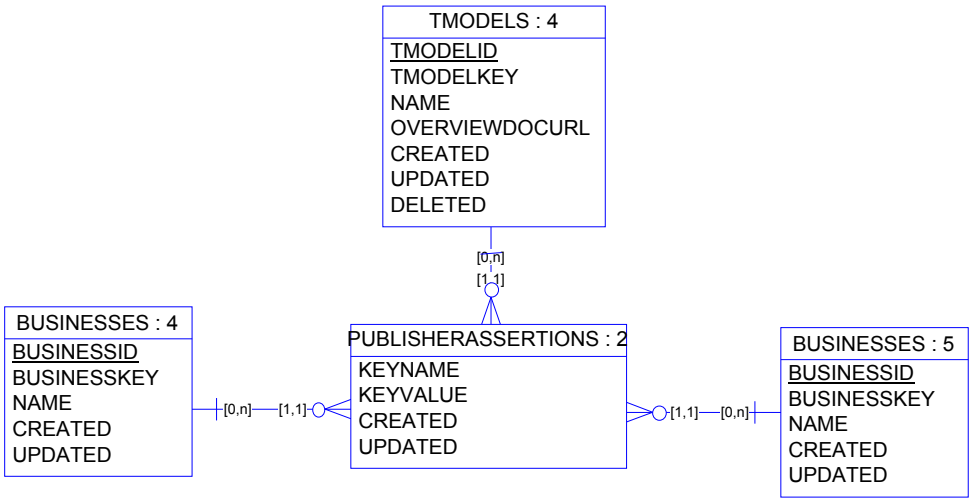


Figura 5.8 – Diagramma concettuale E-R riferito all'asserzione.

4. *Le API (Application Programming Interface) UDDI.*

Le API UDDI [28] sono gli strumenti primari per la pubblicazione e la scoperta delle informazioni sulle aziende, che si sono registrate presso un registro UDDI, e i relativi servizi forniti. La specifica 2.0 definisce un modello di programmazione per inviare e per ricevere approssimativamente 40 messaggi SOAP. Questi consentono di invocare le funzioni per l'accesso al registro e la manipolazione delle informazioni ivi contenute.

Le API si distinguono in:

- API per la pubblicazione (publisher API)
- API per l'interrogazione (Inquiry API)

Le prime forniscono un modello di programmazione per memorizzare e modificare le strutture dati. Le seconde, invece, consentono di consultare le informazioni contenute nel registro.

Per l'utilizzo delle API di pubblicazione è richiesto un accesso autorizzato al registro ed è responsabilità dell'operatore realizzare un protocollo di autenticazione efficiente e compatibile con le tali API.

Per le API di consultazione, invece, non è richiesta alcuna forma di autenticazione in quanto si limitano solo alla lettura di dati pubblici, a tutto vantaggio dei fornitori.

4.1. API di consultazione.

Le API di consultazione (inquiry API) forniscono il supporto per tre forme d'interrogazioni:

- *Browse.* Iniziano tutte per *find_* e servono, appunto, a cercare le informazioni nel registro a partire da requisiti generici.
- *Drill-Down.* Iniziano tutte col prefisso get_ e servono ad acquisire informazioni di dettaglio relative a una particolare entità del registro.
- *Invocation.* Servono per preparare l'applicazione ad interagire con la sua controparte sulla base delle informazioni contenute nel registro. Infatti, per poter integrare il Web Service desiderato nella propria applicazione, l'utilizzatore deve conoscere le informazioni contenute nella struttura

bindingTemplate, scaricarle e adoperarle per contattare il fornitore del servizio.

- I problemi che si possono verificare nell'eseguire l'operazione di negoziazione sul servizio sono inerenti essenzialmente allo spostamento inaspettato del servizio, cosa che può succedere ad esempio quando si effettua un aggiornamento del server, o durante un'operazione di recupero da un disastro, etc. Ovviamente se il provider si è preso la briga di aggiornare le informazioni del *bindingTemplate*, allora l'utente del Web Service può ottenere le nuove informazioni e aggiornare il binding.

- Purtroppo, queste API non sono state ancora specificate.

4.1.1. API di navigazione e acquisizione.

In generale, un browser UDDI è un'applicazione con capacità di navigazione che consente all'utente di esplorare ed esaminare i dati presenti nel registro [33]. Partendo da informazioni generiche sull'azienda o sul servizio da cercare e con riferimento a particolari criteri di ricerca, interroga il registro e se l'esito della ricerca è positivo, fornisce i risultati trovati. Questi, successivamente, possono essere utilizzati per effettuare operazioni di drill-down. Il browser può essere implementato utilizzando le API di navigazione del registro. Il sequence diagram riportato in figura 5.9 illustra una tipica interazione tra il browser UDDI e il registro.

Supponiamo che un utente usa il browser per cercare i servizi offerti da un'azienda della quale ricorda solo i primi tre caratteri del nome,'ita'

per esempio. Il browser invoca il metodo *find_business* passandogli i primi tre caratteri del nome dell'azienda.

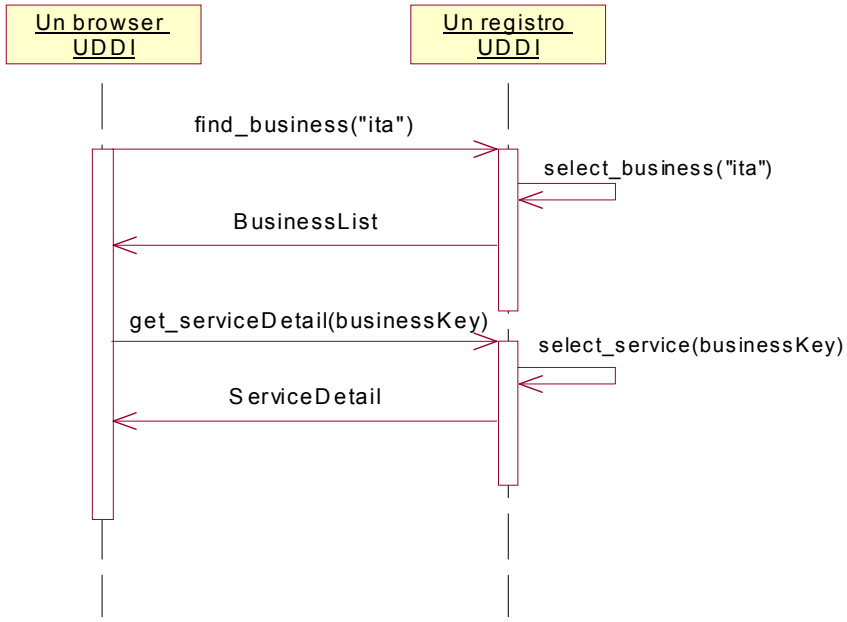


Figura 5.9 – Interazione tra il browser e il registro UDDI.

Il gestore del registro interroga il database e spedisce al richiedente una lista di aziende che corrispondono alla descrizione fornita. L'utente selezione quella desiderata, Italdata, e, di conseguenza, il browser mostra i nomi dei servizi offerti dall'azienda selezionata.

Quando l'utente sceglie il servizio, il browser preleva l'UUID dell'azienda dalla struttura dati *businessEntity* e lo passa come argomento alla funzione *get_serviceDetail*. Il registro, ricevuto il messaggio SOAP, effettua un'interrogazione al database e spedisce al richiedente la lista dei dettagli dei servizi offerti dall'azienda "Italdata".

In figura 5.10 riporto il diagramma di classe concettuale che delinea le associazioni tra i vari oggetti che prendono parte all'interazione.

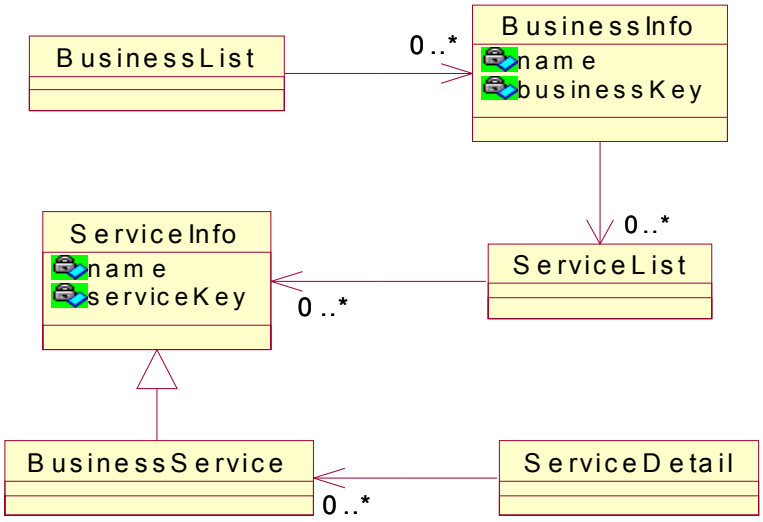


Figura 5.10 – Diagramma di classi relativo alle informazioni estratte dal registro quando si esegue l'operazione *find_business*.

Nella tabella 5.2 sono riportate le descrizione delle funzioni che consentono lo '*scorrimento*' delle informazioni registrate.

Nome API	*Descrizione*
find_binding	Viene usata per localizzare le specifiche tecniche dei servizi. Ritorna una struttura *bindingDetail*.
find_business	Usata per localizzare le informazioni su uno o più aziende. Ritorna una struttura *businessList*.
find_relatedBusiness	Localizza informazioni inerenti alle registrazioni *businessEntity* che sono correlate con una specifica entità o azienda, passata nel messaggio di richiesta. Ritorna una *relatedBusinessList*.

find_Service	Localizza specifici servizi dentro una data *businessEntity*. Ritorna una *serviceList*.
find_tModel	Usata per trovare uno o più strutture *tModel*. Ritorna una struttura *tModelList*.

Tabella 5.2 – API di navigazione (browse).

Il listato 5.10 mostra il formato per l'operazione di ricerca di tutte le specifiche relative al servizio identificato dal valore dell'attributo *serviceKey*, e restituisce il risultato in ordine alfabetico ascendente per nome.

```
<find_binding serviceKey="11556A41-45A4-787B-88EF-C44CDE4AE23A"
 generic="2.0" xmlns="urn:uddi-org:api_v2">
    <findQualifiers>
        <findQualifier>sortByNameAsc</findQualifier>
        </findQualifiers>
</find_binding>
```

Listato 5.10 – *Un esempio di operazione find_binding.*

Dunque, una volta ottenuta la chiave di una struttura dati principale mediante le funzioni di navigazione, questa può essere usata nelle invocazioni dei metodi get_xx per conoscere le inerenti informazioni di dettaglio.

Le API d'acquisizione sono brevemente descritte in tabella 5.3.

Nome API	***Descrizione***
get_bindingDetail	Usata per acquisire tutte le informazioni relative a una specifica struttura bindingTemplate.
get_businessDetail	Viene usato per acquisire tutte le informazioni contenute nella struttura businessEntity.
get_businessDetailExt	Acquisisce informazioni estese relative al businessEntity.
get_serviceDetail	Riporta i dettagli di un dato set di businessService registrati
get_tModelDetail	Usato per acquisire i dettagli per un dato tModel

Tabella 5.3 – *API per l'acquisizione dei dettagli delle informazioni contenute nel registro.*

Il listato 5.11 è un esempio di utilizzo di un'API di acquisizione per estrarre due strutture *businessService* dal registro.

```
<get_serviceDetail generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <serviceKey keyValue="25557A31-55C4-787B-88EF-
        C54CDE4AE33B"/>
    <serviceKey keyValue="24544B32-58B4-989E-8EEF-
        C53CCE4AE34B"/>
</get_serviceDetail>
```

Listato 5.11 – Esempio di API *get_serviceDetail*.

4.2. API di pubblicazione.

La specifica 2.0 definisce 16 API di pubblicazione. Cinque riguardano le relazioni tra entità, quattro sono usate per la pubblicazione delle strutture fondamentali del registro, altrettanti per la loro eliminazione,

due alla gestione dell'accesso al registro e una può essere impiegata per ottenere una lista di entità e modelli pubblicati da un singolo individuo. Conseguentemente, i messaggi SOAP coinvolti nelle API di pubblicazione sono, sostanzialmente, richieste d'accesso autenticato ad un operatore, di pubblicazione e aggiornamento delle informazioni contenute nel registro UDDI.

Lo scambio dei messaggi su cui si basano le API è di tipo sincrono e avviene attraverso SOAP su HTTP-POST o HTTPS.

Nella tabella 5.4 sono descritte le otto funzioni per la pubblicazione e la rimozione delle quattro strutture fondamentali: *businessEntity*, *businessService, bindingTemplate* e *tModel*.

Operazione	Descrizione
delete_binding	Rimuove una struttura bindingTemplate.
delete_business	Rimuove una struttura businessEntity.
delete_service	Elimina una struttura businessService.
delete_tModel	Usato per nascondere le informazioni circa un tModel. Qualunque tModel nascosto in questo modo può ancora essere usato per motivi di riferimento e può essere acquisito mediante l'operazione get_tModelDetail, ma risulta semplicemente nascosto dall'insieme dei risultati restituito dall'operazione find_tModel.
save_binding	Consente di registrare una nuova struttura bindingTemplate o aggiornane una esistente.
save_business	Consente di registrare una nuova struttura businessEntity o aggiornane una esistente..
save_service	Consente di registrare una nuova struttura businessService o aggiornane una esistente.
save_tModel	Consente di registrare una nuova struttura tModel o aggiornane una esistente.

Tabella 5.4 – *API di pubblicazione.*

Ogni editore, ovvero persona autorizzata a pubblicare informazioni nel registro, è identificato da un contrassegno specifico detto *authorization token*. Per richiedere un *gettone* (login) all'operatore, l'editore deve invocare la funzione *get_authToken* con il nome utente e password relativi all'account attivato all'atto della registrazione al registro UDDI. Una volta ottenuto, il contrassegno

potrà essere usato per invocare qualunque altra funzione di pubblicazione dentro una sessione di pubblicazione. Quando questa termina, conviene invocare la funzione *discard_authToken* per invalidare il gettone (logout) in modo che non possa essere riutilizzato da altri.

Per eliminare una struttura dal registro occorre, oltre al token, anche il suo UUID.

Il resoconto di tutte le entità e modelli pubblicati da uno specifico editore, può essere ottenuto con la funzione *get_RegisteredInfo*.

Prima di passare alle API che consentono di stabilire relazioni fra entità, è opportuno chiarire il concetto d'asserzione. Una relazione fra due aziende è attivata attraverso un'asserzione, cioè una dichiarazione pubblica da parte di un'azienda che attesta in qualche modo tale associazione. Quantunque l'asserzione venga concessa, questa per essere visibile all'esterno deve essere confermata dalla controparte. Il meccanismo di pubblicazione di un'asserzione è costituito, dunque, da due fasi. Nella prima un'azienda asserisce di essere in relazione con un'altra e questa, nella seconda fase, conferma l'asserzione. In questo modo le due aziende non devono sincronizzarsi in tempo reale per la pubblicazione.

In generale, un'asserzione può trovarsi in tre stati: *complete*, *toKey_incomplete*, *fromKey_incomplete*. Il primo indica che l'asserzione è stata convalidata da entrambe le parti, gli atri due che una o l'altra parte non hanno ancora confermato la relazione. Un'associazione è visibile solo se si trova nello stato *complete*.

Le API che gestiscono la struttura *publisherAssertion* sono brevemente descritte nella tabella 5.5.

Funzione	Descrizione
add_publisherAssertions	Aggiunge una relazione tra due aziende o entità pubblicate presso il registro.
delete_publisherAsertions	Rimuove una specifica asserzione, e quindi la relazione tra due entità pubblicate.
set_publisherAssertions	Utilizzato per salvare l'intero collezione di asserzioni (collection) per uno specifico account.

Tabella 5.5 – *API per la pubblicazione di asserzioni.*

5. Come usare WSDL in UDDI.

La descrizione del servizio mediante WSDL è un'informazione complementare alle strutture dati *businessService* e *bindingTemplate* definite nel registro UDDI. D'altra parte, il registro non supporta direttamente i documenti WSDL ma fornisce un meccanismo generico per pubblicare e trovare differenti tipi di descrizioni di servizi [31].

In generale, la descrizione del servizio può essere pubblicata in vari modi. Il caso più semplice, ma anche meno efficace, di pubblicazione è

quello diretto o statico in cui il service provider invia la descrizione direttamente al service requestor, ad esempio, attraverso la posta elettronica [2]. Questo caso è illustrato in figura 5.10.

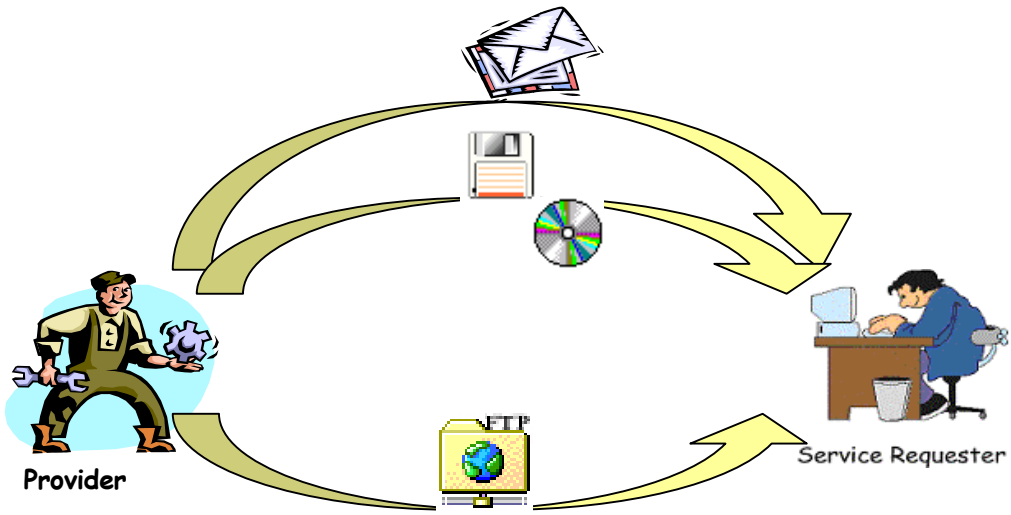


Figura 5.10 – *Pubblicazione diretta della descrizione del servizio offerto.*

La pubblicazione diretta avviene dopo che il consumatore e il fornitore si sono accordati sui termini sanciti dall'e-business, in generale dopo aver stipulato una formula contrattuale. In questo scenario può capitare che il service requestor mantenga una copia locale della descrizione del servizio.

Una pubblicazione leggermente più dinamica, che prevede la presenza di un intermediario, usa meccanismi basati su HTTP-GET, definiti, ad esempio, in DISCO o ADS, che consentono di ottenere le descrizione a partire da uno specifico URL. Degli operatori gestiscono una

repository delle descrizioni dei servizi su cui consentono di effettuare ricerche avanzate secondo certi requisisti.

Nel caso della pubblicazione delle descrizioni dei servizi nel registro UDDI, i service provider possono usare differenti nodi privati UDDI in funzione del tipo di pubblico che si prefiggono di raggiungere.

In seguito è riportato un elenco dei tipi di nodi UDDI [2]:

- *Nodo UDDI per applicazioni interne all'azienda*: I Web Services da usare per integrazioni di applicazioni all'interno dell'azienda (internal interprise application) dovrebbero essere pubblicati su un nodo di questo tipo. Tali nodi si trovano dietro il firewall e consentono un maggiore controllo sull'accesso al registro da parte degli editori, che sono le unità organizzative dell'azienda. Ovviamente, i servizi vengono pubblicate sotto un'unica entità aziendale.

- *Portal UDDI node*. Generalmente, viene utilizzato per pubblicare Web Services destinati ai partner commerciali (Web Services esterni) e perciò è limitato alle sole descrizioni di servizi che la compagnia decide di rendere visibili.

- *Partner Catalog Uddi node*. Solo i partner commerciali sono autorizzati a pubblicare informazioni sul catalogo.

- *E-Marketplace UDDI node*. Se i service provider hanno intenzione di offrire Web Service economicamente competitivi allora dovrebbero pubblicare la loro descrizione in tale registro. Questi sono ospitati e gestiti da organizzazioni o

consorzi e, in generale, sono specifici per determinati settori industriali. I nodi marketplace forniscono anche un sistema di filtraggio degli accessi, garantendo un certo livello di qualità.

- *UDDI Operator node*. E' un registro pubblico in cui gli eventuali fornitori di Web Services possono rivolgersi, previa registrazione, per esporre le informazioni sul proprio conto e sui servizi offerti [29]. Un UDDI Operator node è supportato, replicato e ospitato dall'IBM, Microsoft e Ariba.

5.1. Pubblicare e trovare le descrizioni WSDL.

Un registro UDDI fornisce un meccanismo, indipendente dalle piattaforme hardware e software, per descrivere e trovare informazioni su aziende e servizi. Le strutture dati UDDI forniscono un framework per la descrizione delle informazioni principali sulle aziende e relativi servizi, mentre le API UDDI offrono un modello di programmazione per effettuare operazioni di pubblicazione e di consultazione, indipendentemente dal linguaggio usato per implementarle.

Un WSDL è un linguaggio definito mediante il metalinguaggio XML che permette di descrivere le interfacce dei servizi, i protocolli per la negoziazione, e di raggiungere una particolare implementazione del servizio. Ebbene, il linguaggio WSDL potrebbe essere usato per complementare lo standard UDDI in modo da estendere le informazioni sul servizio alla descrizione della sua interfaccia e le informazioni sulle specifiche tecniche alle descrizioni delle implementazioni. L'obiettivo è quello di separare le informazioni

WSDL riusabili da quelle specifiche di una data istanza di un servizio. Come visto nel capitolo 4, le informazioni comuni a certe categorie di servizi, come formati di messaggi, portTypes (documento astratto), e il protocollo di binding (ad esempio, RPC SOAP), sono incluse in una porzione riusabile (definizione dell'interfaccia del servizio), mentre informazioni relative agli endpoint (l'elemento *wsdl:port*), sono incluse in un documento a parte (definizione dell'implementazione del servizio). Dunque, poiché l'interfaccia del servizio rappresenta una definizione riusabile, essa sarà pubblicata nel registro UDDI come un modello (struttura *tModel*). L'implementazione, invece, è una particolare istanza del servizio e quindi sarà pubblicata come *businessService*. Il modello deve essere pubblicato prima delle informazioni d'implementazione.

In figura 5.11 è riportato il mapping tra gli elementi WSDL e le strutture dati del registro.

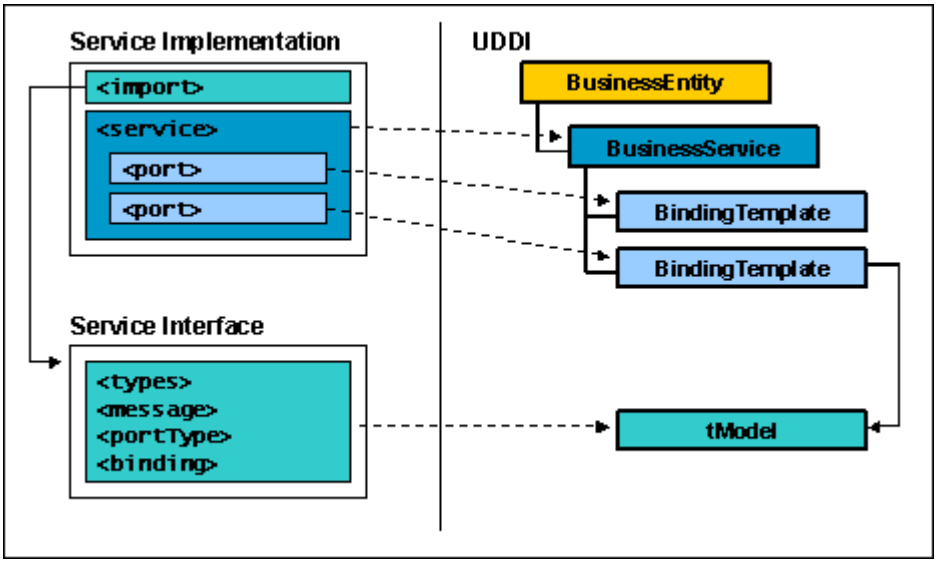


Figura 5.11 – *Mapping tra i documenti WSDL e le strutture dati definite nel registro UDDI.*

Di seguito riporto gli step che consentono di usare WSDL come supporto per la creazione di strutture *businessService*:

1. Il primo passo è la creazione della definizione dell'interfaccia del servizio. Tipicamente, un service interface provider (che generalmente coincide con lo stesso fornitore del servizio) realizza una definizione dell'interfaccia del servizio utilizzando il documento WSDL. Successivamente, tale definizione viene registrata in una struttura dati *tModel* facendola puntare dal valore del campo *overviewDoc*.
2. La figura 5.12 illustra questo concetto. WSDL-serv-spec-model è il modello che fa riferimento alla descrizione dell'interfaccia di un particolare servizio.

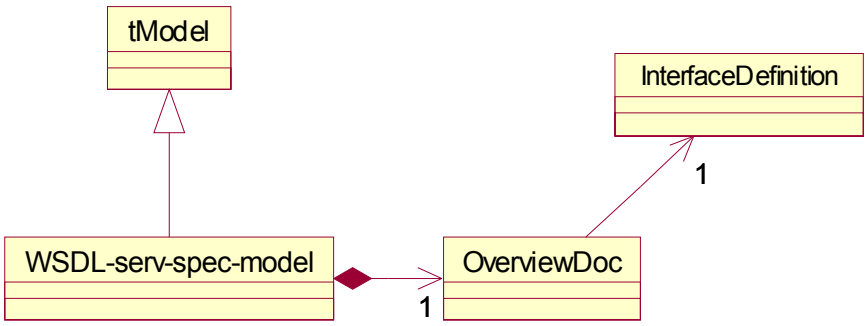


Figura 5.12 – Pubblicazione della definizione dell'interfaccia del servizio.

3. Un service provider, contattando un registro UDDI, ed utilizzando le funzioni di consultazione, può ottenere la descrizione del modello della definizione delle interfaccia del servizio che desidera implementare e fornire. Se non trova un modello che corrisponde ai suoi requisisti, il provider può sempre pubblicarne uno, a patto, però, che abbia già definito l'interfaccia del servizio da offrire. Se, invece, la ricerca ha esito positivo, seguendo il link presente nella struttura *overviewDoc*, otterrà il documento WSDL della corrispondente definizione. Con questo, gli sviluppatori del provider possono implementare il servizio in maniera conforme all'interfaccia proposta dal service interface provider e comunemente condivisa.

4. Infine, il fornitore deve effettuare il deployment dell'implementazione del servizio e la registrazione dello stesso presso il registro. Per la pubblicazione si deve creare una struttura *bindingTemplate* per ciascun elemento *wsdl:port* della definizione dell'interfaccia e impostare il valore del campo *accessPoint* al valore del corrispondente endpoint. Inoltre, ciascun *bindingTemplate* deve puntare al modello WSDL-serv-spec-model specifico per quel servizio, utilizzando la struttura *tModelInstanceInfo*.

Nella figura 5.13 è mostrato un esempio completo di mapping tra i documenti WSDL e le strutture dati UDDI coinvolti nella pubblicazione [31]. Ricordo, tuttavia, che non esiste una vera specifa

per il meccanismo di mapping tra il documento WSDL e UDDI, per cui quello di figura 5.13 è da considerarsi come un suggerimento.

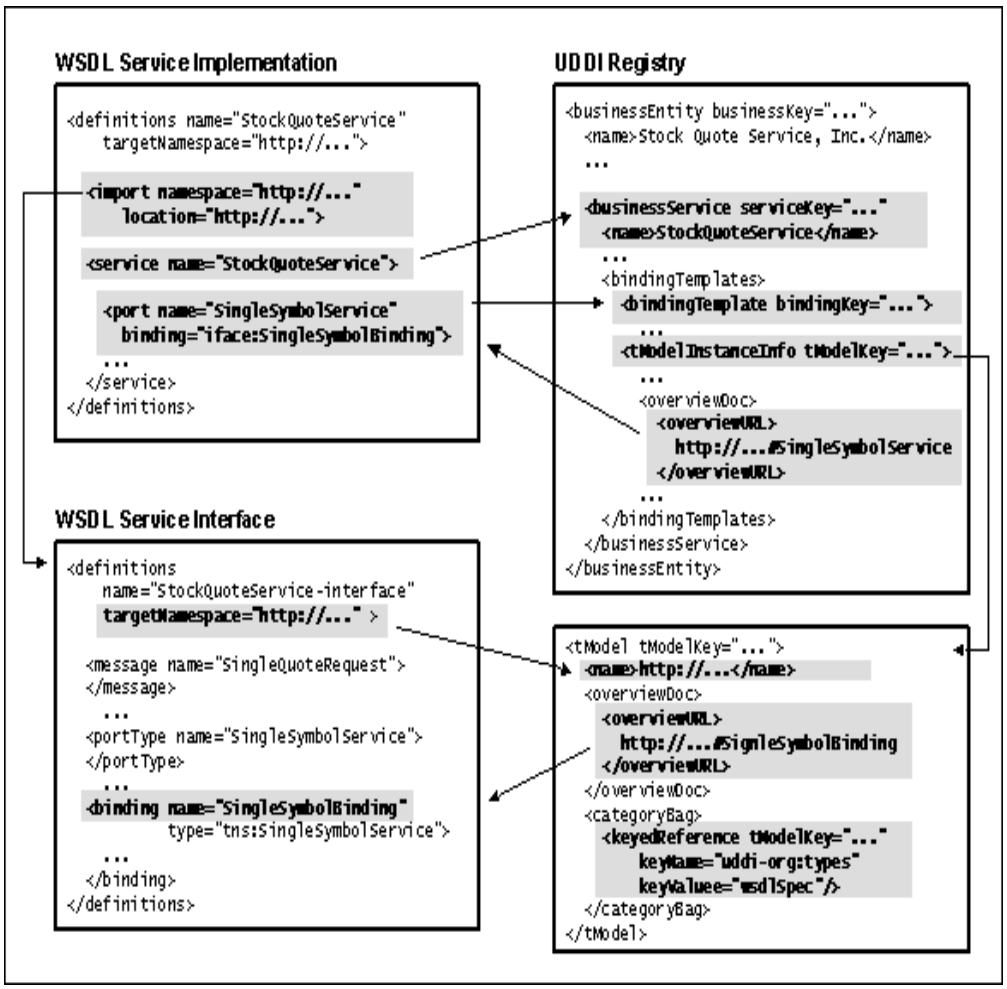


Figura 5.13 – *Esempio di documento WSDL mappato sulle strutture dati del registro UDDI.*

Capitolo VI

RIUSO E WEB SERVICES

1. Introduzione.

Il riuso del software è un processo che consente di riutilizzare il patrimonio software aziendale all'interno del ciclo di vita dei prodotti software. Il software riutilizzabile non è solo codice, ma qualunque manufatto inerente al ciclo di vita di prodotti in precedenza realizzati, e, perciò, include porzioni di specifiche dei requisiti, progetti, piani di test, documentazione, ed altro. Tuttavia, in questo lavoro di studio verranno trattati solo quegli aspetti riguardanti il riuso dei componenti software mediante un approccio basato sui Web Services. A tale proposito è stato realizzato un prototipo che consente di classificare e pubblicare i componenti software in ambiente Intranet, esposti come servizi Web. Il prototipo consente altresì di cercare e utilizzare questi servizi.

Il fulcro dell'applicazione è rappresentato dal registro UDDI che viene usato come libreria condivisa di servizi. Conseguentemente, non escludo la possibilità d'impiego dell'applicazione per riutilizzare anche altri manufatti, come diagrammi UML, documentazione, specifiche dei requisiti, etc. Questo obiettivo può essere raggiunto utilizzando la

struttura tModel messa a disposizione dalla specifica UDDI per modellare concettualmente tali manufatti e per localizzarli fisicamente mediante il campo OverviewDoc (vedi capitolo 5).

Tutti i tool utilizzati sono implementati in Java e utilizzano il protocollo SOAP per realizzare un paradigma client-server. La prima caratteristica assicura la portabilità al livello del codice, la seconda consente di utilizzare l'applicazione client per la ricerca dei componenti riusabili da qualsiasi sistema remoto provvisto di librerie per la comunicazione su HTTP.

La capacità dei Web Service di esporre le funzionalità dei servizi, comunque implementati, in un ambiente Web può essere sfruttata per riutilizzare e condividere codice esistente. Infatti, essi combinano i migliori aspetti della programmazione basata su componenti e la programmazione Web, e possono essere riusati e condivisi senza preoccuparsi di come i servizi sono implementati, quale linguaggio, sistema operativo, modello a componenti o altre risorse sono usati per realizzarli.. Generalmente, questo tipo di riuso è detto "black-box reuse", ovvero si possono usare le funzionalità offerte dal componente senza conoscere cosa c'è nella scatola, e riusare i componenti mediante composizione. Diversamente, il "white-box reuse" consente di riutilizzare gli oggetti mediante il meccanismo dell'ereditarietà, ovvero l'implementazione della sottoclasse può essere definita in termini dell'implementazione della superclasse . In quest'ultimo tipo di riuso l'implementazione della superclasse deve essere visibile alla sottoclasse.

In particolare, si discuterà la possibilità di sviluppare Web Service interni (all'azienda) fornendo un'interfaccia XML a semplici componenti sviluppati ex novo o esistenti.

Questo capitolo è suddiviso sostanzialmente in due parti. Nella prima parte viene descritto il concetto di riuso del software in generale e nella seconda gli strumenti che consentono di esporre i componenti software aziendali come servizi Web, di pubblicarli, di cercarli e di utilizzarli.

2. Vantaggi di una strategia del riuso.

Oggi, le organizzazioni al fine di fronteggiare la crescente pressione competitiva, devono cercare di ridurre il tempo richiesto per portare i loro prodotti sul mercato, i costi di sviluppo e manutenzione del software, e contemporaneamente d'incrementarne la qualità.

Se implementata in modo appropriato, una strategia del riuso può consentire ad un'azienda di raggiungere alcuni o tutti dei seguenti obiettivi:

- *Incremento della produttività*. Dopo un investimento iniziale, il riuso dei manufatti software esistenti permette di diminuire il costo dello sviluppo e manutenzione del software. Il progetto di riuso di HP (Hewlett-Packard) ha consentito incrementi di produttività dal 6% al 40%[38].

- *Riduzione del time-to-market*. Il riutilizzo del software può abbreviare considerevolmente il *critical path* nella consegna di un prodotto.

- *Miglioramento della qualità del software*. E' una convinzione comune che un componente usato molte volte, generalmente, possiede meno difetti di uno appena sviluppato. Di conseguenza, il riutilizzo di componenti esistenti può portare a realizzare prodotti con una densità di difetti minori e, quindi, di qualità migliore.

- *Fornire consistenza e interoperabilità ai prodotti*. Le interfacce standard e l'uso comune di componenti nei prodotti incentivano la facilità d'uso e l'interoperabilità. Per esempio, quando diversi prodotti riusano lo stesso schema d'interfaccia utente, le convenzioni e le caratteristiche d'interazione possono essere più coerenti. Per quanto riguarda l'interoperabilità, si pensi alla possibilità di fornire ai sistemi software comportamenti più consistenti mediante l'uso comune, ad esempio, di routine per la gestione degli errori.

- *Assicurare la conformità dei prodotti software ai requisisti utente mediante la prototipazione*. Poiché la disponibilità dei componenti riutilizzabili facilita la realizzazione di prototipi, i requisisti utenti possono essere convalidati più rapidamente e agevolmente.

Inoltre, il rischio associato alla creazione di nuovi componenti può essere ridotto qualora i componenti riusabili già esibiscono le funzionalità desiderate attraverso interfacce standard che ne facilitano la loro integrazione.

3. Infrastruttura del riuso.

L'esperienza ha dimostrato che il riuso non consiste solamente nella creazione di una repository o libreria di componenti e d'altri manufatti software, accessibile agli sviluppatori e progettisti [39]. Piuttosto, un'implementazione sistematica del riuso richiede un'infrastruttura adeguata che supporti il riutilizzo del software.

Per adottare un approccio metodologico al riuso si devono prendere in considerazione quattro elementi fondamentali, mostrati nella figura 6.5 e di seguito elencati:

- Processo del riuso, che è costituito da varie attività, ruoli organizzativi e responsabilità.
- Automazione, insieme di tool per rendere meccaniche alcune attività.
- Componenti, un nutrito catalogo di risorse software dell'azienda.
- Approccio, inerente alla strategia di riuso intrapresa.

Nei successivi paragrafi saranno discussi gli aspetti basilari di questi elementi quattro elementi.

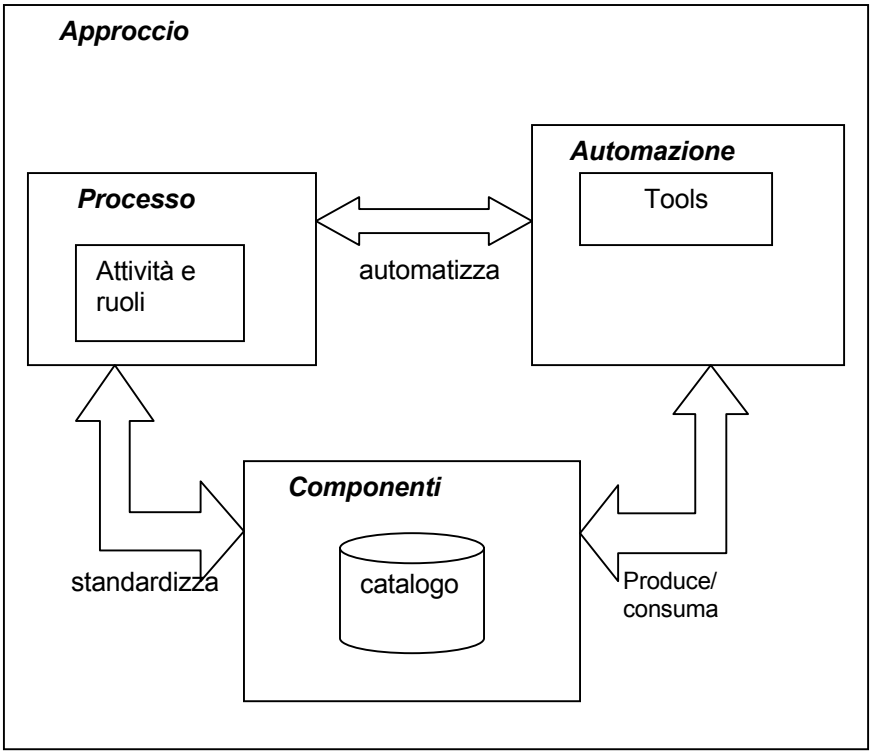


Figura 6.5 – *Infrastruttura per il riutilizzo delle risorse software.*

3.1. Processo del riuso.

L'aspetto critico di quest'infrastruttura è il processo del riuso.

In una forma semplificativa, il processo del riuso consiste di quattro attività fondamentali: gestione dell'infrastruttura del riuso (MRI, manage reuse infrastructure), produzione di risorse riutilizzabili (PRA, produce reusable assets), mediatore di risorse riutilizzabili (BRA, broker reusable assets), e consumare di risorse riusabili (CRA,

consume reusable assets). In figura 6.1 è riportato l'insieme delle attività inerenti al processo di riuso.

Relativamente a queste attività si possono individuare, fondamentalmente, tre ruoli: il fornitore o produttore che crea beni riutilizzabili, il consumatore che li usa o per produrre prodotti per l'utente finale o per produrre ulteriori beni riutilizzabili. Il fornitore e il consumatore sono ruoli logicamente separati ma in pratica possono essere svolti da una stessa figura professionale all'interno dell'azienda. Infine, un gestore che ha il compito di coordinare, in qualche modo, le attività associate ai due precedenti ruoli.

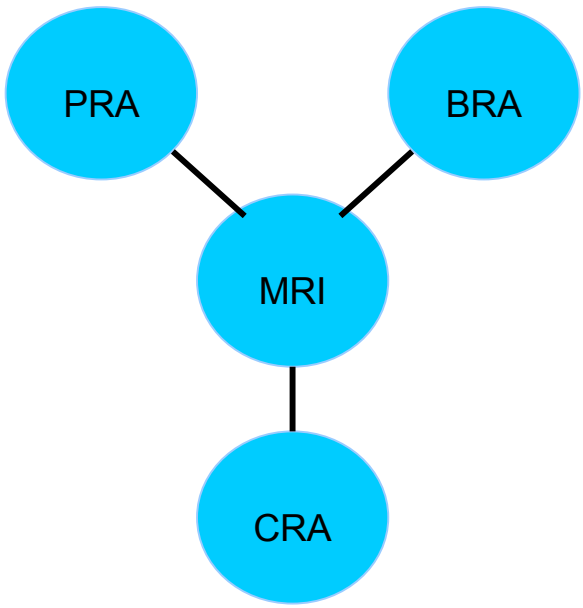


Figura 6.2 – *Attività principali del processo del riuso del software.*

La funzione dell'attività MIR è di stabilire le regole del riuso, ruoli, e obiettivi nell'infrastruttura di supporto al riuso. Alcuni task di questa attività possono essere: stabilire convenzioni e standard; approvare

l'aggiunta, eliminazione e modifica delle informazioni contenute nella libreria o repository aziendale; commissionare la realizzazione di componenti; coordinare programmi e risorse e allineare gli obiettivi del riuso con quelli dell'azienda. Funzioni più avanzate di quest'attività potrebbero stabilire e assegnare incentivi economici, interpretare i dati secondo particolari metriche, e implementare modelli economici.

L'attività PRA riguarda lo sviluppo, la generazione e reingegnerizzazione del patrimonio software aziendale con in mente specifici obiettivi di riusabilità. PRA include l'analisi e l'ingegneria del dominio [39]. L'analisi del dominio è il processo che identifica, colleziona, organizza, analizza e descrive le caratteristiche comuni e variabili tra i sistemi in un certo dominio applicativo e architettura software dallo studio di sistemi esistenti, dalla teoria sottostante, dalle tecnologie emergenti e dalle teorie di sviluppo dentro il dominio d'interesse. L'ingegneria del dominio, in particolare, s'interessa della realizzazione di componenti, metodi, e tool per risolvere i problemi di sviluppo di sistemi e sottosistemi mediante l'applicazione di conoscenza sul modello del dominio e sulle architetture software [36].

L'attività BRA contribuisce allo sforzo del riuso mediante i task di certificazione, configurazione, manutenzione, e mediazione dei manufatti riutilizzabili. A quest'attività è anche delegato il compito di classificare e ricuperare i beni software dalla libreria.

L'attività CRA è utile per la realizzazione di prodotti software usando materiale riusabile. I consumatori utilizzano la libreria e i tool associati per trovare le informazioni inerenti ai materiali riusabili a loro

disposizione, per identificare e recuperare i componenti necessari, e per integrare i componenti trovati nelle loro applicazioni.

3.1.1. Ruoli e compiti.

Il riuso richiede, inevitabilmente, la definizione di nuovi ruoli e differenti compiti per il personale dell'azienda. In particolare, è fondamentale la figura del produttore di materiale riutilizzabile. Realizzare software riusabile comporta dei cambiamenti al regolare ciclo di sviluppo dei prodotti software. In particolare, il cambiamento più evidente consiste nell'aumento del tempo di progettazione [39]. Infatti, progettare software per uso specifico è in generale meno complesso e più rapido che progettare software per uso generico. La fase di progettazione rivolta al riuso deve essere preceduta da un'attività d'analisi del dominio con riferimento a specifici requisiti di progettazione. Un ipotetico analista di dominio dovrebbe intervistare un esperto del dominio, che è una persona ben informata sui prodotti esistenti e pianificati all'interno di particolare dominio, e in base alle informazioni raccolte deve definire un modello del dominio applicativo che forma la base per l'analisi dei requisiti nel processo software [39]. Una volta in possesso dei requisiti, il produttore può iniziare la progettazione di componenti riusabili.

Un altro compito del progettista è l'anticipazione delle condizioni e contesto in cui il materiale può essere riutilizzato in futuro e al fine di facilitarne la comprensione deve fornire un'adeguata documentazione per l'utente.

Il ruolo del consumatore consiste nel riutilizzare i beni software aziendali integrandoli nei propri progetti. Il riuso evita allo sviluppatore la creazione di materiali ridondati, lasciandogli più tempo disponibile per curare gli aspetti innovativi dello sviluppo. Ovviamente, avendo a disposizione già dei componenti prefabbricati l'utente può realizzare più velocemente prototipi software per la convalida dei requisiti inderogabili e desiderabili del cliente.

Chi coordina le attività inrenti al produttore e al consumatore deve contribuire in maniera incisiva allo sforzo del riuso. In particolare, deve assicurare che il patrimonio software sia alimentato e consumato in modo appropriato. Con riferimento alla gestione dell'infrastruttura del riuso si possono individuare tre tipi di ruoli: *reuse sponsor*, *reuse champion* e *reuse manager* [38].

Lo sponsor è usualmente un gestore che autorizza e rafforza il programma di riuso, controllando e assicurando che questo abbia le adeguate risorse.

Il campione (champion) può essere un singolo individuo oppure un gruppo che difende e supporta il programma di riuso, una specie di *predicatore del riuso software*. Di conseguenza, tra i compiti di questo ruolo figura quello dispensare concetti del riuso ai propri sostenitori (sponsor) e perseguendo un'opera di convincimento. Generalmente, questo ruolo è individuale e ricoperto da chi è stimato per la sua capacità di coordinare e dirigere il proprio gruppo.

Il manager è responsabile della pianificazione del riuso e della gestione del flusso d'informazione e materiale riusabile tra i produttori e

consumatori, assicurando contemporaneamente che l'infrastruttura sia adeguata a supportare il processo di riuso. Deve essere abile e capace a comprendere e gestire le questioni inerenti al riuso, coordinare la collaborazione tra le differenti unità organizzative aziendali, riconoscere e bilanciare obiettivi a breve e lungo termine e identificare e risolvere impedimenti di tipo culturale e organizzativo.

Il manager può essere direttamente responsabile per i meccanismi che stanno alla bese della strategia riuso o indirettamente attraverso un comitato direttivo (comitato di saggi).

Ovviamente, la suddivisione dei ruoli è sola logica. Praticamente, il ruolo del gestore può essere ricoperti anche da un singolo individuo.

3.2. Componenti riusabili e servizi Web.

La progettazione di componenti riusabili segue i concetti fondamentali d'astrazione, information hinding, indipendenza funzionale, raffinamento e programmazione strutturata, con aggiunta dei metodi orientati agli oggetti, principalmente l'ereditarietà, ma anche il testing, metodi di SQA (Software Quality Assurance), etc [39]. Tuttavia, questi sono principi di progettazione che normalmente dovrebbero essere applicati durante lo sviluppo di prodotti software, perciò, un componente deve possedere ulteriori caratteristiche per potersi definire riusabile. In particolare, esso deve essere facilmente scoperto e deve fornire tutte le informazioni necessarie per consentire al consumatore di utilizzarlo [40]. Si possono avere i migliori componenti del mondo, ma se non si ha la possibilità di trovarli e conoscerli resteranno i segreti meglio custoditi del mondo.

Per trovare i componenti può essere usata una libreria condivisa e un insieme d'informazioni di classificazioni associato con essi.

Invece, per comprendere le loro funzionalità e il loro contesto d'uso è necessario un opportuno assortimento d'informazioni e di documentazioni. Questo include specifiche funzionali, informazioni per la distribuzione, istruzioni per il deployment e se il componente è destinato ad un uso esterno all'azienda vanno include anche informazioni di tipo commerciali e inerenti alla licenza d'uso.

Nella figura 6.2 è illustrato il concetto che sta dietro ad un componente riusabile.

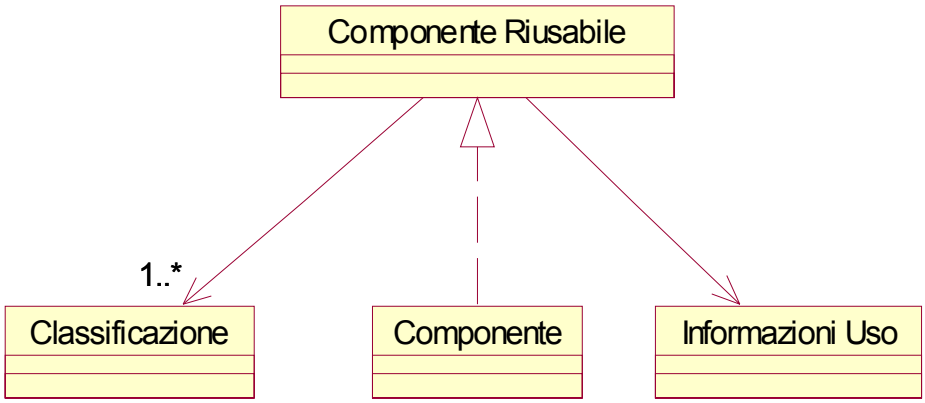


Figura 6.2 – *Le informazioni di classificazioni, che consentono di trovare il componente, e le informazioni d'uso, che permettono di valutarlo ed utilizzarlo, rendono un componente, realizzato secondo i principi della progettazione orientata agli oggetti, un componente riusabile.*

Seguendo la specifica Reusable Component Specification (RCS di ComponentSource [40]) un componente riusabile è organizzato secondo quattro dimensioni principali: funzionalità, tecnologia, distribuzione e commercio, come mostrato dalla figura 6.3.

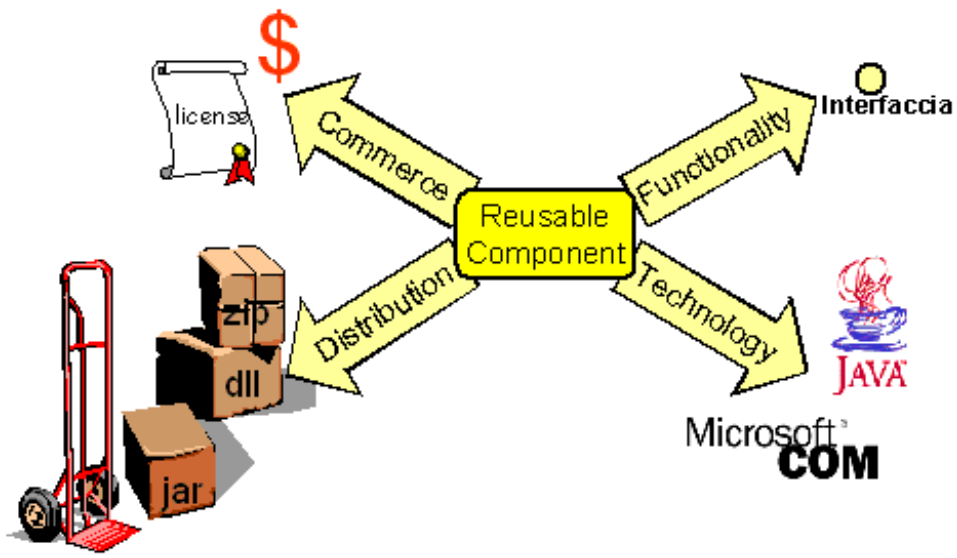


Figura 6.3 – *Definizione di componente riusabile secondo la specifica RCS.*

La tecnologia ricopre gli aspetti dello sviluppo del componente, le piattaforme a cui è conforme (ad esempio, J2EE, COM/.Net) e quindi l'ambiente e l'architettura in cui può essere eseguito.

Le funzionalità si riferiscono alle specifiche funzionali del componente mostrate attraverso la sua interfaccia.

La terza dimensione riguarda la modalità di distribuzione dei file fisici (dll, jar, etc.) in cui il componente è impacchettato, e della relativa documentazione d'installazione comprensive delle eventuali istruzioni per il deployment all'ambiente di runtime.

Infine, la dimensione commerciale aggiunge tutte le informazioni economiche necessarie affinché possa partecipare, ad esempio, ad un mercato on-line di componenti. Queste informazioni sono inerenti alla determinazione del prezzo, alle norme di concessione della licenza, tipologie di contratti, garanzia di qualità, e altro.

Interpretando queste definizioni in termini di Web Services, ovvero se le funzionalità del componente vengono esposte mediante un Web Service, allora si possono suddividere i servizi in esterni ed interni.

3.2.1. Internal Web Services.

Un Web Service è interno quando il servizio che espone e la sua invocazione sul Web appartengono alla stessa organizzazione.

Generalmente, come primo caso di studio, le compagnie usano i servizi Web per integrare applicazioni interne e per esporre applicazioni legacy in un ambiente eterogeneo senza dover riscrivere un significativo ammontare di codice. Logicamente, si suppone che queste organizzazioni abbiano già un'infrastruttura Web tale da consentire di utilizzare la tecnologia Web Service.

Tuttavia, poiché la tecnologia dei Web Services è ancora giovane e immatura, e d'altronde molti aspetti, come quelli legati alle funzionalità di più alto livello (ad esempio, il meccanismo per l'autenticazione multiutente) non sono ancora bene definiti (non esiste uno standard a riguardo), non esiste, attualmente, un'infrastruttura basata sui Web Services veramente solida e completamente *non proprietaria* capace d'integrare i sistemi legacy e componenti leggermente più complessi in maniera agevole ed efficace.

Con riferimento alle quattro dimensioni descritte nel paragrafo precedenze, i servizi Web interni hanno come dimensione tecnologia quella basata sulle tecnologie di base dei Web Service (SOAP, WSDL e UDDI) che sono indipendenti dalle piattaforme, dai linguaggi di

programmazione e sono aperte, e non posseggono una dimensione economica, in quanto il loro uso è solo interno all'azienda.

3.2.2. External Web Service.

Ci troviamo in presenza di un Web Service esterno quando l'invocazione mediante il Web e il servizio appartengono, rispettivamente, a due compagnie differenti. Questi servizi, generalmente, sono creati e pubblicati da un'azienda, service provider, e consumati da un'altra, service requestor.

I Web Service esterni possono essere utilizzati per esporre componenti o servizi interni come prodotti in vendita e accessibili attraverso il Web. Logicamente, un'azienda deve conoscere quali tipi di servizi conviene esporre sul Web. Alcuni servizi (ad esempio applicazioni multimediali) hanno bisogno di molta larghezza di banda per funzionare adeguatamente, mentre altri, come, per esempio, motori di ricerca, sofisticati applicazioni per trasformazioni di immagini, o controllo di carte di credito lavorano con una larghezza di banda minore. Inoltre, il modello adottato dall'azienda deve essere in grado di valutare se un servizio è veramente spendibile oppure no. Comunque, certi servizi hanno sicuramente valore aggiunto se esposti come Web Service (ad esempio il controllo della carta di credito). In questo senso, i Web Service possono essere riguardati come nuovi canali di distribuzione.

Per consentire alle compagnie di utilizzare i servizi offerti da terze parti, molte compagnie stanno seriamente pensando di sviluppare nuovi modelli per la gestione dei cataloghi di Web Service, come

registri UDDI, e altri tipi di sistemi di localizzazione. Attualmente, questi registri sono pubblici e non è garantito alcun controllo sulla qualità dei prodotti pubblicati.

La figura 6.4 illustra uno scenario complesso (detto ecosistema) basato sull'uso misto di servizi Web interni ed esterni. In quest'integrazione di sistemi eterogenei, le compagnie oltre ad utilizzare i servizi interni, integrano nelle loro applicazioni servizi Web forniti da partner commerciali e nello stesso tempo sono anche fornitori.

Tuttavia, la mancanza di specifiche largamente accettate per la sicurezza, l'instradamento dei messaggi XML, le modalità di pagamento, e altre funzionalità limita l'integrazione ai soli casi in cui esiste un'intesa a priori sulla tecnologia usata per implementare queste funzionalità.

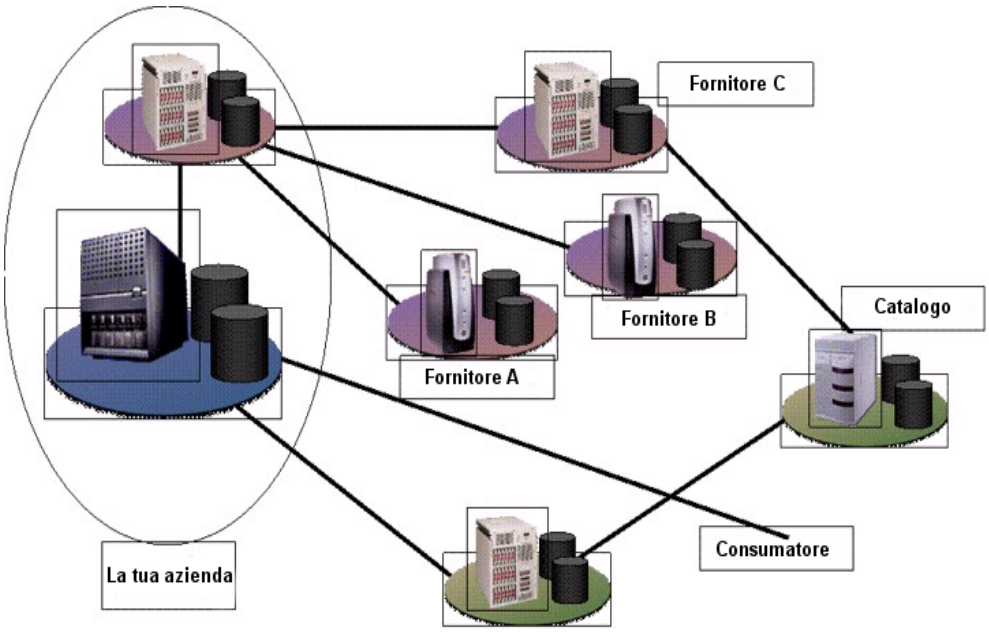


Figura 6.4 – *Interoperabilità tra molteplice compagnie.*

4. Riuso dei Web Services interni.

Per il riuso delle funzionalità esistenti, specialmente per quelle la cui implementazione ha consumato tempo e risorse, ci sono attualmente due problemi da risolvere:

- Refactoring. Poichè le capacità dei sistemi esistenti/legacy sono state progettate e implementate per altri scopi, bisognerebbe spezzarle in pezzi più piccoli per formare delle "micro" funzionalità. Successivamente, queste si devono assemblare e aggregare per formare funzionalità meno complesse e più utili.

- Exposing. Le funzionalità devono essere esposte in modo da essere disponibili alle tecnologie di sviluppo usate all'interno dell'azienda. Ad esempio, un gruppo di progetto che sviluppa in Java deve essere in grado di riutilizzare funzionalità implementate in oggetti COM.

Dare una soluzione al primo problema va oltre l'intento di questo lavoro. Per quanto riguarda il secondo, viene fornita un'indicazione di come sia possibile, mediante un'approccio basato sui Web Services, realizzare componenti riusabili e riutilizzare e condividere codice esistente indipendentemente dall'ambiente di programmazione.

Infatti, l'uso dei servizi Web nell'ambito intranet consente di integrare nelle applicazioni molti tipi di componenti, ad esempio, componenti Java, EJB (Enterprise Java Bean), componenti COM, ma anche

programmi scritti in C, Fortran e script come JavaScript, JScript, Vbscript e altri moduli software.

Questo conferisce a servizi Web interni un forte grado di riutilizzo e condivisione. Gli sviluppatori di componenti Java e componenti COM anziché creare gli stessi servizi in due tipi di linguaggi differenti possono riutilizzare nei loro progetti un unico servizio indipendentemente dal linguaggio usato per l'implementazione.

Tuttavia, non esiste un unico ambiente SOAP per esporre ed eseguire tutti i componenti come servizi Web. Le soluzioni offerte dipendono principalmente dalla scelta del sistema operativo e dal linguaggio di programmazione con cui le funzionalità, che si desiderano esporre come Web Services, sono implementate. Sostanzialmente, un ambiente SOAP (ambiente runtime) deve essere in grado di localizzare il servizio e attivarne le funzionalità richieste. Logicamente la seconda operazione dipende fortemente dal tipo di applicazione e dal tipo di linguaggio con cui è implementata. Ad esempio, se si tratta di una libreria scritta in C, il modulo di attivazione dell'ambiente SOAP deve essere in grado di accedere alla libreria e chiamare la funzione che fornisce la funzionalità desiderata. Se invece si tratta di una classe, bisogna istanziarla ed invocarne il metodo opportuno. Ovviamente, istanziare una classe C++ è un'operazione diversa che istanziare una classe Java. E nel primo caso dipende anche dal sistema operativo utilizzato. Inoltre, non è raro che lo sviluppatore dovrà costruire opportuni wrapper (per accedere alle applicazioni) nello stesso linguaggio di programmazione con cui è stato implementato l'ambiente che espone i Web Services.

Nella figura 6.8 sono mostrati i componenti che in generale dovrebbe possedere un'infrastruttuta che consente di consumare i Web Services.

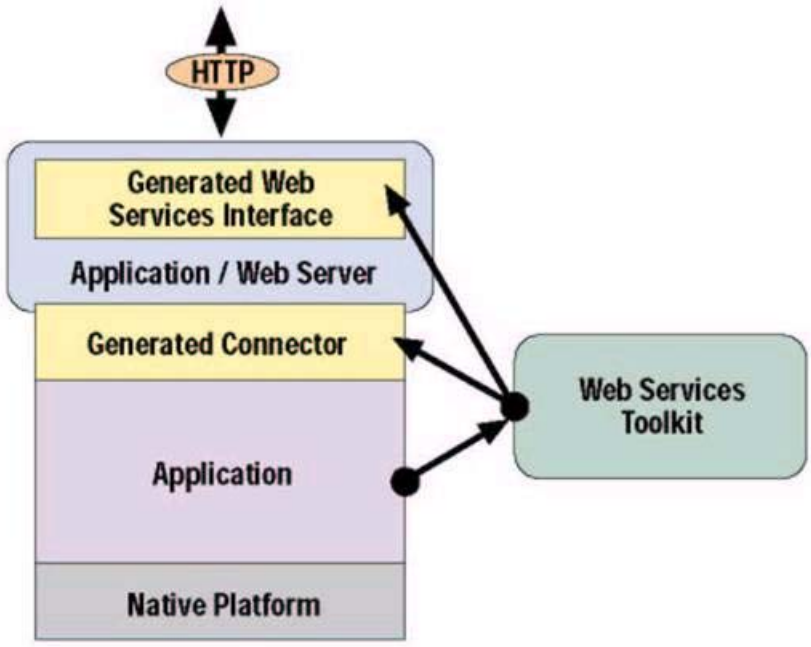


Figura 6.8 – *Componenti per realizzare un'infrastruttura che consenta di consumare i Web Services.*

Un'applicazione che implementa le funzionalità esposte dai Web Services. Questa può già esistere o essere sviluppate da zero usando un Web Services Toolkit provvisto di tutto il necessario per esporre i servizi nel formato Web Services, come ad esempio un server SOAP capace di servire chiamate a metodi remoti, utility per la generazione di proxy, etc.

Un Web Application server (Tomacat, Websphere, Weblogic, IIS) che offra un ambiente di runtime nel cui contesto possano essere eseguite le applicazioni Web che implementano i server SOAP (esempio, applicazioni asp, servlet, cgi). In alcune soluzioni (come SOAP::Lite) non occorre alcun Web server in quanto i Web Services sono

sviluppati come daemon che si mettono in ascolto sulla porta 80 per servire le richieste HTTP.

Ad esempio, un service provider può utilizzare l'ambiente di esecuzione SOAP Apache per esporre automaticamente ed attivare classi Java, EJB, script JavaScript e JPhyton come dei Web Services. Ma tale ambiente non offre alcun supporto per le classi C++, applicazioni C, e solo un timido approccio all'attivazione di oggetti COM. Per utilizzare applicazioni scritte in C bisogna costruire un wrapper utilizzando la Java Native Interface (JNI) (consente di definire delle interfacce per applicazioni native [44]) e costruirsi una classe Java. In figura 6.5 sono mostrate le applicazioni che SOAP Apache è in grado di eseguire come Web Services e i moduli richiesti per ottenere ciò.

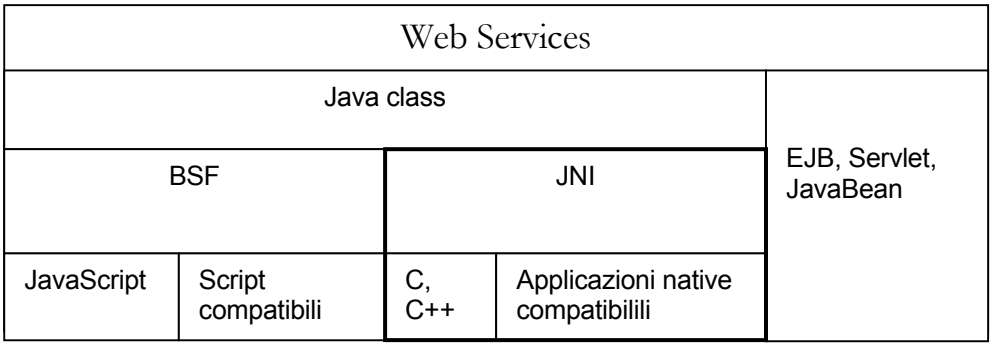


Figura 6.5 – *I componenti supportati dall'ambiente SOAP Apache.*

SOAP Apache utilizza l'architettura Bean Scripting Framework (BSF) [51] per incorporare ed attivare script nelle applicazioni Java e Applet Java. Attualmente, BNF supporta: JavaScript versione 1.5 (Mozilla Rhino, *http://www.mozilla.org/rhino*), JPython versione 1.1 (*http://www.jpython.org/*), Jacl versione 1.2.6 (*http://www.scriptics.com/java*)

, Win32 ActiveScript: JScript, VBScript, PerlScript (*http://msdn.microsoft.com/scripting*), ed altri.

Il discorso per JNI è diverso. Infatti, in questo caso bisogna scrivere un'opportuna classe Java che utilizza particolari chiamate a metodi nativi mediante le librerie dinamiche (.dll nel caso di Windows ed .so nel caso di Unix). Quando si richiamano i metodi dell'applicazione nativa i tipi Java degli argomenti vengono codificati nei rispettivi tipi del linguaggio C ed avviene esattamente il contrario per i valori di ritorno. Le operazioni di codifica e decodifica dei tipi sono possibili grazie ad opportune librerie C fornite dal JDK.

Nella figura 6.6 è mostrato come JNI si colloca tra i due l'ambienti C e Java.

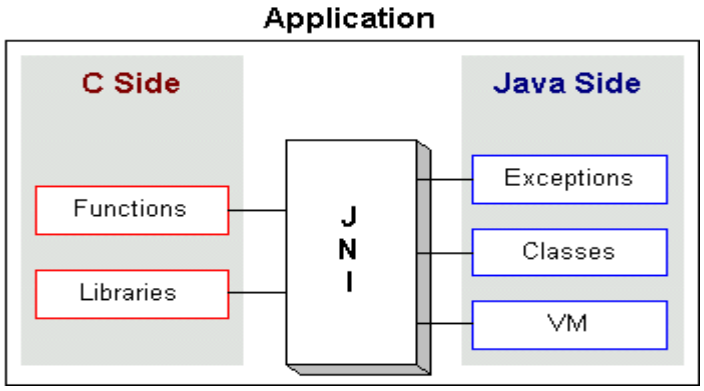


Figura 6.6 – *Interfacciamento mediante JNI tra applicazioni in linguaggio C e applicazioni in Java.*

Se invece il service provider desidera esporre oggetti COM come Web Services, dovrebbe utilizzare il toolkit Mirosoft SOAP. Questa scelta comporta anche l'istallazione del Web server Internet Information

Server (IIS) su una piattaforma Microsoft. In figura 6.7 è mostrata l'architettura di base delle API di alto livello di SOAP Microsoft.

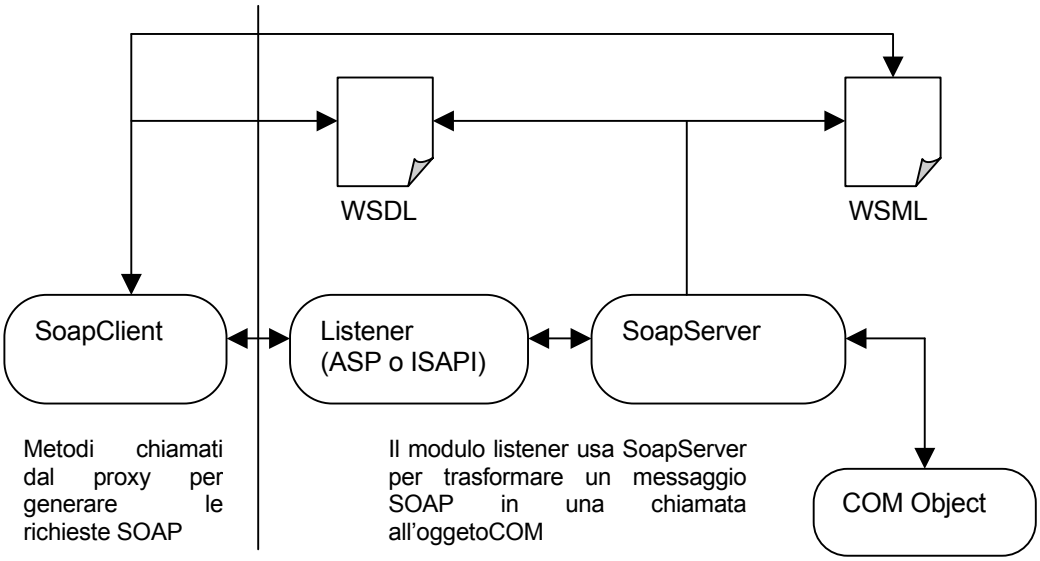


Figura 6.7 – *Diagramma dell'architettura di base del prodotto SOAP Microsoft.*

Il documento Web Services Meta Language (WSML) è un documento XML con una propria grammatica. Esso viene usato per mappare gli elementi WSDL , in particolare le operazioni, sui metodi dell'oggetto COM [8].

Si può adottare anche una strategia basata sull'ambiente Phyton usando i tools forniti per costruire wrapper per le librerie di funzioni C e le classi C++, e nello stesso tempo implementare Web Services nel linguaggio Phyton. Le utility per la generazione di wrapper Phyton di librerie di funzioni C sono fornite dal pacchetto SWIG (Simplified Wrapper and Interface Generator). Per le classi C++ invece si può usare SIP, simile al precedente. Una possibile soluzione potrebbe consistere nel creare wrapper Phyton per funzioni di librerie C, per

esempio, per semplice funzioni si può usare SWIG per generare il codice C per un modulo di estensione Phyton, invece per funzioni più complesse (traslazione dei tipi di dati più complicata) è necessario operare a mano. Una volta ottenuto il modulo Phyton, questo potrà essere tranquillamente esposto come Web Services.

Un altro ambiente SOAP per linguaggio Perl è SOAP::Lite. Fornisce un set di oggetti Perl che possono essere usati per costruire agevolmente server e client SOAP. E' fornito sia per Unix sia per Windows. Un Web Service implementato in Perl è self-contained ed è avviato con un proprio listener HTTP integrato in modalità "daemon" che si gestiscono le richieste che arrivano sulla porta 8080. Il server SOAP:Lite può anche essere avviato come un'applicazione CGI nel costesto di un Web server.

EasySoap++ toolkit è invece indicato per esporre mediante Web Service i metodi delle classe scritte in linguaggio C++. Esiste sia una versione per linux/unix sia per Windows. Il Web Service deve essere costruito includendo le librerie *SOAPCGIServer.h* nella classe C++ che si sta realizzando. Alla fine viene prodotto un eseguibile CGI (.cgi) che deve essere sistemato nell'opportuna repository di un Web Server che supporta le applicazioni CGI.

Infine, SOAPx4 è un ambiente per script PHP e necessita di un Web server PHP-enabled (come Apache e IIS).

Dato che la repository dei Web Services è costituita dal registro UDDI, questo viene usato come catalogo per memorizzare e cercare le

descrizioni sui servizi riusabili, come la definzione dell'interfaccia, i punti d'accesso, ed altre informazioni.

4.1. Utility a supporto del riuso.

L'obiettivo è mettere insieme alcune utility per realizzare un prototipo come supporto all'automazione di certe attività associate al processo del riuso di componenti software.

Queste utility sono state suddivise in tre insiemi in funzione degli attori mostrati dal caso d'uso di figura 6.8.

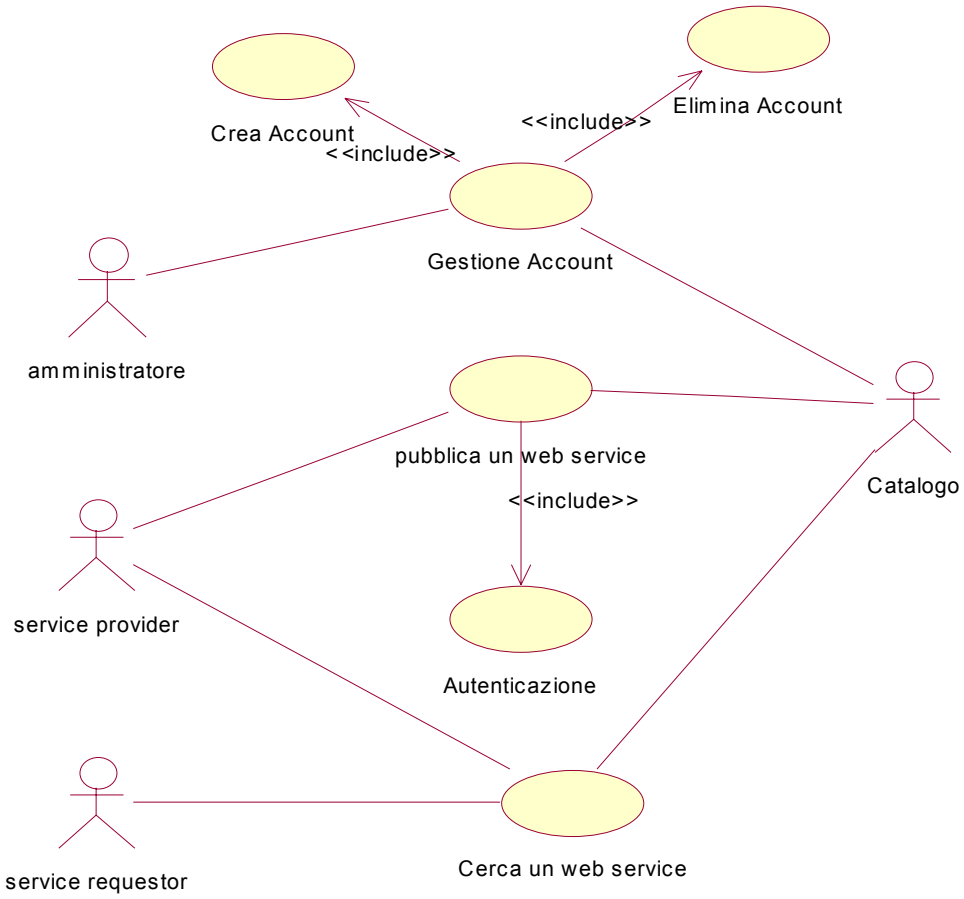


Figura 6.8 – *Use case con riferimento agli scenari fondamentali del prototipo.*

Di seguito vengono dettagliati i vari ruoli e le utility a loro disposizione per realizzare le operazioni fondamentali mostrate nel caso d'uso: pubblicazione dei servizi Web, scoperte degli stessi e gestione degli account per l'autorizzazione alla pubblicazione del catalogo.

Web Service Provider:

Sviluppa il servizio Web utilizzando i metodi definiti nel capitolo 2 (green field, top-down, bottom-up e meet-in-the-middle). In questo contesto, il provider è anche il fornitore della definizione dell'interfaccia del servizio ed è responsabile per la relativa pubblicazione. Deve altresì produrre i documenti d'informazione generale (overview document) sul servizio, le specifiche funzionali, guida utente e le istruzioni d'istallazione del componente (ad esempio come si integra il modulo proxy in un'applicazione o come si effettua il deploy del servizio Web all'ambiente di runtime per eseguirlo). Il service provider è anche responsabile della manutenzione del servizio.

Se il service provider effettua il deploy del servizio all'ambiente di runtime senza pubblicare la definizione dell'interfaccia, e quindi dell'implementazione, nel catalogo, il servizio non può considerarsi riusabile, anche se sarà comunque utilizzabile dagli sviluppatori messi a conoscenza dal provider. Inoltre, un servizio non è detto che sia riusabile anche se la sua definzione dell'interfaccia è stata pubblicata presso il catalogo. Sarà un opportuno comitato di saggi a decidere se un servizio è da ritenersi riusabile oppure no.

Un esempio di service provider è un gruppo di progetto di una unità organizzativa che decide di rendere disponibile agli altri gruppi aziendali un servizio sviluppato all'interno di un proprio progetto.

In figura 6.9 sono mostrate le applicazioni a supporto del service provider.

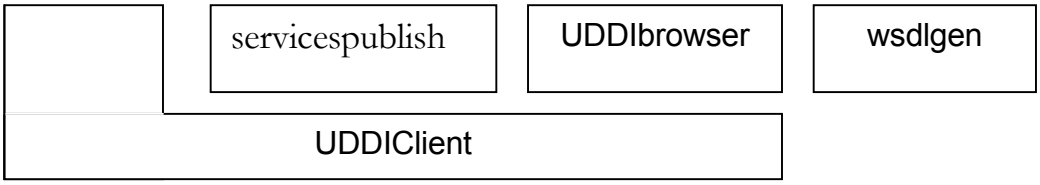


Figura 6.9 – *Insieme di utility che il service provider può utilizzare durante lo sviluppo dei Web Services.*

Tra queste applicazioni non ho incluso quella che consente di esporre i servizi Web perchè dipendente sostanzialmente dal tipo di ambiente SOAP runtime utilizzato e per questo motivo verrà presentata nel capitolo 7 quando sarà illustrato un semplice esempio.

L'applicazione Java *wsdlgen* è fornita dal toolkit IBM WSTK 2.3. Essa è dotata d'interfaccia grafica e consente la generazione della definizione dell'interfaccia da un bean Java, da un EJB e anche da un oggetto COM.

L'utility usa la riflessione per conoscere l'interfaccia pubblica di una classa Java (Java Reflection [50]) con la quale costruisce tre documenti XML: la definizione dell'interfaccia, la definizione dell'implementazione e il deployment descriptor (usato per il deploy del servizio all'engine SOAP).

Nella figura 6.10 è mostrata la finestra utente in cui vengono richiesti i parametri per la generazione automatica dei documenti XML.

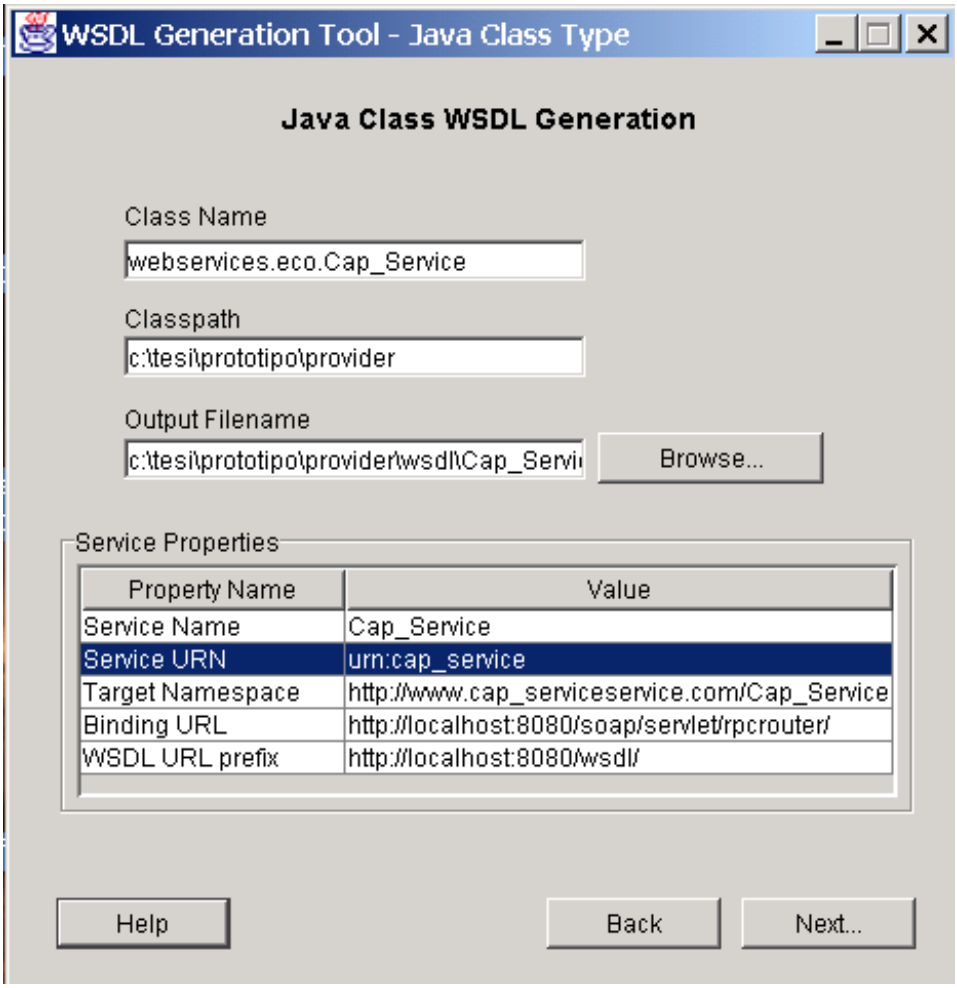


Figura 6.10 – *WSDL Generation tool di IBM.*

Di seguito riporto una breve descrizione.

Service Name: Nome del servizio come definito nella specifica WSDL.

Service URN: Identificatore del servizio nel broker di SOAP Apache.

Target Namespace: namespace per la definizione dell'interfaccia.

Binding URL: indica l'url in cui può essere localizzato il servizio. Nel caso di SOAP Apache, per esempio, questo indirizzo è *http://localhost:8080/soap /servlet/rpcrouter/*.

WSDL prefix: il namespace usato per il prefisso degli elementi WSDL.

In figura 6.11 sono mostrati i documenti XML che l'applicazione wsdlgen genera automaticamente a partire da una classe Java, EJB (file .jar) o da un'interfaccia COM [52].

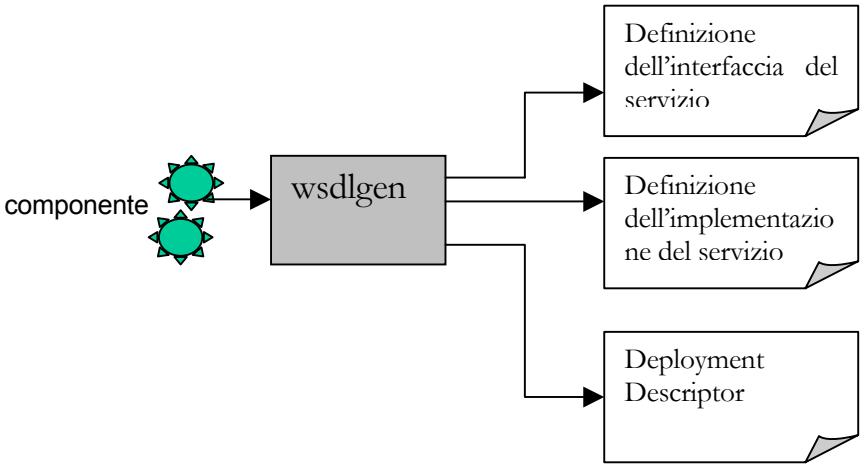


Figura 6.10 – *Documenti XML generati dal programma wsdlgen.*

Il documento XML deployment descriptor è usato dall'ambiente SOAP Apache per esporre il servizio Web e sarà trattato nel capitolo 7.

L'applicazione *servicepublish* è tuttora un prototipo che ho realizzato per consentire la pubblicazione semiautomatica del servizio Web nel catalogo.

Il programma Java è dotato di un'interfaccia utente realizzata utilizzando le librerie di classi Java Swing. Nella figura 6.12 riporto il

diagramma dei package con riferimento alle unità di più alto livello delle classi e le relative dipendenze.

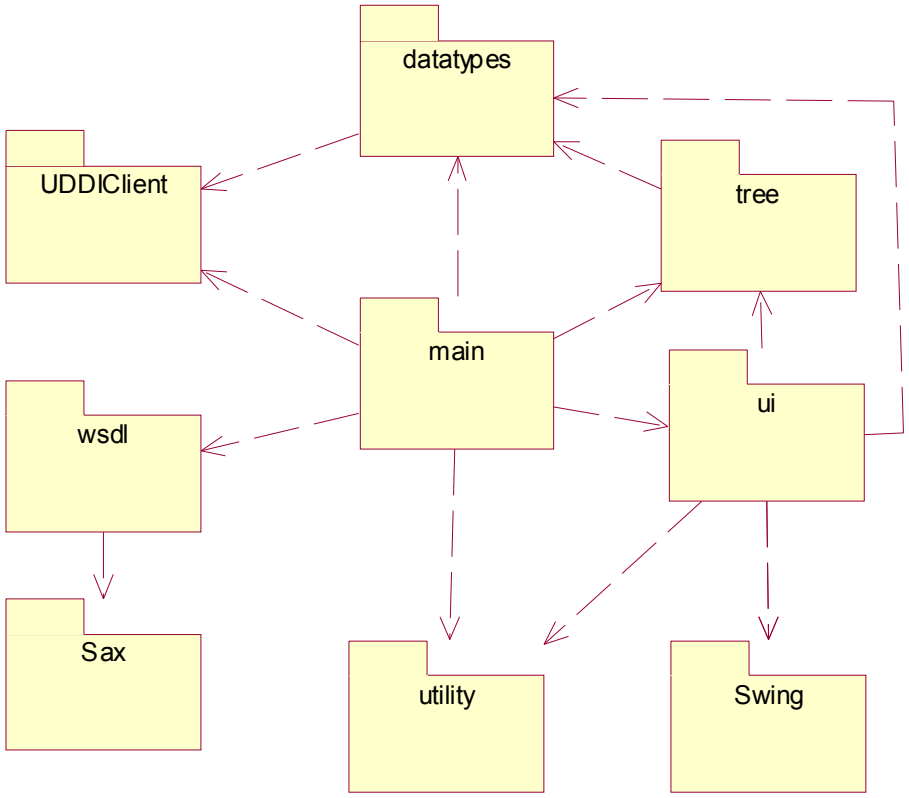


Figura 6.12 – *Diagramma dei package con riferimento al programma servicepublish.*

Le librerie di alto livello SAX e UDDIClient sono esterne all'applicazione e precisamente implementano, rispettivamente, il parser SAX (parte del pacchetto XercesJ di Apache, vedi capitolo 7) e un proxy per la comunicazione via SOAP con il server UDDI che implementa il catalogo dei Web Services. Le librerie UDDIClient implementano le API e le strutture dati della specifica UDDI 2.0 (vedi capitolo 5).

Nella figura 6.13 è riportato il caso d'uso relativo alla pubblicazione del servizio Web.

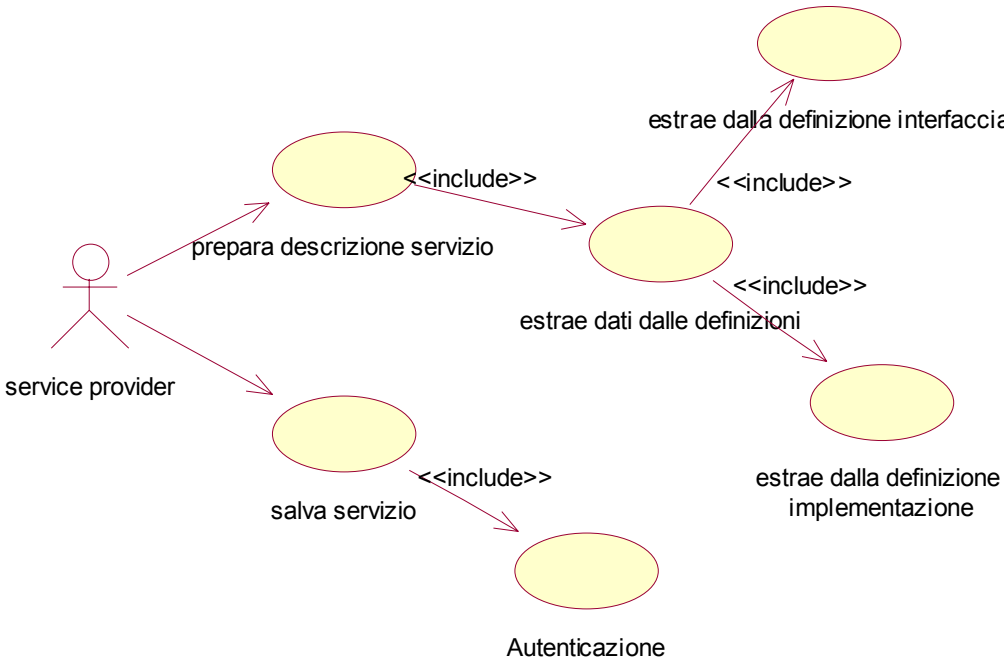


Figura 6.13 – *Caso d'uso relativo alla pubblicazione del servizio a partire dalle sue definizioni.*

Il diagramma di figura 6.14 illustra le interazioni relative allo scenario dell'estrazione dei dati dalla definizione dell'implementazione. Il parser SAX durante l'analisi del documento WSDL (definizione dell'implementazione) genera una serie di callback sul gestore del contenuto. Questo usa tali chiamate per settare le strutture dati *Import* e *Service*. La prima classe contiene l'informazione per raggiungere la descrizione dell'interfaccia, la seconda il nome del servizio e alcuni parametri di binding, come l'endpoint o punto d'accesso. Tali classi sono mostrate nel diagramma di classi concettuale riportato in figura 6.15.

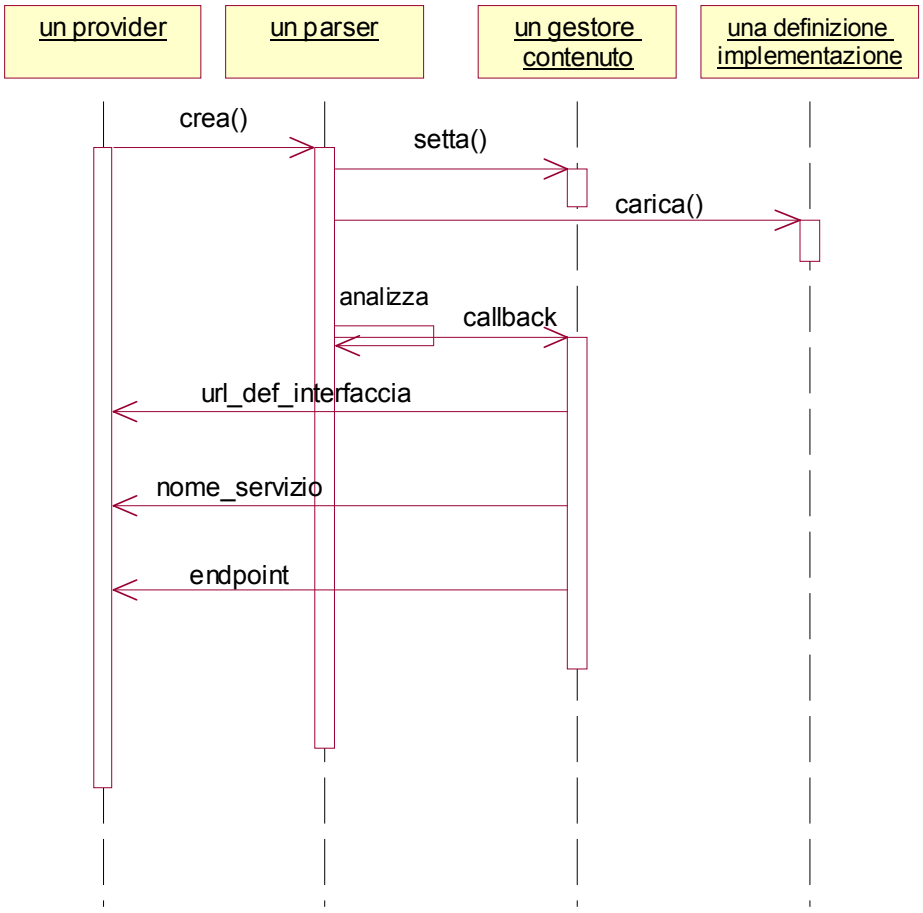


Figura 6.15 – *Estrazione dei dati dalla definizione dell'implementazione.*

Una volta localizzata (mediante l'attributo location di *Import, vedi acpitolo 4*), la descrizione dell'interfaccia viene analizzata dal parser che estrae il targetNamespace (vedi capitolo 5) usato per pubblicare il modello (tModel) rappresentativo di tale definizione. Successivamente, tale modello viene referenziato da un binding template del servizio, come riportato nel paragrafo 5.1. Il caso d'uso mostrato in figura 6.16 (scenari raltivi al caso d'uso "prepara definizione servizio" di figura 6.13) illustra questo concetto.

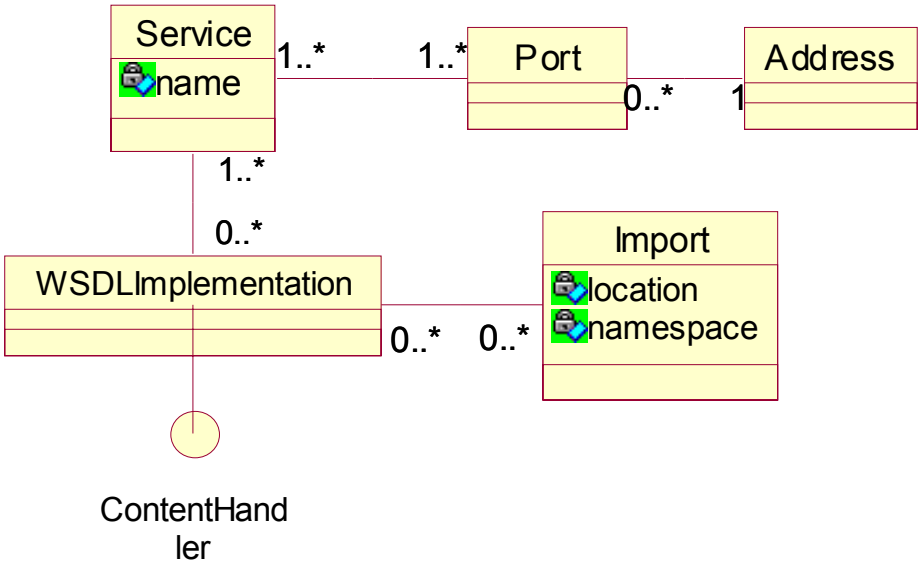


Figura 6.15 – *Diagramma di classe concettuale con riferimento alle strutture dati estratte dal documento della definizione dell'implementazione.*

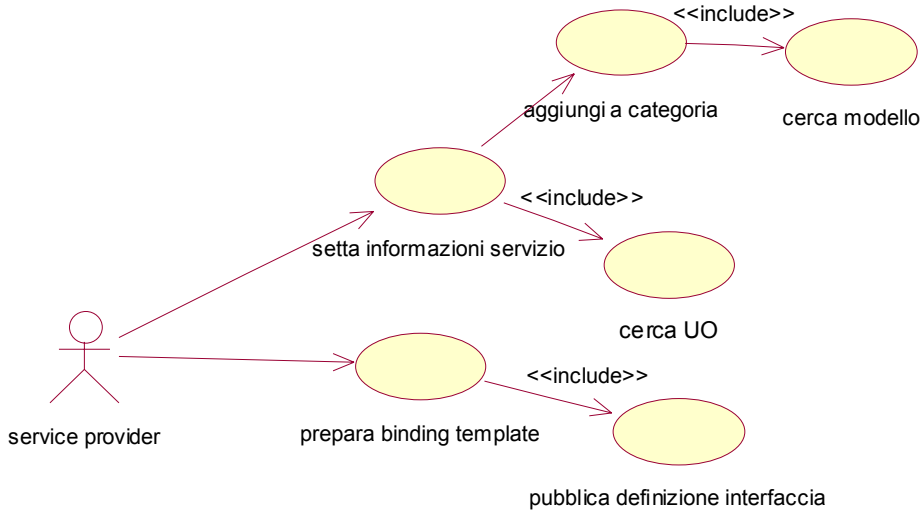


Figura 6.16 – *Caso d'uso relativo alla preparazione della descrizione del servizio.*

Nel listato 6.1 è presentata la classe che implementa il contentHandler del parser SAX e che consente di tirar fuori da documento WSDL gli elementi *import* e *service*.

```
import java.util.Vector;
import org.xml.sax.*;

 /**Implementa il gestore del contenuto di un documento WSDL che definisce
  * l'implementazione del Web Services.
  */
public class WSDLImplementation implements org.xml.sax.ContentHandler{

private Vector listImport=new Vector(); //elemento import
private Vector listService= new Vector();//elementi service
private Vector listPort=new Vector();//elementi port
private Service service=null;
private Port port=null;
private Address address=null;//address.location == accessPoint

  public void setDocumentLocator(org.xml.sax.Locator locator){}
  public void startDocument() throws SAXException{
  System.out.println("Analisi del file di configurazione in corso...");
  }
  public void endDocument() throws SAXException{
  System.out.println("Fine analisi del file");
  }
  public void processingInstruction(String target,String data) throws
SAXException{
  }
  public void startPrefixMapping(String prefix,String uri)
  {
  }
  public void endPrefixMapping(String prefix){
  }

  /** Questo metodo cattura gli eventi associati ai tag iniziali deli elementi.
   *  Per maggiori informazioni, consultare la dicumentazione dell'API SAX 2.0
   *
   */
  public void startElement(String namespaceURI,String localName,String
rawName,Attributes atts) throws SAXException{

    //acquisisce indirizzo per localizzare l'interfaccia.
    if (localName.equals("import")){
```

```
            Import include=new Import();
            include.setLocation(atts.getValue("location"));
            include.setNamespace(atts.getValue("namespace"));
            listImport.addElement(include);

                }
    else if (localName.equals("service")){
            //Servizi inclusi nel definizione dell'implementazione. Tuttavia,
            //in genere include un solo servizio.
            service=new Service();
            service.setName(atts.getValue("name"));}
    else if (localName.equals("port")){
             //setta port.
             port=new Port();
             port.setName(atts.getValue("name"));
            }
    else if (localName.equals("address")){
             address=new Address();
             address.setLocation(atts.getValue("location")); }

  }

public void endElement(String namespaceURI,String localName,String
rawName)throws SAXException{
    if (localName.equals("service")){
            //Servizi inclusi nel definizione dell'implementazione. Tuttavia,
            //in genere include un solo servizio.
            service.addPort(listPort);
            listService.addElement(service);}
    else if (localName.equals("port")){
             //setta port.
             listPort.addElement(port);
            }
    else if (localName.equals("address")){
             port.addAddress(address);
    }
  }

  /** Questo metodo consente di catturare i callback associati ai contenuti degli
elementi
   *
   */
  public void characters(char[] ch,int start,int length) throws SAXException{
   }
  public void ignorableWhitespace(char[] ch,int start,int length) throws
SAXException{
```

```
}
public void skippedEntity(String name) throws SAXException{
}

public Vector listImport(){
return listImport;
}
public Vector listService(){
return listService;
}
}
```

Listato 6.1 – *Implementazione del gestore del contenuto del documento WSDL della definizione dell'implementazione del servizio.*

Il sequence diagram di figura 6.17 rappresenta le interazioni per settare il BusinessService con l'indetificatore unico dell'unità organizzativa (UO) che pubblica il servizio.

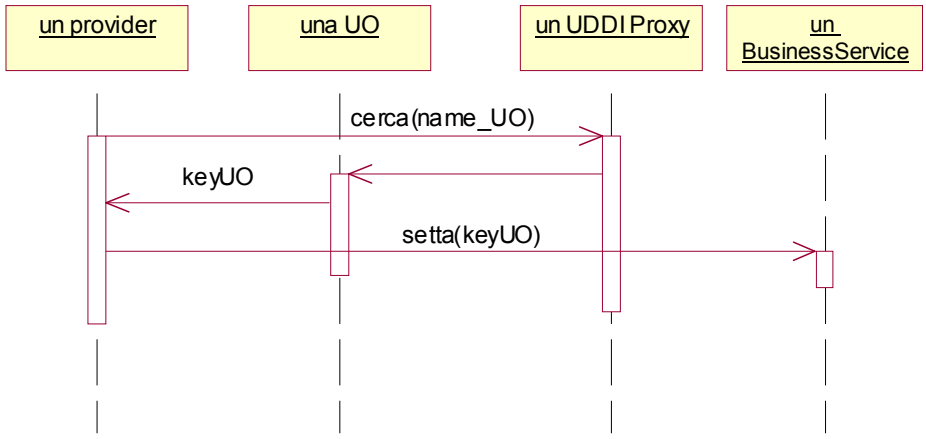


Figura 6.17 - *Sequence diagram relativo all'aggiunta del servizio all'UO che lo fornisce.*

Il sequence diagram di figura 6.18 mostra invece le interazioni coinvolte nel caso di studio di figura 6.16.

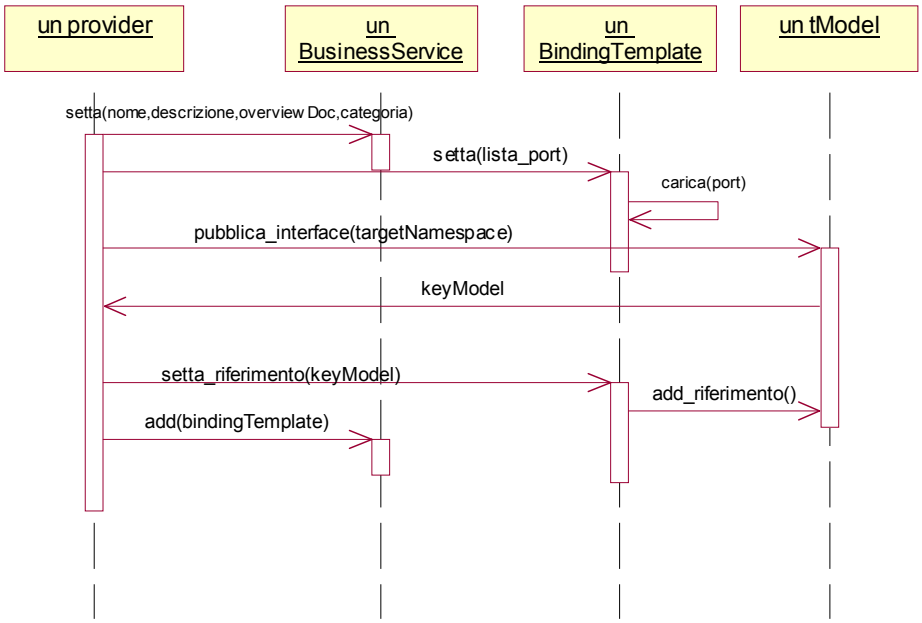


Figura 6.18 – *Sequence diagram che illustra come viene preparata la struttura binding template.*

Il listato 6.2 rappresenta la classe che implementa il contenuto del gestore della definizione dell'interfaccia. Essa consente di estrarre l'attributo *targetNamespace* dell'elemento *definitions* e l'attributo name di *binding* dalla definizione dell'interfaccia. Questa classe implementa il contentHandler (il gestore del contenuto di un documento XML) del parser SAX 2.0 implementato dalla libreria Xerces di Apache.

```
import org.xml.sax.*;

public class WSDLInterface implements org.xml.sax.ContentHandler{

private String targetNamespace=null;
private String[] bindingAtts= new String[2];

  public WSDLInterface(){}

  public void setDocumentLocator(org.xml.sax.Locator locator){
  }
```

```
  public void startDocument() throws SAXException{
  //System.out.println("Analisi del file di configurazione in corso...");
  }
  public void endDocument() throws SAXException{
  //System.out.println("Fine analisi del file di configurazione in corso...");
  }
  public void processingInstruction(String target,String data) throws
SAXException{
  }
  public void startPrefixMapping(String prefix,String uri)
  {
  }
  public void endPrefixMapping(String prefix){
  }

  /** Questo metodo cattura gli eventi associati ai tag iniziali deli elementi.
   *  Per maggiori informazioni, consultare la dicumentazione dell'API SAX 2.0
   *
   */
  public void startElement(String namespaceURI,String localName,String
rawName,Attributes atts) throws SAXException{

    //acquisisce attributi opportuni.
    if (localName.equals("definitions"))
          targetNamespace=atts.getValue("targetNamespace");
    else if (localName.equals("binding")){
           /*estrae solo il nome locale binding con prefisso wsdl (wsdl:binding)
           /*o con senza prefisso*/
           if (namespaceURI.equals("http://schemas.xmlsoap.org/wsdl/") ||
namespaceURI.equals("")){
           for(int i=0;i<atts.getLength();i++){
           bindingAtts[i]=atts.getValue(i);}}
          }
  }

  public void endElement(String namespaceURI,String localName,String
rawName)throws SAXException{
  }

  /** Questo metodo consente di catturare i callback associati ai contenuti degli
elementi
   *
   */
  public void characters(char[] ch,int start,int length) throws SAXException{
   }
```

```
  public void ignorableWhitespace(char[] ch,int start,int length) throws
SAXException{
  }
  public void skippedEntity(String name) throws SAXException{
  }

  public String getTargetNamespace(){
  return targetNamespace;
  }

/**Restituisce il nome dell'elemento binding
 * @return nome dell'elemento binding
 */
  public String getBindingName(){
  return bindingAtts[0];
  }

}
```

Listato 6.2 – *Questa classe implementa parzialmente il gestore del documento WSDL. Dal momento, che occorre estrarre poche informazioni ho ritenuto opportuno usare il parser SAX invece del DOM.*

Il content handler del documento deve essere associato al parser. Ad esempio, si può estendere la classe SAXParser e annettere il gestore, come mostrato di seguito:

```
import java.net.*;
import java.io.*;

import org.xml.sax.*;
import org.apache.xerces.parsers.SAXParser;

public class WSDLParser extends SAXParser{

 public WSDLParser() {

 }

 public static void main(String argv[]){
  try {
   String filename="file:c:/tesi/prototipo/provider/wsdl/Cap_Service-
```

```
        Interface.wsdl";
    XMLReader parser=new WSDLParser(); //crea parser
   WSDLInterface interf=new WSDLInterface(); //crea gestore del documento
    parser.setContentHandler(interf); //associa il gestore al parser
    parser.parse(filename); //affettua il parsing del documento e genera le callback
        sul gestore

   System.out.println(interf.getBindingName());
   System.out.println(interf.getTargetNamespace()+" "+interf.getBindingName());

} catch(MalformedURLException e_url){ //gestione eccezioni
  System.out.println("Errore URL:"+e_url.getMessage());
  } catch(IOException e_io){
  System.out.println("Errore IO:"+e_io.getMessage());
  } catch(SAXException e_sax){
  System.out.println("Errore SAX:"+e_sax.getMessage());
 }
}
}
```

Per facilitare la generazione delle strutture dati che devono essere pubblicate nel catalogo ho implementato la classe mostrata nel listato 6.3. Questa consente di generare il modello della definizione dell'interfaccia a partire dal tergetNamespace, e gli oggetti BusinessService e BindingTemplate dai dati provenienti dalla definizione dell'implementazione.

```
 //import dal client UDDI [43].
 import org.frenchstudio.uddi.client.* ;
 import org.frenchstudio.uddi.datatypes.*;
 import org.frenchstudio.uddi.datatypes.business.*;
 import org.frenchstudio.uddi.datatypes.service.BindingTemplate;
 import org.frenchstudio.uddi.datatypes.service.*;
 import org.frenchstudio.uddi.datatypes.spec.*;
 import org.frenchstudio.uddi.datatypes.tmodels.*;
 import org.frenchstudio.pools.FastPoolVector;
 import org.frenchstudio.soap.SOAPException;
```

```
import org.xml.sax.*;
import org.apache.xerces.parsers.SAXParser;
import java.net.*;

  /**Crea gli oggetti delle classi TModel, BusinessService<BR>
   * e BindingTemplate già configurati con gli attributi che provengono <BR>
   * dalla definizione dell'interfaccia e da quella dell'implementazione.
   */
public class WSDL2UDDI {

 //URL per localizzare la definizione dell'interfaccia
  private URL urlWSDL=null;
 //attributi estratti dalla definizione dell'interfaccia
  private String targetNamespace=null;
  private String bindingName=null;
 //attributi estratti dalla definizione dell'implementazione
  private Service serviceWSDL=null;

  public WSDL2UDDI() {
  }

/**Setta l'url della definizione dell'interfaccia.
 * @param urlWSDL - url della definizione dell'interfaccia
 */
  public void setURLWSDLInterface(URL urlWSDL){

  this.urlWSDL=urlWSDL;
  }

/**Setta il servizio estratto dalla definizione dell'implementazione
 */
  public void setServiceWSDL(Service serviceWSDL){
  this.serviceWSDL=serviceWSDL;
  }

/**Setta il targetNamespace
 */
  public void setTargetNamespace(String targetNamespace){
  this.targetNamespace=targetNamespace;

  }

/**Setta il nome del tipo di binding
 */
  public void setBindingName(String bindingName){
  this.bindingName=bindingName;
```

```
  }

  /** Restituisce il modello che rappresenta l'interfaccia con i parametri
   *  provenienti dalla definizione dell'interfaccia.
   */
  public Modello getModel(){
  Modello model=new Modello();
  model.setName(targetNamespace); //setta il nome
  OverviewDoc urlDoc=new OverviewDoc(); //crea un riferimento alla
definizione
  String urlWSDLString=urlWSDL.toString();
  String urlResult=urlWSDL.toString();
  int len=urlWSDLString.length();
  if (urlWSDLString.charAt(len-1)!='/') //crea un url valido
  urlResult=urlResult+"/";
  urlDoc.setURL(urlResult+bindingName);
  model.setOverviewDoc(urlDoc); //attacca riferimento al documento
  return model;

  }

 /**Restituisce il servizio definito dalla specidica UDDI, settato col nome
  * proveniente dal documento della definizione dell'implementazione.
  */
  public BusinessService getBusinessService(){
  BusinessService businessService=new BusinessService();
  businessService.setName(serviceWSDL.getName()); //setta nome.
  return businessService;
  }

  /** Crea una lista di BindingTemplate per ogni wsdl:port presente nella
   *  definizione dell'implementazione e allega ciascuno ad un modello della
   *  definizione dell'interfaccia.
   *  @param urlWSDLImplem - localizza la definizione dell'implementazione
   *  @param keyModel - identificatore del modello.
   */

  public java.util.Vector getBindingTemplates(URL urlWSDLImplem,String
keyModel){

  java.util.Vector listBindingTemplate=new java.util.Vector(); //crea vettore
  java.util.Vector listPort=serviceWSDL.getPort(); //estrae lista dei Port
  int len=listPort.size();

  for(int i=0;i<listPort.size()-1;i++){
  Port port=(Port)listPort.elementAt(i);
```

```
AccessPoint accessPoint=new AccessPoint();
accessPoint.setURL(port.getAddress().getLocation());
System.out.println(accessPoint.getURL());
//Attualmente sembra l'unico trasporto utilizzato
accessPoint.setURLType("http");
BindingTemplate bindingTemplate=new BindingTemplate();
bindingTemplate.setAccessPoint(accessPoint);

//creazione del riferimento al modello dell'interfaccia.
TModelInstanceInfo ins=new TModelInstanceInfo();
ins.setTModelKey(keyModel);
OverviewDoc doc=new OverviewDoc();
doc.setURL(urlWSDLImplem.toString());
InstanceDetails iDetails=new InstanceDetails();
iDetails.setOverviewDoc(doc);
Description desc=new Description();
desc.setLanguage("it");
desc.setText("Per raggiungere il servizio seguire il documento riferito");
iDetails.addDescription(desc);
ins.setInstanceDetails(iDetails);

bindingTemplate.addTModelInstanceInfo(ins); //allega
listBindingTemplate.addElement(bindingTemplate);
}
return listBindingTemplate;
}
```

Listato 6.3 – *Crea le strutture per la pubblicazione della definizione dell'interfaccia e della definizione dell'implementazione.Si trova nel package wsdl.*

Infine nella figura 6.19 è riportata la sequenza di operazioni per portare a termine la pubblicazione del servizio nel catalogo.

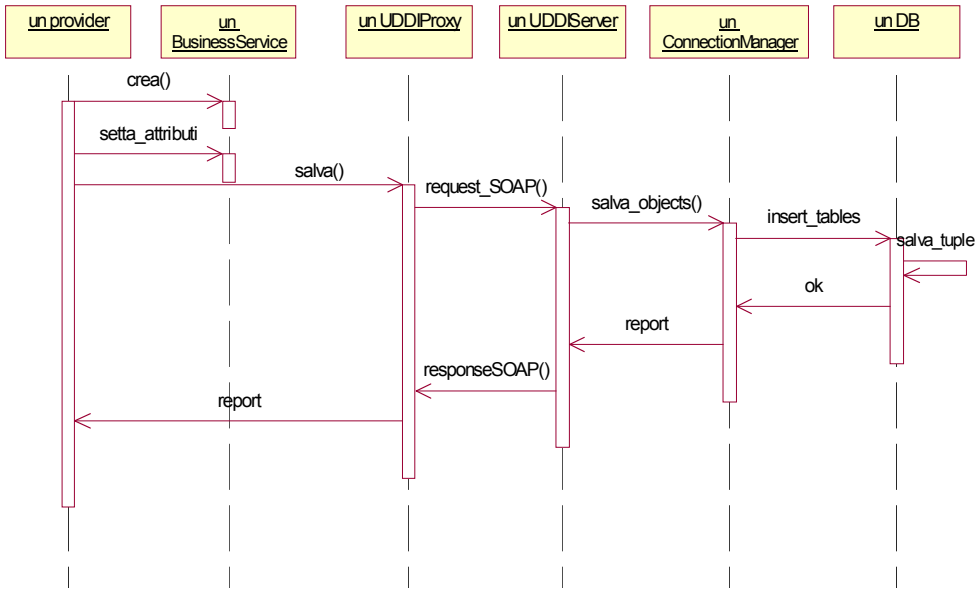


Figura 6.19 – *Sequence diagram relativo alla pubblicazione del servizio.*

Per salvare l'oggetto della classe BusinessService (il servizio Web che si vuole pubblicare nel catalogo) occorre acquisire un proxy UDDI dalla classe statica Proxy, mostrata nel listato 6.4.

```
import java.net.MalformedURLException;
import org.frenchstudio.uddi.client.UDDIProxy;

/**Questa classe inizializza e gestisce il proxy UDDI che implementa
 * le API UDDI 2.0.
 *
 */
 public class Proxy {

  static private String endpoint="http://jack:8080/uddi/servlet/uddi";
//endpoint del server UDDI
  static private UDDIProxy proxy=null;

  /**Restiutisce l'oggetto proxy
   *
   */
  public static UDDIProxy getUDDIConnection() {
  return proxy;
  }
```

```
  /**Inizializza il proxy.
   *
   */
  public static void init(String _endpoint) throws MalformedURLException{
  endpoint=_endpoint; //Setta endpoint
  proxy=new UDDIProxy(); //crea proxy
  proxy.setInquiryEndpoint(endpoint); //setta endpoint per la pubblicazione
  proxy.setInquiryEndpoint(endpoint); //setta endpoint per la consultazione
  }
}
```

Listato 6.4 – *Gestisce il proxy UDDI.*

Nella figura 6.20 è mostrata l'interfaccia della classe UDDIProxy.

Figura 6.20 – *Interfaccia UDDIProxy la cui implementazione consente di salvare, aggiornare, eliminare e trovare le strutture dati definite dalla specifica UDDI 2.0 (vedi capitolo 5).*

L'UDDIbrowser è un'applicazione che consente di navigare tra i servizi offerti dalle varie UO, cercarli, consultarne le descrizioni e scaricare le definizioni dell'interfaccia e dell'implementazione. Viene usato sia dal service provider sia dal service requestor. Nel primo caso, l'utente autorizzato (deve possedere un account) può modificare ed eliminare i servizi pubblicati dalla sua unità organizzativa. Tuttavia, non può modificare o eliminare modelli, unità organizzativa o creare classificazioni, operazioni queste di competenza del gestore del catalogo. Nel secondo caso, l'utente può solo consultare le informazioni.

Le classi implementate per l'UDDIbrowser sono fondamentalmente tutte d'interfaccia utente dipendenti dalla libreria Java Swing, dal proxy UDDI e dalle struttura dati UDDI, come mostrato dal diagramma dei package di figura 6.21.

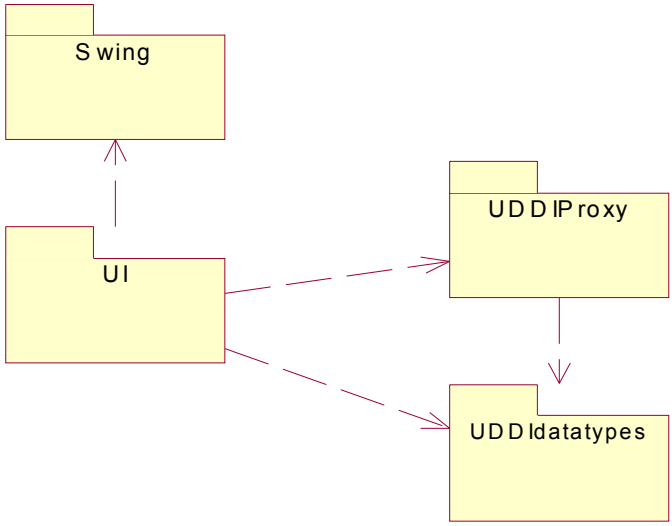


Figura 6.21 – *Diagramma dei package con riferimento all'applicazione UDDIbrowser. Nel pacchetto UI ci sono classi che realizzano le interfacce grafiche utente. L'applicazione è tuttora un prototipo.*

Web Service Requestor

Rappresenta il consumatore di servizi Web. Ricerca il servizio nel catalogo, controlla che si tratta di un servizio riusabile, ne visualizza la descrizione generale e se ritiene che fa al suo caso scarica il necessario per utilizzarlo. Il prototipo realizzato consente di cercare ed eventualmente utilizzare anche servizi che non sono identificati come riusabili, per i quali non si garantisce un'interfaccia stabile, il corretto funzionamento, la completa descrizione, il supporto alla manutenzione, etc.

Il requestor può scaricare il necessario per utilizzare il servizio in tre forme diverse:

- La definizione dell'implentazione (documento WSDL).
- Il modulo proxy.
- Il pacchetto compresivo del servizio e quanto altro occorre per la sua istallazione.

La prima forma di distribuzione, teoricamente, consente di generare il modulo proxy (che consente di negoziare con il servizio Web) nel linguaggio di programmazione desiderato, disaccoppiando completamente l'ambiente di programmazione del requestor dal servizio Web. Per quanto riguarda questo prototipo, è possibile

generare solo proxy implementati in linguaggio di programmazione Java.

La seconda forma di distribuzione pur contravvenendo allo spirito dei Web Services (che si basano sulla distribuzione della definizione del servizio mediante documenti WSDL), può rivelarsi l'unica soluzione in quei casi in cui i proxy non sono generabili automaticamente dai documenti WSDL, o il tool di generazione non fornisce il supporto a quel particolare linguaggio di programmazione usato per implementare il proxy.

Il terzo caso consente di scaricare un file che contiene tutti i moduli e le descrizioni necessarie per usare il servizio nell'ambiente di esecuzione del requestor (non attraverso il proxy). Ad esempio, nel caso di componenti Web Java (servlet, jsp) si possono legare tutti i moduli (compresi pagine html, immagini gif, e deployment descriptor) in un unico file con estenzione .war [22]. Questo file, una volta scaricato, può essere istallato nella repository di un Web server ed essere eseguito nel Web Container [48]. Nel caso degli EJB invece può essere usato il file .jar comprensivo di deployment descriptor, interfacce e classi [48].

Un esempio di service requestor può essere un gruppo di progetto di una unità organizzativa che ricerca nella intranet aziendale la disponibilità di servizi sw da utilizzare all'interno di un proprio progetto.

In figura 6.22 sono mostrate le applicazioni a supporto del service requestor.

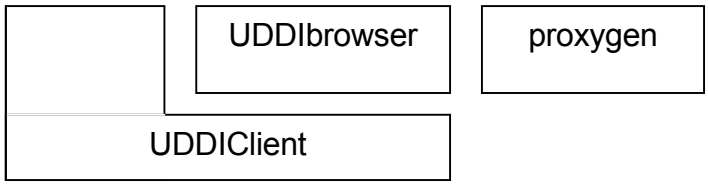


Figura 6.22 – *Utility a disposizione del service requestor.*

Proxygen fa parte del toolkit IBM WSTK 2.3 e consente di generare il modulo proxy client in Java dalla definizione dell'interfaccia o dalla definizione dell'implementazione del servizio. Nel primo caso, la classe generata ha tutti i metodi che consentono di comunicare via SOAP con il Web Services, ma non conosce l'endpoint per raggiungerlo, che dovrà essere settato dall'utente. Nel secondo caso, il proxy generato possiede anche l'informazione sul punto d'accesso del servizio Web. Infatti, in questo caso, dalla definizione dell'implementazione del servizio, l'applicazione proxygen ricava l'indirizzo della definizione dell'interfaccia. Questa, dopo essere stata analizzata, viene usata per generare il sorgente .java con le librerie SOAP client e gli opportuni metodi. Successivamente, il programma compila il sorgente con l'informazione relativa al punto d'accesso ricavata dal parsing della definizione dell'implementazione. Un esempio di proxy verrà mostrato nel capitolo 7.

Web Service Catalog Manager

Gestisce gli account dei service provider, decidendo chi può pubblicare i servizi e le relative descrizione nel catalogo. Definisce e cura uno o più schemi di classificazione dei servizi che consente ai service requestor di trovare rapidamente e agevolmente i servizi desiderati. E

infine, stabilisce quali servizi tra quelli pubblicati soddisfano i requisiti di riusabilità.

Esempio: un "comitato di saggi" (comitato direttivo) si prende in carico di standardizzare il processo di pubblicazione e di catalogare i servizi.

In figura 6.23 sono mostrate le applicazioni a supporto del service catalog manager.

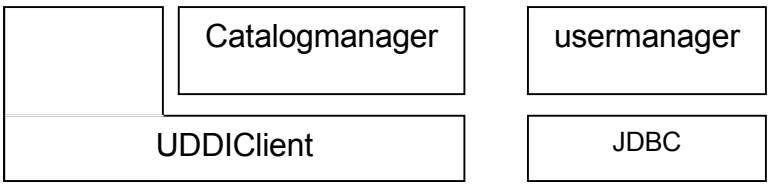


Figura 6.23 – *Utility a supporto del gestore del catalogo dei Web Services.*

L'applicazione Catologmanager dovrebbe consentire ,fondamentalmente, la creazione e la gestione dei modelli del registro, delle tassonomie dei servizi e delle informazioni sulle unità organizzative che pubblicano i servizi. Quest'applicazione, attualmente, non è dotata d'interfaccia utente, e le operazioni suddette devono essere effettuate nel *main* del sorgente Catalogmanager.java attraverso sequenze d'istruzioni. Nel capitolo 7 mostrerò come creare la struttura unità organizzativa e come effettuare una classificazione dei servizi.

Per quanto riguarda usermanager, nel listato 6.5 è mostrata la classe che crea e gestisce le connessioni al database tramite il protocollo JDBC.

```
import java.sql.*;
import org.eco.uddi.admin.datatypes.Admin;
import org.eco.uddi.admin.exception.ADMINException;

/**Questa classe crea e gestesce le connessioni al DB
 *
 */
public class ConnectionManager {

 private static Connection oConn=null;
 //Driver Oracle
 private static String driver="oracle.jdbc.driver.OracleDriver";
 private static Admin _admin=null;
 private static String _pathConnection=null;

public static void connect(String pathConnection, Admin admin) throws
ADMINException{

  _admin=admin;
  _pathConnection=pathConnection;

  ADMINException adminException=new ADMINException();
   try {

  //carica driver
  Class.forName (driver);
  // si connette
  oConn
=DriverManager.getConnection("jdbc:oracle:thin:"+admin.getUserID()+
  "/"+admin.getPassword()+"@"+pathConnection);

  System.out.println("Connessione OK");

   }  catch(SQLException sql_e)
       {adminException.setMessage(sql_e.getMessage());
        throw adminException;
       }
      catch (ClassNotFoundException e)
       {adminException.setMessage(e.getMessage());
        throw adminException;}
   }

  /**Restituisce una connessione al DB
   *
   */
  public static Connection getConnection() throws ADMINException{
```

```
ADMINException adminException=new ADMINException();
try{
if (oConn.isClosed()||oConn==null)
connect(_pathConnection,_admin);
} catch(SQLException e_close){
adminException.setMessage(e_close.getMessage());
throw adminException;}
return oConn;
}

}
```

Listato 6.5 *Classe che gestisce la connessione al database.*

Invece il listato 6.6 mostra il gestore degli oggetti utente (della classe User) che si occupa di caricarli, salvarli, aggiornarli ed eliminarli dal database.

```
import java.sql.*;
import java.util.Vector;


import org.eco.uddi.admin.datatypes.user.User;
import org.eco.uddi.admin.datatypes.DateEntity;
import org.eco.uddi.admin.datatypes.user.Quote;
import org.eco.uddi.admin.exception.ADMINException;

/**Questa classe consente di rendere permanenti alcuni oggetti, come user, etc.
 *
 */
public class DBCommands {

  //costanti
  public final String ALLUSERS="all";

  private Connection oConn=null;


  public DBCommands() throws ADMINException {
  oConn=ConnectionManager.getConnection();
  if (oConn==null){
  //System.out.println("Connessione inesistente");
```

```
ADMINException adminException=new ADMINException();
adminException.setMessage("Connessione inesistente");
throw adminException;
}

}



public Vector getUsers(String id) throws ADMINException{

ADMINException adminException=new ADMINException();
Vector arrayUsers= new Vector();
String baseQuery="SELECT * FROM users";
String query= id.equals("all")?baseQuery :baseQuery+" where userid="+id;
//String query="SELECT * FROM prova";

try {
Statement stmt=oConn.createStatement();
ResultSet rs= stmt.executeQuery(query);

while (rs.next()){
User user=new User();
Quote quota=new Quote();
//riempie i campi
//user.setID(rs.getInt(1));
user.setID(rs.getString(1));

user.setNome(rs.getString(2));
user.setUserID(rs.getString(3));
user.setPassword(rs.getString(4));
DateEntity data=new DateEntity(rs.getDate(5),rs.getDate(6));
user.setDate(data);
quota.setMaxtModels(rs.getInt(7));
quota.setMaxBusinessEntities(rs.getInt(8));
quota.setMaxBusinessServices(rs.getInt(9));
quota.setMaxBindingTemplate(rs.getInt(10));
user.setEmail(rs.getString(11));
user.setLinguaggio(rs.getString(12));
quota.setMaxPublishAssertions(rs.getInt(13));
quota.setMaxSpace(rs.getInt(14));
user.setValidateURL(rs.getString(15));
user.setQuota(quota);
arrayUsers.addElement(user);
}
stmt.close();
```

```
 rs.close();
 //oConn.close();
  }
 catch(SQLException sql_e)
      {adminException.setMessage(sql_e.getMessage());
       throw adminException;}
 return arrayUsers;
}

/**Questo metodo consente di inserire (user)un utente nel db
 * e restituisce 0 se l'operazione non è avvenuta
 * 1 se invece la riga è stata inserita
 * @param -user , utente da inserire.
 */
public int insertUser(User user) throws ADMINException{

ADMINException adminException=new ADMINException();
String commandInsert="INSERT INTO users VALUES (userid_seq.nextval,'"
                                    +user.getNome()+"','"
                                    +user.getUserID()+"','"
                                    +user.getPassword()+"',"
                                    +"sysdate,"
                                    +"sysdate,"
                                    +user.getQuota().getMaxtModels()+","
                                    +user.getQuota().getMaxBusinessEntities()+","
                                    +user.getQuota().getMaxBusinessServices()+","
                                    +user.getQuota().getMaxBindingTemplate()+",'"
                                    +user.getEmail()+"','"
                                    +user.getLinguaggio()+"',"
                                    +user.getQuota().getMaxPublishAssertions()+","
                                    +user.getQuota().getMaxSpace()+",'"
                                    +user.getValidateURL()+"')";
 int nrorow=0; //numero di righe inserite (0 o 1)
 try{
 Statement stmt=oConn.createStatement();
 nrorow=stmt.executeUpdate(commandInsert);
 stmt.close();
 }catch(SQLException sql_e)
      {adminException.setMessage(sql_e.getMessage());
       throw adminException;}
return nrorow;
}

/**Questo metodo consente di cancellare un utente dal db noto il suo
 * identificatore e restituisce 0 se l'operazione non è avvenuta
 * 1 se invece la riga è stata eliminata
```

```
 * @param -ID , chiave dell'utente
 */
 public int deleteUser(String ID)throws ADMINException{
 int nrorow=0;
 ADMINException adminException=new ADMINException();
 String commandDelete="DELETE FROM users WHERE userid="+ID;
 try{
  Statement stmt=oConn.createStatement();
  nrorow=stmt.executeUpdate(commandDelete);
  stmt.close();
  }catch(SQLException sql_e)
       {adminException.setMessage(sql_e.getMessage());
        throw adminException;}
 return nrorow;
 }

  /**Questo metodo consente di aggiornare un utente nel db.
   * Restituisce 0 se nessuna riga è stata aggiornata
   *            1 se la riga è stata aggiornata.
  * @param -user , utente da aggiornare.
  *
  */
 public int updateUser(User user) throws ADMINException{
 ADMINException adminException=new ADMINException();
 String commandUpdate="UPDATE users SET
name="'+user.getNome()+"',"+
                              "login="'+user.getUserID()+"',"+
                              "password="'+user.getPassword()+"',"+

"maxmodels="'+user.getQuota().getMaxtModels()+"',"+

"maxbusinesses="'+user.getQuota().getMaxBusinessEntities()+"',"+

"maxservices="'+user.getQuota().getMaxBusinessServices()+"',"+

"maxbindings="'+user.getQuota().getMaxBindingTemplate()+"',"+
                              "email="'+user.getEmail()+"',"+
                              "la="'+user.getLinguaggio()+"',"+

"maxassertions="'+user.getQuota().getMaxPublishAssertions()+"',"+
                              "maxmsgsize="'+user.getQuota().getMaxSpace()+"',"+
                              "validateurl="'+user.getValidateURL()+"'"+
                              "WHERE userid="+user.getID();
  int nrorow=0;
  try{
  Statement stmt=oConn.createStatement();
```

```
 nrorow=stmt.executeUpdate(commandUpdate);
 stmt.close();
 }catch(SQLException sql_e)
      {adminException.setMessage(sql_e.getMessage());
       throw adminException;}
return nrorow;

 }

}
```

Listato 6.6 – *Gestore della persistenza degli oggetti della classe User.*

Nella figura 6.24 è mostrato il diagramma di classe relativo alla classe User.

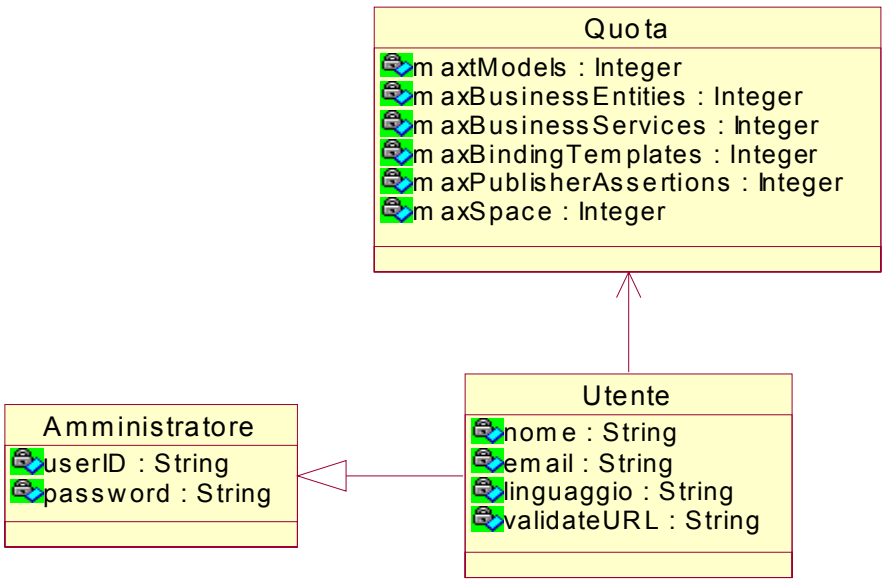


Figura 6.24 – *Diagramma di classe con riferimento alla classe User (Utente).*

L'amministratore del catalogo per poter gestire gli utenti (service provider) deve possedere l'autorizzazione d'accesso al database.

Nella figura 6.25 è mostrato il diagramma di sequenza con riferimento alla registrazione del service provider nel catalogo.

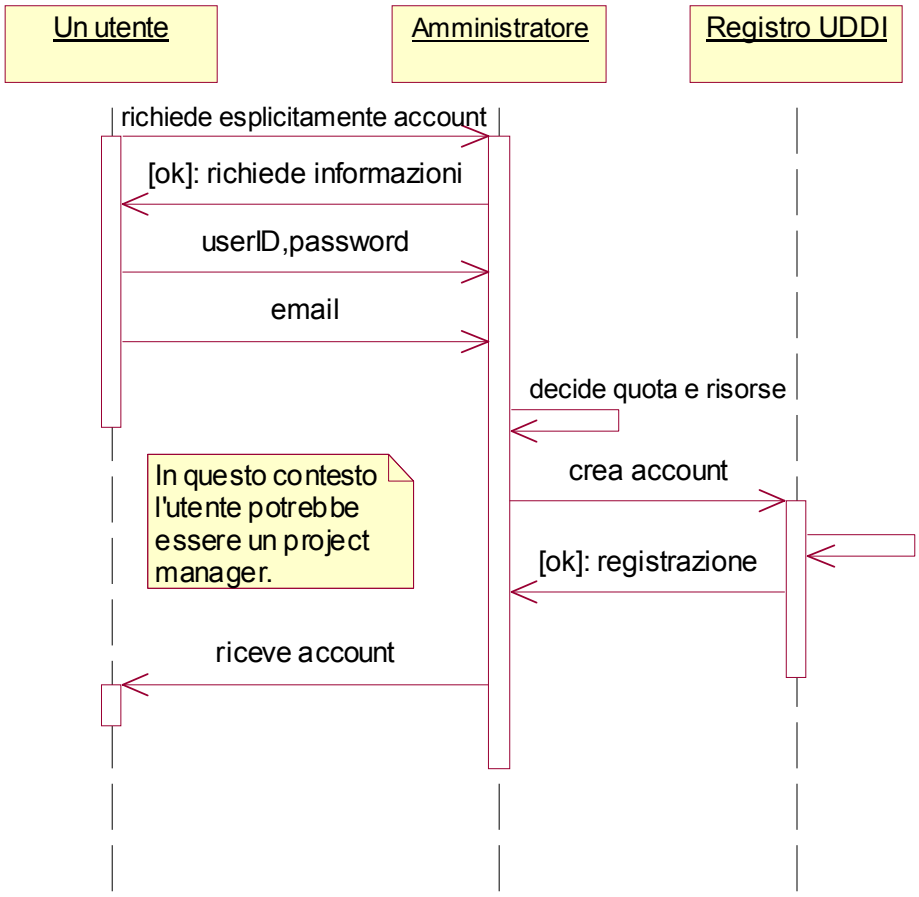


Figura 6.25- *Sequence diagram che illustra la registrazione del service provider. L'account ricevuto dall'amministratore del catalogo gli consente di pubblicare i servizi nel catalogo.*

Capitolo VII

UN SEMPLICE ESEMPIO

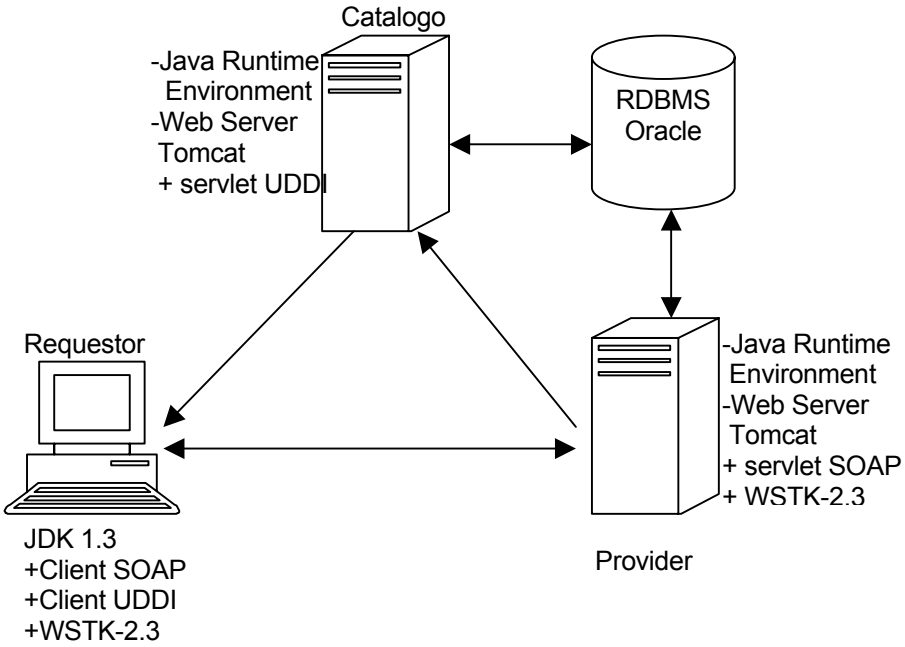
1. Introduzione.

In questo capitolo sarà mostrato un semplice esempio di sviluppo di un Web Service e descritto come configurare l'ambiente d'esecuzione ed usare i vari tool.

Il caso di studio preso in considerazione è molto semplice e riguarda un servizio che fornisce il Codice Avviamento Postale (CAP) dato il nome di un comune italiano. Esso è un componente Java esposto come servizio Web.

Ribadisco che quest'esempio è solo un mezzo per mostrare quale tipologia di strumenti e ambienti sono necessari per realizzare e sviluppare servizi Web e suggerire una serie di passi da seguire per raggiungere quest'obiettivo.

La seguente figura mostra tutti i componenti usati per eseguire questo caso di studio.



2. Configurazione del service provider.

Il service provider, con riferimento a quest'esempio, ospita un ambiente Java per l'esecuzione dei Web Services. In una strategia del riuso questo ruolo potrebbe coincidere con il produttore e manutentore del servzio. Ad esempio, in un'azienda potrebbe essere rivestito da un'unità organizzativa.

L'ambiente Java runtime è costituito sostanzialmente dai seguenti componenti:

- *Web Application Server*. Per quest'esempio, un server HTTP provvisto di tutto il necessario per eseguire servlet Java.
- *SOAP RPC router*. Un modulo software che ha il compito d'instradare le richieste SOAP (dopo essere state deserializzate

da un opportuno modulo) verso un componente provider (anche detto dispatcher). Questo s'incarica di eseguire la chiamata al servizio e di ritornare la risposta che viene serializzata e spedita al destinatario. Convenzionalmente, il modulo router è una servlet eseguita nel contesto del Web Application server.

- *Interfaccia plugable provider*. Fornisce estensibilità al server SOAP, consentendo di aggiungere provider personalizzati, ovvero moduli software che incapsulano i dettagli relativi alla particolare implementazione del servizio Web target [46]. Dunque, un plugable provider fornisce il necessario per localizzare, caricare ed invocare l'implementazione del servizio con riferimento alla richiesta SOAP. Se il messaggio è basato su RPC, il provider passa il risultato al modulo che si occupa di serializzarlo e quindi di trasformarlo in un risposta SOAP.

- Un'applicazione che consente di generare la descrizione di base del servizio (vedi par. 2.3.1 per la descrizione di un servizio).

- Un'applicazione che consente di pubblicare la descrizione completa del servizio (vedi par. 2.3.2) nel catalogo dei Web Services.

Nella figura 7.1 sono mostrati i componenti attivati per portare a termine una chiamata al servizio target. Questi fanno parte di un caso semplice dell'architettura AXIS di Apache (successore di SOAP Apache) [47].

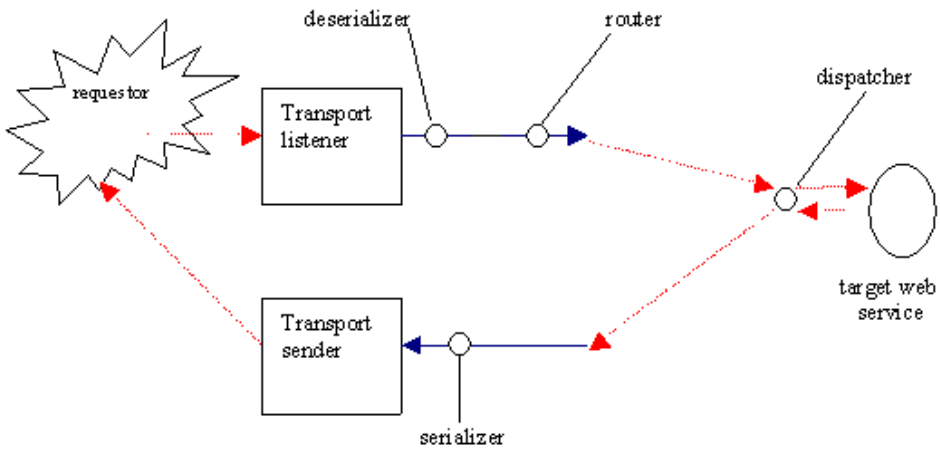


Figura 7.1 – Una request/responce RPC.

Il trasport sender è un componente che riceve i messaggi in uno specifico formato di protocollo a livello di trasporto (esempio, HTTP, SMTP, FTP, etc.) e li converte in un InputSource XML, ovvero un oggetto Java che incapsula tutti i dettagli relativi ad un documento XML (API Java [44]).

Il deserializer è un componente che converse l'InputSource XML (applicando uno schema di decodifica) in oggetti nel linguaggio di programmazione specifico (in Java in questo caso). Ad esempio, converte il body di un messaggio SOAP in un oggetto che contiene informazioni riguardanti il nome del metodo invocato, i valori degli argomenti e il loro tipo.

Il modulo router entra con la chiave (identificatoreServizio, nomeMetodoInvocato) in una tabella ed estrae il provider o dispatcher che si occupa d'invocare il Web Service target.

Il componente serializzatore trasforma il risultato fornito dal servizio in un InputStream XML. Ad esempio, se il risultato è un vettore,

questo sarà trasformato in un elemento XML, secondo lo schema di codifica SOAP.

Infine, il transport sender invia l'InputSorce (l'envelope SOAP) al destinatario usando uno specifico protocollo di trasporto.

In riferimento a questo specifico esempio, il service provider che ospita il servizio Web è costituito dal sistema operativo Windows 2000, dal Web server Apache Tomcat 3.21 [41] con l'estensione per eseguire le servlet Java [22]. Questo server sarà utilizzato per eseguire sia l'engine SOAP Apache 2.1 [42] che consente di attivare i servizi sia il registro UDDI [43] che invece permette di pubblicarli e scoprirli. Queste due applicazioni sono entrambe servlet Java.

Il Web server Tomcat è stato istallato sul sistema operativo Windows 2000, tuttavia, essendo sviluppato completamente in Java, dovrebbe essere codice portabile anche su altre piattaforme software. Il discorso resta valido anche per SOAP Apache e il registro UDDI.

2.1. Configurazione del Web server.

Tomcat richiede un Java Runtime Environment conforme a JRE 1.1 o successiva, includendo qualsiasi piattaforma Java 2. Nel caso specifico è stata utilizzato Java 2 SDK v1.3.1 [45], che include JRE 1.3.

Prima di far partire il server bisogna creare la variabile d'ambiente TOMCAT_HOME e settarla in modo che punta alla sottodirectory in cui è stato istallato tomcat. Successivamente, si devono usare i due file startup.bat e shutdown.bat per far partire e stoppare, rispettivamente, il server.

Se la procedura d'istallazione è andata a buon fine digitando l'indirizzo *http://localhost:8080/* nel browser Web dovrebbe comparire la home page di tomcat, mostrata in figura 7.2.

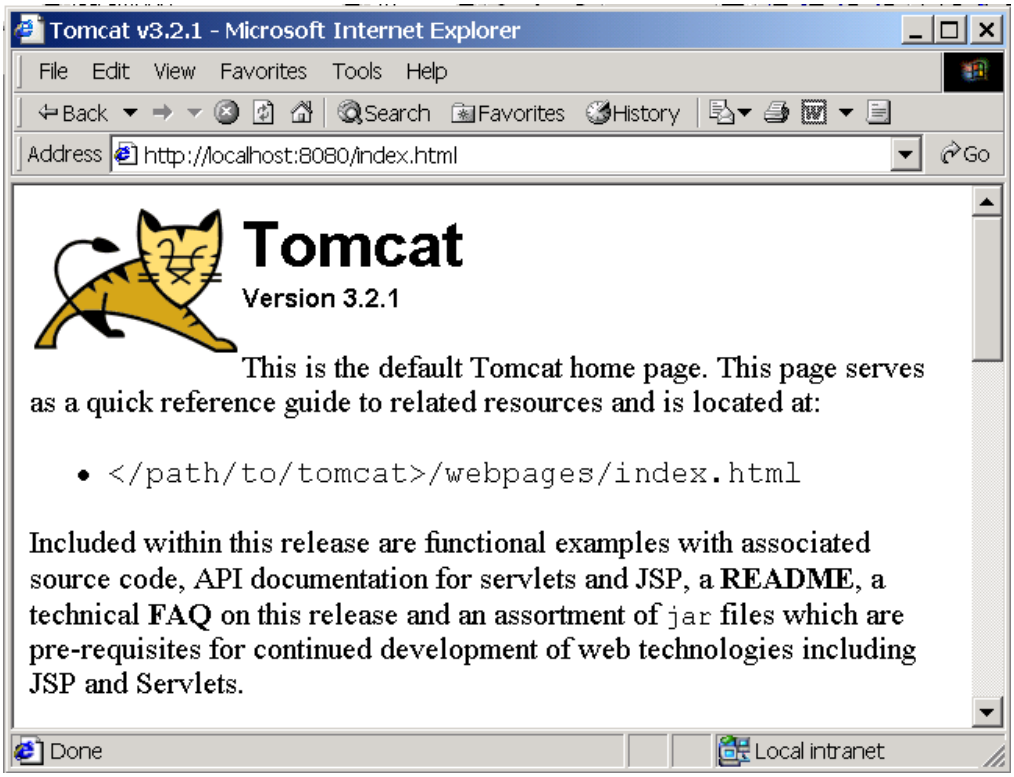
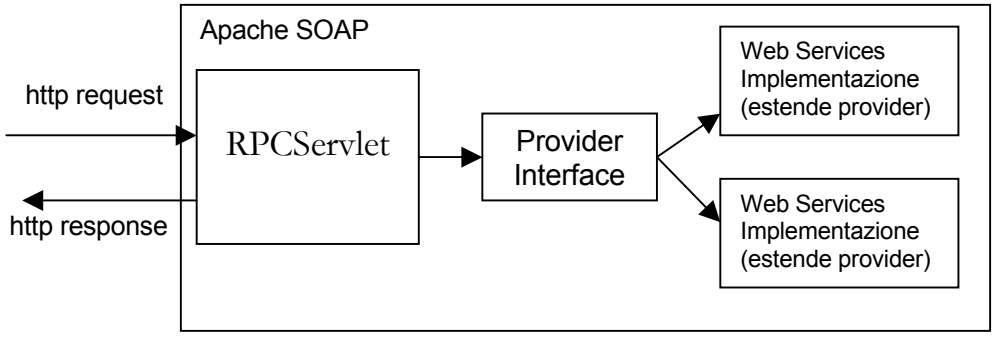


Figura 7.2 – Home Page di tomcat.

2.2. Configurazione del server SOAP Apache.

La libreria SOAP Apache implementa un SOAP Web Services framework (solo la parte relativa all'attivazione del servizio remoto) usando la tecnologia Java. Come detto nel paragrafo precedente, il bridge tra il server SOAP e il servizio è chiamamo provider. Nella seguente figura è riportato il diagramma del flusso di messaggio attraverso SOAP Apache e sono mostrati i puggable provider:



Il pacchetto SOAP Apache (essenzialmente la libreria soap.jar) richiede altre librerie Java per funzionare: Xerces-J, JavaBeans Activation Framework [45] (di cui occorre solo la libreria activation.jar) e JavaMail [46] (occorre solo mail.jar). Le ultime due librerie sono necessarie per eseguire i meccanismi per la codifica MIME e per allegare file ai messaggi SOAP e vanno comunque istallate, anche se non si creano servizi che utilizzano le loro funzionalità.

La servlet SOAP Apache utilizza il parser XML DOM level 2 fornito da Xerces-J di Apache, mentre tomcat usa il parser xml.jar che ha implementa un'interfaccia DOM level 1. Ora, dal momento che tomcat 3.2 carica prima le sue librerie e successivamente quelle contenute nella variabile d'ambiente CLASSPATH (usata dalla JRE per localizzare le classi) si hanno problemi di conflitto tra la servlet e il server (si fa riferimento sempre alla libreria caricata per prima). Per evitare quest'inconveniente, bisogna fare in modo che tomcat carichi prima la libreria Xerces-J.jar e poi xml.jar.

Le altre librerie soap.jar, activation.jar e mail.jar vanno messe nel CLASSPATH del sistema o in alternativa nel file tomacat.bat che viene eseguito da startup.bat per l'esecuzione di tomcat. Nella stessa variabile

d'ambiente va incluso il percorso che consente di raggiungere la radice della directory in cui vengono messi i servizi Web.

Infine, per fare in modo che la servlet venga eseguita dal server tomcat, bisogna modificare il file di configurazione server.xml di quest'ultimo.

Nel listato 7.1 vengono mostrate le sezioni dei due file(rispettivamente tomcat.bat e server.xml) modificate con riferimento a quest'esempio.

Tomcat.bat

```
…
rem ----- Set Up The Runtime Classpath ------------------------------------

SET JAVA_HOME=c:\jdk1.3.1
SET PROTOTIPO_HOME=c:\tesi\prototipo
SET SOAP_HOME=%PROTOTIPO_HOME%\provider\soap

:setClasspath
set CP=%TOMCAT_HOME%\classes
set CP=%SOAP_HOME%\lib\xerces.jar;%CLASSPATH%;%CP%
set CLASSPATH=%SOAP_HOME%\lib\mail.jar;
%SOAP_HOME%\lib\activation.jar;%SOAP_HOME%\lib\soap.jar
;%SOAP_HOME%;%PROTOTIPO_HOME%\provider\webServices
set CLASSPATH=%CLASSPATH%;%PROTOTIPO_HOME%\registry\server-
uddi\webapps\uddi\web-inf\lib; %PROTOTIPO_HOME%\registry\server-uddi\oracle-
jdbc\classes111.zip
…
```

server.xml

```
…
<Context reloadable="false" path="/soap"
docBase="c:\tesi\prototipo\provider\soap\webapps/soap" debug="0"/>
…
```

Listato 7.1 – Modifica da apportare al file tomcat.bat e server.xml.

Affinché tomcat possa eseguire la servlet SOAP è necessario aggiungere un contesto al file server.ml (mostrato nel listato 7.1). La

sezione <Context> mappa un percorso dentro il server tomcat in cui l'applicazione risiede. Nella tabella 7.1 sono riportate brevi descrizioni degli attributi dell'elemento xml <Context>.

Attributo	Descrizione
Path	Il prefisso di una richiesta http che identifica il contesto che tomcat deve usare.
DocBase	Punta alla directory base per l'applicazione Web.
Reloadable	Impostato a *true* impone al server di ricaricare la servlet ogni volta che è modificata.
debug	Se '0' sopprime tutto l'output sulla console, se '9' imposta la modalità *verbose*.

Tabella 7.1 – Attributi dell'elemento Context

Se la configurazione dell'engine SOAP è stata eseguita correttamente, sul browser, all'indirizzo *http://localhost:8080/soap/admin/index.html* , dovrebbe comparire l'home page mostrata in figura 7.3.

Figura 7.3 *Home page di SOAP Administrator.*

Per la produzione della descrizione di base del servizio (par. 2.3.1) occorre il toolkit IBM WSTK 2.23 [47] in cui è inclusa (oltre al SOAP Apache) un'applicazione *wsdlgen* che consente di generare automaticamente le definizioni del servizio da alcuni componenti (Java Bean, EJB [48] e COM).

3. Configurazione del service registry o catalogo dei servizi Web.

Come repository dei servizi Web è stato utilizzato un registro UDDI. Questo è realizzato mediante una servlet Java che esegue chiamate

(attraverso il protocollo JDBC) a stored procedure di un RDBMS Oracle 8.1 (database aziendale) per accedere ai dati rappresentativi delle strutture dati UDDI presentate nel capitolo 5.

Brevemente, la servlet nella sezione init (alla prima richiesta utente) carica il file di configurazione e in seguito effettua un pool di collegamenti al database aziendale, Oracle 8.1 appunto. Dopo l'inizializzazione si pone in ascolto di eventuali richieste del tipo HTTP POST. Quando arriva un messaggio SOAP (codifica in XML delle strutture dati e API presentati nel capitolo 5), questo viene deserializzato (parser) e processato mediante chiamata a stored procedure (un modulo, Uddi request, in base al tipo di richiesta crea l'oggetto orasp addetto all'attivazione della stored procedure che serve quella particolare richiesta). Il risultato viene restituito al serializzatore e da questo alla servlet che lo invia al destinatario. Nella figura 7.8 è mostrato uno schema di massima dei moduli attivati durante l'interazione col server UDDI.

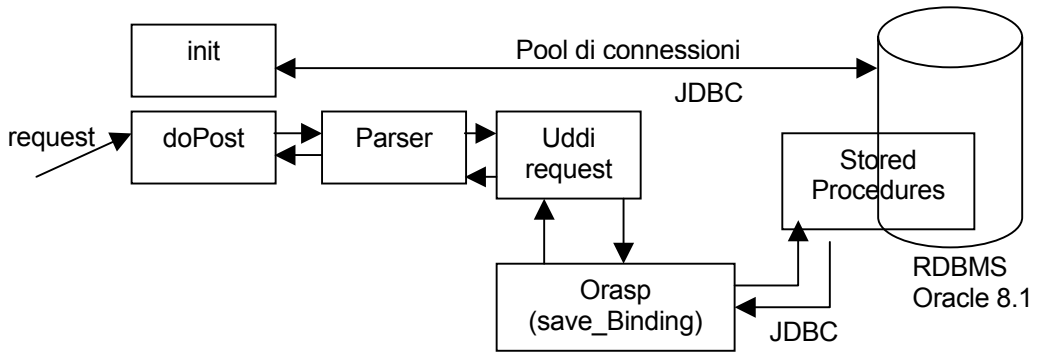


Figura 7.8 – Interazione con il server UDDI.

I moduli che implementano il server UDDI [43] sono free software, e possono essere modificati e/o distribuiti sotto i termini della licenza GNU Lesser General Public License, secondo quanto riportato dall'autore. Personalmente, ho modificato il modulo relativo al parser per adattarlo alla nuova versione del parser SAX 2.0 (implementazione fornita da Xerces-J). Il motivo della modifica è legato al conflitto tra la versione del parser usato dal Web server Tomcat 3.2 e quello UDDI.

Per la configurazione della servlet sotto il Web server bisogna aggiungere il contesto:

<Context reloadable="false" path="/uddi"
docBase="c:\tesi\prototipo\registry\server-uddi\webapps/uddi" debug="0"/>

al file server.xml

4. Configurazione del service requestor.

In questo esempio il consumatore di servizi Web usa un ambiente d'esecuzione Java ed accede al servizio mediante il protocollo SOAP (libreria di SOAP Apache lato client). Per cercare i servizi disponibili utilizza un prototipo di client UDDI provvisto d'interfaccia grafica Quest'applicazione si basa essenzialmente sulle librerie dal lato client del pacchetto pUDDIng 2.0 [43] (che implementano le strutture dati e le API UDDI 2.0 presentate nel capitolo 5), sulle quali sono state realizzate le interfacce grafiche (GUI Java Swing [49]) per fornire all'utente un ambiente amichevole.

5. Realizzazione del Web Service.

IL metodo di sviluppo seguito per realizzare il servizio Web è il green field (vedi par. 2.4.1). Di seguito riporto le fasi di cui è costituito:

1. Realizzazione dell'applicazione Java che implementa il servizio.
2. Definizione dell'interfaccia del servizio e successiva pubblicazione.
3. Deploy del servizio all'engine SOAP.
4. Generazione e pubblicazione della definizione dell'implementazione.
5. Esecuzione. Il servizio Web può essere eseguito nel contesto del Web application server (tomcat) oppure in un costesto diverso. Ad esempio, nel caso d'implementazione mediante EJB, questi vengono eseguiti un ambiente runtime separato (EJB container) dal contesto del Web application server in cui viene eseguito SOAP [2]

5.1. Il servizio Cap_Service.

Il servizio realizzato per quest'esempio fornisce il C.A.P. del comune a partire dal suo nome. In figura 7.11 è mostrato il diagramma di classe che descrive l'applicazione che implementa il servizio. Invece nei listati 7.2, 7.3 e 7.4 sono mostrate le relative classi Java.

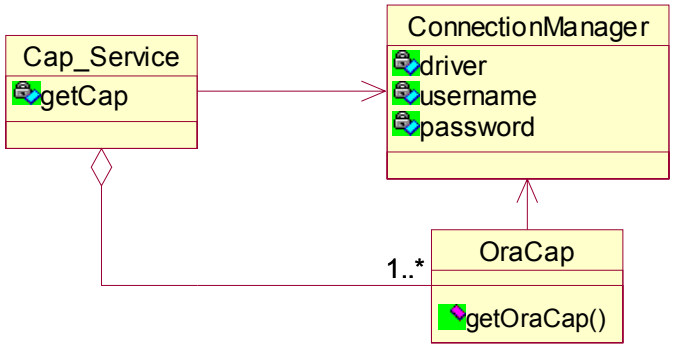


Figura 7.11 – *Diagramma di classe (notazione UML) dell'applicazione che realizza il servizio.*

```
/** Questa classe implementa il servizio Cap che fornisce il cap dato il<BR>
 *  nome di un comune d'italia, che deve essere scritto per intero.
 *
 *  @autor Antonio Valentino
 *
 */
public class Cap_Service {

  private OraCap oraCap=null;
  private boolean isSuccess=false;
  public Cap_Service() {
  //Inizializzazione. In SOAP APACHE il costruttore vuoto della classe viene
invocato
  //una sola volta, quando la servlet viene eseguita per la prima volta.
  //Connessione al database
   isSuccess=ConnectionManager.init("oracle.jdbc.driver.OracleDriver",
   "jdbc:oracle:thin:@jack:1521:sample","antonio","A90Pa91A",1);
   if (!isSuccess){
   System.out.println("Errore durante la connessione al database");
  }
  }

  /**Restituisce il cap dato il nome del comune.
   * @param comune - nome del comune
   * @return il cap.
   */
  public String getCap(String comune){

  if (!isSuccess) return null;
```

```
  oraCap=new OraCap();
  return oraCap.getOraCap(comune);
  }
}
```

Listato 7.2 – Classe che implementa il servizio.

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Vector;

/**
* Gestore del pool di connessioni, evita di aprire e chiudere ogni volta una
connessione al database
* per effettuare qualsiasi operazione.<BR>
*
*
* @author Antonio Valentino
* @version 1.0
**/
public abstract class ConnectionManager extends Object {
  /**
  * Stringa di connessione al db
  **/
  private static String connectionString;
  /**
  * username
  **/
  private static String username;
  /**
  * password
  **/
  private static String password;


  /**
  * vettore delle connessioni disponibili
  **/
  private static Vector listConns = new Vector();
```

```
 /* static {
    try {
       DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
    } catch (SQLException sqle){
       sqle.printStackTrace();
    }
 }*/

  /**Metodo d'inizializzazione delle connessioni
   * @param driver - driver JDBC.
   * @param url - Stringa di connessione
   * @param user - username relativo all'account per l'accesso al db
   * @param password - password utente
   * @param nConn - numero si connessioni iniziali.
   */
  public static boolean init(String driver, String url, String user, String _password,
int nConn){
    try {
      DriverManager.registerDriver ((Driver)Class.forName(driver).newInstance());
//carica driver
    } catch (Exception e){//gestisce eccezione
      System.err.println("Connection Manager Error:"+e.getMessage());
      e.printStackTrace(System.err);
      return false;
    }

    connectionString=url;
    username=user;
    password=_password;
    //Inizializza il vettore delle connessioni
    for (int a=0;a<nConn;a++){
      try {
        openConnection();
      } catch (SQLException sql_e){
        System.err.println("Errore durante l'inizializzazione di ConnectionManager:
"+sql_e.getMessage());
        sql_e.printStackTrace(System.err);
        }
    }
    if (listConns.size()==0){//non ci sono connessioni disponibili.
      return false;
    }
      return true;

  }
```

```
  /**
   * Apre una connessione al database e la inserisce nel pool
   *
   * @exception SQLException - non è possibile aprire la connessione
   **/
  private static void openConnection() throws SQLException{
      listConns.addElement(getNewConnection()); //inserisce nel vettore una
nuova connessione
    }

  /**
   * Apre una nuova connessione al database.
   * @exception SQLException - non è possibile aprire la connessione
   **/
  private static Connection getNewConnection() throws SQLException {
    Connection oConn
=DriverManager.getConnection(connectionString,username,password);
    return oConn;
  }

  /**
   * Ritorna una connessione dal pool, e controlla che sia ancora valida prima di
usarla.<BR>
   * Se non sono disponibili connessioni ne apre una nuova.
   * @return una Connection
   * @exception SQLException - la connessione non può essere aperta
   **/
  public static Connection getConnection() throws SQLException {
    Connection oConn=null;
    if (listConns.size()>0){//allora estrae connessione già creata.
        oConn=(Connection)listConns.remove(listConns.size()-1);

     //controlla la connessione, se non è valida allora ne crea una nuova
     //(il n.ro di connessioni deve sempre rimanere constante)
     try {
       Statement s=oConn.createStatement();//prova
       s.close();
     } catch (SQLException sqle) {
       //connessione non valida --> crea nuova connessione
       oConn=getNewConnection();
     }
    } else {
     oConn=getNewConnection(); //connessione non disponibile --> crea nuova
connessione
    }
    return oConn;
```

```
 }

 /**
 *  Rimette la connessione nel pool.
 *  @param oConn - connessione da rimettere nel pool
 **/
 public static void releaseConnection(Connection oConn){
  if (oConn!=null){
      listConns.addElement(oConn);
    }
 }

}
```

Listato 7.3 – *Classe che implementa ConnectionManager che consente di creare e gestire un pool di connessioni al database.*

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;
import java.sql.Types;

 /**Questa classe consulta il database per estrarre il CAP di un dato comune.
  *
  */
 public class OraCap {

  private Connection oConn=null;

  public OraCap() {
  }

  /**Restituisce il cap di un comune italiano.
   * @param nomeComune - nome del comune di cui si vuole conoscere il cap.
   */
  public String getOraCap(String nomeComune){
  String query="SELECT cap FROM ComunItalia WHERE
nome='"+nomeComune+"'";
  String cap=null;

  try {
     oConn=ConnectionManager.getConnection(); //acquisisce una connessione
     Statement stmt=oConn.createStatement();  //crea un comando
     ResultSet rs= stmt.executeQuery(query);  //interroga
```

```
    while (rs.next()){//estrae il risultato
        cap=rs.getString(1);
        System.out.println(cap);
    }
    stmt.close();
    rs.close();

  }catch (SQLException sql_e){
   System.err.println("Errore durante il collegamento al database:
"+sql_e.getMessage());
   sql_e.printStackTrace(System.err);
  }

  if (oConn!=null){
    ConnectionManager.releaseConnection(oConn); //rimette la connessione nel
pool
    }

   return cap;

}
}
```

Listato 7.4 – *La classe OraCap che consulta il database per ottenere il CAP del comune.*

Quando il servizio viene attivato la sezione d'inizializzazione (racchiusa nel costruttore vuota della classe) crea un pool di connessioni al database che possono essere utizzate dagli oggetti OraCap per interrogare la base dati (il servizio è stato pensato come un'applicazione Web).

Il modulo OraCap acquisisce una connessione dal pool e la usa per interrogare il database dopodiché la rimette nel vettore che gestisce il pool di connessioni.

5.2. Definizione dell'interfaccia e pubblicazione nel catalogo.

La definizione dell'interfaccia può essere effettuata a mano a partire dall'intefaccia dell'implementazione del servizio Cap_Service, oppure utilizzando l'utility *wsdlgen* del pacchetto WSTK di IBM, descritta nel capitolo precedente. Per ovvi motivi ho scelto la seconda strada.

Il documento WSDL generato dal programma è mostrato dal listato 7.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Cap_Service"
 targetNamespace="http://www.cap_service.com/Cap_Service-interface"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://www.cap_service.com/Cap_Service"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<message
    name="IngetCapRequest">
 <part name="meth1_inType1"
  type="xsd:string"/>
</message>

<message
    name="OutgetCapResponse">
 <part name="meth1_outType"
  type="xsd:string"/>
</message>


<portType
    name="Cap_Service ">
 <operation name="getCap">
  <input
     message="IngetCapRequest"/>
  <output
  message="OutgetCapResponse"/>
 </operation>
</portType>
```

```
<binding name="Cap_ServiceBinding"
    type="Cap_Service">
 <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation
     name="getCap">
  <soap:operation
      soapAction="urn:cap_service"/>
  <input>
   <soap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:cap_service"
      use="encoded"/>
  </input>
  <output>
   <soap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:cap_service" use="encoded"/>
  </output>
 </operation>
</binding>

</definitions>
```

Listato 7.5 – *Documento WSDL della definizione dell'interfaccia del servizio*

La pubblicazione della definizione dell'interfaccia nel catalogo consiste nella creazione di un modello con il nome pari all'attributo *targetNamespace* dell'elemento *definitions*, con un riferimento (mediante url) al documento WSDL che rappresenta la definizione dell'interfaccia e una descrizione. Inoltre, questo modello viene classificato come appartenente al tipo wsdl-spec (specifica WSDL) di uddi-org:types, ovvero viene etichettato come un documento WSDL (vedi capitolo 5 per maggiori chiarimenti).

Di seguito viene mostrata la sequenza di istruzioni che consentono di creare il modello della definizione dell'interfaccia Cap_Service-Interface.wsdl.

```
//questo url consente di localizzare la definizione dell'interfaccia.
URL url=new URL("http://jack:8080/interfacce/");

//prepara parser
XMLReader parser=new WSDLParser();
WSDLInterface interf=new WSDLInterface();
String filename="file:c:/WSTK-2.3/BIN/Calc_Service-Interface.wsdl";
parser.setContentHandler(interf); //setta il gestore della definizione
parser.parse(filename);//esegue il parsing

//crea oggetto per creare la struttura Modello
WSDL2UDDI wsdl2Uddi=new WSDL2UDDI();

//prepara autorizzazione e proxy per le operazioni sul registro UDDI
//crea Autorizzazione
Auth auth=new Auth("a.Valentino","susy");

//crea proxy (invia messaggi soap al server UDDI secondo la specifica 2.0, vedi
capitolo 5 e [43]
UDDIProxy proxy=new UDDIProxy();

//setta endpoint del service registry
String endPointUDDI="http://localhost:8080/uddi/servlet/uddi";
proxy.setPublishingEndpoint(new URL(endPointUDDI)); //per la pubblicazione
proxy.setInquiryEndpoint(new URL(endPointUDDI)); //per la consultazione

//Crea e prepara modello.
Modello model=null;
wsdl2Uddi.setBindingName(interf.getBindingName());//setta nome binding
wsdl2Uddi.setTargetNamespace(interf.getTargetNamespace());
wsdl2Uddi.setURLWSDLInterface(url); //setta url della definizione
model=wsdl2Uddi.getModel(); //restituisce modello parzialmente preparato
//setta autorizzazione proxy per pubblicare il modello
model.setAuth(auth);
model.setProxy(proxy);
//Aggancia una descrizione al modello.
Description desc=new Description();
desc.setText("Modello per la definizione dell'interfaccia Cap_Service");
desc.setLanguage("it");
model.addDescription(desc);
```

```
//classifica interfaccia come wsdl-spec.
KeyedReference category=new KeyedReference();
//setta coppia nome valore
category.setName("uddi-org:types");
category.setValue("wsdl-spec");
//setta chiave modello di riferimento.
category.setTModelKey("UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4");
model.addCategory(category);
model.save();//pubblica modello.
```

Nella figura 7.13 è mostrato un diagramma concettuale che evidenzia il legame della classe Modello con il pacchetto UDDI parte client [43].

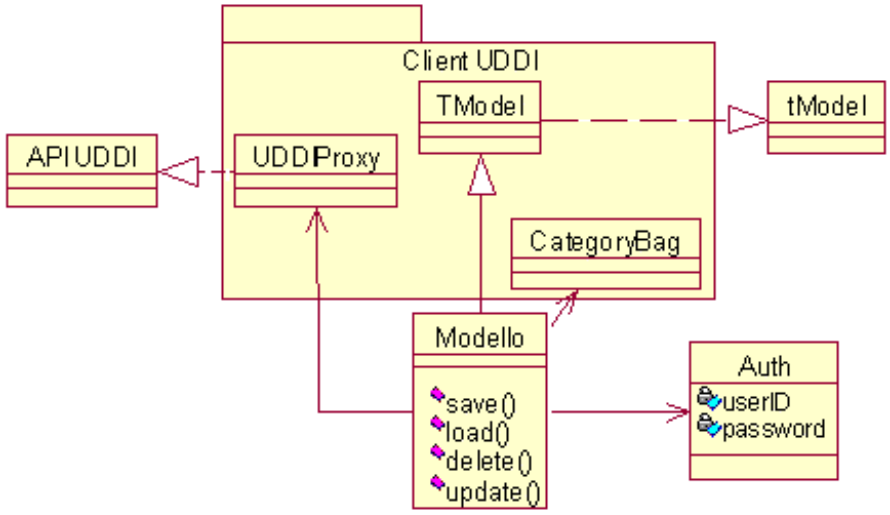


Figura 7.13 – *Diagramma concettuale con riferimento alla classe Modello.*

Nel listato 7.6 è mostrata l'implementazione della classe Modello.

```
/**Estende la classe TModel implementata dal client UDDI [43]
 *
 */
public class Modello extends org.frenchstudio.uddi.datatypes.tmodels.TModel{

  private UDDIProxy proxy=null; //proxy client
  private Auth auth=null; //autorizzazione

  public Modello() {

  }
```

```
  //setta il proxy per effettuare le operazioni sul registro.
  public void setProxy(UDDIProxy proxy){
  this.proxy=proxy;
  }

  //setta l'autorizzazione per la pubblicazione
  public void setAuth(Auth auth){
  this.auth=auth;
  }

  //pubblica il modello.
  public void save() throws UDDIException, SOAPException{
  //prende un gettone per la pubblicazione (vedi capitolo 5)
  AuthToken
authToken=proxy.get_authToken(auth.getUserID(),auth.getPassword());
    if (authToken!=null){
         //pubblica
         TModelDetail detail=proxy.save_tModel(authToken,new TModel[]{this});
         //elimina gettone
         proxy.discard_authToken(authToken);
         //preleca identificatore assegnato.
         FastPoolVector models=detail.listTModels();
         //setta la chiave del modello.
         this.setTModelKey(((TModel)models.elementAt(0)).getTModelKey());
         }

  }

  /** Aggiorna modello
   *
   */
  public void update()throws UDDIException, SOAPException{
  AuthToken
authToken=proxy.get_authToken(auth.getUserID(),auth.getPassword());
    if (authToken!=null){
       TModelDetail detail=proxy.save_tModel(authToken,new TModel[]{this});
       proxy.discard_authToken(authToken);
       UDDIPool.releaseObject(authToken,UDDIPool.AUTHTOKEN);
  }
  }

  /**Carica modello data un identificatore
   *
   */
  public void load() throws UDDIException,SOAPException{
```

```
    TModelDetail detail=proxy.get_tModelDetail(this.getTModelKey());
    FastPoolVector listModel=detail.listTModels();
    TModel tModel= (TModel)listModel.elementAt(0);
    this.setName(tModel.getName());
    this.setOverviewDoc(tModel.getOverviewDoc());

  }

  /**Cancella modello dal registro.
   *
   */
  public boolean delete() throws UDDIException, SOAPException{
    AuthToken
authToken=proxy.get_authToken(auth.getUserID(),auth.getPassword());
    DispositionReport rapporto=null;
    boolean esito=false;
    if (authToken!=null){
     rapporto=proxy.delete_tModel(authToken,this.getTModelKey());
     proxy.discard_authToken(authToken);
     UDDIPool.releaseObject(authToken,UDDIPool.AUTHTOKEN);

    }
    esito=rapporto.isSuccess();
    return esito;
  }

}
```

Listato 7.6 – *Classe Modello.*

5.3. Esposizione del servizio Web al server SOAP.

Per il deploy del servizio all'ambiente di runtime SOAP si possono seguire due strade. La prima porta all'home page dell'administrator in cui bisogna settare alcuni campi, come mostra la figura 7.8.

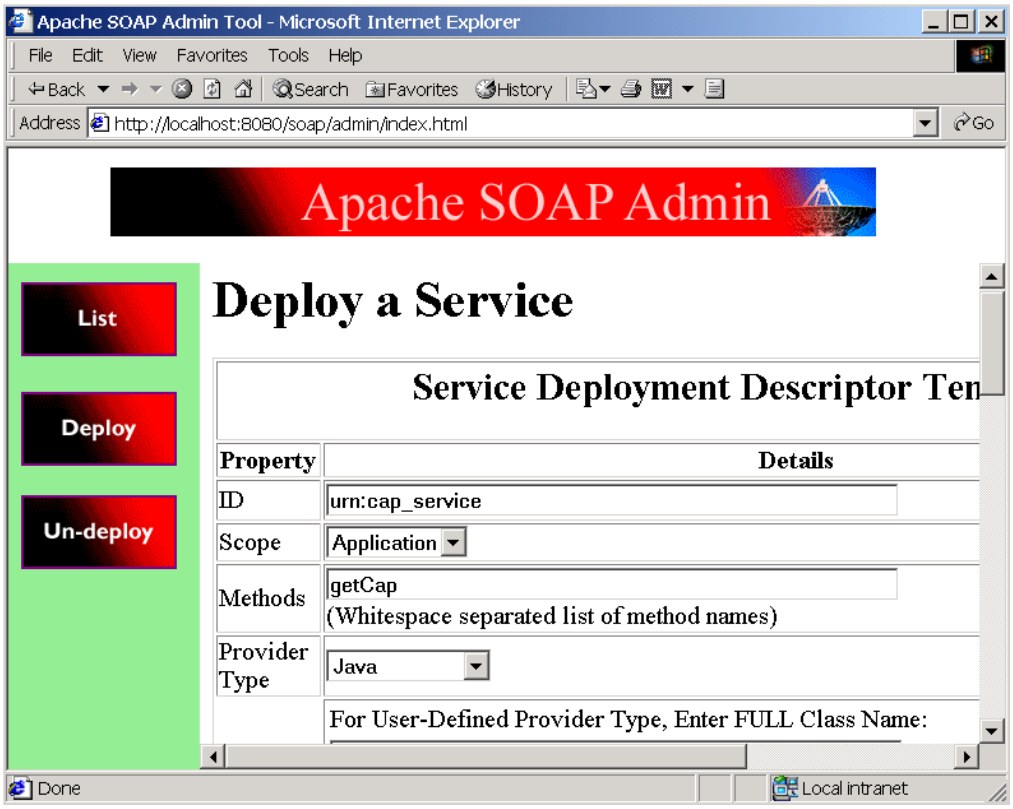


Figura 7.14 – *SOAP Administrator. Consente il deploy dei servizi Web impostando gli opportuni campi.*

La seconda strada invece semplifica ulteriormente la vita. Infatti, basta utilizzare il deployment descriptor generato dall'utility *wsdlgen* e la classe *ServiceManagerClient* fornita da Apache, nel seguente modo:

```
java org.apache.soap.server.ServiceManagerClient  \
 java org.apache.soap.server.ServiceManagerClient  \
   http://localhost:8080/soap/servlet/rpcrouter deploy
DeploymentDescriptor.xml
```

per notificare il servizio all' Apache-SOAP listener. All'indirizzo *http://localhost:8080/soap/servlet/rpcrouter* risponde la servlet SOAP che provvede ad attivare il giusto provider per negoziare con il servizio. Il provider (vedi par. 7.2) viene prelevato dalla tabella dei descrittori dei servizi mediante la coppia (urn:cap_service, getCap). Nel caso

dell'esempio, viene usato il privider di default (*RPCJavaProvider* che carica le classi Java, invoca il metodo desiderato e converte il risultato in un messaggio SOAP). L'attributo *scope* indica il lifetime dell'istanza della classe che è stata caricata. "Request" indica che l'oggetto sarà rimosso dopo che la richiesta è stata completata, "Session" indica che l'oggetto durerà per il corrente lifetime della sessione http (questo può essere sfruttato per il mantenimento dello stato), e "Application" indica che l'oggetto durerà finché la servlet che serve la richiesta non sarà terminata. Il listato 7.7 rappresenta il deployment descriptor del servizio Cap_Service.

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
id="urn:cap_service" >
  <isd:provider type="java" scope="Application" methods="getCap">
    <isd:java class="webservices.eco.Cap_Service" static="false"/>
  </isd:provider>
</isd:service>
```

Listato 7.7 – *Deployment descriptor del servizio Cap_Service.*

Per visualizzare la lista dei servizi disponibili si può usare l'amministratore SOAP, come visualizzato nella figura 7.15.

Figura 7.15 – Lista dei servizi esposti al server SOAP.

5.4. Pubblicazione del servizio e della definizione dell'implementazione.

In questo paragrafo sarà mostrato come creare una classificazione dei servizi, come pubblicare l'unità organizzativa aziendale che fornisce il servizio e infine come pubblicare il servizio con le specifiche di binding annesse.

Il listato 7.8 rappresenta la definizione dell'implementazione generata dall'utility *wsdlgen*.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Cap_Service "
 targetNamespace="http://www.cap_serviceservice.com/Cap_Service"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://www.cap_serviceservice.com/Cap_Service"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<import
   location="http://localhost:8080/wsdl/Cap_Service-interface.wsdl"
   namespace="http://www.cap_serviceservice.com/Cap_Service-interface">
</import>

<service name="Cap_Service ">
 <documentation>IBM WSTK 2.0 generated service definition file
 </documentation>
 <port binding="Cap_ServiceBinding" name="Cap_ServicePort">
  <soap:address location="http://localhost:8080/soap/servlet/rpcrouter"/>
 </port>
</service>
</definitions>
```

Listato 7.8 – *Definizione dell'implementazione del servizio.*

Da questo documento si possono estrarre informazioni utili per la pubblicazione del servizio (il nome) e dei binding template (l'endpoint o punto d'accesso, rappresentato dall'attributo location dell'elemento soap:address) (vedi par. 5.3.2).

Tuttavia, prima di pubblicare il servizio bisogna creare una classificazione e pubblicare l'unità organizzativa che fornisce il servizio. Per creare la classificazione mostrata nella figura 7.16 (è solo una semplice classificazione a scopo esplicativo) si possono utilizzare le strutture dati tModel e CategoryBag della specifica UDDI.

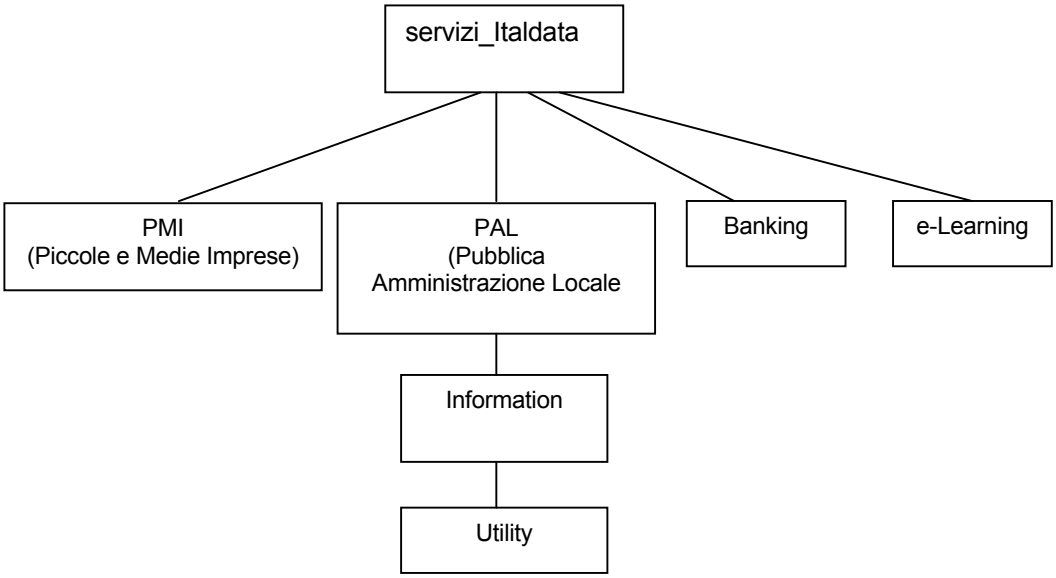


Figura 7.16 – *Struttura gerarchica rappresentativa della classificazione usata per quest'esempio.*

Il servizio verrà pubblicato nella categoria utility. Per creare la struttura di figura 7.16 si può utilizzare la struttura Modello definita in precedenza. Il listato 7.9 mostra la sequenza di istruzioni che consente di realizzare la suddetta classificazione. Mentre la figura 7.17 mostra la sequenza di operazioni che consentono di creare una tassonomia nel registro UDDI (vedi capitolo 5) e di aggiungere il modello dei servizi PAL alla radice servizi_Italdata.

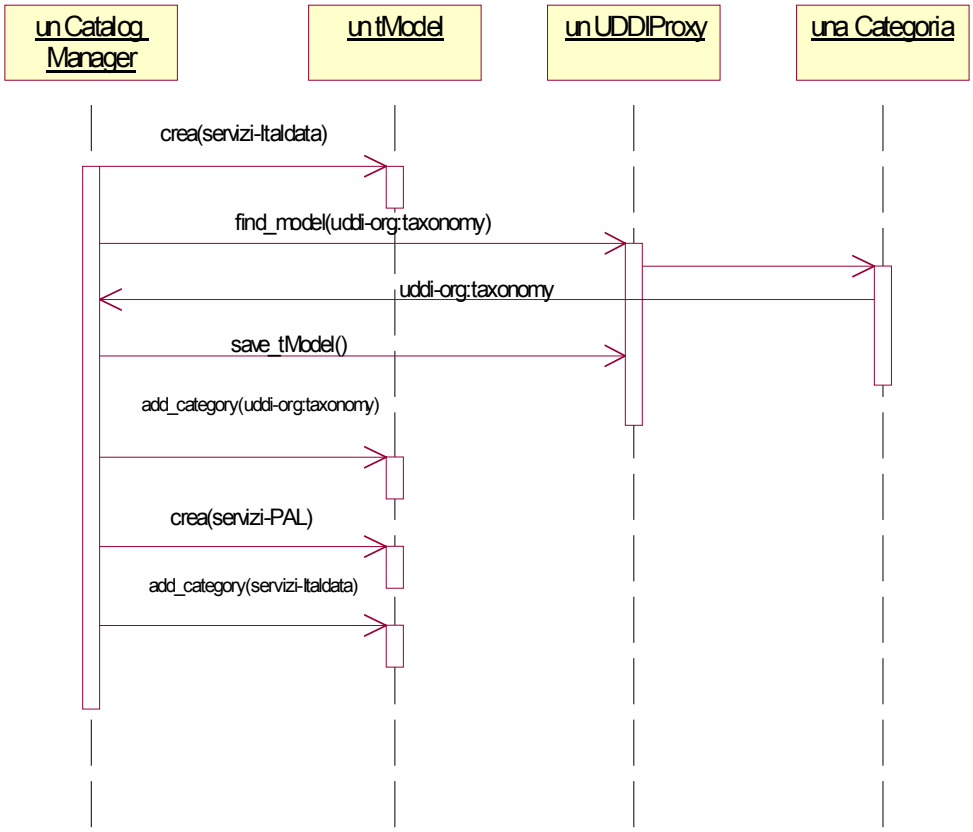


Figura 7.17 – *Creazione di una tassonomia per i servizi nel catalogo.*

```
//crea modello servizi_Italdata
  Modello model1=new Modello(); //crea oggetto modello servizi_Italdata
  model1.setProxy(proxy);
  model1.setAuth(auth);
  model1.setName("servizi_Italdata");
  model1.addDescription("it","Modello di riferimento per i servizi Italdata");
  //creazione documento di riferimento per questo modello.
  OverviewDoc doc=new OverviewDoc();
  doc.setURL("http://jack:8080/webservices/index.html");
  doc.addDescription("it","Dettali relativi a questo modello");
  //aggiungi documento al modello
  model1.setOverviewDoc(doc);

  //crea modello PAL (Pubblica Amministrazione Locale)
  Modello model2=new Modello(); //crea oggetto modello.
  model2.setProxy(proxy);
  model2.setAuth(auth);
  model2.setName("PAL");
  model2.addDescription("it","Modello di riferimento per la Pubblica
Amministrazione Locale");
  //creazione documento di riferimento per questo modello.
  doc=new OverviewDoc();
  doc.setURL("http://jack:8080/webservices/PAL/index.html");
  doc.addDescription("it","Dettali relativi a questo modello");
  //aggiungi documento al modello
  model2.setOverviewDoc(doc);

  //crea modello Information (Servizi di informazione)
  Modello model21=new Modello(); //crea oggetto modello.
  model21.setProxy(proxy);
  model21.setAuth(auth);
  model21.setName("Information");
  model21.addDescription("it","Modello di riferimento per \nservizi
d'informazione");

  //creazione documento di riferimento per questo modello.
  doc=new OverviewDoc();
  doc.setURL("http://jack:8080/webservices/PAL/information/index.html");
  doc.addDescription("it","Dettali relativi a questo modello");
  //aggiungi documento al modello
  model21.setOverviewDoc(doc);

  //crea modello Information (Servizi di informazione)
  Modello model211=new Modello(); //crea oggetto modello.
  model211.setProxy(proxy);
  model211.setAuth(auth);
```

```
  model211.setName("Utility");
  model211.addDescription("it","Modello di riferimento per \nservizi di utility
nell'area information");
  //creazione documento di riferimento per questo modello.
  doc=new OverviewDoc();
doc.setURL("http://jack:8080/webservices/PAL/information/utility/index.html");
  doc.addDescription("it","Dettali relativi a questo modello");
  //aggiungi documento al modello
  model211.setOverviewDoc(doc);

  //crea modello PMI (Piccole e Medie Imprese)
  Modello model3=new Modello(); //crea oggetto modello.
  model3.setProxy(proxy);
  model3.setAuth(auth);
  model3.setName("PMI");
  model3.addDescription("it","Modello di riferimento per la Piccole e Medie
Imprese");
  //creazione documento di riferimento per questo modello.
  doc=new OverviewDoc();
  doc.setURL("http://ejbean:8080/webservices/PIM/index.html");
  doc.addDescription("it","Dettali relativi a questo modello");
  //aggiungi documento al modello
  model3.setOverviewDoc(doc);

  //crea modello Banking (Servizi per la banca)
  Modello model4=new Modello(); //crea oggetto modello.
  model4.setProxy(proxy);
  model4.setAuth(auth);
  model4.setName("Banking");
  model4.addDescription("it","Modello di riferimento per servizi bancari");

  //creazione documento di riferimento per questo modello.
  doc=new OverviewDoc();
  doc.setURL("http://jack:8080/webservices/Banking/index.html");
  doc.addDescription("it","Dettali relativi a questo modello");
  //aggiungi documento al modello
  model4.setOverviewDoc(doc);

  //crea modello e-Learning (Corsi di formazione on-line)
  Modello model5=new Modello(); //crea oggetto modello.
  model5.setProxy(proxy);
  model5.setAuth(auth);
  model5.setName("e-Learning");
  model5.addDescription("it","Modello di riferimento per i corsi di formazione on-
line");
  //creazione documento di riferimento per questo modello.
```

```
doc=new OverviewDoc();
doc.setURL("http://jack:8080/webservices/e-Learnig/index.html");
doc.addDescription("it","Dettali relativi a questo modello");
//aggiungi documento al modello
model5.setOverviewDoc(doc);
//crea una tassonomia per i servizi Italdata.
KeyedReference category=new KeyedReference();
category.setName("uddi-org:taxonomy"); //categoria definita da UDDI 2.0
category.setValue("servizi");
//setta chiave modello di riferimento.
category.setTModelKey("UUID:3FB66FB7-5FC3-462F-A351-C140D9BD8304");
model1.addCategory(category);
model1.save(); //pubblica il modello servizi_italdata

 if (model1.getTModelKey()!=null){
//aggiunge categoria
KeyedReference category=new KeyedReference();
category.setName(model1.getName());
category.setValue("servizi");
//setta chiave modello di riferimento.
category.setTModelKey(model1.getTModelKey());
model2.addCategory(category);//aggiunge servizi PAL
model2.save(); //pubblica modello PAL

 if (model2.getTModelKey()!=null){
 //aggiunge categoria
 KeyedReference category=new KeyedReference();
 category.setName(model2.getName());
 category.setValue("servizi");
//setta chiave modello di riferimento.
category.setTModelKey(model2.getTModelKey());
model21.addCategory(category);//aggiunge Information
 model21.save();// pubblica Information

 if (model21.getTModelKey()!=null){
 //aggiunge categoria
 KeyedReference category=new KeyedReference();
 category.setName(model21.getName());
 category.setValue("servizi");
 //setta chiave modello di riferimento.
 category.setTModelKey(model21.getTModelKey());
 model211.addCategory(category);//aggiunge utility
 model211.save();//pubblica utility}}

 //aggiunge categoria
KeyedReference category=new KeyedReference();
```

```
category.setName(model1.getName());
category.setValue("servizi");
//setta chiave modello di riferimento.
category.setTModelKey(model1.getTModelKey());
model3.addCategory(category);//aggiunge PMI a sevizi_Italdata
 model3.save();//pubblica

//aggiunge categoria
KeyedReference category=new KeyedReference();
category.setName(model1.getName());
category.setValue("servizi");
//setta chiave modello di riferimento.
category.setTModelKey(model1.getTModelKey());
model4.addCategory(category);//aggiunge Banking a servizi_Italdata
model4.save();//pubblica

 //aggiunge categoria
KeyedReference category=new KeyedReference();
category.setName(model1.getName());
category.setValue("servizi");
//setta chiave modello di riferimento.
category.setTModelKey(model1.getTModelKey());
model5.addCategory(category);//aggiunge e-Learning a servizi_Italdata
 model5.save();//pubblica
```

Listato 7.9 – *Pubblicazione della classificazione di figura 7.16.*

Per pubblicare le informazioni relative all'unità organizzativa si deve utilizzare la struttura *BusinessEntity* della specifica UDDI, come mostrato nel listato 7.11.

```
//creare il proxy (consente di comunicare con il server UDDI)
UDDIProxy proxy=new UDDIProxy();

//settare l'endpoint del registro UDDI.
String endPointUDDI="http://localhost:8080/uddi/servlet/uddi";
try {
proxy.setPublishingEndpoint(new URL(endPointUDDI));
proxy.setInquiryEndpoint(new URL(endPointUDDI));
//creazione degli oggetti associati all'entità: unità organizzativa.

//descrizione dell'U.O.
Description
desc=(Description)UDDIPool.getObject(UDDIPool.DESCRIPTION);
```

```
  desc.setLanguage("it"); //linguaggio adoperato per il testo
  desc.setText("ECO, unità organizzativa interna all'azienda Italdata.\nProgetti
attivi:FinanzaLocale e BancaReale");
  Address add=new Address();
  add.setSortCode("1");
  add.setUseType("line 1");
  //Crea e setta una linea indirizzo.
  AddressLine line=new AddressLine();
  line.setLine("Centro Direzionale - Collina Liguorini \nEdificio D 83100
Avellino");
  line.setKeyName("line 1");
  line.setKeyValue("1");
  add.addAddressLine(line);//aggiunge linea indirizzo.

  //setta n.ro telefonico
  Phone phone=new Phone();
  phone.setPhoneNumber("08257712xx");

  //imposta email
  Email email=new Email();
  email.setEmailAddress("roberto.capobianco@italdata.it");

  //crea un contatto.
  Contact contact=new Contact();
  contact.setPersonName("Roberto Capobianco"); //responsabile del contatto
  contact.addAddress(add);
  contact.addEmail(email);
  contact.addPhone(phone);

  //Crea un indirizzo Web dove localizzare ulteriori descrizioni
  DiscoveryURL disc=new DiscoveryURL();
  disc.setURL("www.italdata.it");
  disc.setUseType("HTTP");

  //crea entità U.O.
  BusinessEntity biz= new BusinessEntity();
  //imposta il nome.
  biz.addName("it","Italdata-ECO"); //it: lingua italiana

  //aggiunge la descrizione, il contatto e indizizzo Web
  biz.addDescription(desc);
  biz.addContact(contact);
  biz.addDiscoveryURL(disc);

  /*Per pubblicare un'entità, il provider ha bisogno di un token, cioè di un'
   *autorizzazione.
```

```
 */
//richiesta di un token
AuthToken authToken=proxy.get_authToken(auth);
//registrazione dell'U.O.
BusinessDetail detailBusiness=proxy.save_business(authToken,biz);
//distrugge il token, per evitare che qualcuno non autorizzato lo possa utilizzare.
proxy.discard_authToken(authToken);
}catch(MalformedURLException e_url){
System.out.println("URL Errore "+e_url.getMessage());
}catch(UDDIException e_uddi){
System.out.println("UDDI Errore "+e_uddi.getMessage());
}catch(Exception e_soap){
System.out.println("SOAP Errore "+e_soap.getMessage());
e_soap.printStackTrace();
}
```

Listato 7.11 – *Pubblicazione delle informazioni circa l'unità organizzativa.*

Nella figura 7.18 è mostrata la descrizione dell'unità organizzativa come viene presentata dall'interfaccia utente del UDDIbrowser.

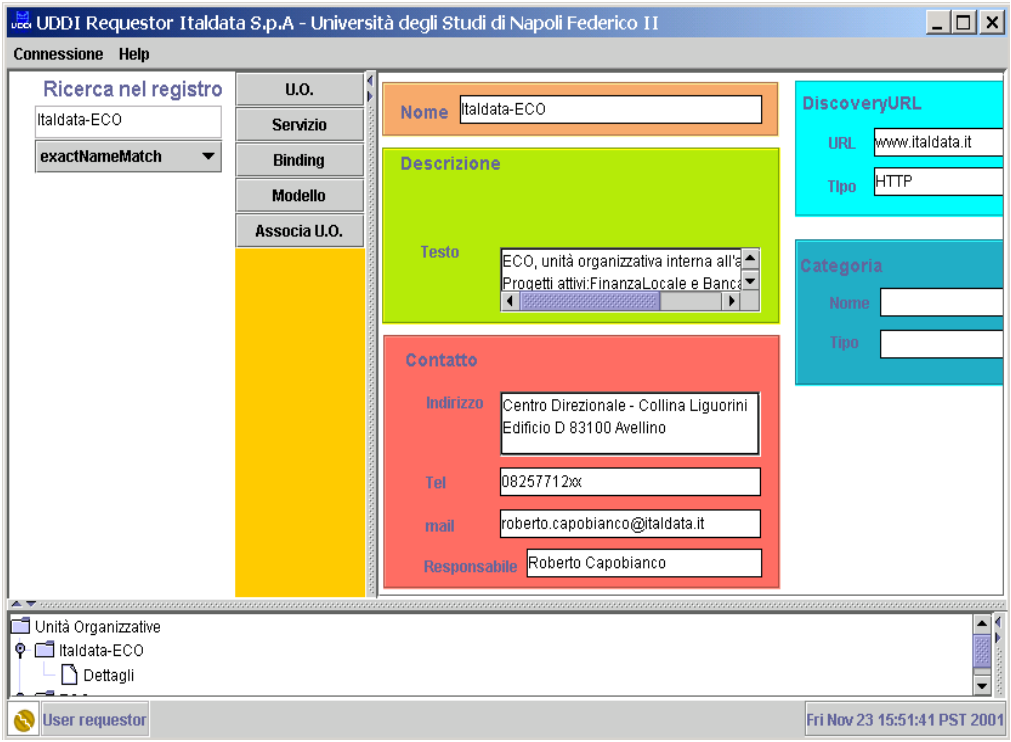


Figura 7.18 – *Finestra dell'interfaccia utente del catalogo. Mostra le informazioni sull'unità organizzativa E.C.O.*

Per la pubblicazione del servizio, può essere usato il programma *servicepublish* presentato nel capitolo precedente. La figura 7.19 mostra la finestra utente che richiede di impostare il percorso del documento della definizione dell'implementazione (dalla quale carica il nome del servizio, i punti d'accesso e la definizione dell'interfaccia). L'utente deve altresì specificare l'unità organizzativa e la categoria a cui tale servizio deve appartenere.

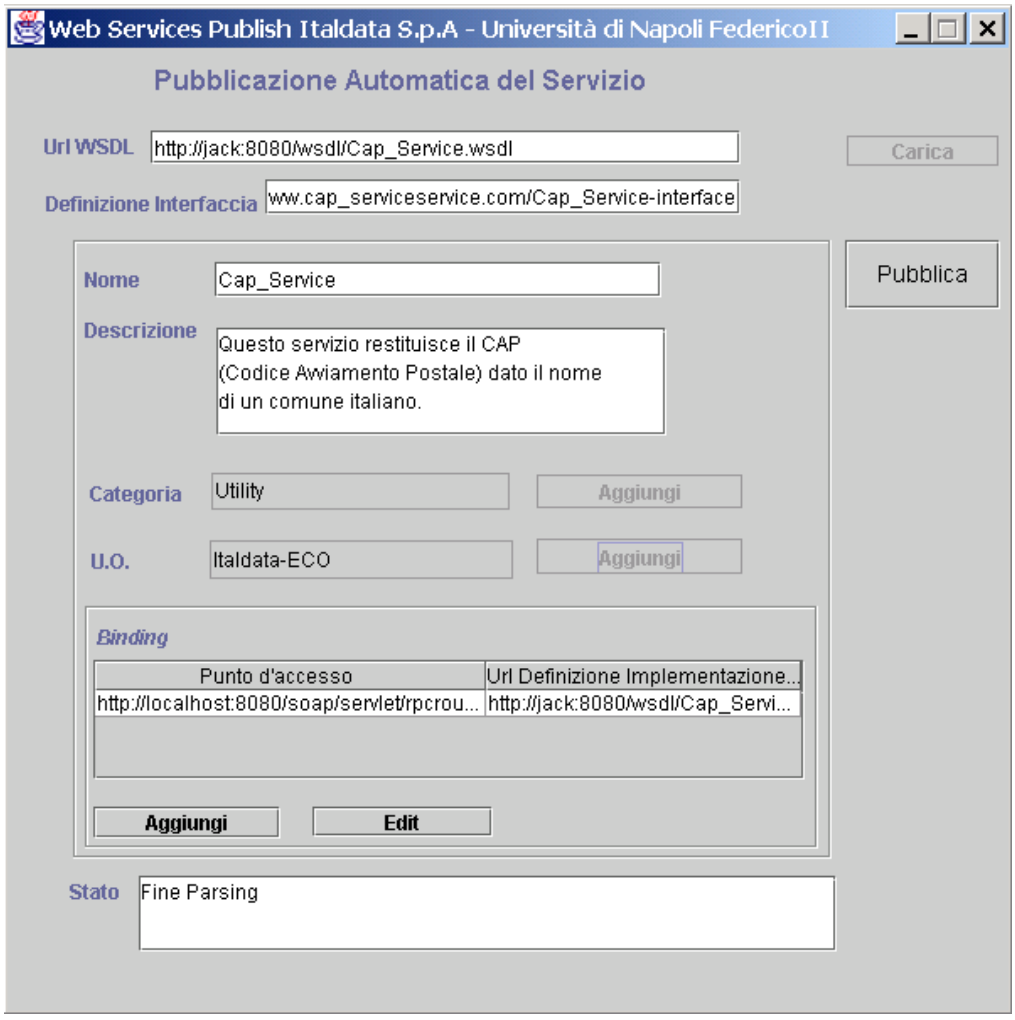


Figura 7.19 – *Applicazione servicepublish che consente di pubblicare un servizio a partire dalla sua definizione dell'implementazione.*

Tuttavia, tale applicazione non consente ancora di aggiungere alle strutture BindingTemplate le istanze ai modelli. Queste possono essere utilizzate per impostare, per ciascun binding, un riferimento ad un modello che descrive un certo concetto. Ad esempio, supponiamo che il service provider abbia realizzato una pagina html che descrive in dettaglio il servizio (interfaccia del servizio, modalità d'uso, etc.) e che voglia renderla disponibile ad un eventuale service requestor insieme alla definizione del servizio (quella mostrata dal catalogo). Allora, il provider deve semplicemente realizzare un'istanza al modello che definisce il documento html, come mostrato in figura 7.20 e settare la descrizione e l'indirizzo della pagina .html in tale istanza.

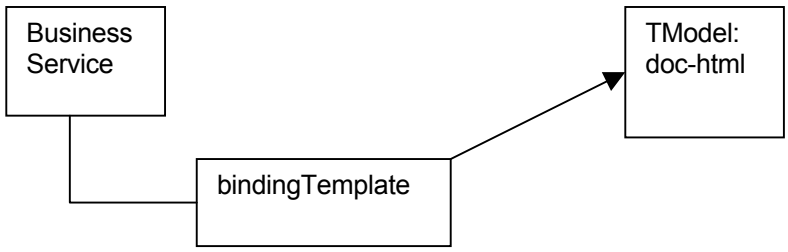


Figura 7.20 – *Il bindingTemplate del servizio punta al modello doc-html per includere nel binding una descrizione dettagliata del servizio.*

Il listato 7.12 crea le istanze ai quattro modelli: doc-html per i documenti html, java-class per i servizi implementati come classi Java, al modello della definizione dell'interfaccia (targetNamespace), e al modello della definizione dell'implementazione.

```
//aggiunge un'istanza per determinare la posizione in rete
//della definzione dell'implementazione del servizio Cap_Service

//identificatore del modello rappresentativo delle definizioni delle
//implementazioni dei servizi
   TModelInstanceInfo ins0=new TModelInstanceInfo();
   ins0.setTModelKey("UUID:624DFD00-DF2A-1D51-98A1-
     BFA70101AA77");
//crea dettagli istanza al modello.
   InstanceDetails iDetails0=new InstanceDetails();
   OverviewDoc overviewDoc0=new OverviewDoc();
//setta indirizzo.
   overviewDoc0.setURL("http://jack:8080/wsdl/Cap_Service.wsdl");
   iDetails0.setOverviewDoc(overviewDoc0);
//descrive l'istanza
   Description desc0=new Description();
   desc0=new Description();
   desc0.setLanguage("it");
   desc0.setText("Definizione dell'implementazione del servizio");
   iDetails0.addDescription(desc0);
   ins0.setInstanceDetails(iDetails0);
//aggiunge istanza al binding.
   bindingTemplate.addTModelInstanceInfo(ins0);


//aggiunge un'istanza alla definzione dell'interfaccia
// del servizio Cap_Service

//identificatore del modello rappresentativo della definizione dell'
//interfaccia del servizio
   TModelInstanceInfo ins1=new TModelInstanceInfo();
   ins1.setTModelKey("UUID:624DFD00-DF2A-1D51-98A1-
      BFA70101AA77");
   InstanceDetails iDetails1=new InstanceDetails();
   OverviewDoc overviewDoc1=new OverviewDoc();
   overviewDoc1.setURL("www.cap_service_service.com/Cap_Service-
       Interface");
   iDetails1.setOverviewDoc(overviewDoc1);
   Description desc1=new Description();
   desc1.setLanguage("it");
   desc1.setText("Definizione dell'interfaccia del servizio");
   iDetails1.addDescription(desc1);
   ins1.setInstanceDetails(iDetails1);
   bindingTemplate.addTModelInstanceInfo(ins1);

//aggiunge un'istanza del modello java-class per definire l'indirizzo del
```

```
//package di distribuzione del servizio.
   TModelInstanceInfo ins2=new TModelInstanceInfo();
//identificatore del modello java-class
   ins2.setTModelKey("UUID:5E5E6AE0-E3A6-1D51-A82F-
      7F000001AA77");
   InstanceDetails iDetails2=new InstanceDetails();
   OverviewDoc overviewDoc2=new OverviewDoc();
   overviewDoc2.setURL("http://jack:8080/webservices
            /implementazioni/cap/cap.zip");
   iDetails2.setOverviewDoc(overviewDoc2);
   Description desc2=new Description();
   desc2.setLanguage("it");
   desc2.setText("Implementazione del servizio");
   iDetails2.addDescription(desc2);
   ins2.setInstanceDetails(iDetails2);
   bindingTemplate.addTModelInstanceInfo(ins2);

//aggiunge un'istanza del modello doc-html per definire l'indirizzo della
//hompage che descrive in maniera dettagliata il servizio.
   TModelInstanceInfo ins3=new TModelInstanceInfo();
   ins3.setTModelKey("UUID:65EB5680-DD52-1D51-B3B2-
      BFA70101AA77");
   InstanceDetails iDetails3=new InstanceDetails();
   OverviewDoc overviewDoc3=new OverviewDoc();
   overviewDoc3.setURL("http://jack:8080/webservices
     /implementazioni/cap/cap.html");
   iDetails3.setOverviewDoc(overviewDoc3);
   Description desc3=new Description();
   desc3.setLanguage("it");
   desc3.setText("Descrizione dettagliata del servizio");
   iDetails3.addDescription(desc3);
   ins2.setInstanceDetails(iDetails3);
   bindingTemplate.addTModelInstanceInfo(ins3);
```

Listato 7.12 – *Mostra come creare e aggiungere istanze di modelli al bindingTemplate del servizio Cap_Service.*

6. Utilizzo del Web Service.

Il caso di utilizzo considerato in questo esempio è un binding di tipo statico che avviene al tempo di sviluppo. Esso consiste di quattro fasi:

1. Trovare la definizione dell'implementazione del servizio.
2. Generare un modulo proxy per utilizzare il servizio remoto.
3. Integrare il proxy nell'applicazione.
4. Testare il Web Service Cap_Service.

6.1. Scoperta della definizione dell'implementazione del servizio.

Il service requestor può utilizzare l'UDDIbrowser per cercare il servizio desiderato. Questo consente di navigare tra le unità organizzativa, visualizzando i relativi servizi realizzati, tra le categorie dello schema di classificazione dei servizi, e cercare una unità organizzativa o un servizio noto il nome. In verità, la API di ricerca della specifica UDDI 2.0 (implementate dall'UDDIProxy [43]) consentono anche altri tipi di ricerca più avanzate, per i quali si rimanda alla suddetta specifica [28].

Nella figura 7.21 è riportata la finistra dell'UDDIbrowser che visualizza i servizi relativi a ciascuna unità organizzativa.

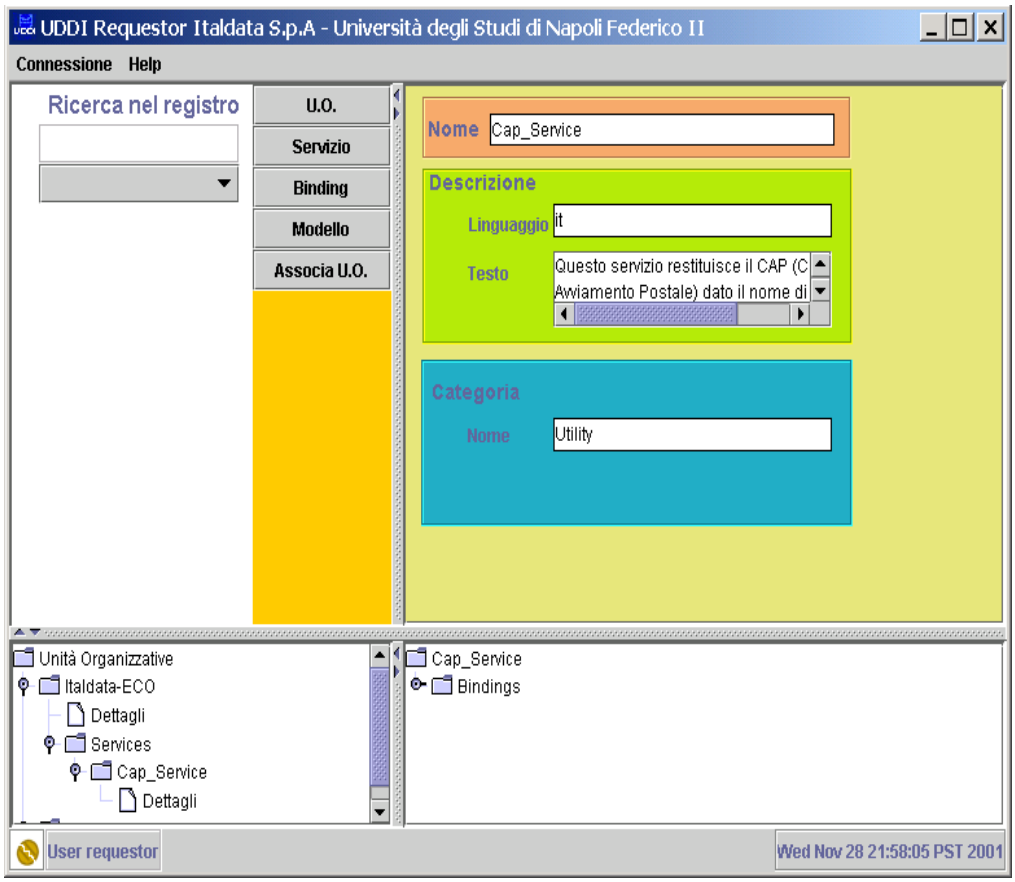


Figura 7.21 – *Ricerca navigando tra i servizi offerti dalle unità organizzative.La finestra mostra le informazioni sul servizio Cap_Service.*

Mentre nella figura 7.22 è riportato un caso di ricerca per categoria di servizi. In essa è riportata anche la lista di istanze create nel listato 7.12, oltre all'url del server SOAP Apache (punto d'accesso) e alla descrizione del binding. Dalla lista delle istanze (rappresentate dalla cappia (descrizione, indirizzo)) l'utente può scegliere di scaricare la definizione dell'implementazione (in realtà basta solo l'indirizzo), di visualizzare la pagina html che descrive i dettagli relativi al servizio (in tal caso deve copiare l'indirizzo e incollarlo nella barra indirizzo del

suo browser HTML) , o di scaricare l'implementazione del servizio (cap.zip).

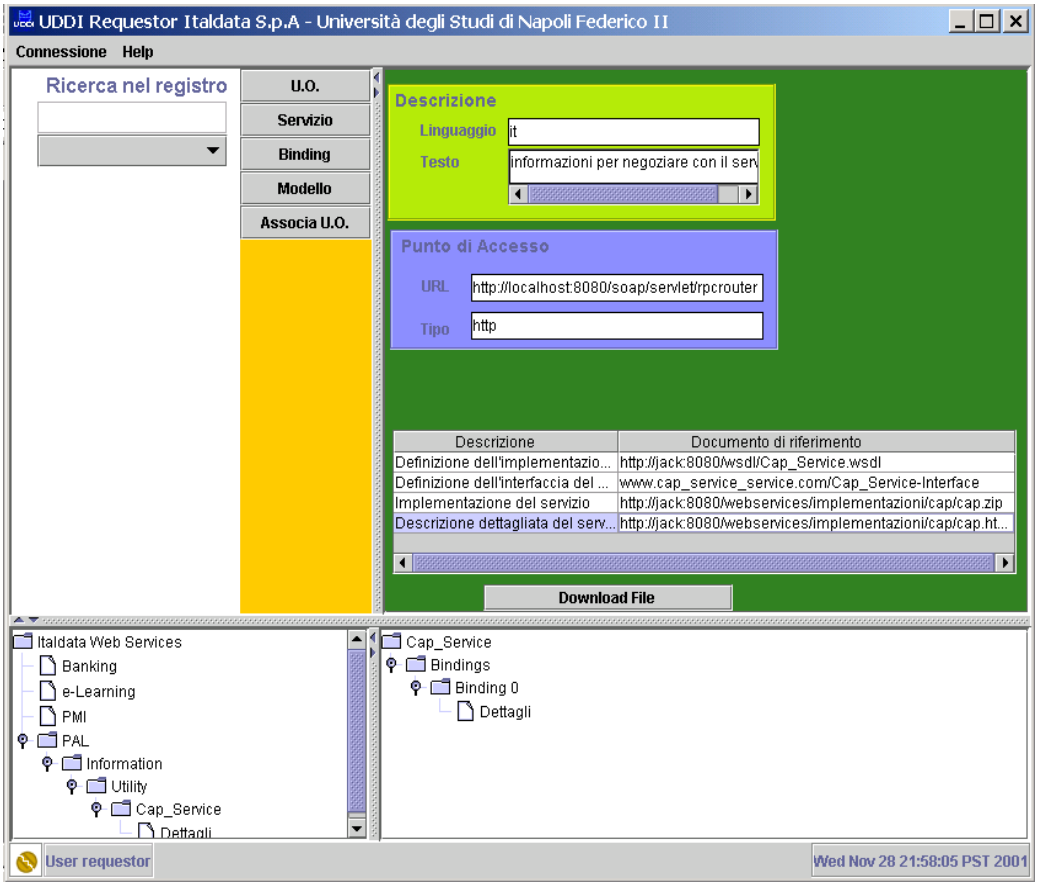


Figura 7.22 – *Ricerca di un servizio navigando tra le categorie di servizi.*

6.2. Generazione del proxy.

Una volta noto l'indirizzo url della definizione dell'implementazione il service requestor può utilizzare il programma proxygen del toolkit WSTK 2.3 per generare automaticamente il proxy. Il programma (applicazione Java da linea di comando) scarica la definizione

dell'implementazione, e, successivamente, mediante l'indirizzo url, estratto dall'elememto *import,* scarica anche la definizione dell'interfaccia. Questa, dopo essere stata analizzata dal parser XML, viene utilizzata per generare il sorgente Java che viene compilato e salvato su disco.

Di seguito è riportata la linea di comando per eseguire l'utility proxygen:

```
C:\WSTK-2.3\BIN>proxygen http://jack:8080/wsdl/Cap_Service.wsdl
'DB2_HOME' is not recognized as an internal or external command,
operable program or batch file.
>> Importing http://jack:8080/wsdl/Cap_Service-interface.wsdl ..
>> Transforming WSDL to NASSL ..
>> Generating proxy ..
Created file C:\WSTK-2.3\BIN\Cap_ServiceProxy.java
Compiled file C:\WSTK-2.3\BIN\Cap_ServiceProxy.java
Done.
```

Il listato 7.13 invece rappresenta il proxy Java che consente di eseguire le richieste SOAP al servizio remoto.

```
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.encoding.*;
import org.apache.soap.rpc.*;
import org.apache.soap.util.xml.*;

public class Cap_ServiceProxy
{
  private Call call = new Call();
  private URL url = null;
```

```
private String SOAPActionURI = "";
private SOAPMappingRegistry smr = call.getSOAPMappingRegistry();

public Cap_ServiceProxy() throws MalformedURLException
{
 call.setTargetObjectURI("urn:cap_service");
 call.setEncodingStyleURI("http://schemas.xmlsoap.org/soap/encoding/");
 this.url = new URL("http://jack:8080/soap/servlet/rpcrouter");
 this.SOAPActionURI = "urn:cap_service";
}

public synchronized void setEndPoint(URL url)
{
 this.url = url;
}

public synchronized URL getEndPoint()
{
 return url;
}

public synchronized java.lang.String getCap
 (java.lang.String meth1_inType1) throws SOAPException
{
 if (url == null)
 {
  throw new SOAPException(Constants.FAULT_CODE_CLIENT,
  "A URL must be specified via " +
  "Cap_ServiceProxy.setEndPoint(URL).");
 }

 call.setMethodName("getCap");
 Vector params = new Vector();
 Parameter meth1_inType1Param = new Parameter("meth1_inType1",
  java.lang.String.class, meth1_inType1, null);
 params.addElement(meth1_inType1Param);
 call.setParams(params);
 Response resp = call.invoke(url, SOAPActionURI);

 // Check the response.
 if (resp.generatedFault())
 {
  Fault fault = resp.getFault();

  throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
 }
```

```
    else
    {
     Parameter retValue = resp.getReturnValue();
     return (java.lang.String)retValue.getValue();
    }
   }

}
```

Listato 7.13 – *Il proxy Java che consente di utilizzare il Web Service Cap_Service (urn:cap_service è l'identificatore unico del servizio usato dall'ambiente SOAP Apache).*

Una chiamata al metodo *getCap* viene trasformata in una richiesta SOAP mediante un oggetto della classe Call .

Di seguito riporto i passi di base per invocare un metodo remoto mediante gli oggetti della classe Call e Response:

1. Creare l'oggetto org.apache.soap.rpc.RPCMessage.Call (la classe Call è l'interfaccia principale per il codice SOAP RPC)
2. Settare il target URI dell'oggetto Call usando il metodo setTargetObjectURI(…) con l'URN che viene usato per identificare il servizio (urn:Cap_Service in questo caso).
3. Settare il nome del metodo che si desidera invocare nell'oggetto Call usando il metodo setMethodName() (per esempio, setMethodName("getCap").
4. Creare un oggetto Parameter per ciascun parametro del metodo da invocare (per la firma del metodo si può consultare la definizione dell'interfaccia del servizio) e aggiungerli all'oggetto Call usando il metodo setParameter(…). Il costruttore della classe Parameter prende quattro valori: il

nome del parametro, il tipo, il valore, e l'URI dello stile di codifica del tipo [26].

5. Eseguire il metodo invoke(…) sull'oggetto Call e catturare l'oggetto della classe Response che è ritornato dal metodo invoke(…). Questo metodo prende due parametri. Il primo è l'URL che identifica l'endpoint presso il quale il servizio risiede (per esempio: *http://jack:8080/soap/servlet/rpcrouter/*) e il secondo è il valore che deve essere inserito nell'header SOAPAction. Inoltre, si osservi che la richiesta SOAP è sincrona, e quindi può essere rievocata solo dopo la sua conclusione.

6. Controllare l'oggetto Response per vedere se è stato generato un fallimento (fault) usando il metodo generatedFault(…).

7. Se è stato ritornato un errore, questo può essere recuperato col metodo getFault(…), altrimenti estrarre il risultato oppure i parametri ritornati usando il metodo getReturnValue() e getParams(…) rispettivamente.

6.3. Integrazione e utilizzo del Web Service.

Nel listato 7.14 è riportata un'applicazione che usa il proxy Cap_ServiceProxy per effettuare le chiamate al metodo remoto getCap del servizio Web Cap_Service. E' una semplice interfaccia che richiede il nome di un comune e restituisce il relativo CAP.

Si osservi che quando si crea il proxy bisogna catturare le eventuali eccezioni MalformedURLException. Queste vengono lanciate

dall'istruzione new URL("http://jack:8080/soap/servlet/rpcrouter") (vedi listato 7.13) che crea un oggetto URL e lo setta al valore dell'url dell'endpoint del server SOAP Apache.

Invece, se si chiama il metodo getCap bisogna catturare l'eccezione SOAPException. Questa può essere lanciata quando:

- si chiama il metodo getCap con l'attributo url dell'oggetto Cap_ServiceProxy nullo.
- è il metodo invoke dell'oggetto della classe Call a lanciarla.
- se la richiesta ritorna un fallimento.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import javax.swing.border.*;

import java.io.*;
import java.net.*;
import java.net.MalformedURLException;
import org.apache.soap.*;

/**Questa classe può essere usata per testare il Web Service Cap_Service.
 * E' dotata d'interfaccia grafica.
 */
public class ProvaCap_Service extends JFrame {

  Container contentPane=null;
  JPanel panel= new JPanel();
  JButton jB=new JButton("Avvia Ricerca");
  JTextField searchComune = new JTextField();
  JLabel jL= new JLabel("Immetti il nome del comune/città da cercare");
  private Cap_ServiceProxy cap_serviceProxy=null;

  public ProvaCap_Service() {
```

```
 init();
 try {
 cap_serviceProxy= new Cap_ServiceProxy();
  }catch (MalformedURLException url_e){
        System.out.println("Errore URL: "+url_e.getMessage());
        }
 }

 //setta impostazioni finestra
 private void init(){
 Dimension dimF= new Dimension(350,200);
 this.setSize(dimF);
 this.setTitle("Esempio d'integrazione di un Web Service");
 contentPane=this.getContentPane();
 contentPane.setLayout(new BorderLayout());
 contentPane.setBackground(Color.white);
 panel.setLayout(new BorderLayout());
 panel.setBackground(Color.white);
 panel.add(jB,BorderLayout.NORTH);
 panel.add(searchComune,BorderLayout.SOUTH);
 panel.add(jL,BorderLayout.CENTER);
 contentPane.add(panel,BorderLayout.CENTER);


/****************Aggiungi Listener su windowClose***************/
 addWindowListener(new WindowAdapter(){
   public void windowClosing(WindowEvent e_win){
        System.exit(0);
        }
   });
/****************Aggiungi Listener sul pulsante jB***************/
 jB.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(ActionEvent e) {
       jB_actionPerformed(e);
     }
   });

 }

/**Questo metodo gestisce l'evento actionPerforme associato al pulsante
 * jB
 */

 void jB_actionPerformed(ActionEvent e){
  String cap=null;
```

```
 if (searchComune.getText()!=null){

try {

 cap=cap_serviceProxy.getCap(searchComune.getText());

 if (cap!=null)
  jL.setText(cap);
 else
  jL.setText("Nessun risultato");
   } catch (SOAPException soap_e){
 System.out.println("Errore Soap: "+soap_e.getMessage());
  }
 }
}

//Metodo main
 public static void main(String argv[]){
  ProvaCap_Service prova=new ProvaCap_Service();
  prova.show();
}
}
```

Figura 7.14 – *Applicazione per provare il funzionamento del Web Service Cap_Service.*

Nella figura 7.23 è riportato la finestra dell'applicazione che mostra il cap del paese Montepulciano.

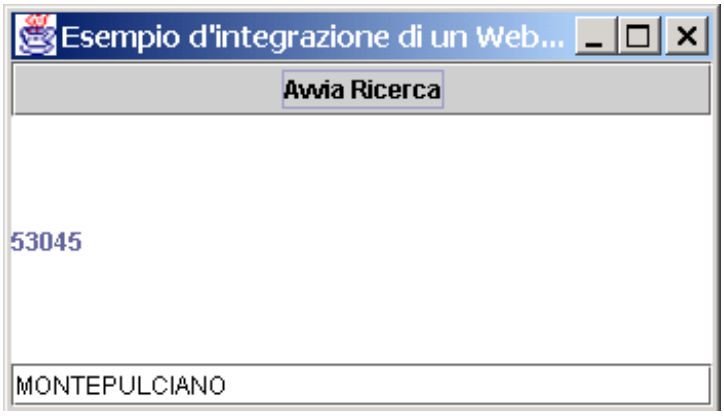


Figura 7.23 – *Applicazione in cui è stato integrato il servizio Web.*

7. Conclusioni

I Web Services sono costruiti sulle specifiche XML, SOAP, WSDL e UDDI che forniscono il fondamento all'integrazione e aggregazione di applicazioni (application-to-application). Su queste specifiche di base le compagnie stanno sviluppando nuove soluzioni per cercare di fornire valore aggiunto ai loro servizi esponendoli come Web Services. Conseguentemente, la necessità di estendere l'infrastruttura di base dei Web Services sta crescendo rapidamente.

Per evitare che alcune funzionalità di alto livello come la gestione della sicurezza, il routing e l'affidabilità dei messaggi, e le transazioni vengano sviluppate in maniera proprietarie e quindi non interoperabile, la Microsoft e IBM hanno sottomesso in visione al W3C Workshop on Web Services 11-12 April 2001 uno schizzo di un comune framework dei Web Services che dovrebbe fornire le specifiche (sempre basate su XML) con cui sviluppare queste funzionalità. Il framework individua i componenti e le funzioni necessarie per ottenere interoperabilità tra i Web Service, in modo particolare nell'interazione programm-to-program con riferimento alle azienda [53].

Microsoft identifica l'intera collezione di specifiche costruite su quelle di base e che insieme a queste consentono di realizzare Web Service in maniera completa, nella Global XML Web Services Architecture (mostrata in figura 7.24).

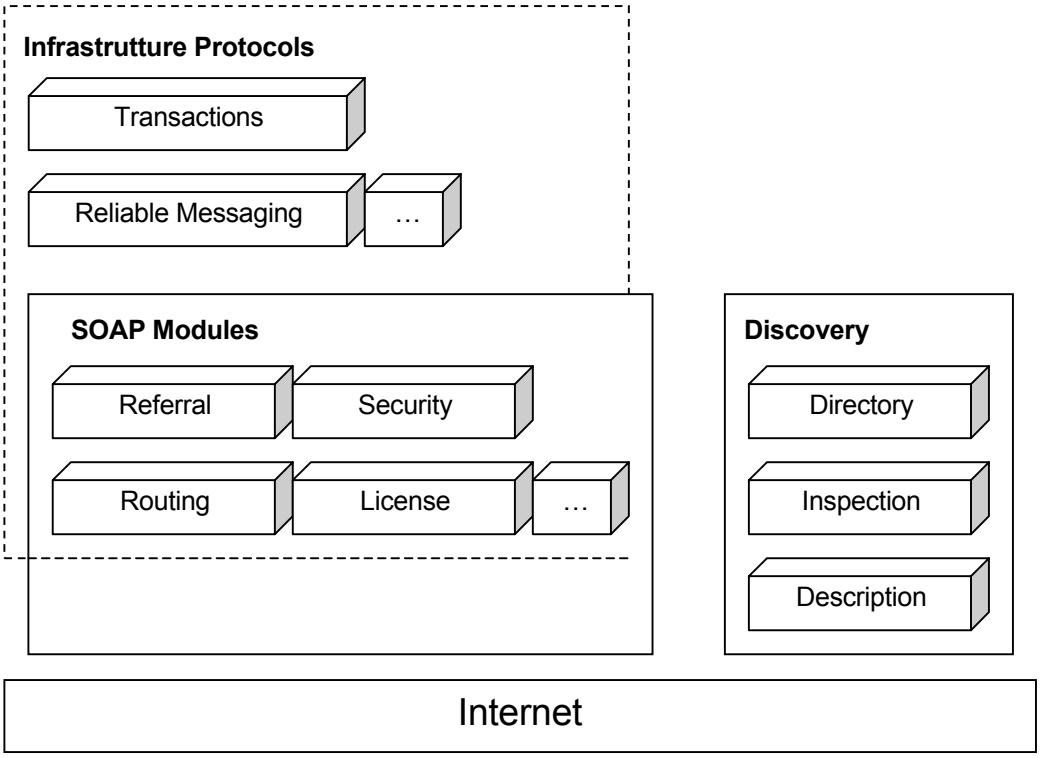


Figura 7.24 – *Global XML Web Services Architecture proposta da Microsoft.*

Le Global XML Web Services Specification sono riportate di seguito:

- WS-Inspection
- WS-License
- WS-Referral
- WS-Routing
- WS-Security

Una breve descrizione.

WS-Inspection.

L'ispezione è il processo d'interrogazione di un server per scoprire quali servizi offre, le relative descrizioni, e dove sono localizzato. Questo consente ad un'applicazione di trovare dinamicamente nuovi servizi.

WS-Inspetion (Web Service Inspection Language) è un linguaggio in formato XML che mette a disposizione un insieme di regole per creare informazioni ispezionabili. Un documento WS-Inspection fornisce i mezzi per realizzare riferimenti alle descrizione dei servizi esistenti che sono stati editati in qualunque formato [54]. Questi documenti d'ispezione sono resi disponibili presso il fornitore del servizio.

La specifica WS-Inspection è stata pubblicata da IBM e Microsoft il primo novembre del 2001.

WS-Inspection definisce gli elementi XML basandosi principalmente sulle specifiche WSDL e UDDI.

Un esempio di documento d'ispezione:

```
<?xml version="1.0"?>
<inspection              xmlns="http://schemas.xmlsoap.org/wsil/"
xmlns:wsiluddi="http://schemas.xmlsoap.org/wsil/uddi/">
  <service>
    <abstract>
       A stock quote service with two descriptions
    </abstract>
    <description
       referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
       location="http://example.com/stockquote.wsdl"/>
    <description referencedNamespace="urn:uddi-org:api">
       <wsiluddi:serviceDescription
           location="http://www.example.com/uddi/inquiryapi">
         <wsiluddi:serviceKey>
                   4FA28580-5C39-11D5-9FCF-BB3200333F79
         </wsiluddi:serviceKey>
       </wsiluddi:serviceDescription>
    </description>
  </service>
  <service>
    <description
```

```
    referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
    location="ftp://anotherexample.com/tools/calculator.wsdl"/>
  </service>
  <link referencedNamespace="http://schemas.xmlsoap.org/wsil/"
        location="http://example.com/moreservices.wsil"/>
</inspection>
```

Quest'esempio mostra come WS-Inspection ritorna una lista di servizi con i relativi metadati.

I documenti WS-inspection sono semplici da costruire, leggeri, e facili da manutenere.

WS-License

La licenza è vista come una specie di credenziale che contiene un insieme di dichiarazioni firmate (assertion signed) da un'authority.

WS-License (Web Services License Language) è un linguaggio XML con una propria grammatica che descrive un set di tipi di licenze comunemente usate [55] e un meccanismo d'estensione per aggiungere altre caratteristiche alle licenze inviate nel messaggio SOAP.

Esso è costruita sulla specifica WS-Security che definisce i meccanismi per trasmettere messaggi SOAP sicuri.

WS-Referral

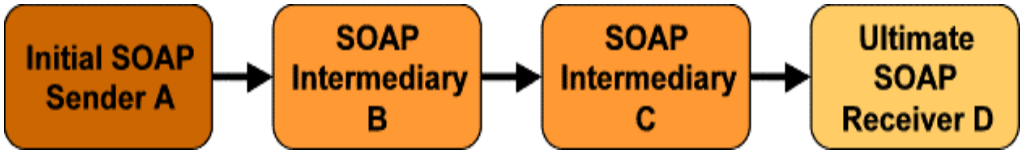
SOAP definisce un modello in cui i messaggi possono contenere alcune porzioni (elemento header) destinate a specifici nodi elaborativi, detti intermediari [9]. Ebbene, WS-Referral fornisce un set di istruzioni, in formato XML, per la configurare il comportamento degli

intermediari rispetto ai messaggi che arrivano e che devono essere inoltrati [56].

WS-Routing

SOAP è un protocollo di rete leggero che definisce i meccanismi di serializzazione per trasportare chiamate a metodi remoti. Tuttavia, attualmente, non definisce un meccanismo per inviare un messaggio da una parte all'altra. WS-Routing (formalmente conosciuto come SOAP-RP) è un protocollo stateless che estende SOAP definendo i mezzi per specificare un message path che va dal creatore del messaggio all'ultimo ricevente del messaggio, attraversando gli intermediari [57].

WS-Routing definisce un nuovo elemento SOAP header e un modello di elaborazione associato. Ad esempio, si consideri un sender SOAP A che desidera inviare un messaggio SOAP ad un server SOA D attraverso gli intermediari B e C, come mostrato nella sequente figura:



Per esprimere il percorso del messaggio, WS-Routing definisce un nuovo header SOAP chiamato "path", e i seguenti elementi:

- un *from* per il creatore del messaggio (A)
- un *to* per l'ultimo ricevente del messaggio (D)

- un *fwd* che contiene il percorso che il messaggio deve seguire (forward message path)
- un *rev* che contiene un riverse message path.

Di seguito è mostrato il messaggio SOAP inviato da A.

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://www.w3.org/2001/06/soap-envelope">
   <SOAP-ENV:Header>
      <wsrp:path xmlns:wsrp="http://schemas.xmlsoap.org/rp/">
         <wsrp:action>http://www.im.org/chat</wsrp:action>
         <wsrp:to>soap://D.com/some/endpoint</wsrp:to>
         <wsrp:fwd>
            <wsrp:via>soap://B.com</wsrp:via>
            <wsrp:via>soap://C.com</wsrp:via>
         </wsrp:fwd>
         <wsrp:from>soap://A.com/some/endpoint</wsrp:from>
         <wsrp:id>uuid:84b9f5d0-33fb-4a81-b02b-
                  5b760641c1d6</wsrp:id>
      </wsrp:path>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      ...
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

WS-Security

Fornisce un linguaggio per rendere sicuri Web Services. La specifica è suddivisa in tre parti: credential transfer, message integrity, e message confidentiality [58]. Tuttavia, queste capacità da sole non forniscono una soluzione completamente sicura. WS-Security è un building-block che può essere usato in congiunzione con altri protocolli per affrontare una ampia varietà di requisiti di sicurezza.

In generale, per scopi di autenticazione, si possono inserire il nome utente e password in una richiesta SOAP. In questo modo, il Web Service può autenticare l'utente ogni volta che questo effettua la

richiesta, interrogando un'appropriata sorgente dati (database, LDAP, un file, etc.) in cui è memorizzata la lista degli utenti.. Quest'approccio è molto semplice da implementare, e anche se incrementa un po' il carico di trasmissione, è abbastanza efficiente.

Per miglioare il modello, invece di sottomettere le credenziali di autenticazione con la richiesta del servizio (per ogni richiesta il Web Service deve effettuare l'autenticazione), si dovrebbe realizzare, dentro il Web Service, un meccanismo che consegna all'utente, una volta autenticato, una chiave di sessione (con una data di scadenza). Questa può essere usata per autenticare le successive richieste al servizio.

Tuttavia, le credenziali dell'utente vengono inviate dentro il messaggio SOAP (nel body generalmente) e quindi in testo chiaro (plain text). Per risolvere questo inconveniente e nell'ipotesi che il protocollo a livello di trasporto usato è HTTP, si può combinare quest'autenticazione di base con SSL (Secured Sockets Layers).

SSL codifica le trasmissioni tra due punti della rete, crittografando il traffico tra il client e il server. La comunicazione avviene in HTTPs o HTTP Secured [8].

Ad esempio, si potrebbe usare SSL solo per l'autenticazione dell'utente, dopodiché il Web Service potrebbe restituire una chiave di sessione valida per un certo periodo di tempo. Allora, l'utente può usare questa chiave per chiamare il servizio Web in una regolare comunicazione HTTP invece che HTTPS.

In generale, il metodo adottato dipende dal livello di sicurezza richiesto dal servizio e si possono usare tutti i metodi di autenticazione previsti per sistemi di reti aperte.

BIBLIOGRAFIA

[1] Paul Allen, *"Service-based component architecture"*, Component Development Strategies (June 2001) – vol.XI,No. 6

[2] IBM Web Services Architecture Team, *"Web Services architecture Overview".*
www-106.ibm.com/developerworks/library/w-ovr/

[3] HP Web Services Team, *"Web Services Concepts - A technical overview".*
h20008.www2.hp.com/developers/ welcome2.asp

[4] Massimiliano Bigatti, *"Web Services III parte: La tecnologia UDDI", Mokabyte 56, Luglio/Agosto 2001*
www.mokabyte.it/2001/07/webservices.htm

[5] Mary Kirtland, *"A Platform for Web Services — Microsoft Developer Network".*
msdn.microsoft.com

[6] Mary Kirtland, "*The Programmable Web: Web Services Provides Building Blocks for the Microsoft .NET Framework*". *msdn.microsoft.com*

[7] e-Speak, HP. *http://* www.e-Speak.net

[8] Cauldwell P., Chawla R., Chopra V., Damshen G., Dix C., Hong T., Norton F., Ogbuji U., Olander G., Richman M., Saunders K., Zaev Z., *"Professional XML Web Services"*, Programmer to Programmer Wrox Press.

[9] Simple Object Access Protocol (SOAP) 1.1, W3C working draft. *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/*

[10] Benoit Marchal *"Pratical XML"* - Jackson Libri, 2000.

[11] Extensible Markup Language (XML) 1.0, *W3C Recommendation 10 Febbraio 1998. http://www.w3.org/TR/1998/REC-xml-19980210*

[12] XML Schema , W3C Raccomandation 2 Maggio 2001 Primer: *http://www.w3.org/TR/xmlschema-0/*

Structures: *http://www.w3.org/TR/xmlschema-1/*
DataTypes: *http://www.w3.org/TR/xmlschema-2/*

[13] Berners-Lee T., *"Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web"*, RFC 1630, CERN, June 1994.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1630.html

[14] Berners-Lee T., *"Uniform Resource Locators (URL)"*, RFC 1738, CERN, December 1994.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1738.html

[15] Sollins k., *"Funtional Requiremnts for Uniform Resource Names"*, RFC 1737, MIT/LCS, December 1994.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1737.html

[16] Brett McLaunghlin, *"Java e Xml"*, Apogeo 2001.

[17] API SAX 2.2.
http://www.megginson.com/SAX/SAX2

[18] API Document Object Model (DOM).

Livel 1: ***http://www.w3c.org/TR/REC-DOM-Level-1/***

Livel 2: ***http://www.w3c.org/TR/REC-DOM-Level-2/***

[19] **Scribner & Stiver, *"Understanding SOAP"*, *Sams*, 2000**

[20] **Berners-Lee T., Fielding R., et al., *"Hypertext Transfer Protocol (HTTP 1.1)"*, RFC 2616, W3C/MIT, June 1999.**
http://www.ietf.org/rfc/rfc2616.txt

[21] **N. Freed, Borenstein, *"Multipurpose Internet Mail Extensions , Part One"*, RFC 2045, Innosoft e First Virtual, November 1996.**
http://www.ietf.org/rfc/rfc2045.txt

[22] **Java Servlet Technology.**
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html

[23] **Web Service Description Language (WSDL) 1.1, W3C submission.**
http://www.w3.org/TR/wsdl

[24] **Massimiliano Bigatti, *"Web Services II parte: Capire i WSDL"*, Mokabyte 53, Giugno 2001.**
http://www.mokabyte.it/2001/06/webservices.htm

[25] **Patrick Naughton, "Il manuale Java", McGraw-Hill, 1996.**

[26] **SOAP APACHE v2.1 Documentation.**
http://xml.apache.org/soap

[27] **UDDI Version 2.0 Data Structure Specification.**
http://www.uddi.org/specification.html

[28] **UDDI Version 2.0 Programmer's API Specification.**
http://www.uddi.org/specification.html

[29] **UDDI Version 2.0 Operator's Specification .**
http://www.uddi.org/specification.html

[30] **UDDI Technical White Paper.**
http://www.uddi.org/whitepapers.html

[31] **Using WSDL in a UDDI Registry 1.05.**
http://www.uddi.org/bestpractices.html

[32] **Dario de Judicibus, "UDDI 2.0 Parte I: Strutture Dati", Internet New 10, Ottobre 2001.**

[33] **ISO/IEC 11578:1996 standard.**

[33] **Dario de Judicibus, "UDDI 2.0 Parte II: le API", Internet New 11, Novembre 2001.**

[34] **North American Industry Classification Sistem (NAICS)** ***http://eccma.org/unspsc/***

[35] **Universal Standard Products and Services Classification (UNSPSC).** ***http://eccma.org/unspsc/***

[36] **ISO 3166 Geographic Taxonomy.** ***http://dmoz.org/regionaltaxonomy.html***

[37] **ISO/IEC 11578:1996 standard.** ***www.iso.ch***

[38] **Wayne C. Lim, "What is Software Reuse?", Flashline.** ***http://www.flashline.com/services/***

[39] **Roger S. Pressman, "Ingegneria del Software", McGraw-Hill, II edizione.**

[40] **Enterprise Reuse Solution (ERS), ComponentSource.** ***http://www.componentsource.com/EnterpriseReuse/***

[41] **The Jakarta Project, Tomcat 3.2**
http://jakarta.apache.org/tomcat/index.html

[42] **Apache SOAP 2.1**
http://xml.apache.org/soap

[43] **pUDDIng 2.0 (beta)**
http://www.opensorcerer.org/

[44] **Java 2 SDK, Standard Edition Version 1.3.1**
http://java.sun.com/j2se/1.3/

[45] **JavaBeans Activation Framework.**
http://java.sun.com/products/javabeans/glasgow/jaf.html

[46] **JavaMail.**
http://java.sun.com/products/javamail/

[47] **Axis Apache.**
http://xml.apache.org/axis

[48] **Enterprise Java Bean.**

[49] **S.Horstmann Cay, Cornell Gary, "Java2 Tecniche Avanzate ", McGraw-Hill.**

[50] **Java Reflection.**
http://developer.java.sun.com/developer/technicalArticles/ALT/Reflection/

[51] **Bean Scripting Framework (BSF).**
http://java.sun.com/products/beans/glasgow/jaf.html

[52] **IBM WSTK 2.3**
http://www6.software.ibm.com/dl/wstk

[53] **Web Services Framework for W3C Workshop on Web Services 11-12 April 2001, San Jose, CA USA**
http://www.w3.org/2001/03/wsws-popa/paper51

[54] **Web Services Inspection Language (WS-Inspection).**
http://msdn.microsoft.com/ws/2001/10/Inspection/

[55] **Web Services License Language (WS-License).**
http://msdn.microsoft.com/ws/2001/10/Licence/

[56] **Web Services Referral Language (WS-Referral).**
http://msdn.microsoft.com/ws/2001/10/Referral/

[57] **Web Services Routing Language (WS-Routing).**
http://msdn.microsoft.com/ws/2001/10/Routing/

[58] **Web Services Security Language (WS-Security).**
http://msdn.microsoft.com/ws/2001/10/Secutity/