Alma Mater Studiorum - Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in Informatica (1674)

Materia di tesi: Interazione Uomo-Macchina

SOAP vs. REST
in un contesto di Informatica Gestionale

Tesi di Laurea di:
MELISSA POLI

Relatore: Chiar.mo Prof.
FABIO VITALI

Sessione II

Anno Accademico 2011 – 2012

Un vincitore è un sognatore

che non si è mai arreso.

(Nelson Mandela)

Ai miei fantastici genitori

ABSTRACT

Il web è cambiato!

In questi ultimi anni internet ha subito grandi cambiamenti, è passato da un web formato da sole pagine ad un web fornitore di servizi.

Questi servizi chiamati 'Web Service' vengono utilizzati ampiamente per sviluppare la maggior parte delle applicazioni web.
A questo proposito, viene presentata brevemente la tecnologia dei Web Service, per passare successivamente ad analizzare due tipi di standard: quelli basati su SOAP con disegni strettamente accoppiati simili a chiamate di procedura e quelli basati su REST con disegni debolmente accoppiati, simili alla navigazione di link.

Questa tesi si pone l'obiettivo di fornire informazioni sulle caratteristiche, l'interoperabilità, le differenze e l'implementazione dei due standard, descrivendo l'utilizzo di queste architetture in un contesto di informatica gestionale, nello specifico, lo scenario di Infor ERP LN.

INDICE

ELENCO DELLE FIGURE

1. INTRODUZIONE

Il Web è sempre più presente nella nostra vita quotidiana, è diventato quasi uno strumento fondamentale. La sua evoluzione è stata aiutata dall'utilizzo di strumenti tecnologici che hanno messo a disposizione sempre più facilmente la reperibilità delle informazioni in esso contenuto.

Ad oggi si parla di Web 2.0 che ha, come caratteristica principale, la trasformazione dell'interazione tra uomo-macchina, introducendo il cambiamento dell'approccio con cui gli utenti si rivolgono al Web.

Si passa fondamentalmente dalla semplice consultazione alla possibilità di contribuire con propri contenuti, spesso creati in cooperazione tra essi. Questa cooperazione è orientata allo scambio di file ed alle chiamate a procedure remote.

Questo approccio ha facilitato lo svolgere di determinate operazioni comuni, come quelle di effettuare acquisti di qualsiasi tipo e consultare servizi informativi di uso generale.

Una tecnologia che sta prendendo piede in questi anni, segnatamente orientata ad affrontare le nuove frontiere del Web, è quella orientata ai servizi. [12]

> Per **servizio** si intende "una risorsa astratta che rappresenta la capacità di eseguire dei compiti che formano un insieme coerente di funzionalità dal punto di vista del fornitore e dell'utilizzatore del servizio".

> Per **Web Service** "WS" si intende "l'implementazione software di un servizio". I servizi Web sono una tecnologia basata su XML e sono progettati per supportare l'interoperabilità e l'interazione tra macchine attraverso una rete.

La tecnologia Web Services, nel documento chiamata semplicemente "servizi" o "servizi Web" e l'utilizzo di architetture orientate ai servizi (SOA) stanno ottenendo un enorme successo.

Questo nuovo approccio alla rete Web consente di erogare informazioni disaccoppiate rispetto alla rappresentazione grafica. I servizi sono dinamici in quanto l'esito dell'elaborazione scaturita dalla richiesta può risultare diversa in base al contesto e/o a determinati criteri.

L'uso comune di Web Service, come descritto nel prossimo capitolo, consente di far comunicare sistemi diversi tra loro, rendendo l'interazione indipendente dalla tecnologia e dalla piattaforma utilizzata. La caratteristica fondamentale di un Web Service è quella di offrire un'interfaccia software che consente alle applicazioni che vi si collegano di usufruire delle funzioni che mette a disposizione.

I punti di forza sono: la possibilità di creare servizi di supporto al normale Web, la veloce progettazione ed implementazione e la possibilità di fornire solo Server (Client di terze parti). D'altro lato emergono anche degli svantaggi, ad esempio l'utilizzo del protocollo HTTP per trasportare altri protocolli più complessi, la trasmissione dei dati diviene più lunga sprecando banda e non sempre si garantisce la compatibilità con tutti i framework di sviluppo.

All'interno del mondo Internet esistono molti servizi, ma la loro eterogeneità rispetto alla realizzazione ed una scarsa descrizione portano ad una difficile comprensione dell'utilizzo da parte dell'utente e rende ancora più complessa la comunicazione e lo scambio dei dati. Proprio per questo, come sottolineato nello stato dell'arte del documento, vengono messi a confronto due dei principali approcci per l'interfacciamento con il Web, che pur lavorano sullo stesso dominio di informazioni, presentano molte diversità, anche se la creazione si basa sull'utilizzo di linguaggi standard che ne permette una descrizione circostanziata.

Quello basato su SOAP che risulta concettualmente più difficile ed ha una curva di apprendimento più ripida mentre quello basato su REST che risulta più intuitivo, più leggero ed ha una curva di apprendimento più lieve. Come illustrato nella successione dei capitoli la differenza principale è data dall'universo sul quale operano. SOAP viene definito come un universo di messaggi trasportati, mentre REST come un

universo di informazioni accessibili. Da qui risulta intuitivo che la semplicità e la fruibilità delle informazioni siano le caratteristiche vincenti di una architettura REST.
Per contro SOAP ha il vantaggio di essere il servizio il più sperimentato e utilizzato sfruttando tecnologie Web già conosciute.

Al fine di verificare la loro efficienza nel tempo e le loro prestazioni è stato preso in esame uno scenario specifico; un sistema di gestione descritto mediante ERP che offre specifiche soluzioni per il business alle aziende orientate all'innovazione, introducendo una maggiore dinamicità nello scambio di informazioni all'interno di processi aziendali quali essi siano: vendite, acquisti, gestione magazzino, contabilità.
Analizzando lo scenario descritto nel terzo capitolo, nel mettere a confronto le due architetture si sono ottenuti risultati completamente diversi, anche se le aspettative erano largamente diverse dalle conclusioni riportate.
L'utilizzo dei Web Service che siano SOAP o REST è sicuramente la via più semplice per l'interfacciamento tra Client e Server ma risulta necessario studiare a fondo i casi specifici di utilizzo poiché il fallimento è un possibile risultato ottenibile.

Il documento, dopo una prima introduzione dove è possibile conoscere l'argomento che andrà a prendere forma nello svolgersi della tesi.
Si articola in questo modo:
Nel capitolo 2 verrà presentato il concetto di servizio Web, soffermandosi sulle caratteristiche e sulla descrizione delle due tipologie evidenziate SOAP e REST, esponendo architettura, vantaggi e problematiche.
Nel capitolo 3 si introduce l'ambiente in cui verranno comparate le due architetture.
Nel capitolo 4 verrà descritto nei particolari un caso d'uso per la realizzazione dell'obiettivo illustrato nel documento, tale per cui si confronta l'implementazione delle due tipologie di architettura.
Concludendo si terranno alcune considerazioni personali e alcuni spunti futuri sull'approccio esposto.

2. STATO DELL'ARTE

2.1 Web Service: Architettura Service-Oriented

L'architettura dei Web Service è stata pensata per lo sviluppo e l'integrazione di applicazioni distribuite, supportando l'interoperabilità tra diversi elaboratori su di una medesima rete, basando la comunicazione tra i diversi soggetti su un linguaggio comune.
Questa architettura ha spostato la concezione del web dal modello centrato sull'utente, verso un modello *application-centric*, dove la comunicazione avviene direttamente tra applicazioni, portando così non solo ad uno scambio di informazioni, ma anche ad una condivisione di risorse computazionali che vengono utilizzate a richiesta.
Questo modello è rappresentato da una architettura denominata SOA (Service-Oriented Architecture)[1] e la sua implementazione più diffusa ad oggi include tutte quelle tecnologie che vengono classificate come Web Service.
I Web Services sono delle applicazioni Web che cooperano fra loro, indipendentemente dalla piattaforma sulla quale si trovano, attraverso lo scambio di messaggi. Ognuna di queste applicazioni viene chiamato Web Service (Servizio Web), o più semplicemente servizio, del quale il Web Services Architecture Working Group (del W3C) dà la seguente definizione [2]:

> *"Un Web Service e un'applicazione software identificata da un URI (Uniform Resource Identifier), le cui interfacce pubbliche e collegamenti sono definiti e descritti come documenti XML, in un formato comprensibile alla macchina (specificatamente WSDL). La sua definizione può essere ricercata da altri agenti software situati su una rete, i quali possono interagire direttamente con il Web Service, con le modalità specificate nella sua definizione,*

utilizzando messaggi basati su XML (SOAP), scambiati attraverso protocolli Internet (tipicamente HTTP).”

Va altresì sottolineato che il ruolo dei Web Service non si limita solo a un ambito di interoperabilità, bensì aiuta a superare la limitazione della “dipendenza” permettendo di descrivere nuovi servizi realizzati ad hoc, sempre però con l’intento di fornire una soluzione *platform-independent*.
In primo luogo va sottolineato la separazione tra linguaggio per descrivere il contenuto informativo ed uno per descrivere la presentazione.
I Web Service, infatti, si occupano della logica applicativa e non si preoccupano di descrivere l’interfaccia. Il compito del fruitore del servizio è di presentare i dati ottenuti servendosi di stili e grafica propri spostando il focus esclusivamente sul contenuto.

I vantaggi offerti dai Web Services sono:

- *Indipendenza dalla piattaforma*: i Web Services possono comunicare fra loro anche se si trovano su piattaforme differenti.

- *Indipendenza dall’implementazione del servizio*: l’interfaccia che un Web Service presenta sulla rete è indipendente dal software che implementa tale servizio. In futuro tale implementazione potrà essere sostituita o migliorata senza che l’interfaccia subisca modifiche e quindi senza che dall’esterno (da parte di altri utenti o servizi sulla rete) si noti il cambiamento.

- *Riuso dell’infrastruttura*: per lo scambio di messaggi si utilizza SOAP che fa uso di HTTP, grazie al quale si ottiene anche il vantaggio di permettere ai messaggi SOAP di passare attraverso sistemi di filtraggio del traffico di rete, quali i Firewall.

- *Riuso del software*: è possibile riutilizzare software implementato precedentemente e renderlo disponibile attraverso la rete. Il concetto di Web Services implica un modello di architettura ad

oggetti distribuiti (oggetti intesi come applicazioni), che si trovano localizzati in punti diversi della rete.

I pregi sopra elencati sono i punti di forza dei servizi web, ma essi portano con se anche qualche limitazione.

Una delle critiche principali che viene fatta a questa tecnologia è la scarsità di meccanismi di sicurezza.[18] L'utilizzo di HTTP come protocollo di trasporto infatti non permette di criptare la comunicazione e i messaggi scambiati tra fornitore e cliente vengono trasmessi in chiaro, quindi facilmente intercettabili da terzi.
Inoltre l'utilizzo di un servizio web all'interno di un'applicazione può portare ad una riduzione delle performance causata da diversi fattori.
Come per tutti i sistemi che utilizzano la rete come mezzo di comunicazione si deve tenere in considerazione il ritardo (*delay*) e i problemi che questa introduce, purtroppo essi non sono controllabili e variano a seconda delle condizioni del traffico (*jitter*) con l'arrivo a tempi differenti dei pacchetti. Infine è da tenere in considerazione il fatto che l'utilizzo di XML come standard per la costruzione di messaggi è ottimo per quanto riguarda la flessibilità e la manutenibilità, ma per contro la verbosità del linguaggio, riduce la velocità di comunicazione ed elaborazione dei servizi.

Prima di addentrarci nei dettagli tecnici e inquadrare meglio lo spazio d'azione, si vuole dare una definizione di più ampio respiro di Web Service.
A tal proposito, si partirà da una architettura di riferimento, la Service Oriented Architecture (SOA) grazie alla quale è possibile identificare gli scopi, gli utilizzi e gli sviluppi futuri che caratterizzano i Web Service. Focalizzando l'attenzione sul concetto di servizio è ovvio immaginare come gli attori in causa siano necessariamente il fornitore e il fruitore. Questo tipo di paradigma è il medesimo che si riscontra nella tipica interazione *client-server*. Attraverso la SOA questa interazione viene arricchita con un ulteriore attore detto *Service Registry* che, come mostrato in Figura:2-1, si inserisce all'interno della comunicazione tra

fornitore *Service Provider*, ovvero chi si occupa della pubblicazione del servizio con le specifiche per l'interrogazione, visibile a chiunque voglia utilizzarlo e fruitore del servizio *Service Requestor*. Per implementare quest'ultimo si utilizza la risorsa messa a disposizione dal Provider, interrogando l'application server. L'output ottenuto fornisce il layout grafico all'interno dell'applicazione richiesta e tramite il *Service Registry* consente di ricercare un servizio sulla base di caratteristiche con le quali è stato definito e memorizzato. Naturalmente esso può seguire politiche di controllo degli accessi sulle interrogazioni, in modo da limitare la visibilità sui servizi inseriti.

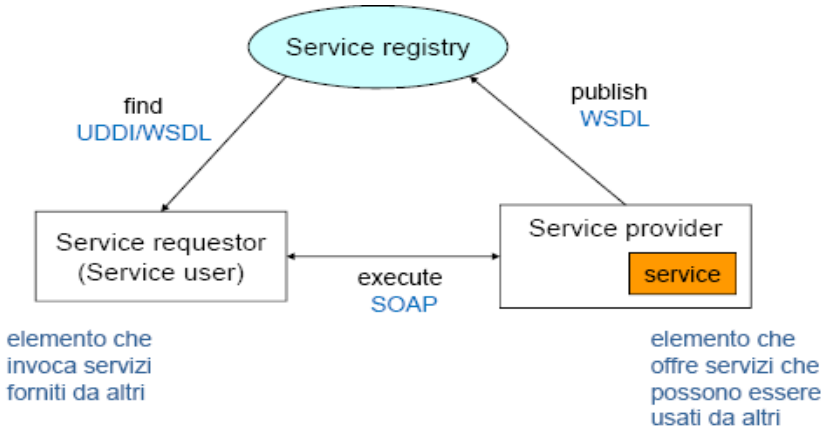


Figura 2-1: Modello Web Service

Va notato che i tre attori interessati possono essere distribuiti ed utilizzare piattaforme tecnologiche differenti, con l'unico vincolo di servirsi di un canale trasmissivo comune. L'approccio adottato dalle SOA ha il vantaggio di potersi integrare con diversi ambienti, permettendo in tal modo di realizzare applicazioni multi-canale fruibili tramite dispositivi diversi. Questo avviene mediante uno scambio di informazioni descritte su un formato XML.[3] (Figura: 2-2)

Questo formato è indipendente dalle varie piattaforme e ciò è dovuto, oltre che all'essere universalmente riconosciuto come standard, al fatto che tale tecnologia si basa sul formato testo e quindi un documento XML può essere letto con facilità su qualsiasi sistema operativo. Il contenuto di

un documento XML è costituito da marcatori e dati strutturati secondo un ordine logico determinato da una struttura ad albero.
Il compito di un documento XML è memorizzare i dati all'interno di una struttura gerarchica che rappresenti le relazioni esistenti fra di essi, senza curarsi minimamente della loro rappresentazione visuale.

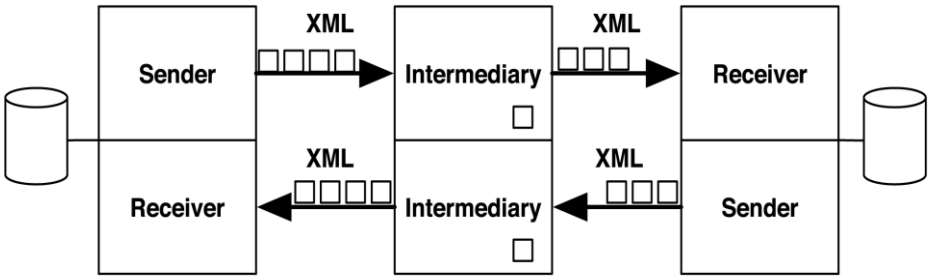


Figura 2-2: Comunicazione basata su Xml

Oltre a questo linguaggio è utile soffermarci a descrive un altro linguaggio, basato su XML, utilizzato per descrivere un Web service, chiamato WSDL, ovvero Web Services Description Language [4]. Più precisamente un documento WSDL fornisce informazioni riguardanti l'interfaccia del Web Service in termini di:

- Servizi offerti dal Web Service,
- URL ad esso associato,
- Modi per l'invocazione,
- Argomenti accettati in ingresso e modalità con cui devono essere passati,
- Formato dei risultati restituiti,
- Formato dei messaggi.

In altre parole si può dire che un file WSDL contiene, relativamente al Web Service descritto, informazioni su:

- *cosa* può essere utilizzato, le "operazioni" messe a disposizione dal servizio;
- *come* utilizzarlo, il protocollo di comunicazione da utilizzare per accedere al servizio, il formato dei messaggi accettati in input e

restituiti in output dal servizio ed i dati correlati, più precisamente i vincoli del servizio;

- *dove* utilizzare il servizio, il cosiddetto *endpoint* del servizio che solitamente corrisponde all'indirizzo - in formato URI - che rende disponibile il Web Service.

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

2.1.1 WSDL – Linguaggio descrittivo

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

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

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/servi"
          xmlns:tns="http://example.com/stockquote/service"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns:defs="http://example.com/stockquote/definitions"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

 <import namespace="http://example.com/stockquote/definitions"
         location="http://example.com/stockquote/stockquote.wsdl"/>

<binding name="StockQuoteSoapBinding"type="defs:StockQuotePortType">
    <soap:binding    style="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>
      <service name="StockQuoteService">
          <documentation>My first service</documentation>
          <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding"
          name=”StockQuoteSoapPort”>
                <soap:address location="http://example.com/stockquote"/>
          </port>
       </service>
</definitions>
```

Il componente fondamentale di un file WSDL è l'elemento <*service*> che identifica un insieme logico di servizi che possono essere più di uno. Ogni servizio specificato, viene identificato dal nome e il tag opzionale <*documentation*> permette di specificare una breve descrizione del servizio. All'interno del tag <*service*> viene specificato l'indirizzo su cui risponde il servizio mediante il tag <*soap:address*> il quale si trova all'interno del tag <*port*> e identifica, mediante l'attributo 'binding' le caratteristiche del servizio, specificatamente, identifica il protocollo di trasporto (SOAP, HTTP o SMTP), il nome dell'operazione che deve essere specificato all'intero del messaggio SOAP e la sintassi dei messaggi di input e output che possono essere definiti all'interno di questo tag o mediante il tag <*portType*>.

Il tag <*portType*> il cui compito è quello di specificare la reale sintassi del servizio, ha, quindi, il ruolo di dire cosa il servizio fa, mentre <*port*> come il servizio possa essere acceduto.

Con <*portType*> si caratterizza la tipologia di operazione, infatti, vengono identificate quattro diverse classi di operazioni (Fig. 2-3).

1. *One_Way*: l'operazione è composta da un solo messaggio in ingresso al fornitore del servizio.
2. *Request_Response*: l'operazione prevede una risposta del fornitore del servizio successiva ad un messaggio ricevuto dall'utente.

3. *Solicit_Response*: l'operazione prevede l'attesa da parte del fornitore del servizio, di una risposta a fronte di una richiesta effettuata dal fornitore stesso.
4. *Notification*: l'operazione è composta da un solo messaggio in uscita al fornitore del servizio.

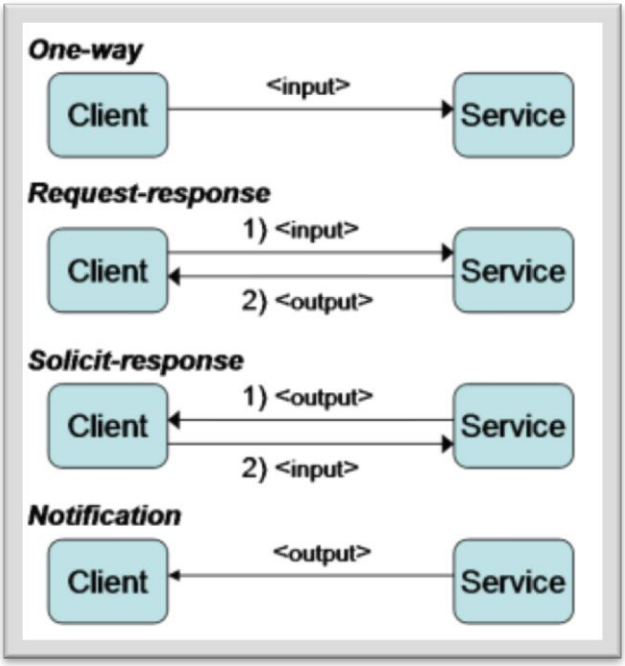


Figura 2-3: Tipologia di Operazioni

Riferendosi, quindi, all'esempio, viene specificato un solo servizio (StockQuoteService) così come appare nel tag <port>. Questo servizio è accessibile all'indirizzo ' http://example.com/stockquote' e per invocarlo è necessario che il client inoltri le proprie richieste secondo il protocollo SOAP.

Al di là delle specifiche sintattiche di WSDL, soffermandosi ulteriormente sulla relazione che esiste tra <port> e <portType> è possibile dire che una <port> può essere vista come una specializzazione di una <portType> operata secondo un particolare protocollo di comunicazione. È quindi possibile che all'interno del medesimo file WSDL la stessa <portType>, quindi il servizio, possa essere reso accessibile, grazie ai '*binding'*, su diversi protocolli di trasporto.

Esistono molte implementazioni tecnologiche che rispecchiano i concetti dell'architettura orientata ai servizi, ma non tutte riescono ad esprimere il concetto fondamentale di implementare entità che descrivano se stesse, tranne la tecnologia che sfrutta il protocollo chiamato SOAP (Simple Object Access Protocol) e definito per assunto come il "Web Service".

Quando ci si riferisce ad un Web Service, infatti si fa subito riferimento al protocollo SOAP e alla potenza del suo linguaggio di definizione.

Oggi grazie alla diffusione del Web 2.0, quando si parla di Servizio Web si intende un sistema software progettato per supportare l'interoperabilità tra sistemi e per erogare contenuti. Per ragioni di comodità e altri fattori che analizzeremo, oltre al classico SOAP, si è diffuso l'utilizzo di un'altra architettura denominata REST (REpresentational State Transfer). Ed è sul confronto tra questi due servizi che si concentra il lavoro descritto in questo documento.

2.2. In Principio era SOAP

SOAP (Simple Object Access Protocol)[5] è un protocollo leggero che fornisce un meccanismo semplice per lo scambio di messaggi tra componenti software, tramite XML. La parola Object contenuta nell'acronimo, manifesta che l'uso del protocollo dovrebbe effettuarsi secondo il paradigma della programmazione orientata agli oggetti, infatti SOAP descrive un meccanismo semplice per esprimere la semantica dell'applicazioni fornendo dei meccanismi per la codifica dei dati all'interno dell'entità coinvolte. SOAP è una struttura operativa (framework) che può operare sopra varie pile protocollari e le chiamate alle procedure remote possono essere modellizzate come uno scambio di messaggi SOAP.

Le specifiche di SOAP contengono in realtà quattro diversi aspetti:

1) la definizione del messaggio vero e proprio ed i modelli di scambio;

2) la “codifica SOAP”, uno standard per strutturare le informazioni articolate in XML;
3) le regole per utilizzare SOAP su http;
4) le regole per utilizzare SOAP per eseguire chiamate remote a procedure (RPC).

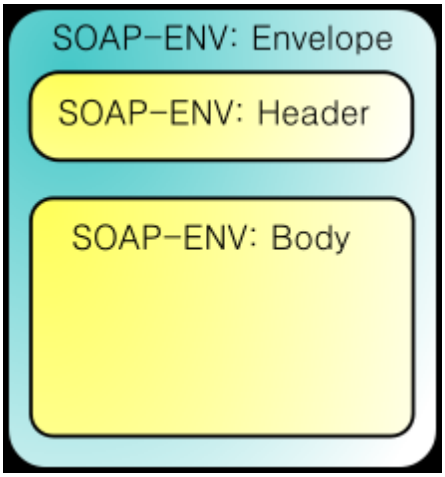


Figura 2-4: La struttura di un messaggio SOAP

Utile vedere nel dettaglio la struttura e il funzionamento per capire a fondo i WS.

SOAP è costituita da quattro componenti principali (Figura 2-4) :

- “SOAP Envelope” è l'elemento principale, rappresenta la busta del messaggio che funge da raccoglitore di tutti gli altri elementi. include due attributi caratteristici del documento. Il primo rappresenta il namespace relativo al messaggio in modo da distinguere gli elementi appartenenti allo standard SOAP, Il secondo attributo soap:encodingStyle specifica regole per la serializzazione dei dati, può essere applicato a qualsiasi elemento, e tutti i figli dell'elemento che specifica tale attributo, lo ereditano. Questo definisce il documento XML come un messaggio SOAP. Esprime ciò che si trova nel messaggio e che cosa tratta.

- “SOAP Header” è un elemento che contiene le informazioni specifiche dell’applicazione. E’ opzionale, il suo scopo è

quello di trasportare informazioni non facenti parte del messaggio, destinate agli 'attori', cioè alle varie parti che il messaggio attraverserà per arrivare al suo destinatario finale, come dati sull'autentificazione o sulla transazione distribuita in corso. Viene anche utilizzato per definire messaggi con diversi destinatari nel caso il messaggio dovesse attraversare più punti di arrivo.

- "SOAP Body" è un elemento indispensabile che è il vero deputato a contenere le informazioni applicative che è necessario comunicare al ricevente

Inoltre:

- "SOAP Fault" è un elemento anch'esso opzionale, che fornisce informazioni riguardo eventuali errori manifestati durante la lettura del messaggio.

La busta del messaggio SOAP così strutturata viaggia su un canale di trasmissione (Figura 2-5) utilizzato per invocare la procedura, proponendo come standard l'utilizzo del protocollo HTTP [6], senza tuttavia precludere l'utilizzo di altri protocolli, a patto che possano trasportare testo.

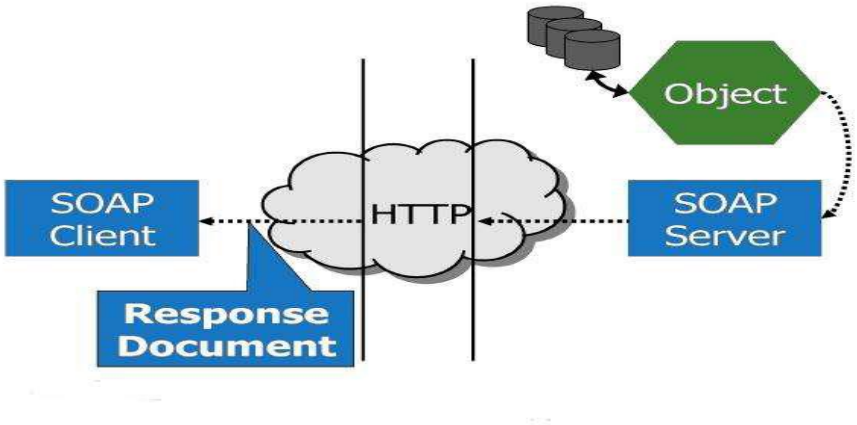


Figura 2-5: Trasmissione SOAP – HTTP

SOAP quindi, stabilisce la conformazione dei singoli messaggi che vengono scambiati tra i nodi, definendo una serie di tag XML specifici. Utilizzando come riferimento questi elementi, chi riceve il messaggio è in grado di conoscere informazioni importanti sul messaggio stesso; nella forma più semplice, la struttura di un messaggio SOAP consente di mettere ordine e trasmettere le informazioni in modo strutturato. Prendiamo ad esempio lo scenario della quotazione della borsa, richiamiamo il servizio 'GetStockQuote' che inserita l'azione 'IBM' ci restituisce la quotazione.

The request:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
  xmlns:show=" http://hostbridge.com/stock-service">
  <soapenv:Body>
    <show:GetStockQuote>
        <show:TickerSymbol>IBM</ show:TickerSymbol>
    </show:GetStockQuote>
  </soapenv:Body>
</ soapenv:Envelope>
```

Il messaggio creato, viene spedito su un canale di trasmissione che permette di raggiungere l'applicazione da interrogare e la risposta che otterremo sarà simile a quella riportata qui sotto:

The response:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
  xmlns:show=" http://hostbridge.com/stock-service">
  <soapenv:Body>
    <show:GetStockQuoteResponse>
        <show:StockPrice>45.265</show:StockPrice>
    </show:GetStockQuoteResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Come si nota nei due messaggi scambiati, il contenuto dell'elemento <soap:Body> non è descritto nelle specifiche del protocollo SOAP, ma è definito dall'applicazione.

Abbiamo detto che la busta SOAP viaggia su un canale di trasmissione, ma non abbiamo definito nulla su di esso; questa è un'ulteriore caratteristica dei Web Service, infatti tutti i concetti finora espressi, rimangono ancora validi anche se il protocollo di trasmissione utilizzato per invocare la procedura è diverso. Principalmente, i canali trasmissivi maggiormente usati, sono gli stessi diffusi nel Web.

2.3 REST (REpresentational State Transfer)

REST (REpresentational State Transfer) è stato coniato nel 2000 nella tesi di dottorato "Architectural Styles and the Design of Network-based Software Architectures" [7] di Roy Fielding, uno dei principali autori delle specifiche dell' Hypertext Transfer Protocol (HTTP).
REST rappresenta uno stile architetturale, ovvero un insieme di principi e metodi di progettazione i quali delineano come le risorse sono definite e indirizzate.
E' importante notare che non si riferisce ad un sistema concreto e ben definito né si tratta di uno standard stabilito da un organismo di standardizzazione. Fielding la definisce come:

> *'Representational State Transfer is intended to evoke an image of how a welldesigned Web application behaves: a network of web pages (a virtual statemachine),where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.'* [14]

Un'applicazione che rispetta completamente le restrizioni REST è generalmente indicata come RESTFul.
Lo stile architettonico REST fu sviluppato in parallelo con l'HTTP basandosi su di un progetto esistente. La larga implementazione di un sistema conforme allo stile architetturale di REST è il WWW.

REST semplifica la caratterizzazione architetturale del Web rispettando le macro-interazioni delle quattro componenti del Web, ovvero i server di origine, i gateway, i proxy ed i client, senza però imporre limiti ai singoli elementi. In definitiva, l'architettura REST regola il corretto comportamento dei partecipanti.

La Filosofia REST evocata da Roy Fielding si basa su questi punti chiave:

- Lo stato dell'applicazione è suddiviso in risorse
- Le risorse sono indirizzabili utilizzando una sintassi universale
- Esiste un'interfaccia uniforme per il trasferimento di stato racchiuso in un set di operazioni
- Il protocollo di trasferimento è *client/server*, *stateless*, *cacheable*, and *layered*. (senza stato, memorizzabile nella cache, e stratificato.)
- URI per l'accessibilità e HTTP per lo stato di trasferimento

I concetti di architettura software e stile che caratterizzano il design pattern denominato REST sono come riportate in figura 2-6:

I vincoli REST	
I	Client / Server
II	Comunicazione stateless
III	Cacheable
IV	Organizzato a livelli
V	Codice a richiesta (opzionale)
VI	Interfaccia uniforme

Figura 2-6: Vincoli REST

I. *Client-Server*:

un’interfaccia separa i Client dai Server, questa separazione di rapporti significa che vengono separate le competenze, in tal modo migliora la portabilità e la scalabilità.

II. *Stateless*:

Ogni ciclo di request/response deve rappresentare un'interazione completa tra client e server. Ogni nuova richiesta è indipendente da quelle precedenti, pertanto deve contenere tutte le informazioni necessarie per la gestione. In questo modo non è necessario mantenere informazioni sulla sessione utente, minimizzando l'uso di memoria del server e la sua complessità. Rafforza la scalabilità, l’affidabilità e la visibilità.

III. *Caching*:

i client possono memorizzare in cache le risposte, questo permette di eliminare parzialmente o completamente alcune interazione tra Client e Server, migliorando la scalabilità e performance

IV. *Code on-demand*:

un componente Client ha accesso a un insieme di risorse, ma non al know-how su come elaborarle. Invia una richiesta a un Server remoto per il codice che rappresenta il know-how, ricevuto il codice viene eseguito in locale. Questo semplifica il lavoro dei Client e aumenta le possibilità di estensione.

V. *Uniform interface*:

ogni risorsa deve avere un indirizzo univoco e ogni risorsa di ogni sistema presenta la stessa interfaccia, precisamente quella individuata dal protocollo HTTP. Semplifica l’architettura e fa sì che ogni parte si evolva indipendentemente.

VI. *Layered system* (Sistemi a più livelli):

Un Client non può, normalmente, dire se è connesso direttamente al Server oppure a uno stato intermedio. I Server intermedi sono in grado di migliorare la scalabilità del sistema rendendo il carico bilanciato e

provvedendo alla condivisione della cache. Questo inoltre rafforza le politiche di sicurezza.

Bisogna affrontare un piccolo approfondimento sul concetto di "*stateless*", questo vincolo deriva direttamente dalla parte di acronimo ST che sta per State Transfer, infatti ad ogni interazione, devono essere trasmesse tutte le informazioni necessarie al Server per elaborare la richiesta senza controllare l'history della comunicazione.
Questo atteggiamento compromette le performance portando ad un degrado delle prestazioni a causa del maggior numero di dati da presentare, ma permette di introdurre la terza caratteristica elencata, il "*caching*", quindi di poter sfruttare tecnologie proxy tra il canale trasmissivo migliorando la scalabilità, facendo attenzione ad attuare dei meccanismi robusti nel controllo delle informazioni memorizzate dato che possono essere non aggiornate.

Un sistema che rispetta questi vincoli viene definito RESTful.

Usando questa architettura si sposta la complessità dal protocollo alla rappresentazione astratta dello stato di una risorsa, naturalmente non è necessario trasmettere tutto lo stato della risorsa ma solo quelle porzioni di stato che interessano. Da specificare è che nell'architettura Web i singoli servizi sono considerati "*risorse*", individuate da URI (Uniform Resource Identifier) [8], tramite il protocollo HTTP che è fornito di un unico schema d'indirizzamento. Un URI è un identificatore univoco di una particolare risorsa in rete, come ad esempio un documento HTML, un'immagine, un video oppure un servizio web. Un particolare URI è l'URL (Uniform Resource Locator) [9] che fornisce maggiori informazioni sulla risorsa referenziata, ovvero dove la stessa si trova fisicamente ed il suo tipo di dato.
Questi servizi vengono definiti, come implementazione di architetture ROA, ovvero "Architettura Orientata alla Risorsa"[10].

Risorsa

La Risorsa (*resource*) è l'elemento chiave di un design RESTful, in opposizione ai "*metodi*" ed ai "*servizi*" rispettivamente usati nelle richieste SOAP dei servizi WEB.
Una risorsa è qualunque entità che possa essere indirizzabile tramite Web, cioè accessibile e trasferibile tra Client e Server. Spesso rappresenta un oggetto appartenente al dominio del problema che stiamo trattando.
Per descrivere una risorsa, bisogna individuare tre elementi che si possono identificare con un triangolo come mostrato in Figura 2.7.

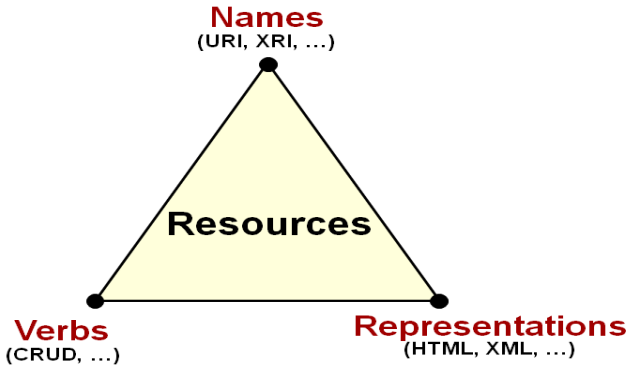


Figura 2-7: Il triangolo delle Risorse

Names : è l'identificativo univoco della risorsa, permette di identificarla in maniera non ambigua e di raggiungerla da qualsiasi altro servizio. Il nome può essere visto come l'indirizzo della risorsa e quindi la possibilità di dare alla risorsa la proprietà di essere indirizzabile. All'interno di REST questa viene definita mediante l'URI (Uniform Resource Identifier)
Un Client che usufruisce del servizio REST, manipola la risorsa connettendosi al Server che la ospita e ne invia il percorso per raggiungerla. L' URI deve essere descrittiva, e deve essere ben strutturata.

Verbs: Abbiamo descritto la risorsa con un nome univoco, ma la cosa più importante che un servizio web mette a disposizione, sono le operazioni

che consentono di manipolare le informazioni. Oltre a digitare gli URI per accedere alle risorse, è possibile interagire con esse per mezzo di azioni che vengono messe a disposizione del protocollo di comunicazione. Il protocollo di trasmissione che viene utilizzato dai servizi REST è il protocollo standard HTTP. I metodi utilizzati (Figura 2-8) sono quelli minimi messi a disposizione GET, POST, PUT e DELETE.
Queste operazioni hanno analogia con le quattro operazioni base utilizzate nei sistemi di storage persistenti, identificate con l'acronimo CRUD (Create, Retrieve, Update e Delete).

HTTP	CRUD
POST	Create
GET	Retrieve
POST	Update
DELETE	Delete

Figura 2-8: Metodi

Representations: è il linguaggio e la struttura con cui il servizio mostra al destinatario lo stato corrente della risorsa. La maggior parte delle volte, le risorse stesse sono i dati e quindi una "*Rappresentazione*" evidenzia la risorsa stessa. Quando si crea una rappresentazione di una risorsa, è necessario rappresentare qualsiasi informazione di essa, in maniera tale, da mostrane completamente lo stato. La rappresentazione può essere destinata all'interazione umana (ad esempio una pagina HTML) o all'interazione machine-to-machine (ad esempio XML o formati binari), infatti può essere usata anche per richiedere al servizio, di creare una nuova risorsa o di aggiornare quella esistente, inviando una rappresentazione di essa.

Di seguito riporto l'esempio precedente, scritto utilizzando il servizio RESTful, per comprendere meglio l'implementazione e cominciare ad introdurre le differenze tra le due architetture.

Un Provider (HostBridge) fornice il servizio Web:

The Request:

```
GET /Quote/IBM HTTP/1.1
Host: hostBridge.com
Accept: text/xml
Accept-Charset: utf-8
```

Richiedente che utilizza il servizio:

The Response:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<show:Quote xmlns: show="http://hostbridge.com/stock-service">
    <show:TickerSymbol>IBM</ show:TickerSymbol>
    <show:StockPrice>45.25</ show:StockPrice>
</show:Quote>
```

Vediamo come l'URL è stato inviato al server utilizzando una semplice richiesta GET, e la risposta da parte del protocollo HTTP è il campo con i dati con il risultato. I dati della response sono trasmessi in maniera tale da essere subito disponibili per un eventuale altro utilizzo.
Si nota come la versione RESTful è più semplice e più concisa, e come questi servizi sono volutamente più vicini al design e stile del Web stesso.

Un passaggio fondamentale per apprezzare REST appunto è la semplicità caratteristica appena menzionata.
La semplicità è un valore da perseguire e la tecnologia può essere il mezzo per ottenere la semplicità dalla complessità del reale. In "The Laws of Simplicity" [11] John Maeda, professore del MIT, definisce delle linee guida per realizzare la semplicità in un prodotto tecnologico, dalle automobili di lusso all'iPod (Figura. 2-9). Gli stessi principi possono

essere applicati per comprendere lo stile REST e il suo rapporto con la semplicità.

Leggi della semplicità		
I	**Riduci**	Il modo più semplice per conseguire la semplicità è attraverso una riduzione ragionata.
II	**Organizza**	L'organizzazione fa sì che un sistema composto da molti elementi appaia costituito da pochi.
III	**Tempo**	I risparmi di tempo somigliano alla semplicità.
IV	**Impara**	La conoscenza rende tutto più semplice.
V	**Differenze**	La semplicità e la complessità sono necessarie l'una all'altra.
VI	**Contesto**	Ciò che sta alla periferia della semplicità non è assolutamente periferico.
VII	**Emozione**	Meglio emozioni in più che in meno.
VIII	**Fiducia**	Noi crediamo nella semplicità.
IX	**Fallimento**	Ci sono cose che non si possono semplificare.
X	**Unica**	Semplicità significa sottrarre l'ovvio e aggiungere il significato.

Figura 2-9: Le leggi della semplicità

Riduci: REST riduce la complessità restringendo l'insieme dei termini del vocabolario utilizzati per indicare azioni sulle risorse. Nel protocollo HTTP sono definiti pochi verbi e i più usati sono GET, POST, PUT e DELETE. In modo analogo, molti servizi web definiscono di solito solo quattro azioni: CREATE, READ, UPDATE e DELETE. Lo stile non dice come si devono chiamare e quale deve essere il significato di queste azioni. Per essere RESTful occorre soltanto che l'interfaccia sia uniforme. L'aspetto forse sorprendente di un'architettura RESTful è che quei pochi verbi molto spesso sono sufficienti per descrivere il comportamento di un sistema.

Organizza: in linea con la prima legge della semplicità, un'architettura RESTful ha un solo oggetto: la risorsa. Però un sistema composto da tanti elementi uguali può essere complesso da gestire. Quindi è opportuno

organizzare le risorse in tipi di risorse e pensare a un meccanismo semplice e intuitivo per formare gli indirizzi che identificano le risorse.

Tempo: i Client possono tenere traccia delle risposte (*caching*). Un buon meccanismo di caching permette di ridurre le interazioni tra Client e Server con ripercussioni positive sulla scalabilità e sulle performance del sistema.

Impara : una buona architettura RESTful non ha bisogno di essere imparata e non necessita di un corposo manuale di istruzioni. L'utilizzo dei servizi deve essere intuitivo e può essere compreso immediatamente perché sembra familiare.

Differenze: la complessità con la quale abbiamo a che fare quando progettiamo un'applicazione Web è la complessità della realtà. Lo stile REST, per prima cosa, semplifica tutto questo, costringendoci a focalizzarci solo sul concetto di risorsa e riducendo il numero di azioni a un insieme piccolo ma significativo. Grazie al confronto con la realtà, riusciamo ad apprezzare meglio lo stile REST. La semplicità di REST viene evidenziata anche nel confronto con la complessità di altre tecnologie o altre filosofie concorrenti.

Contesto: identificare le risorse di un sistema RESTful è un compito delicato che richiede un certo "gusto" nello scegliere i concetti adatti per raccontare qual'è l'esperienza d'uso che si intende offrire. Il contesto deve trasmettere in modo intuitivo e familiare quello che il sistema vuole e deve essere. In altre parole, guida da lontano la progettazione dei Client che seguono un modello semantico comune implicito nella descrizione delle risorse e delle azioni dell'architettura RESTful. La progettazione di servizi RESTful mette in relazione concetti diversi creando metafore intuitive e di immediata comprensione.

Emozione: "la settima legge non è per tutti: ci sarà sempre il modernista duro e puro che rifiuta ogni oggetto che non sia bianco o nero". C'è emozione nel freddo mondo della programmazione? REST rappresenta le

risorse in un formato logico e formale, ma comprensibile da un essere umano.

Fiducia: " Don’t be evil ", dicono quelli di Google. Non so quanto il motto di un’azienda sia rispettato dall’azienda stessa. Però il semplice motto contiene una verità. Il successo di Google è legato alla fiducia degli utenti per Google. La semplicità è implementata quando un sistema fa quello che ci si aspetta che faccia senza mostrare la complessità delle operazioni necessarie per portare a termine un compito apparentemente semplice come può essere una ricerca sul Web. Un sistema RESTful fa proprio questo, non si sa bene quello che accade dietro le open API (Application Programming Interface). Però a noi sta bene così perché tutto sembra più semplice. Questo comporta però una certa dose di fiducia da parte nostra verso il sistema. Abbiamo fiducia non solo che i nostri desideri vengano soddisfatti senza troppi fronzoli, ma anche che il sistema non faccia del male a noi o ad altri. Attenzione: anche se sembra che si tratti di un tema solo "etico", questo aspetto è invece importante almeno quanto quelli più tecnici o di design. L’economia delle reti è un’economia che si basa sulla simbiosi e sulla fiducia. Gli utenti e il sistema sono vicendevolmente utili gli uni agli altri. Il legame che li unisce è la fiducia.

Fallimento: finora ho solo parlato bene di REST, ma farei un errore se dicessi che REST risolve tutti i problemi. Un’architettura RESTful non è adatta ad ogni situazione. Dipende da quello che si vuole fare e spesso quello che si vuole fare non ha motivazioni tecniche. Per quanto riguarda il tema della semplicità esistono cose che non possono essere semplificate. Realizzare l’interoperabilità tra i sistemi informativi di una o più aziende non è banale. Spesso richiede l’utilizzo di strumenti molto complessi per orchestrare i diversi processi. In questo caso forse REST potrebbe fallire. Quando REST è destinato a fallire? Non lo so e non credo che esista una regola generale. L’unica strategia è quella di provare a progettare e sviluppare un’architettura RESTful.

Unica: l’ultima legge riassume tutte le altre. La decima legge potrebbe anche essere l’unica.

Un sistema è semplice quando fa quello che ci aspettiamo in modo intuitivo. Un sistema di questo tipo lo costruiamo mettendo in relazione concetti e utilizzando metafore di utilizzo. Dobbiamo capire cosa è necessario e cosa possiamo lasciare fuori. Solo così facciamo risaltare in modo intuitivo il significato implicito del sistema. Non bisogna nemmeno credere che REST sia un martello dorato che possiamo usare per risolvere ogni problema. Ci sono cose che non possono essere semplificate e dove lo stile REST può fallire.

2.4 Comparazione SOAP vs REST

In questa sezione vogliamo analizzare le differenze che le due tipologie di servizi hanno, per poter identificare meglio i vantaggi e svantaggi delle due metodologie.

Differenze principali:

SOAP
Il Web è un universo di messaggi trasportati, le applicazioni hanno la possibilità di interagire ma rimangono ‘al di fuori del Web.

REST
Il Web è un universo di informazioni accessibili, le applicazioni sono pubblicate tramite URI.

Un servizio web basato sulla metodologia REST, ha come punto di forza l’utilizzo di un interfaccia che è diffusamente conosciuta: URI. Qualsiasi Client e Server che supporta invocazioni HTTP, può facilmente invocare un servizio REST.
In un servizio SOAP le richieste vengono descritte da un linguaggio chiamato WSDL, quindi questo comporta che per poter usare al meglio un servizio SOAP, si abbia la competenza di saper leggere un file XML che descrive tutte le informazioni necessarie per invocare le operazioni.

Avere una rigida descrizione del servizio, è una caratteristica che in alcuni scenari può essere un vantaggio, come ad esempio la comunicazione di scambio nei grandi Data Center dove le informazioni, per motivi di sicurezza devono essere tipizzate.
Un servizio SOAP, indirizza le richieste sempre verso un unico indirizzo chiamato "*endpoint*" ed all'interno del messaggio vengono definite le operazioni che vengono passate dall'application server ed invocate insieme ai dati sempre dichiarati all'interno del messaggio.

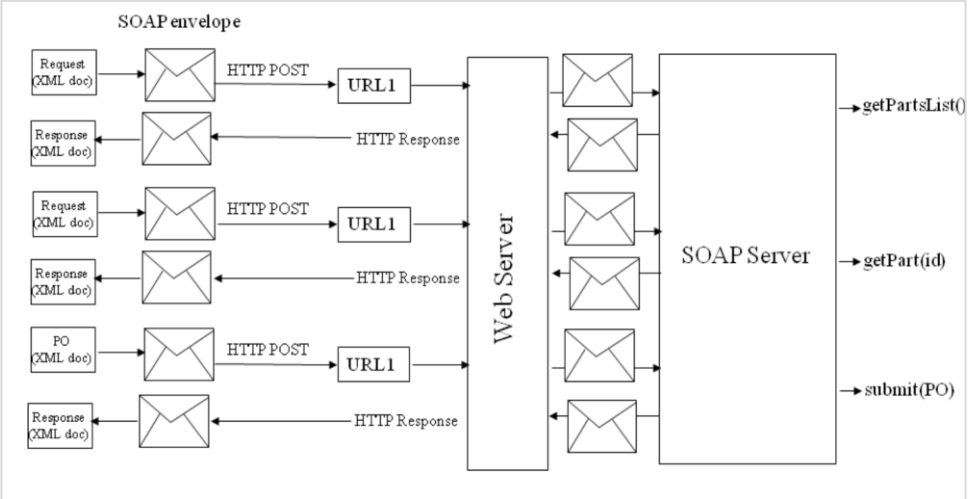


Figura 2-10: Servizio SOAP

Nella figura 2-10 si nota come tutte le richieste vengono effettuate verso un'unica URI (URL1). Tutte le richieste vengono effettuate sempre con il metodo POST e questo porta ad un maggior traffico verso l'endpoint in quanto in tutte le richieste viene sempre inviato un documento XML.
Nei servizi REST le richieste, come detto precedentemente, vengono indirizzate verso URI differenti che si mappano sulle risorse. Il consumo di banda che porta un servizio REST è ridotto al minimo, infatti viene inviato insieme alla richiesta un documento XML o altre informazioni oltre all'URI solo quando bisogna creare o aggiornare lo stato di una risorsa (Figura 2-11).

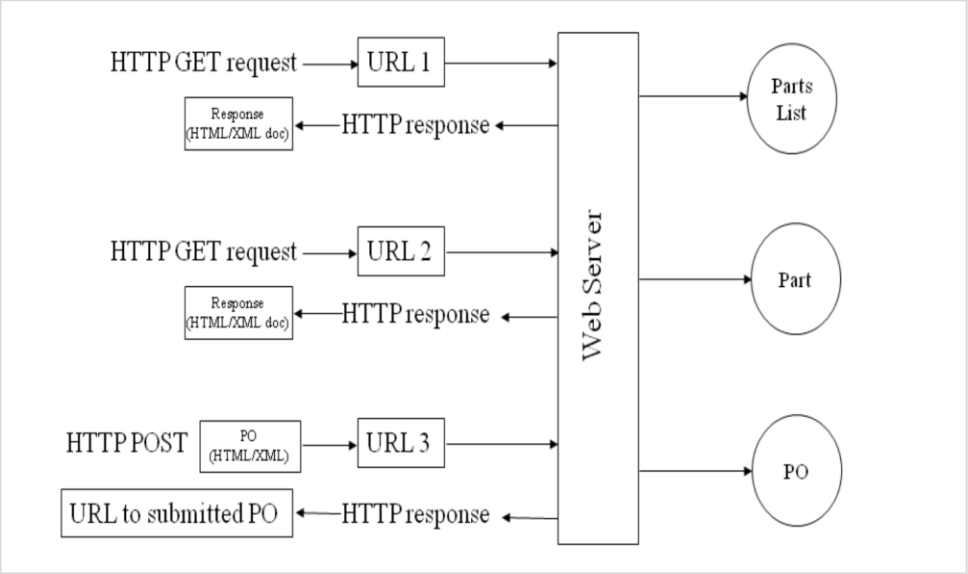


Figura 2-11 : Servizio REST

I messaggi di un servizio SOAP sono complessi e includono informazioni che non possono essere salvate poiché non riusabili, questo non permette l'utilizzo di proxy e ne compromette la scalabilità.

L'aspetto più interessante da analizzare tra le due tipologie di servizi riguarda la sicurezza.

Nella comunicazione di un servizio REST, gli apparati di sicurezza come il firewall sono in grado di discernerne l'intento per ciascun messaggio, analizzando il comando HTTP utilizzato nella richiesta. Ad esempio, una richiesta GET può sempre essere considerata sicura, in quanto non può, per definizione, modificare nessun dato. Dall'altra parte, una tipica richiesta SOAP utilizza sempre il metodo POST per comunicare con i servizi senza un'analisi completa del controllo dei messaggi, non si è in grado di predire se è una richiesta che può modificare le informazioni sul Server e quindi attuare gli opportuni meccanismi di controllo.

Per la parte di autenticazione e autorizzazione, SOAP utilizza una sua metodologia di autenticazione inserita nella specifica del protocollo di comunicazione demandando tutto alla fase di progettazione del servizio.

La metodologia REST, invece, si basa sul fatto che i Web Server supportano già questo tipo di operazioni tramite l'uso di standard come lo scambio di certificati e comuni sistemi di gestione dell'identità (come ad esempio un LDAP server).

Questo disaccoppiamento a volte può aiutare lo sviluppatore del servizio a demandare la problematica dell'autenticazione ad applicazioni esterne che sono facilmente integrabili con l'applicazione RESTful. Altre caratteristiche che differenziano i due tipi di servizio sono la complessità che si incontra nella costruzione delle applicazioni lato Client e lato Server. Effettuare chiamate HTTP ad un servizio è molto semplice e non richiede l'utilizzo di librerie complesse per creare e leggere pacchetti di messaggi come avviene in un servizio SOAP.
Dal lato Server, la complessità si incontra nell'esposizione delle API del servizio, in quanto la gestione del messaggio di una richiesta SOAP avviene mediante librerie del Server, mentre in un servizio REST bisogna gestire la serializzazione del file XML in uscita dalle richieste. Un punto critico è l'esposizione del servizio all'utente che dovrà utilizzarlo, infatti nel SOAP viene messa a disposizione il WSDL, che permette di serializzare in un formato standard le API del servizio. Nei servizi REST viene, ad oggi, utilizzata una pagina in HTML human-readable e quindi è impossibile automatizzare la creazione di richieste da parte di un calcolatore fornendo solo la sua descrizione.

Queste differenze fanno intuire che un servizio RESTful non permette la sostituzione di un servizio creato su SOAP, ma ne è un'alternativa. Ogni tipologia ha vantaggi e svantaggi nella sua applicazione. L'uso di REST o di SOAP è sicuramente determinato dal contesto in cui si vuole utilizzarlo.

3. SCENARIO PROPOSTO

3.1 Il sistema Infor ERP LN

Per poter raggiungere il nostro obiettivo è utile descrivere che cosa si intende per sistema di gestione ERP ed in particolare uno dei maggiori Sistemi ERP in commercio: Infor ERP LN. [19]

Infor LN, implementato dall'americana Infor (terzo produttore mondiale di ERP) è basato su una tecnologia Client-Server proprietaria (linguaggio di programmazione 4GL), supporta tutti i principali database in commercio fino a dimensionamenti enormi (Boeing, Daimler, Ferrari) e si appoggia su piattaforme Web-Based (Sharepoint, Apache, Java, IIS) per realizzare il front-end e l'interfacciamento utente.

ERP (Enterprise Resource Planning) è un sistema di gestione, che integra tutti i processi di business rilevanti di un'azienda (vendite, acquisti, gestione magazzino, contabilità etc.).

Il concetto fondamentale alla base degli ERP fu per la prima volta messo in atto alla fine degli anni '80, quando la tecnologia Client-Server era abbastanza matura da permettere la comunicazione tra i vari componenti costituenti il sistema. E' possibile definire un sistema ERP come la spina dorsale che sorregge, integra e automatizza la maggior parte dei processi industriali, coinvolgendo i settori delle vendite e della distribuzione, della produzione, della logistica, della fatturazione e delle risorse umane.

Inizialmente i sistemi software che si occupavano della gestione di queste sezioni esistevano e funzionavano molto bene nel loro campo specifico. Il problema fondamentale stava nel fatto che, quando questi programmi dovevano collaborare tra loro, la mole di lavoro generata con il solo scopo di produrre dati interscambiabili era tale da invalidare i vantaggi apportati dall'introduzione di tali programmi, in quanto generava eccessivi ritardi per mantenere una buona coordinazione tra i settori. I sistemi ERP non si limitano esclusivamente a coordinare tra loro diverse divisioni, ma permettono alle compagnie di condividere l'informazione tra i vari processi industriali. E' l'idea di condivisione il vero vantaggio di tali sistemi. La connettività è garantita da un corredo di software secondari che condividono i dati del sistema. Questo tipo di organizzazione offre

vantaggi immediati e visibili. Ad una prima e semplicistica analisi sembrerebbe che il processo informativo generato da un evento di partenza si ramifichi ai vari settori dell'impresa secondo una sorta di ordine gerarchico: partendo, ad esempio da un ordine di produzione, si scatena una successione di operazioni che coinvolgono gli altri reparti come "nodi figli" del primo evento. In realtà, non si può parlare di una struttura ad albero, bensì di una struttura a rete. La differenza fondamentale risiede nelle modalità di interazione tra i vari reparti: ognuno genera dati che servono al nodo sottostante, al nodo fratello e anche al nodo padre, creando connessioni con tutti gli apparati interessati al processo in atto ed è questa la vera attuazione del concetto di integrazione. Infor realizza questa idea tramite l'organizzazione logica del software package e moduli e tecnica layer sovrapposti.
La grande potenza di Infor ERP LN è proprio questa, ovvero la possibilità di mantenere e personalizzare un'implementazione con un potentissimo sistema di versioning, completo, estensibile e completamente automatizzato.

Un sistema Infor ERP LN è composto da una serie di moduli funzionali tra loro integrati, con lo scopo di offrire un supporto ai processi d'impresa. Sono possibili varie classificazioni ma i moduli principali possono essere suddivisi in tre grandi gruppi:

- Finanza
- Logistica
- Risorse Umane

Ognuno di questi è formato, al suo interno, da altri sottogruppi più piccoli, con funzionalità che ricoprono capillarmente tutte le possibili operazioni utilizzabili dall'impresa.

3.1.1 Architettura di un sistema Infor ERP LN

La possibilità di creare dei legami tra i vari elementi della struttura informativa, grazie all'utilizzo di un unico database, rappresenta la principale caratteristica del sistema informativo integrato, che lo rende potenzialmente in grado di rispondere a qualunque tipo di richiesta. Infor ERP LN è definito come l'architettura software che facilità il flusso d'informazioni tra tutte le funzioni interne alla società. In questa parte si 2vuole dare un'idea generale di come è costituita l'architettura software di un sistema ERP per poi integrare questo scenario con l'introduzione di servizi SOA, nonché SOAP.
In pratica l'architettura software viene divisa su più livelli in generale su tre livelli, infatti viene anche detta Three Tier come rappresentato nella figura sottostante (Figura 3-1):

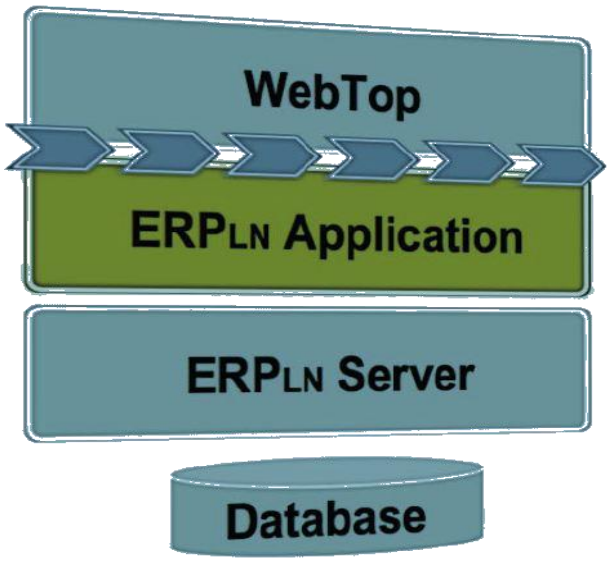


Figura 3-1 : Architettura Infor ERP LN

L'architettura Three Tier è sempre basata sul paradigma Client-Server quindi oltre ad acquisire tutti i vantaggi derivanti da esso offre la possibilità di poter separare le logiche di funzionamento in modo tale da consentire ulteriori miglioramenti in termini di performance.

Presentation Layer

- *Web Top / ERP ln Application*: utilizzato per la rappresentazione dei dati che vengono richiesti dall'utilizzatore (Client) del programma gestionale, il livello di presentazione o presentation layer è caratterizzato da una interfaccia grafica con la quale è possibile visualizzare tutti i dati relativi ai moduli software che sono stati installati nel sistema ERP Server. Viene quindi data la possibilità chiaramente in base alla tipologia di utente che usufruisce dell'accesso al sistema, di avere un controllo effettivo su tutti i dati che sono scritti in un unico database posto al livello (DataBase Layer) e che rispettano i modelli di coerenza e consistenza dei dati.

Application Layer

- *ERP ln Ser*ver: Contiene la logica che interpreta ed elabora i dati passati dal client. In questo livello vengono inseriti il cuore del sistema ERP ovvero il Server in cui viene posizionato il kernel del sistema gestionale ed anche tutti i moduli supplementari (finanza, risorse umane, ecc) che si desiderano installare ai fini di una customizzazione del sistema stesso sulla base delle esigenze di modellazione dell'azienda. E' anche il livello dove avvengono le operazioni già orientate alle transazioni che sono l'insieme di task che consentono la gestione del sistema e di tutte le operazioni che si possono compiere al suo interno e che sono dirette verso il livello database.

DataBase Layer

- *DataBase Layer*: Questo livello si distingue dai precedenti in quanto avviene tutta la gestione dei dati che sono utilizzati dal sistema mediante un database centralizzato. Sostanzialmente all'interno di questo livello è situato un RDBMS (Relational DataBase Management System) a cui viene affidata la gestione dei dati ed è di fondamentale importanza il suo corretto funzionamento ai fini della coerenza dei dati che vengono elaborati al livello Application e visualizzati successivamente al

livello Presentation. In questo livello è importante installare un RDBMS sicuro ed affidabile per rispondere correttamente alle richiesta da parte delle applicazioni ma anche per evidenziare quello che risulta essere un grande punto di forza che ha sempre caratterizzato i sistemi ERP, ovvero l'unicità dell'informazione.

3.2 ERP SOA e SOAP

Le aziende in un mercato globalizzato sentono sempre più spesso la necessità di rendere accessibili le loro applicazioni e questo comporta l'esigenza di avere qualcosa in più di un semplice front-end posto sul nucleo del sistema centrale: nasce quindi l'esigenza di realizzare una forte integrazione tra tutte le applicazioni ed i dati, indipendentemente dalla loro posizione geografica o logica.
Infor ERP LN per necessità come sopra descritto, è stato inserito in un'architettura orientata ai servizi (SOA), permettendole così di integrarsi con le applicazioni di partner, fornitori e in particolare è stato introdotto un software proprietario, chiamato ERP Business Studio che viene utilizzato come piattaforme di sviluppo per le interfacce di Business Infor. Queste interfacce consentono di comunicare i dati tra ERP e le altre applicazioni aziendali. Business Studio è implementato come un insieme di funzioni del framework Eclipse e offre la possibilità di includere vari componenti all'interno di una interfaccia di definizione e implementazione con entità collegate quali tabelle, tipi di dati, funzioni e moduli.

E' possibile utilizzare Business Studio per le seguenti attività principali:

- Per definire una nuova interfaccia di business.
- Per visualizzare e modificare l'interfaccia esistente di business.
- Per implementare un'interfaccia di business per ERP Enterprise generando il codice runtime.
- Per testare una implementazione dell'interfaccia business.
- Per esportare o importare un'interfaccia di business.

Business Studio produce una interfaccia chiamata BDE (Business Data Entity) che consiste in una collezione e combinazione di dati, tabelle, DLL e metodi per l'accesso ai dati.

La creazione di una BDE è sostanzialmente un Web Service che viene pubblicato attraverso un connettore basato su scambio di messaggi SOAP. Ciò implica che un'applicazione esterna può accedere alla BDE inviando un messaggio di richiesta (request) e la BDE restituirà una risposta (response). Più precisamente un Client Web invia un messaggio SOAP al connettore richiamando la BDE che è uno specifico messaggio XML e lo invia al back-end. L'applicazione ERP LN passando dalla BDE recupera i dati richiesti e restituisce un altro BDE che invia al Web Service che lo trasforma in un messaggio SOAP e infine lo inoltra al Client che ha fatto al richiesta. Il formato dei messaggi che vengono scambiati è il linguaggio WSDL (Figura:3-2). Nella figura viene inserita anche un'altra interfaccia chiamata BOI (Business Object Interface), il predecessore delle BDE, che utilizza strumenti interni a LN.

Il limite di questa interfaccia riguarda la non possibilità di utilizzare Business Studio. In questo documento facciamo riferimento solo alle BDE proprio perché associata ad esse è possibile utilizzare questa tecnologia esterna. Questo insieme di tecnologie ci permetteranno di analizzare il caso di studio descritto.

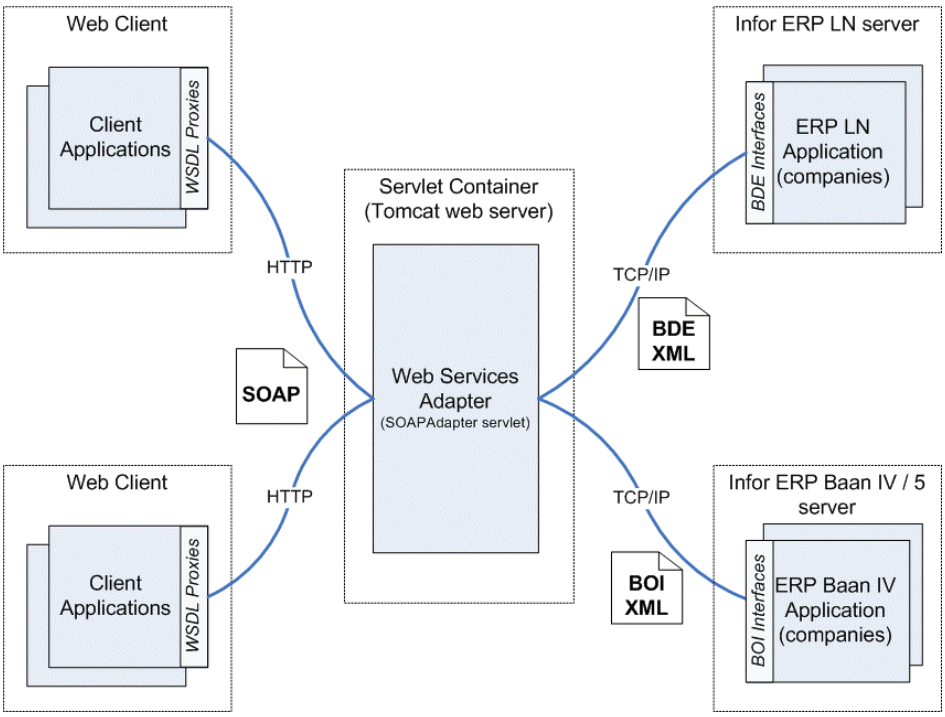


Figura: 3-2 Schema Business Studio

Il connettore menzionato posto tra il Client Application ed il Server ERP Ln è una servlet java gestita in ambiente TomCat Server.

Per l'implementazione di un Web Service specifico, si utilizza una pagina di configurazione, dove vengono elencati tutti i Web Service. In questa pagina viene visualizzato lo stato attuale dei servizi pubblicati e vengono monitorate le richieste in entrata. Infatti il connettore definisce anche una pagina di stato, mostrando i servizi web attualmente attivi e specifica , per ogni servizio, la Infor ERP back-end, il BDE, l'URL e il numero di richieste effettuate per quello specifico servizio.

E' possibile inoltre estrarre il file WSDL per utilizzarlo nella generazione di un eventuale proxy.

L'approccio proposto da Infor quindi è quello di utilizzare l'architettura SOAP per lo scambio di informazioni tra ERP e le applicazioni esterne.

Nel capitolo seguente verrà descritto un esempio di implementazione sulla base degli studi e delle considerazioni riportate precedentemente.

4. PROGETTAZIONE E IMPLEMENTAZIONE

Vediamo ora con un esempio pratico come avviene l'utilizzo di queste strutture e come sia possibile o meno utilizzare anche l'architettura REST.

4.1 Scenario di riferimento

Per poter spiegare e capire al meglio tutte le informazioni riportate in questo capitolo, vogliamo descrivere un particolare scenario utilizzandolo per esprimere alcuni concetti fondamentali.

Partiamo dalla costruzione di una BDE utilizzando il tool di Infor Business Studio descritto nel capitolo precedente.

Il nostro obiettivo è produrre un Web Service che faccia le veci della funzione 'Ordine di acquisto', che nello specifico produce un inserimento, una vista ed una lista di un ordine fatto dal magazzino.

Mostriamo l'interfaccia di Business Studio Figura 4-1 e creiamo un nuovo progetto. Chiameremo il nostro Web Service 'PurchaseOrderAdvice', che si interfaccerà con le tabelle di Infor ERP Ln.

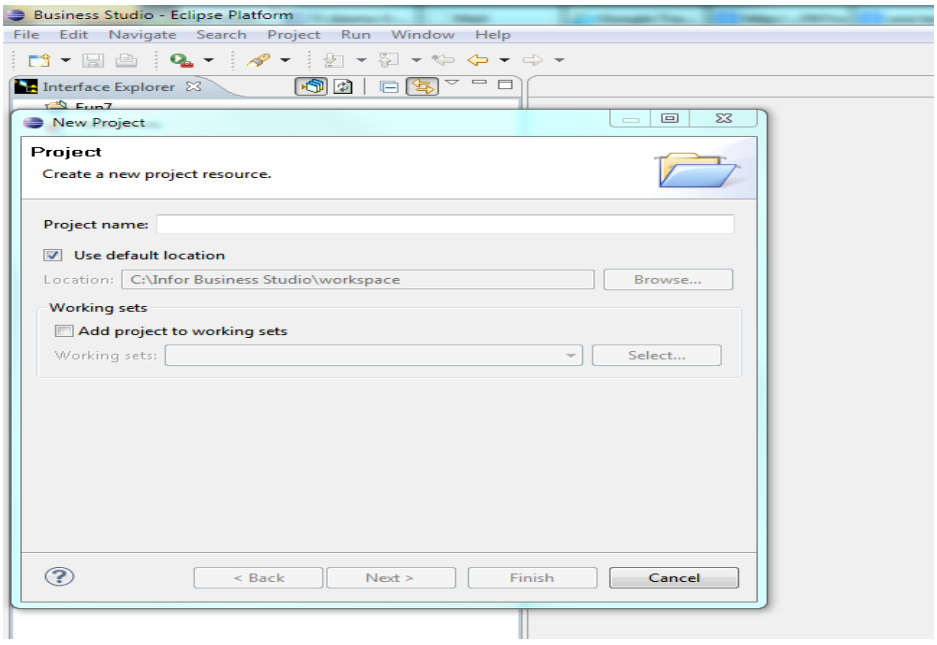


Figura 4-1: Interfaccia Business Studio

Successivamente attraverso l'interfaccia di implementazione nell'applicativo chiamata Interface Implementations importiamo la tabella dal database di Infor ERP Ln, corrispondente agli Ordini di acquisto, definiamo l'attributo chiave 'Item' e i metodi standard. Per ogni metodo implementato ogni componente deve essere disponibile come parametro di input/output, solo input o solo output.

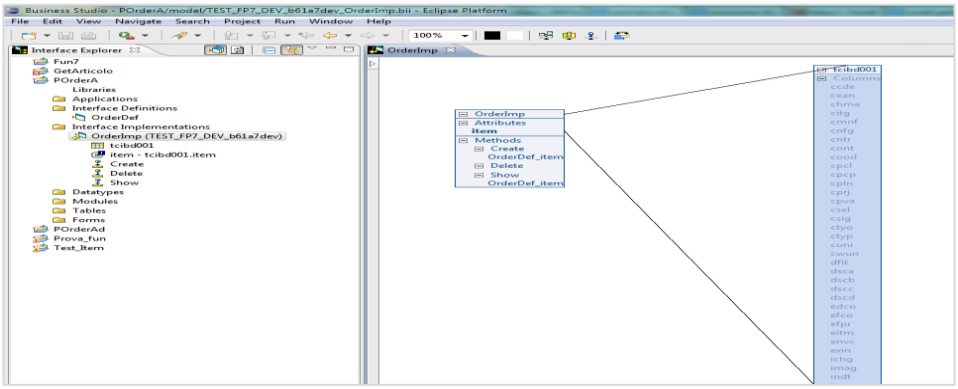


Figura 4-2: Creazione BDE

Nella Figura 4-2 è mostrato lo schema della BDE appena creata, dove si evince la corrispondenza con i campi della tabella e i metodi (Create, Delete, Show), che saranno le azioni possibili attraverso la chiamata al nostro Web Service.

Un volta definita la BDE attraverso un comando 'Generate ERP Enterprise Implementation' viene pubblicata sul connettore WebTop. (Figura 4-3)

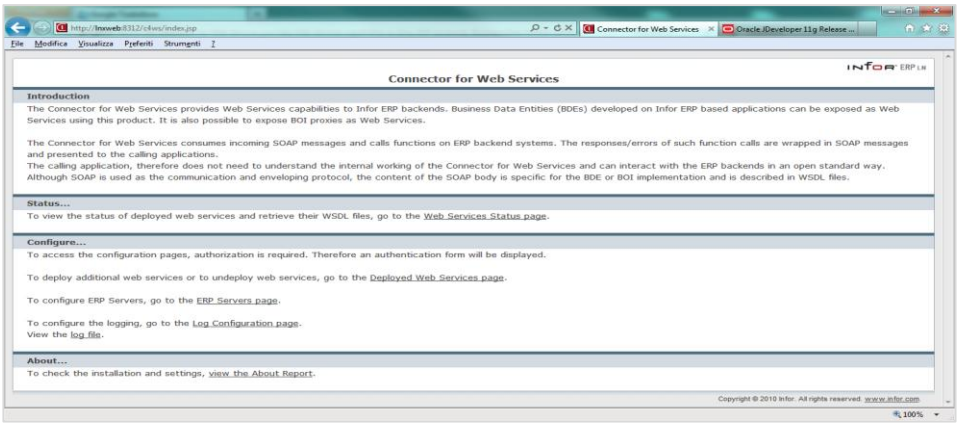
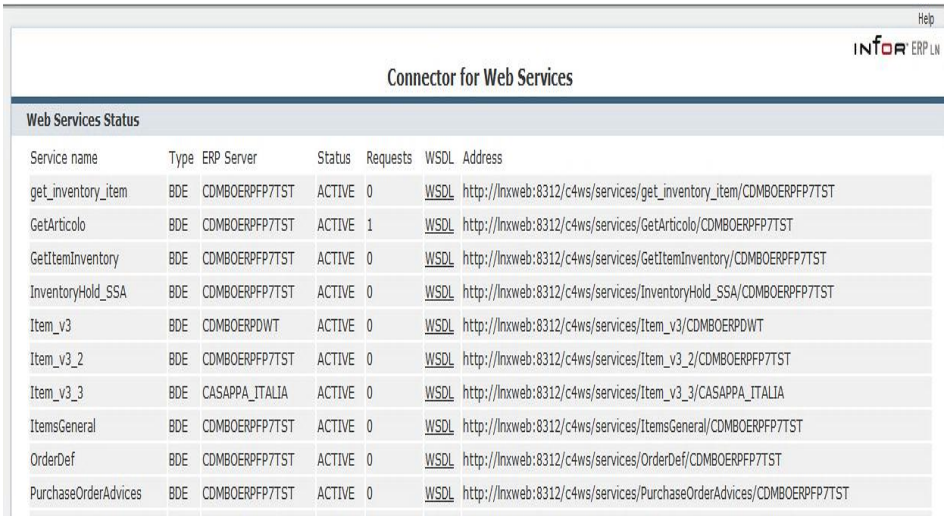


Figura 4-3: Connettore per Web Service



Service name	Type	ERP Server	Status	Requests	WSDL	Address
get_inventory_item	BDE	CDMBOERPFP7TST	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/get_inventory_item/CDMBOERPFP7TST
GetArticolo	BDE	CDMBOERPFP7TST	ACTIVE	1	WSDL	http://lnxweb:8312/c4ws/services/GetArticolo/CDMBOERPFP7TST
GetItemInventory	BDE	CDMBOERPFP7TST	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/GetItemInventory/CDMBOERPFP7TST
InventoryHold_SSA	BDE	CDMBOERPFP7TST	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/InventoryHold_SSA/CDMBOERPFP7TST
Item_v3	BDE	CDMBOERPDWT	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/Item_v3/CDMBOERPDWT
Item_v3_2	BDE	CDMBOERPFP7TST	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/Item_v3_2/CDMBOERPFP7TST
Item_v3_3	BDE	CASAPPA_ITALIA	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/Item_v3_3/CASAPPA_ITALIA
ItemsGeneral	BDE	CDMBOERPFP7TST	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/ItemsGeneral/CDMBOERPFP7TST
OrderDef	BDE	CDMBOERPFP7TST	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/OrderDef/CDMBOERPFP7TST
PurchaseOrderAdvices	BDE	CDMBOERPFP7TST	ACTIVE	0	WSDL	http://lnxweb:8312/c4ws/services/PurchaseOrderAdvices/CDMBOERPFP7TST

Figura 4-4: Elenco Web Service

A questo punto è possibile visualizzare e utilizzare il WSDL per interfacciare qualsiasi applicazione esterna che necessita della funzione Ordine di acquisto 'PurchaseOrderAdvice'. (Figura 4-4)

Prendiamo ora un tool di test chiamato SoapUI, che ci permetterà di verificare l'effettiva efficienza di quanto descritto finora. Creando un nuovo progetto inserendo il file WSDL, l'applicativo restituirà quanto visualizzato in Figura 4-5.

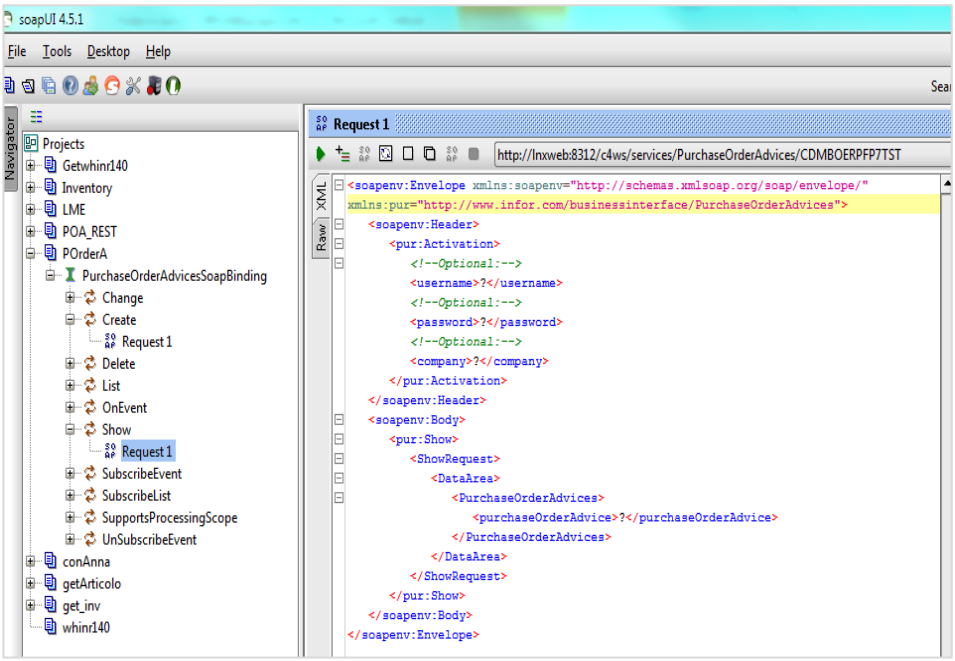


Figura 4-5: SoapUI

Prendendo in esame il metodo ‘Show’, è sufficiente inserire l’etichetta dell’ordine di acquisto che vogliamo visualizzare nella sezione <DataArea>, compilando il tag <PurchaseOrderAdvice> con il codice richiamato dal metodo.

Il codice sotto elencato, riporta la Request e successivamente all’esecuzione del metodo la Response con i dati richiesti:

Request SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:pur="http://www.infor.com/businessinterface/PurchaseOrderAdvices">
  <soapenv:Header>
    <pur:Activation>
      <!--Optional:-->
      <username>bsp</username>
      <!--Optional:-->
      <password>duedipicche</password>
      <!--Optional:-->
      <company>090</company>
    </pur:Activation>
  </soapenv:Header>
  <soapenv:Body>
    <pur:Show>
      <ShowRequest>
        <DataArea>
          <PurchaseOrderAdvices>
            <purchaseOrderAdvice>PUA000006</purchaseOrderAdvice>
          </PurchaseOrderAdvices>
        </DataArea>
      </ShowRequest>
    </pur:Show>
  </soapenv:Body>
</soapenv:Envelope>
```

Response SOAP

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ShowResponse xmlns="http://www.infor.com/businessinterface/PurchaseOrderAdvices">
      <ShowResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="">
        <DataArea>
          <PurchaseOrderAdvices>
            <confirmed>no</confirmed>
            <buyer/>
            <planner/>
            <warehouse>WHAMFS</warehouse>
            <deliveryDate>2012-09-14T15:27:37Z</deliveryDate>
            <effectivityUnit>0</effectivityUnit>
            <item>PCX-POWER CORD</item>
            <orderedQuantityInOrderUnit>48</orderedQuantityInOrderUnit>
            <orderDate>2012-09-14T15:27:37Z</orderDate>
            <purchaseOrderAdvice>PUA000006</purchaseOrderAdvice>
            <orderUnit>pcs</orderUnit>
            <line>0</line>
            <purchaseOrder/>
```

```
            <orderedQuantityInInventoryUnit>48</orderedQuantityInInventoryUnit>
            <relatedBomComponent>0</relatedBomComponent>
            <relatedOrder/>
            <line0>0</line0>
            <relatedOrderSequence>0</relatedOrderSequence>
            <returnOrder>no</returnOrder>
            <shipfromBusinessPartner>BPG000003</shipfromBusinessPartner>
            <buyfromBusinessPartner>BPG000003</buyfromBusinessPartner>
            <purchaseOrderAdviceText/>
          </PurchaseOrderAdvices>
        </DataArea>
      </ShowResponse>
    </ShowResponse>
  </S:Body>
</S:Envelope>
```

Il Web Service richiamato si interfaccia con il database di Infor ERP Ln, riportando le informazioni richieste. Apriamo quindi il menu browser dell'interfaccia e vediamo da dove vengono prese le informazioni. (Figura 4-6, 4-7).

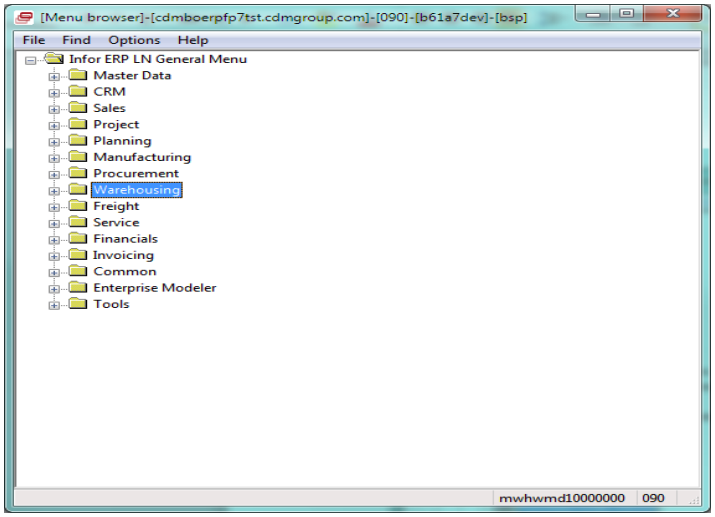


Figura 4-6: Menu Browser del DB di Infor ERP Ln

Questo menù organizzato in sotto menù, permette l'accesso a tutte le funzioni del gestionale che rispettano la suddivisione dei package/moduli. E' possibile visualizzare quindi tutte le informazioni che riguardano la struttura dei moduli e le specifiche sessioni. Le singole sessioni mostrano i dati e il corredo di strumenti che il gestionale offre.

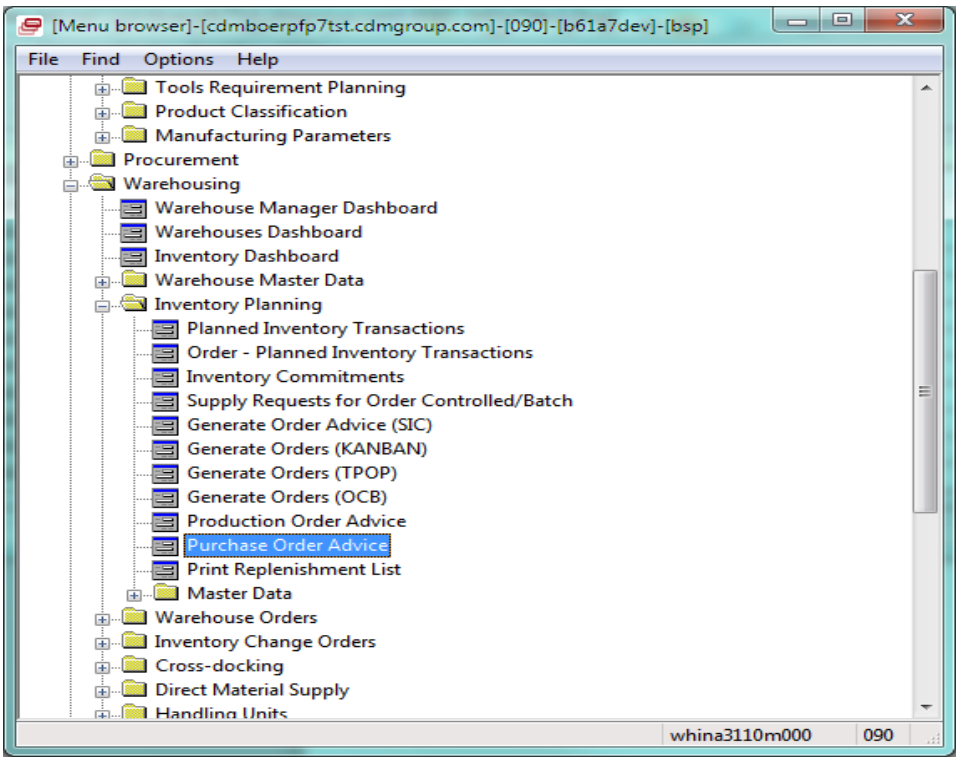


Figura 4-7: Sessione interrogata

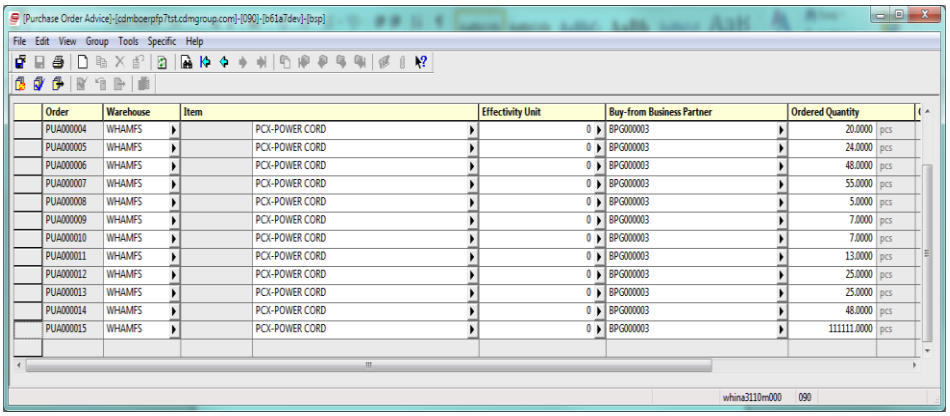


Figura 4-8: Maschera Ordini di Acquisti

Le informazioni passate dalla Response SOAP corrispondono alle informazioni presenti in banca dati associate all'ordine con il codice precedentemente inserito. (Figura 4-8).

Abbiamo scelto di descrivere il metodo Show per semplicità, ma le stesse operazioni vengono effettuate per gli altri metodi, il Web Service creato con chiamate SOAP permette quindi di inserire, visualizzare, cancellare,

le informazioni riguardanti gli ordini di acquisto senza interfacciarsi con il database. Questi Web Service creati ad hoc sulle funzioni di Infor ERP Ln permettono alle applicazioni Client come terminali RDF (RaDioFrequenza – Figura: 4-9) di migliorare il processo produttivo, consentendo la riduzione degli errori legati alle operazioni di magazzino, controllando l'efficienza di ciascuna operazione, trasmettendo dati senza perdita di tempo e senza spreco di banda.

Figura 4-9: Terminale RaDioFrequenza

4.2 ERP Ln e REST

Come abbiamo detto nei capitoli precedenti ciò che propone REST è creare server capaci di accettare in input tutti i metodi HTTP definiti dal protocollo ridefinendoli ed utilizzandoli e non più l'utilizzo di funzioni ma solamente di risorse.

Riprendendo quindi l'esempio implementato precedentemente, sviluppiamo il caso d'uso utilizzando REST.

Con il tool SoapUI creiamo un nuovo progetto inserendo direttamente l'end-point o 'Address' ricavato dal Connettore per Web Service, l'indirizzo è:

http://lnxweb:8312/c4ws/services/PurchaseOrderAdvices/CDMBOERPFP7TST. (Figura: 4-10)

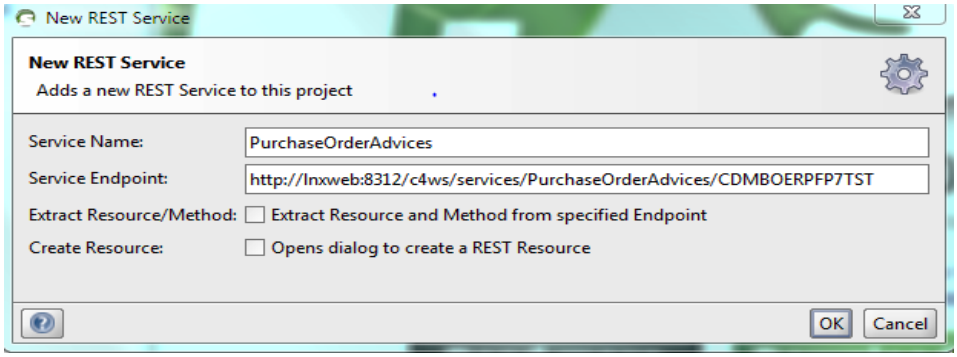


Figura 4-10: Inserimento end-point

Successivamente viene inserita la risorsa della quale si vogliono visualizzare i dati, con il metodo 'Show' – PurchaseOrderAdvice = 'PUA000006', poi si procede definendo il metodo HTTP, in questo caso 'GET' che permette il ritorno dei dati corrispondenti alla risorsa inserita. (Figura:4-11)

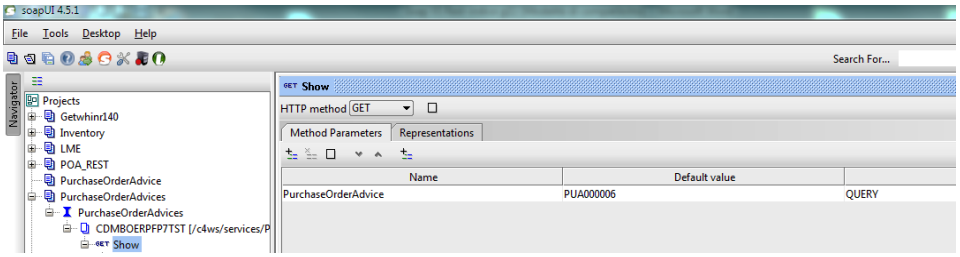


Figura 4-11: Metodo GET

La Request risulta essere questa :

GET
http://lnxweb:8312/c4ws/services/PurchaseOrderAdvices/CDMBOERPFP7TST?PurchaseOrderAdvice=PUA000006
HTTP/1.1

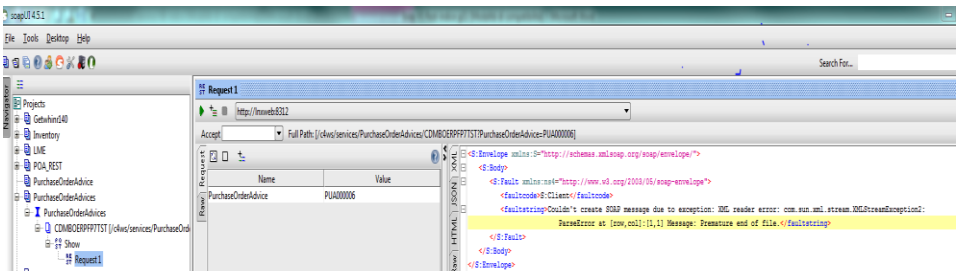


Figura 4-12: Risultato Request

Eseguendo la Request sul metodo Show, il risultano non è quello aspettato, il sistema restituisce un <Fault> riportato qui sotto

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>S:Client</faultcode>
          <faultstring>Couldn't create SOAP message due to exception: XML reader
        error: com.sun.xml.stream.XMLStreamException2:
                  ParseError at [row,col]:[1,1] Message: Premature end ofile.
      </faultstring>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

Come abbiamo illustrato approfonditamente nelle sezioni precedenti, HTTP definisce la semantica delle operazioni e definisce anche un insieme di codici di risposta, per esempio 200, che significa 'Ok' e 404, che indica che la risorsa non è stata trovata. Nel nostro messaggio di ritorno, possiamo notare che non viene restituito nessun codice, questo significa che la chiamata REST non arriva nemmeno ad interfacciarsi con la BDE.

L'errore dice che la comunicazione tra BDE e chiamata REST non avviene con lo stesso linguaggio. La richiesta non è in un formato SOAP valido, e dà un errore nella lettura del file XML. Le librerie di Infor Ln supportano solo SOAP, non entrambi, il modello di comunicazione è troppo diverso per giustificare l'efficienza dei due scenari.

Pur essendo molto più semplice la semantica di questa architettura, non è possibile interfacciare REST con le BDE create con l'architettura SOAP di Infor Ln.
Per scelte aziendali Infor non ha ancora sviluppato un software proprietario che permetta l'implementazione di servizi sviluppati con REST.

5. CONCLUSIONI

I servizi costruiti sul protocollo SOAP, possono affidarsi alla rigidità delle specifiche del protocollo e la sua applicazione storica permette di avere la certezza che le regole che costituiscono la sua costruzione non subiranno grosse modifiche. I servizi web che si basano sui principi di design REST non sono regolamentati da specifiche rigide e questo può portare al cambiamento continuo di utilizzo e ad una esplosione di modalità d'uso.

L'architettura RESTful risulta sufficientemente flessibili per implementare servizi di natura differente, ed il motivo della grande diffusione, è data dalla sua flessibilità di utilizzo e il modello su cui poggia è di uso comune (HTTP). Chi vuole costruire un servizio REST non ha bisogno di sapere come si costruisce un WSDL e i requisiti minimi sono quelli di saper costruire una pagina Web ben leggibile dall'utente umano. Per questo, quando si lavora e si prendono in considerazione i servizi web RESTful, bisogna fare attenzione a non appesantirlo troppo, con molte regole complicate, altrimenti decade il principale vantaggio di utilizzo. Infatti aspetti importanti tipo la qualità del servizio, le transazioni, la sicurezza e l'affidabilità messaggistica, non sono ancora supportati da REST. Altre caratteristiche importanti, come la composizione di servizi e la definizione del contratto, sono ancora in fase di studio e richiedono ulteriore sviluppo.

Questi problemi possono impedirne l'utilizzo in scenari di integrazione complessi in cui queste caratteristiche sono di grande importanza.

Entrambi gli approcci sono in grado di risolvere i problemi, ma la scelta tra di loro nel processo di standardizzazione non si basa solo su argomenti tecnici ma bensì anche su scelte aziendali.

Quanto visto in questo documento di tesi, riporta la necessità appunto delle aziende di affidarsi al servizio SOAP già largamente diffuso e conosciuto, piuttosto che interfacciarsi con REST vista la mole di lavoro che risulterebbe dal costruire da capo un software che si interfaccia

perfettamente alla complicata e pesante mole di dati di cui fa parte un gestionale come Infor ERP Ln.

BIBLIOGRAFIA

[1] W3C Working Group. Web Services Architecture. http://www.w3.org/TR/ws-arch/.

[2] W3C Working Group. Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[3] W3C Recommendation 10 Febbraio 1998. Extensible Markup Language (XML) 1.0, W3C. http://www.w3.org/TR/1998/REC-xml-19980210

[4] G. Meredith S. Weerawarana E. Christensen, F. Curbera. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl

[5] W3C Working Group. Simple object access protocol (soap) 1.1 http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_ Toc478383487.

[6] W3C/MIT. Hypertext transfer protocol – http/1.1. http://tools.ietf.org/ html/rfc2616.

[7] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. Master's thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

[8] Tim Berners-Lee. Universal Resource Identifiers – axioms of web architecture. . http://www.w3.org/DesignIssues/Axioms.

[9] Tim Berners-Lee. Uniform Resource Locators (URL) http://www.w3.org/Addressing/URL/url-spec.txt

[10] Leonard Richardson, Sam Ruby. RESTful web services O'Reilly, Sebastopol 2007

[11] John Maeda. The Lows of Simplicity, The MIT Press (August 21, 2006)

[12] W3C Working Group. Web Services Architecture http://dev.w3.org/2002/ws/arch/wsa/wd-wsa-arch.html#whatisws

[13] Alessia Cuppini. Infor E Red Hat Collaborano (October 19, 2012) http://www.comunicati-stampa.net/com/infor-e-red-hat-collaborano-per-distribuire-infor-ln-su-piattaforma-full-open-source-con-database-mysql-e-mariadb.html

[14] Hao He. What is Service-Oriented Architecture? http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[15]IBM developerWorks. New to SOA and Web services. http://www-106.ibm.com/developerworks/webservices/newto/

[16] Roy Thomas Fielding. Representational State Transfer (REST) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[17] Jon Flanders. MSDN Magazine. More On REST. http://msdn.microsoft.com/en-us/magazine/dd942839.aspx

[18] Elisabetta Scaramuzza. Java Web Service (April 2004). Web Services Security. http:// http://javawebservice.net/

[19] Enterprise Resource Planning. http://www.infor.com/solutions/erp/

[20] Brennan Spies. Soap vs. Rest. http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest

[21] Daniele Bonetta, Achille Peternier, Cesare Pautasso, Walter Binder. University of Lugano, USI, Lugano Switzerland. A scripting language for high-performance RESTful web services. http://dl.acm.org/citation.cfm?doid=2145816.2145829

[22] Kennedy Sean; Molloy Owen; Stewart Robert; Jacob Paul; Maleshkova Maria; Doheny Frank. A Semantically Automated Protocol Adapter for Mapping SOAP Web Services to RESTful HTTP Format to Enable the Web Infrastructure, Enhance Web Service Interoperability and Ease Web Service Migration. *Future Internet* 2012, *4*, 372-395. http://www.mdpi.com/1999-5903/4/2/372

[23] Robert Dizon**.** Applies to: Windows Communication Foundation. Operation-based SOAP vs. Resource-based REST. http://msdn.microsoft.com/enus/library/vstudio/hh323724%28v=vs.100%29.aspx

[24] Wilson A. Higashino, M. Beatriz Felgar de Toledo, Miriam A. M. Capretz. Published in: Proceedings First International Symposium on Services Science ISSS'09, Logos, Berlin, 2009 (www.logos-verlag.de). REST and Resource-Oriented Architecture.

[25] HostBridge Technology. Revision date: 3/2/2009. SOAP and REST: Choosing formal and informal Web services for CICS integration

[26] Michael zur Muehlen, Jeffrey V. Nickerson, Keith D. Swenson. Developing web services choreography standards—the case of REST vs. SOAP. Decision Support Systems 40 (2005)

[27] David Chappell, Chappell & Associates. SOAP vs. REST: Complements or Competitors? www.davidchappell.com (2009)