ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Informatica per il management

Grounding Di Servizi
Per Sistemi Adattativi

Relatore:
Chiar.mo Prof.
Davide Rossi

Correlatore:
Francesco Poggi

Presentata da:
Vittoria Emma Maria
Emiliani

I Sessione
2016/2017

ABSTRACT

Negli ultimi anni sono nati numerosi servizi Web che implementano un nuovo tipo di approccio che rispecchiano l'insieme dei principi denominati REST. Nel creare composizioni di servizi, si necessita spesso di eseguire una serie di invocazioni verso servizi di entrambe le tecnologie sia SOAP che REST.

Questa tesi si pone l'obbiettivo di fornire una rappresentazione semantica di servizi di stessa natura ma implementati sia con architettura REST che con architettura SOAP, per poter effettuare una chiamata trasparente di un servizio o dell'altro.

Sono stati creati due web services di stessa natura ma con due protocolli diversi, implementati con le librerie JAX-WS e JAX-RS.

E' stata creata poi una rappresentazione semantica di tali servizi, attraverso ontologie OWL-S, che avessero stesso Service, Profile e Process ma che supportassero sia un Grounding REST che uno SOAP.

Tutto ciò è stato fatto per permettere quindi di poter richiamare il servizio con architettura REST piuttosto che quello con architettura SOAP in modo trasparente in base alle necessità.

Per poter fare ciò, è stato necessario creare un Manager che si occupasse di leggere e interpretare le ontologie, in modo tale da poter effettuare la chiamata al servizio opportuno.

Indice

Elenco delle figure 6

1 INTRODUZIONE 9

1.1 Obiettivo 12

2 IL WEB: DA WEB 1.0 a WEB 3.0 15

2.1 Il Web 1.0 15

2.2 Il Web 2.0 17

2.2.1 Web Services 18

2.2.2 Web Services: SOAP e REST 20

2.3 Web 3.0 - Semantic Web 24

3 I SERVIZI NEL SEMANTIC WEB 29

3.1 Ontologie 29

3.2 OWL 31

3.2.1 Individuals 31

3.2.2 Proprietà 32

3.2.3 Classi 34

3.3 OWL-S 35

3.3.1 Service Profile 39

3.3.2 Process Service Model 42

3.3.3 Grounding 44

3.3.4 Soap Grounding 44

3.3.5 Grounding RESTful 49

4 PROG & IMP 57

4.1 Ontologies Manager 57

4.2 Servizi 66

4.2.1 JAX-WS - Calculator SOAP service 67

4.2.2 JAX-RS - Calculator REST Service 73

4.3 Ontologie Calculator - OWL 78

4.3.1 Calculator Profile 79

4.3.2 Calculator Process 81

4.3.3 CalculatorSoapGrounding 84
4.3.4 CalculatorRestGrounding 88
4.3.5 Calculator Service 91

5 CONCLUSIONI **93**
5.1 Tecnologie Utilizzate 95

Riferimenti bibliografici **99**

Elenco delle figure

1 Web Services 10
2 Web 1.0 16
3 Web Services : application-centric 18
4 Web 2.0: architettura 19
5 Richiesta SOAP 21
6 Richiesta REST 22
7 Web of data 24
8 Semantic Web layered architecture [4] 25
9 What is an ontology? 29
10 Rappresentazione di Individuals [10] 32
11 Rappresentazione delle Proprietà [10] 32
12 Le diverse proprietà di OWL [10] 33
13 Rappresentazione delle Classi (contenenti Individuals) [10] . 34
14 Service Ontology 36
15 Structure of the OWL-S Profile 40
16 Structure of the OWL-S Process [8] 42
17 Mapping tra OWL-S e WSDL [8] 47
18 Struttura di un web service REST 49
19 Estensione di OWL-S per OWL-S/WADL Grounding 54
20 RDF Data Model 58
21 SPARQL triple pattern 59
22 Gerarchia Grounding 62
23 Calculator OWL-S 78
24 Diagramma Di Attività 94

CAPITOLO 1

Introduzione

1 INTRODUZIONE

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

Si parla di Web 2.0 che ha, come caratteristica principale,la trasformazione dell'interazione tra sito-utente, introducendo il cambiamento dell'approccio con il quale gli utenti si rivolgono al Web, che passa fondamentalmente dalla semplice consultazione alla possibilità di contribuire popolando e alimentando il Web con propri contenuti spesso creati in cooperazione tra essi. Questo tipo di approccio ha facilitato lo svolgere di determinate operazioni comuni come quello di effettuare acquisti di qualsiasi tipo e consultare servizi informativi di uso generale.

Una tecnologia che sta prendendo piedi in questi anni, orientata ad affrontare le nuove problematiche e sfide del Web di oggi, è quella orientata ai servizi.

- 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 utilizzatore del servizio.
- Per **Web Service (WS)** si intende un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete ovvero in un contesto distribuito.

I servizi Web sono una tecnologia basata su Xml e sono progettati per supportare l'interoperabilità e interazione tra macchine attraverso una rete. La tecnologia Web Services, spesso nel documento chiamata semplicemente servizi o servizi Web, e l'utilizzo di architetture orientate ai servizi (SOA), sembra ottenere un enorme successo.

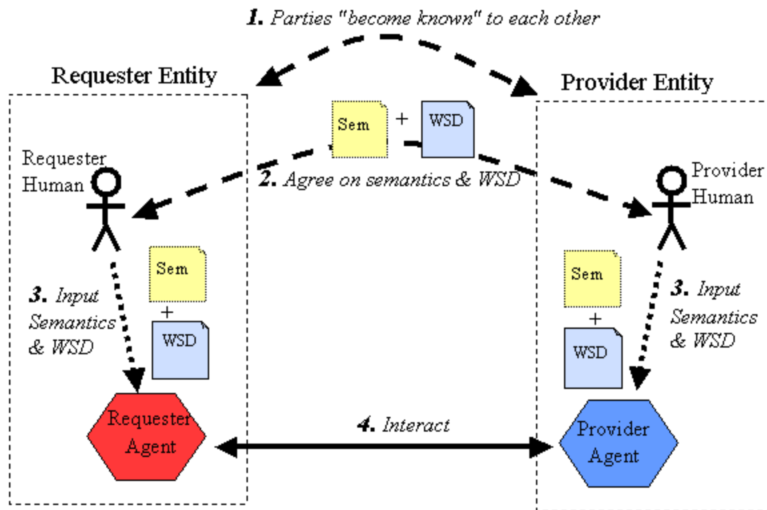

Figura 1: Web Services

Con il Web 3.0 o semantic web si vogliono diminuire compiti e decisioni umane e lasciarli alle macchine stesse. Questo rendendo i contenuti web "machine-readable". In generale il web 3.0 è formato da due piattaforme principali: le tecnologie semantiche, sulle quali ci focalizzeremo principalmente in questa tesi, e il social computing.

Con il Web 3.0 la meccanica di scambio di messaggi è documentata in un WSD - Web Service description.

Il WSD è un documento che consente al fornitore di servizi di comunicare le specifiche per l'avvio del servizio Web al richiedente del servizio.

Per quanto riguarda i web service SOAP, il documento viene espresso con il linguaggio WSDL, per quanto riguarda invece i web service REST, si parla di linguaggio WADL.

Di fondamentale importanza nel Web 3.0 è ciò che viene conosciuto co-

me Web Semantico. Si intende la trasformazione del World Wide Web in un ambiente dove i documenti pubblicati (pagine HTML, file, immagini, e così via) sono associati ad informazioni e dati (metadati) che ne specificano il contesto semantico in un formato adatto all'interrogazione e l'interpretazione (es. tramite motori di ricerca) e, più in generale, all'elaborazione automatica.

Il linguaggio trattato in tale tesi che permette di effettuare una descrizione semantica di un web service, è OWL-S.

1.1 Obiettivo

I Web Services, e si differenziano in due proposte diverse.

La prima, è basata sull'uso del protocollo **SOAP**, di cui parleremo estesamente più avanti, per l'imbustamento dei contenuti applicativi, e sulla definizione di numerosi e spesso eccessivamente complessi formati di header della busta SOAP, utilizzati per contenere le metainformazioni necessarie a vari servizi di infrastruttura (sicurezza, indirizzamento, transazionalità, etc.).

La seconda, solitamente riferita come Restful Web Services o semplicemente **REST** parte invece dall'assunzione che il protocollo http possa essere usato nativamente per realizzare applicazioni, senza bisogno dell'ulteriore livello di imbustamento SOAP. REST impone però un insieme di vincoli molto rigidi alla progettazione delle applicazioni.

In una composizione di servizi potremmo avere la possibilità di poter richiamare servizi di entrambe le tecnologie SOAP e REST in maniera dinamica e trasparente dal punto di vista dell'invocazione.

Lo scopo principale di tale tesi è stato quello di concentrarsi maggiormente su quello che oggi viene chiamato **Web 3.0** o **Semantic Web** e le sue tecnologie annesse.
Si ha la necessità di creare dei servizi che possano essere machine-readable e che possano essere interpretati dalla macchina e quindi richiamati, in base alle richieste del client, in modo trasparente.

In particolare in questa tesi ci si è focalizzati principalmente nel creare due servizi con stesse funzionalità ma di tipologia sia REST che SOAP, in modo tale da mostrare la possibilità di poter descrivere semanticamente entrambi i due servizi in modo tale da richiamare o l'uno o l'altro in base alle necessità.

La descrizione semantica di tali servizi si focalizzerà soprattutto nella creazione dei due Grounding diversi che saranno supportati dal servizio: Grounding RESTful e Grounding SOAP.

E' stato inoltre creato un manager, che permettesse di leggere e interpretare le ontologie create, in modo tale da poter richiamare il servizio opportuno in base alla richiesta del client.

CAPITOLO 2

Da Web 1.0 a Web 3.0

2 IL WEB: DA WEB 1.0 a WEB 3.0

2.1 Il Web 1.0

Nel 1989, Tim Burners-Lee suggerì di creare uno spazio ipertestuale globale in cui ogni informazione presente in rete potesse essere accessibile da un unico UDI (Universal Document Identifier).

Il sogno dietro all'idea era quello di creare uno spazio comune di informazione in cui le persone potessero comunicare condividendo esse stesse delle informazioni.

Il Web 1.0 era principalmente un **"read-only" web**, non c'era quindi iterazione, ma era possibile solo leggere informazioni. Era statico e monodirezionale.

I siti web includevano pagine HTML statiche che venivano aggiornate poco frequentemente. Lo scopo principale di tali, era quello di pubblicare informazioni che ogni persona potesse consultare in ogni momento.

Gli users e i visitatori dei siti web, potevano solo visitarli senza poter contribuire o interagire con essi.

Il Web 1.0 si basava soprattutto sugli hyperlink, che permettevano di connettere più documenti diversi. La grande creazione fu l'internet cloud: non bisognava più preoccuparsi di dove fossero collocati i documenti, bastava che avessero un indirizzo internet e che ci fossero gli hyperlink che ne permettessero la connessione.

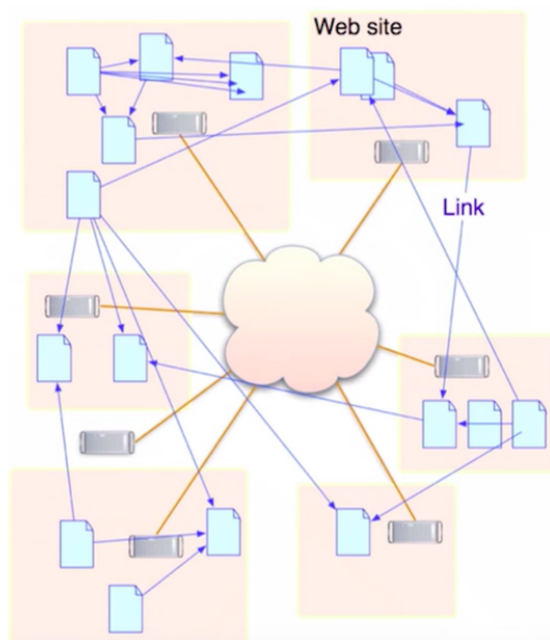

Figura 2: Web 1.0

Il vero grande vantaggio di internet era che permetteva un'astrazione dal livello macchina: non bisognava più preoccuparsi di dove fossero collocati i servers.

2.2 Il Web 2.0

Il termine web 2.0 è stato definito ufficialmente nel 2004 da Dale Dougherty.

Il Web 2.0 è conosciuto anche come people-centric web e read-write web. Permettendo oltre alla lettura anche la scrittura, il web è diventato bidirezionale. La differenza sostanziale risiede nell'approccio con cui gli utenti si rivolgono al Web: dalla semplice consultazione passiva dei contenuti alla produzione dinamica e attiva di pagine web e informazioni che vanno ad arricchire, popolare e alimentare la rete.

Come molti importanti concetti, Web 2.0 non ha un confine definito, ma un'anima gravitazionale. Si può vedere il web 2.0 come un set di principi e pratiche che legano un vero sistema solare di siti che manifestano alcuni o tutti di questi principi, al variare della distanza da tale centro.

Uno dei principali principi è quello del Web come piattaforma. Pensare al web in questi termini vuol dire che, nel passaggio da Web 1.0 a 2.0, gli sviluppatori progettano e gli utenti usano tecnologie web per compiti e funzioni che prima si basavano su altre piattaforme, il che comporta per gli utenti accedere sempre più spesso a software e dati che stanno in rete, mentre prima risiedevano nel pc.

Con il Web 2.0 i siti diventano applicazioni, ovvero hanno una interfaccia, una facilità e velocità d'uso che li rendono simili ad applicazioni desktop.

Nasce così il concetto di Web Services, anche se in realtà sono nati prima della definizione di Web 2.0, ora ne sono parte integrante.

2.2.1 Web Services

Un Web Service è un servizio disponibile in internet che utilizza una sistema di comunicazione tramite messaggi XML e non è legato ad alcun sistema operativo o linguaggio di programmazione.

Lo sviluppo dei server Web che li hanno portati ad essere dei veri Application Server, 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 non significa che gli umani sono interamente fuori dalla figura, ma che queste comunicazioni possono partire direttamente tra applicazioni come avveniva tra browsers e servers.

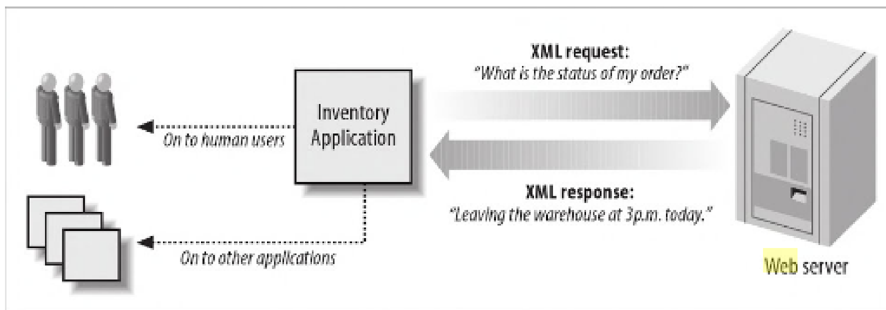

Figura 3: Web Services : application-centric

Con il Web 2.0 si è passati ad 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.

L'architettura si sposta verso un modello orientato ai servizi con comunicazioni basate su protocolli aperti e standard, facili da comporre.

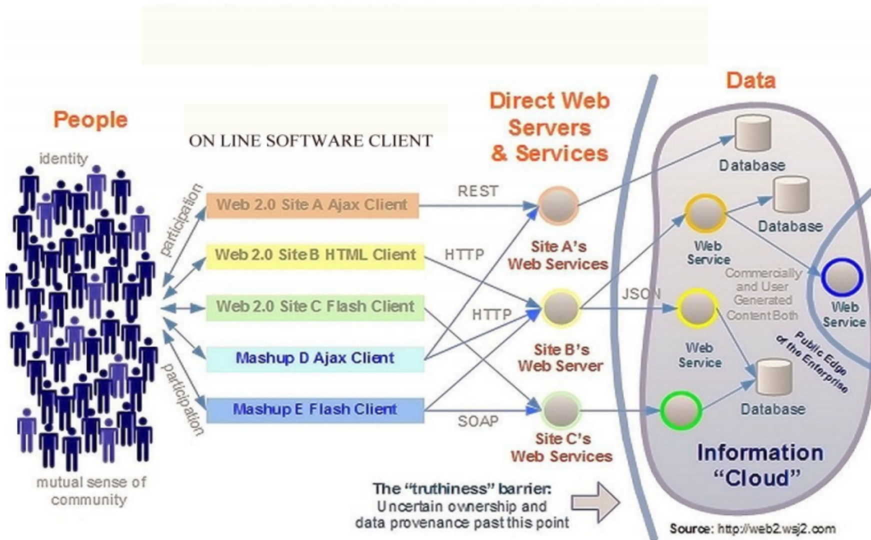

Figura 4: Web 2.0: architettura

I Web Service vengono visti come soluzione tecnologica adatta all'interoperabilità dei sistemi poichè con l'introduzione di diversi linguaggi di programmazione orientati al web, le Web application (API) sono sempre più dipendenti dalla piattaforma con cui vengono sviluppati.

Va altresì sottolineato che il ruolo dei Web Service non si limita solo a un discorso 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.

2.2.2 Web Services: SOAP e REST

Fondamentale per lo scopo di tale tesi è quello di fare una distinzione tra quelli che sono i due approcci per la realizzazione di Web Service: REST e SOAP.

Anche se l'obiettivo dei due approcci è pressochè identico, cioè l'adozione del Web come piattaforma di elaborazione, la loro visione e la soluzione suggerita sono totalmente differenti.

Risulta utile fare un confronto tra i due approcci per comprendere queste differenze.

La prima evidente differenza tra i due tipi di Web Service è la visione del Web proposta come piattaforma di elaborazione. REST propone una visione del Web incentrata sul concetto di risorsa mentre i SOAP Web Service mettono in risalto il concetto di servizio:

- Un Web Service RESTful è custode di un insieme di risorse sulle quali un client può chiedere le operazioni canoniche del protocollo HTTP;
- Un Web Service SOAP espone un insieme di metodi richiamabili da remoto da parte di un client.

L'approccio dei SOAP Web service ha mutuato un'architettura applicativa denominata SOA, Service Oriented Architecture, a cui si è recentemente contrapposta l'architettura ROA, Resource Oriented Architecture, ispirata ai principi REST. [7]

Un servizio SOAP indirizza le richieste sempre verso un unico indirizzo chiamato "end point" e all'interno del messaggio vengono definite le operazioni che vengono parsate dall'application server ed invocate insieme ai dati sempre dichiarati all'interno del messaggio.

Tutte le richieste vengono effettuate sempre con il metodo POST e questo porta ad un maggior traffico verso l'end-point in quanto in tutte le richieste viene sempre inviato un documento XML.

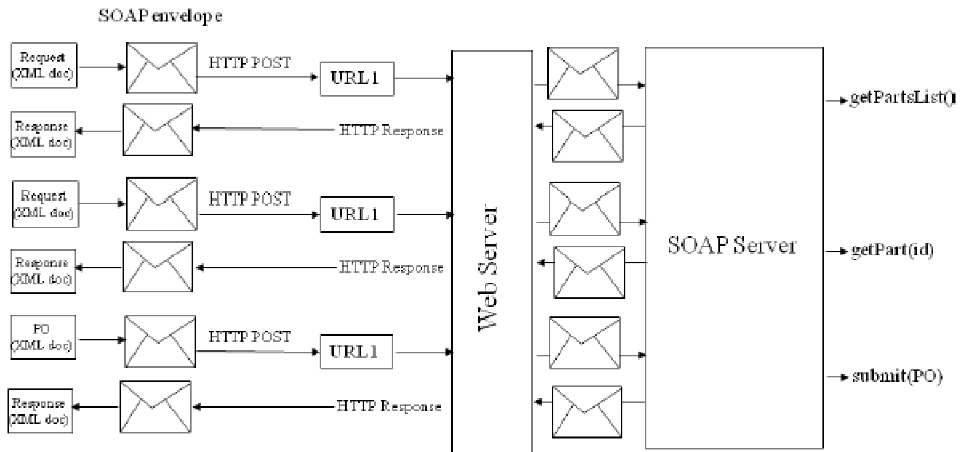

Figura 5: Richiesta SOAP

Nei servizi REST le richieste, 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.

I Web Service basati su SOAP prevedono lo standard WSDL, Web Service Description Language, per definire l'interfaccia di un servizio. Questo comporta che per poter utilizzare 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 tipizzati.

REST non prevede esplicitamente nessuna modalità per descrivere come interagire con una risorsa. Le operazioni sono implicite nel protocollo

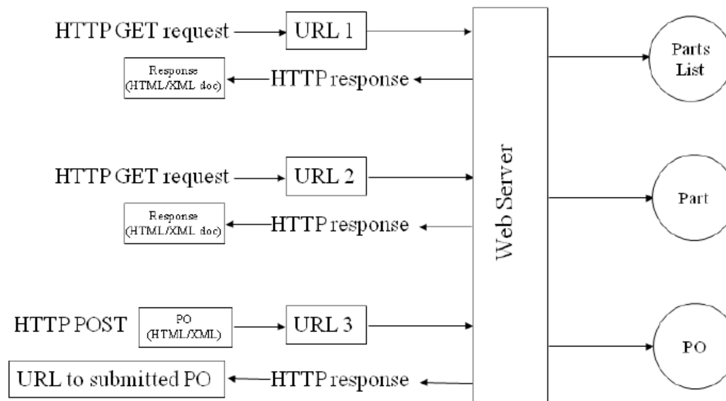

Figura 6: Richiesta REST

HTTP. Qualcosa di analogo a WSDL è WADL, Web Application Definition Language, un'applicazione XML per definire risorse, operazioni ed eccezioni previsti da un Web Service di tipo REST.

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 il metodo POST per comunicare con un servizio e gli apparati senza un'analisi completa dei messaggi, controllandone il contenuto, non si è in grado di predire se è una richiesta che può modificare le informazioni e quindi attuare gli opportuni meccanismi di controllo.

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.

2.3 Web 3.0 - Semantic Web

"Il Web sarà un luogo in cui l'improvvisazione dell'essere umano, e il ragionamento della macchina coesisteranno in una miscela ideal e potente." [2]

Il Web 3.0 è conosciuto anche come semantic web.
"Semantic Web" è stato pensato da Tim Berners-Lee, inventore del World Wide Web.

L'idea base del web 3.0 quello di definire una struttura formata da dati e link che li connettono in modo tale da rendere più efficace la ricerca, l'automazione, integrazione e riuso attraverso diverse applicazioni.

Il Semantic Web cerca di collegare, integrare e analizzare data da diversi data sets per ottenere nuove informazioni. E' capace di migliorare la gestione dei dati, supportare l'accessibilità di internet mobile, simulare creatività e innovazione, incoraggiare fenomeni di globalizzazione, accrescere la soddisfazione dei consumatori e aiutare a organizzare la collaborazione nei social web. [3]

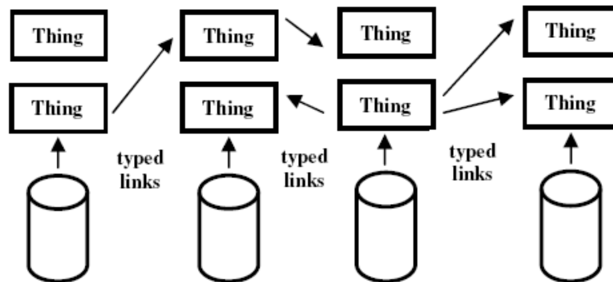

Figura 7: Web of data

Il semantic web è stato sviluppato per superare i problemi del web 2.0 - web di documenti.

Il semantic web può essere definito **web of data** (web dei dati), in un qualche modo visto come un database globale: il suo principale scopo è la macchina principalmente, in seguito gli umani.

L'oggetto principale sono i dati, quindi connettere dati. La semantica dei contenuti e i links sono espliciti e la struttura tra dati è basata su un modello RDF.

Tim Berners-Lee propose una architettura a livelli per il semantic web che viene rappresentato usando un diagramma. La figura sottostante ne da una tipica rappresentazione.

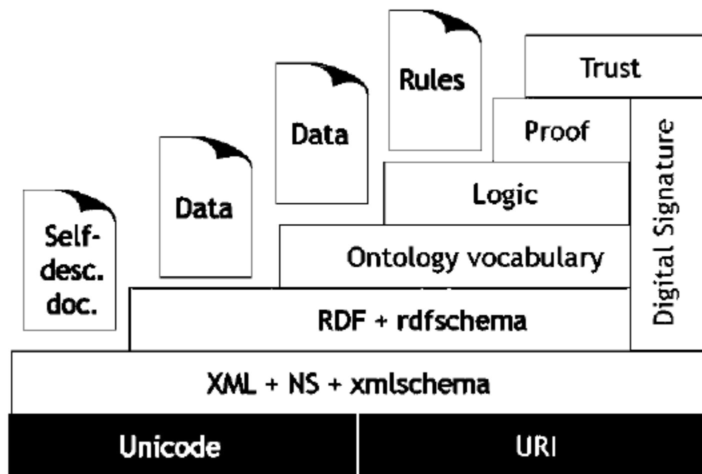

Figura 8: Semantic Web layered architecture [4]

I livelli dell'architettura sono:

- **URI** (Universal Resource Identifier): sono stringhe che vengono utilizzate per definire nomi e indirizzi di risorse astratte o fisiche nel web. Si orientano a creare sintassi di accesso unificata alle risorse di dati disponibili.

Tutte le istruzioni di accesso ai vari specifici dati disponibili secondo un dato protocollo sono codificate come una stringa di indirizzo.

Gli URI sono per definizione:

- **URN** (Uniform Resource Name): sintassi che permette un'etichettatura permanente e non ripudiabile della risorsa. Il nome deve definire un'informazione certa e affidabile sulla esistenza e accessibilità.
- **URL** (Uniform Resource Locator): sintassi che contiene informazioni immediatamente utilizzabili per accedere alla risorsa.

- **Unicode**: fornisce un numero unico per tutti i caratteri, indipendentemente dalla piattaforma sottostante, programmi o linguaggio.

- **XML** (Extensible Markup Language): XML fornisce una sintassi per strutturare i documenti senza imporre vincoli semantici sul significato di tali documenti.

  I Name Space sono usati per identificare e distinguere elementi differenti all'interno dell'XML.

  XML schemas e' un linguaggio che permette di definire lastruttura di un documento XML;

- **RDF** (Resource Description Framework): è un modello per rappresentazione di dati che sfrutta gli URIs per identificare le risorse sul web e per descrivere le relazioni tra le risorse in termini di proprietà e valori.

  E' basato su triple: soggetto - predicato - oggetto.

- **RDF Schema**: fornisce un vocabolario di modellazione di dati per i dati RDF.

  RDF schema è un'estensione semantica di RDF. Esso fornisce meccanismi per descrivere i gruppi di risorse correlate e le relazioni tra queste risorse. [5]

- **Ontologia**: è il modello concettuale della rappresentazione. Più precisamente in informatica, il termine ontologia è definito dall'unione dei seguenti concetti:
  - una rappresentazione di elementi di un dominio;
  - le relazioni che intercorrono tra gli elementi considerati.
- **Logica e Prova**: questi livelli sono al di sopra di quello delle ontologie p si fa riferimento alla necessità di astrarre dal contenuto informativo il senso delle frasi, considerando soltanto la loro forma o struttura. [3]
- **Fiducia**: quest'ultimo livello della pila fornisce una garanzia della qualità dell'informazione nel web e il grado di confidenza nelle risorse fornite. [3]

Il Semantic Web non si limita solo di pubblicare i dati nel web, ma tratta soprattutto di creare links che permettono di connetterli.

Berners-Lee ha introdotto una serie di regole che sono conosciute come principi **Linked Data** per pubblicare e connettere dati nel web:

- Usare gli URI come nomi per i dati;
- Usare URI HTTP per cercare questi nomi;
- Fornire informazioni utili usando gli standard RDF e SPARQL;
- Includere links a altri URIs per trovare maggiori dati.

CAPITOLO 3

I Servizi Nel Semantic Web

3 I SERVIZI NEL SEMANTIC WEB

Gli sforzi verso la crazione di un web semantico stanno prendendo slancio: poter riuscire ad accedere alle risorse web attraverso i contenuti piuttosto che dalle parole chiavi.

Una forza significativa è lo sviluppo di un linguaggio di markup web chiamato OWL. Questi linguaggi permettono la creazione di ontologie per ogni tipo di dominio e permettono di instanziarle nella descrizione dei siti web. [8]

3.1 Ontologie

Tutta l'introduzione sulla trasformazione dal Web 1.0 al 3.0 è servita per poter capire meglio quello che è stato fatto in questa tesi, e per introdurre quello che è il web semantico e quelle che sono le sue tecnologie associate.

In particolare in tale tesi si parla di Ontologie.

Cosa è una ontologia? L'intelligenza artificiale contiene diverse definizioni di ontologia, alcune delle quali contraddicono delle altre.

Figura 9: What is an ontology?

Per lo scopo di tale tesi, una ontologia è una descrizione formale esplicita di concetti nel dominio di un discorso (classi). La proprietà di ogni concetto descrive: caratteristiche e attributi dei concetti (slots - a volte chiamati ruoli o proprietà) e restrizioni di tali slots.

Una ontologia insieme ad un set di istanze individuali di classi costituiscono una conoscenza base.

Nella realtà, esiste una linea sottile dove finisce l'ontologia e dove inizia la conoscenza di base. [6]

Le **classi** sono l'elemento centrale delle ontologie. Le classi descrivono i concetti nel dominio.

Per esempio: una classe "vini", rappresenta tutti i vini. I vini specifici, sono **istanze** di tali classi.

Una classe può avere delle **subclassi** che rappresentano concetti che sono più specifici rispetto alle superclassi.

Per esempio: possiamo dividere la classe di vini in: vini rossi, bianchi e rosè.

Gli **slots** descrivono proprietà di tali classi e istanze.

In termini pratici, sviluppare una ontologia include:

- definire le classi dell'ontologia,
- organizzare le classi in una tassonomia gerarchica (superclass - subclass),
- definire gli slots e descrivere i valori permessi per tali,
- riempire i valori per gli slot con istanze. [6]

3.2 OWL

Ci sono differenti linguaggi per rappresentare le ontologie.

In particolare in questa tesi esse vengono implementate attraverso il linguaggio standard **OWL** che deriva dal World Wide Web Consortium (W3C).

OWL ha un insieme ricco di operatori, come ad esempio l'intersezione, l'unione e la negazione.

Si basa su un modello logico diverso che consente di definire e descrivere i concetti. I concetti complessi possono quindi essere costruiti in definizioni su concetti semplici.

Inoltre, il modello logico consente l'uso di un reasoner che può verificare se tutte le affermazioni e le definizioni dell'ontologia siano coerenti e possono anche riconoscere quali concetti si inseriscono in base a tali definizioni.

Il reasoner può pertanto contribuire a mantenere in modo corretto la gerarchia. Ciò è particolarmente utile quando si tratta di casi in cui le classi possono avere più genitori.

Una ontologia OWL, è formata da: classi, individuals e proprietà. [10]

3.2.1 Individuals

Gli **individuals** rappresentano oggetti del dominio di riferimento ai quali siamo interessati.

Essi sono conosciuti anche come istanze. Essi possono essere definiti anche come "istanze delle classi".

La figura sottostante dimostra una rappresentazione di alcuni individuals all'interno di un dominio.

Figura 10: Rappresentazione di Individuals [10]

3.2.2 Proprietà

Le **proprietà** sono relazioni binarie tra individuals: esse possono collegare due individuals insieme.

Esse hanno tre caratteristiche:

- Possono avere un inverso;
- Possono avere il limite di riferirsi ad un solo valore;
- Possono essere transitive o simmetriche.

Le proprietà sono l'equivalente di quelli che chiamavamo nella sezione generale delle ontologie "slots".

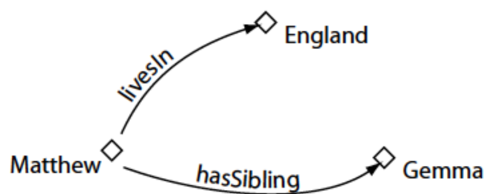

Figura 11: Rappresentazione delle Proprietà [10]

Ci sono tre tipi di proprietà: object properties, datatype properties e annotation properties.

Le **Object Properties** rappresentano il collegamento tra due individuals.

Le **Datatype Properties** collegano un individual a un XML Schema Datatype value o a un rdf literal. In altre parole, descrivono la relazione tra un individual e un valore.

Infine, le **Annotation Properties**, vengono utilizzate per aggiungere informazioni alle classi, individuals e alle object/datatype properties.

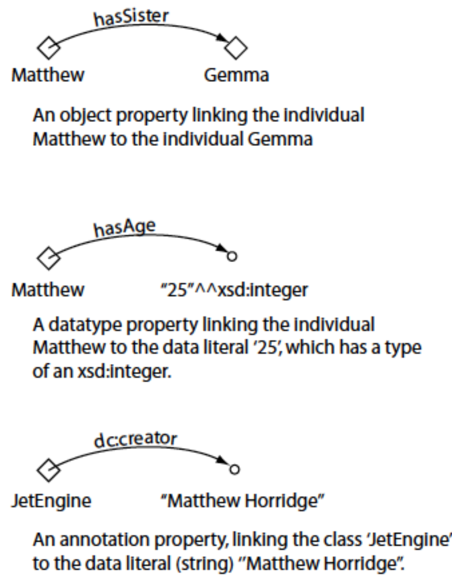

Figura 12: Le diverse proprietà di OWL [10]

3.2.3 Classi

Le classi sono intese come un contenitore che contiene degli individuals.

La parola "concetto" a volte viene utilizzata al posto di classe, in quanto le classi sono rappresentazioni concrete di quello che sono i concetti.

Le classi possono essere organizzate in super-classi e sub-classi che sono anche conosciute come "tassonomie".

Le sub-classi specializzano le loro super-classi. [10]

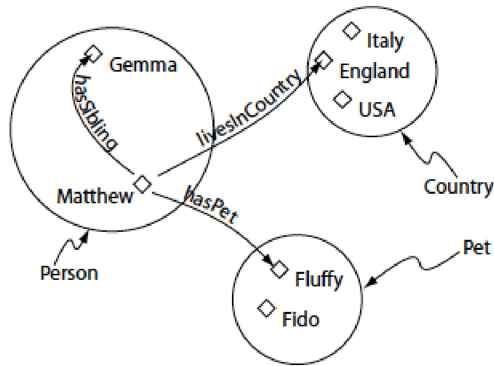

Figura 13: Rappresentazione delle Classi (contenenti Individuals) [10]

3.3 OWL-S

Gli utenti e gli agenti software dovrebbero permettere un accesso non solo al contenuto ma anche ai servizi del web. Essi dovrebbero essere in grado di scoprire, invocare, comporre e monitorare le risorse Web offerte da particolari servizi e che hanno determinate proprietà; inoltre dovrebbero essere capaci di di farlo con un alto grado di automazione se desiderato.

OWL-S è una ontologia di servizi che permette che queste funzionalità siano possibili.

Per utilizzare un web service, un agente software necessita di una descrizione machine-interpretable del servizio, e che il significato di esso sia accessibile.

Un importante obiettivo di OWL-S è quello di stabilire un framework all'interno del quale sono create e condivise queste descrizioni.

I Web Service dovrebbero essere in grado di implementare una ontologia standars, formata da classi e proprietà per descrivere e dichiarare servizi. OWL è l'ontologia che fornisce una struttura linguistica di rappresentazione compatibile con il web all'interno del quale fare tutto ciò.[8]

Si possono considerare due tipi di servizi: atomici o compositi. Gli **atomic services** sono quei servizi in cui un singolo programma, sensore o dispositivo accessibile a un Web viene richiamato da un messaggio di richiesta, esegue il suo compito e produce una sola risposta al richiedente.
Con i servizi atomici non c'è una continua interazione tra servizio e utente.

I **composite services** sono composti da molteplici servizi primitivi e potrebbero richiedere una estesa interazione o conversazione tra il richiedente e il set di servizi che saranno utilizzati.

OWL supporta entrambi le categorie di servizi.

La struttura delle ontologie di servizi è motivata dalla necessità di fornire tre tipi di conoscenze sui servizi che sono caratterizzate da tre domande:

- Cosa fornisce il servizo ai clienti?
- Come viene utilizzato?
- Come si interagisce con esso?

Queste tre domande vengono risposte dalle tre sub ontologie incluse in OWL-S, che sono: Service Profile, Service Model e Service Grounding.

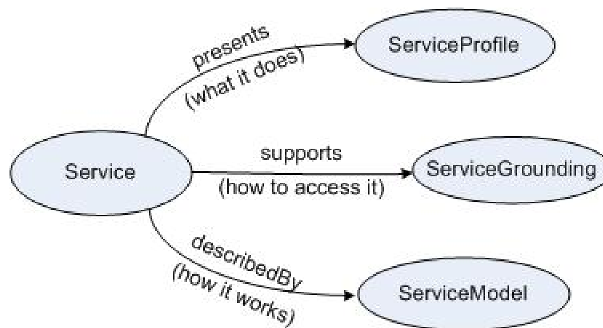

Figura 14: Service Ontology

Nella figura viene rappresentata la relazione tra le classi di alto livello dell'ontologia.

L'ovale rappresenta una classe OWL e l'arco rappresenta la proprietà OWL. [9]

Un **Service Profile** indica "quello che il servizio fa", in modo tale che l'agente che effettua una ricerca di servizi possa determinare se tale sia quello di cui necessita.

Questa forma di rappresentazione include una descrizione di quello che viene compiuto dal servizio, limitazioni e qualità del servizio, e requisiti dei

quali il servizio necessita per utilizzarlo in modo corretto.

Un **Service Model** spiega al client come utilizzare il servizio, dettagliando il contenuto semantico delle richieste, le condizioni necessarie per far si che un risultato si verifichi, e, quando necessario, i processi step by step per determinare tali risultati.

Descrive come richiamare tale servizio e cosa succede quando il servizio viene portato a termine.

Un **Service Grounding** specifica i dettagli di come un agente può accedere a tale servizio.

Solitamente un grounding specifica:

- protocollo di comunicazione
- il formato dei messaggi
- altri dettagli specifici del servizio.

Inoltre, il grounding deve specificare, per ogni tipo semantico riguardante gli input e gli output specificati nel Service Model, un modo non ambiguo per scambiare dati di quel tipo con il servizio.

Sono presenti delle cardinalità da rispettare quali:

- un servizio può essere descritto da al massimo un Service Model;
- un Grounding può essere associato al massimo a un servizio.

Non è specificata una cardinalità minima per le proprietà "presents" e "described by".

Non è specificata, inoltre, una cardinalità massima per le proprietà "presents" e "supports", quindi un servizio può avere più profili o grounding. [8]

Questo punto è fondamentale, in quanto nella tesi proposta ci si focalizza soprattutto sull'idea di creare due grounding per servizio, uno che rappresenta il servizio con interfaccia SOAP e l'altro che rappresenta quella

REST.

Ogni servizio descritto attraverso OWL-S è rappresentato da un'istanza della classe OWL "Service", che possiede le proprietà per associare se stessa ad un "Process Model", a uno o più "Grounding" e opzionalmente a uno o più "Profile".

3.3.1 Service Profile

Una transazione all'interno di un web service si compone di tre parti: il richiedente del servizio, l'erogatore del servizio e i componenti dell'infrastruttura.

Il richiedente del servizio, che si può identificare come l'acquirente, che ricerca il servizio.

Il fornitore del servizio, che si può identificare come il venditore, che fornisce il servizio ricercato dal richiedente.

In un ambiente come internet, il richiedente non conosce prima del tempo l'esistenza del fornitore, perciò esso fa affidamento sui componenti dell'infrastruttura che lavorano come registri per individuare il fornitore appropriato. [8]

All'interno di OWL-S quindi, un Service Profile, fornisce una metodologia per descrivere i servizi offerti dai fornitori e i servizi di cui necessitano i richiedenti. [8]

La struttura generale del Profile, è rappresentata dalla Figura sottostante (Figura 15).

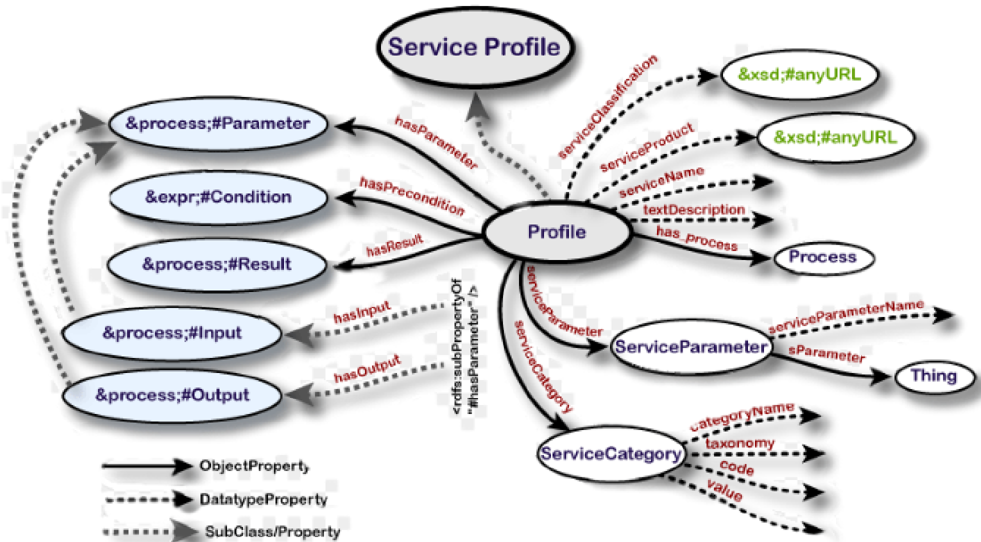

Figura 15: Structure of the OWL-S Profile

Un Profile, descrive un servizio come una insieme di tre tipi di aspetti: quello funzionale, quello di classificazione ed infine quello non funzionale.

L'**aspetto funzionale** del servizio, specifica gli input, gli output, le precondizioni e gli effetti. Esso rappresenta:

- la trasformazione compiuta dalle informazioni durante in servizio: dagli input che il servizio si aspetta agli output generati;
- la trasformazione nel dominio, da un set di precondizioni che il servizio necessita per essere eseguito correttamente, agli effetti che produce durante la sua esecuzione.

Un esempio potrebbe essere l'acquisto online: gli inputs sono i nomi dei prodotti desiderati e i dati della carta di credito; l'output è la ricevuta della vendita; la precondizione è che la carta di credito sia valida ed infine gli

effetti prevedono che la carta di credito venga addebitata e le merci spedite al cliente.

L'**aspetto di classificazione**, supporta la descrizione del tipo di servizio.

La classificazione del servizio è indicata dal tipo di profilo e / o dalle categorie di servizi. Utilizzando questi elementi, il fornitore di servizi può specificare esplicitamente quale tipo di servizio esso fornisce e il client può specificare il tipo di servizio che necessita.

Infine il Profile presenta un **aspetto non funzionale**, che fa distinzione tra i servizi che fanno la stessa cosa ma in modi diversi o con differenti caratteristiche di prestazione.

Esempi di aspetti non funzionali di un servizio comprendono i requisiti di sicurezza e privacy, la precisione e la tempestività (Quality of Service o QoS) del servizio offerto, la struttura dei costi e gli aspetti della provenienza. [9]

3.3.2 Process Service Model

Una volta che un agente software ha identificato un servizio pertinente con i suoi obiettivi, ha bisogno di un modello dettagliato del servizio per determinare se questo può soddisfare le sue esigenze e, in caso affermativo, quali restrizioni devono essere soddisfatte e quali interazioni sono necessarie per utilizzarlo.

Sostanzialmente il process esplicita come lavora il servizio: specifica le possibili interazioni con i web services.

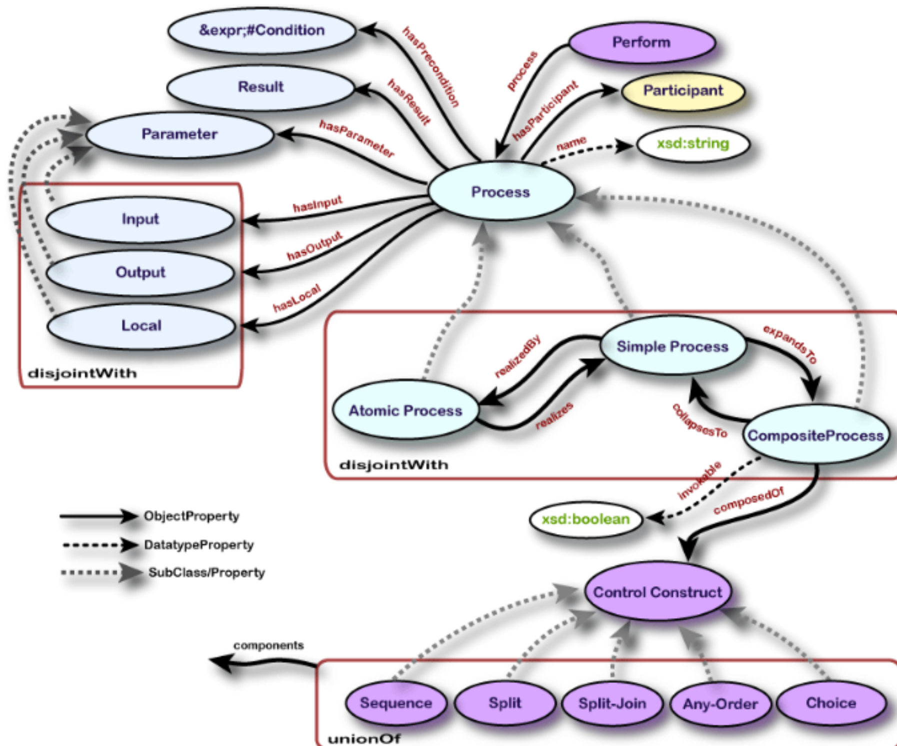

Figura 16: Structure of the OWL-S Process [8]

E' importante spiegare che un processo non è un programma che deve essere eseguito, ma è una specificazione di come il client dovrebbe interagire con il servizio.

Possiamo vedere due tipi di processi che possono essere invocati: atomic, simple e composite process.

Un **atomic process** è la descrizione di un servizio che si aspetta un solo messaggio in input e ritorna un unico messaggio in output (possibilmente entrambi complessi).

E' invocabile direttamente passandogli l'appropriato messaggio. Esso non ha sotto-processi e viene eseguito in un unico step.

Per ogni atomic process deve essere previsto un grounding.

Infine, per un atomic process, ci sempre solo due partecipanti al servizio: "TheClient" e "TheServer".

Un **simple process** non è invocabile e non è associato a un grounding, ma come l'atomic process, essi vengono eseguiti in un unico singolo step.

Vengono usati come elementi di astrazione: un simple process può essere usato sia per procurare una vista di alcuni atomic process, sia per semplificare la rappresentazione di alcuni composite process.

Nel primo caso, il simple process è realizzato da ("realizedBy") un atomic process; nel secondo caso, il simple process si espande ("expandTo") al composite process.

Un **composite process** consiste in un insieme processi collegati tra loro tramite flussi di controllo e strutture di flusso di dati.

Il flusso di controllo viene descritto usando tipici linguaggi di programmazione o i workflow come: sequenze, rami condizionali e cicli.

Il flusso di dati è la descrizione del modo in cui le informazioni vengono acquisite e utilizzate nei passaggi successivi del processo.

3.3.3 Grounding

Il grounding di un servizio specifica i dettagli riguardanti le modalità di accesso al servizio. In particolare, dettagli che riguardano: il protocollo e il formato dei messaggi, trasporto e indirizzamento.

Un grounding può essere pensato come una mappatura da una specificazione astratta a una concreta della descrizione degli elementi richiesti dal servizio per interagire con esso stesso - nel nostro caso, si tratta degli input e degli output di un atomic process.

Sia il ServiceProfile che il ServiceModel (process) sono pensati come rappresentazioni astratte, solo il ServiceGrounding offro un concreto livello di specificazione. [8]

I messaggi concreti, tuttavia, sono specificati esplicitamente nel grounding. La funzione centrale di un grounding OWL-S è quella di mostrare come gli input e gli output (astratti) di un processo atomico devono essere realizzati concretamente come messaggi, che li portano attraverso un determinato formato trasmissibile.

Arrivati fino a qui, bisogna fare una grande distinzione fra grounding, nonchè punto fondamentale di questa tesi:

- Restful Grounding
- Soap Grounding

3.3.4 Soap Grounding

Come detto in precedenza, i Web Service basati su SOAP prevedono lo standard WSDL, per descrivere il servizio stesso.

Cos'è precisamente WSDL?

Web Services Description Language (WSDL) è linguaggio basato su XML utilizzato per descrivere un servizio web.

In particolare viene utilizzato per specificare determinati aspetti dei servizi quali: l'indirizzo fisico del servizio, le operazioni supportate, la tipologia dei messaggi per ogni operazione.

Mediante WSDL è quindi possibile descrivere un'interfaccia pubblica di un Web Service che fornisca le informazioni necessarie per poter interagire con un determinato servizio.

Un documento WSDL contiene infatti, 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) ovvero i "vincoli" (bindings) del servizio.
- dove utilizzare il servizio (cosiddetto endpoint del servizio che solitamente corrisponde all'indirizzo - in formato URI - che rende disponibile il Web Service)

Un documento WSDL può essere suddiviso tra definizione logica e concreta.

La prima descrive le interfacce, le operazioni ed i messaggi.

La seconda definisce il trasporto, il binding e gli end point.

Per realizzare una descrizione WSDL bisogna definire tutte le sue parti:

- Tipi (types): ogni messaggio scambiato nell'ambito delle operazioni WSDL deve essere tipato. Il modo più comune utilizatto per definire i tipi utilizzati nei documenti WSDL è il formalismo degli schemi XML. Possono essere definiti i tipi come simpleType e complexType, ma anche sotto forma di elementi.
- Messaggi (message): i messaggi sono alla base della costruzione di un servizio web con WSDL. Sono definiti uno o più elementi message che seguono la sezione types. Ciascun messaggio ha un nome univoco ed è costituito da una o più parti (part) ciascuno con un nome ed un tipo distinti.

- Operazioni (operation): sono le funzionalità esposte dall'interfaccia del servizio, e sono definite all'interno degli elementi operation. Gli elementi operation contengono elementi di input. output e fault che specificano i messaggi scambiati durante l'operazione. Ci sono quattro tipi di operazioni:
  - One-way: il client spedisce un messaggio al servizio.
  - Request-Response: il client spedisce un messaggio al servizio e riceve una risposta.
  - Notification: il servizio invia un messaggio a un client.
  - Solicit-Response: il servizio invia un messaggio al client e quest'ultimo risponde.
- Port-type: la descrizione astratta di una porta, intesa come insieme di operazioni è affidata agli elementi PortType. Ogni port-type ha un nome ed è composta da un insieme di elementi operation rappresentanti le operazioni associate.
- Binding: è l'istanziazione di una porta astratta. Ogni elemento binding contiene un attributo type che si riferisce al particolare portType istanziato. WSDL definisce gli elementi per esprimere vari tipi di binding quali: SOAP, HTTP, MIME.
- Porte (port): una porta è l'istanza di un portType ottenuta tramite un binding. Esse sono specificate tramite l'elemento port.
- Servizi (service): le porte sono dichiarate all'interno di servizi. Service è l'elemento di livello più alto in WSDL. Esso dichiara un servizio web con un particolare nome, che rappresenta il nome del servizio web.

Questa piccola infarinatura sul WSDL è stata necessaria per entrare più nello specifico per quanto riguarda i grounding soap, in quanto sono necessari sia il linguaggio OWL-S che WSDL per una definizione completa di un grounding, poichè questi non coprono lo stesso spazio concettuale.

Come descritto dall'immagine sottostante, i due linguaggi si sovrappongono nel fornire la specifica di ciò che WSDL chiama "tipi astratti", che a

loro volta vengono utlizzati per caratterizzare gli input e gli output dei servizi.

WSDL, per impostazione predefinita, specifica tipi astratti utilizzando XML Schema, mentre OWL-S consente la definizione di tipi astratti come classi OWL.

Tuttavia, WSDL / XSD non è in grado di esprimere la semantica come una classe OWL. Allo stesso modo, OWL-s non ha alcun mezzo per esprimere le informazioni che WSDL invece fornisce.

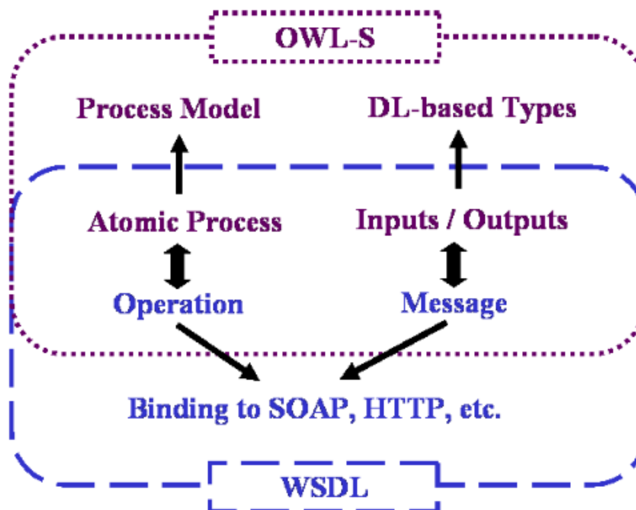

Figura 17: Mapping tra OWL-S e WSDL [8]

Un grounding OWL-S/WSDL si basa su le seguenti tre corrispondenze tra OWL-S e WSDL.

1. Un OWL-S atomic process corrisponde a un "operation" di WSDL.
2. Gli insiemi di input e output di un atomic process in OWL corrispondono al concetto di "message" in WSDL. Ogni input e output di OWL devono corrispondere a ogni message part di WSDL.

3. I tipi di input e output in OWL-S all'interno di un atomic process (classi) corrispondono alla nozione di "abstract type" in WSDL.

Rimane da definire i meccanismi attraverso i quali i costrutti WSDL vengono referenziati in OWL-S.

La classe *WsdlGrounding*, che è una subclasse di *Grounding*, serve proprio per questo obiettivo.

Ogni istanza di *WsdlGrounding* contiene una lista di istanze *WsdlAtomicProcessGrounding*.

Un'istanza di *WsdlAtomicProcessGrounding* si riferisce a specifici elementi presenti all'interno del documento WSDL usando le seguenti proprietà:

- *wsdlVersion* : riporta un URI che indica la versione di WSDL in uso.
- *wsdlDocument* : riporta l'URI del documento WSDL al quale si riferisce il grounding.
- *wsdlOperation* : riporta l'URI dell'operazione WSDL che corrisponde all'atomic process in questione.
- *wsdlService wsdlPort* : riporta l'URI del servizio WSDL che offre l'operazione in questione.
- *wsdlInputMessage*: è un oggetto che contiene l'URI della definizione del messaggio WSDL che contiene gli input necessari per tale atomic process.
- *wsdlInput*: oggetto che contiene il mapping dei parametri in input dell'operazione. Contiene una coppia di elementi *wsdlMessagePart* e *owlsParamaters* che rispettivamente indicano l'elemento in input definito nel WSDL (il part del message) e l'elemento input definito nel grounding OWL-S. Ogni coppia viene rappresentata utilizzando un'istanza di *WsdlInputMessageMap*.
- *wsdlOutputMessage*: simile a *wsdlInputMessage* ma riguardante gli outputs.

- *wsdlOutput*: simile a *wsdlInput* ma per gli output. In questo caso ogni coppia però è rappresentata da un'istanza di *WsdlOutputMessageMap*. [8]

3.3.5 Grounding RESTful

Con lo sviluppo del web e delle nuove tecnologie, è nato quella che è la nuova architettura REST.

I servizi REST formano la maggiorparte dei servizi web sviluppati all'interno del contesto del Web 2.0. E' stato così necessario di parlare anche di Grounding connessi a servizi web REST.

In primo luogo, OWL-S è usato come base delle ontologie di servizi anche in questo caso, e invece WADL è linguaggio che viene utilizzato per descrivere sintatticamente il servizio.

In secondo luogo, il protocollo HTTP è usato per il trasferimento dei messaggi, definendo le azioni che vengono eseguite e definendo lo scopo di tale esecuzione.

Infine sono presenti degli identificatori rappresentati da URI responsabili di definire l'interfaccia del servizio. [11]

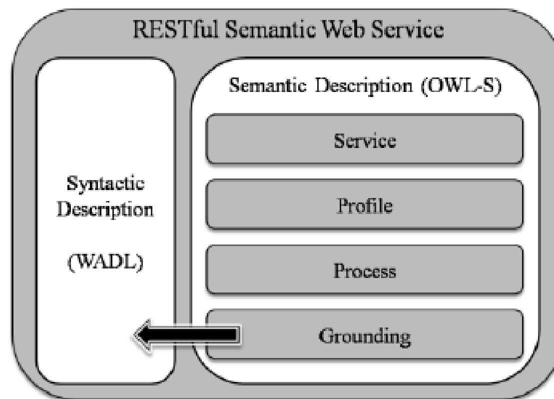

Figura 18: Struttura di un web service REST

Facciamo prima una piccola introduzione a WADL.

Il **Web Application Description Language - WADL** è un vocabolario XML che consente di descrivere le caratteristiche di una risorsa che un servizio RESTful espone. Questo linguaggio ha molte analogie con il WSDL utilizzato nei servizi SOAP, partendo da una somiglianza con il nome fino ad arrivare al risultato che si ottiene applicandolo.

Il WADL non è ancora utilizzato in maniera massiva per poterlo indicare come uno standard, a differenza del WSDL, poichè la maggior parte dei servizi RESTful vengono descritti in HTML perchè non sono pensati per una elaborazione automatizzata, e l'utilizzo del WADL consisterebbe nel convertire tutte le descrizioni in questo linguaggio, cosa che va contro i principi di semplicità d'utilizzo del servizio REST.

Per ogni risorsa che il web service tratta, può essere specificato un WADL e oltre alle caratteristiche sintattiche di esso, ne descrive la modalità con cui si può manipolarla. Supporta la descrizione dell'URI-Template e tutti i metodi HTTP.
Una caratteristica molto importante che si mette in luce del WADL, è quella di favorire la descrizione del formato di rappresentazione della risorsa andando più in là della solita coppia chiave/valore.
Si possono descrivere diversi formati di rappresentazione, ad esempio si può descrivere una rappresentazione in forma XML facendo riferimento al suo Xml-Schema.

Il file ottenuto dal WADL per descrivere un servizio, si può suddividerlo in tre sezioni: la definizione della risorsa, la definizione del metodo utilizzato e la definizione della rappresentazione.

Uno schema semplice di una descrizione di un servizio utilizzando WADL ha questa conformazione:

Listing 1: Sample Schema WADL

```
<application>
     <resources>
          <resource>
               <method>
                    <request />

                    <response />
                </method>
          </resource>
     </resources>
</application>
```

Esaminiamo tutti gli elementi di tale schema:

- Resources: questo elemento contiene un attributo "base" che va ad indicare l'URI si riferisce all'end point REST, cioè il percorso da cui partire per raggiungere le risorse. Può essere aggiunto all'interno di tale elemento, opzionalmente, una breve descrizione del servizio.

- Resource: ogni elemento *resources* contiene elementi *resource* che contengono il path relativo della risorsa. Concatenendo il valore dell'attributo base di resources e il path di resource, si ricava il path completo della risorsa.

- Method: i metodi associati a una risorsa REST sono descritti nell'elemento method. Un elemento *resource* può avere uno o più elementi *method*. In tale elemento viene identificato il nome univoco del metodo (tramite un attributo *id*) e viene evidenziata il metodo del protocollo HTTP applicato a tale risorsa (tramite un attributo *name*).

- Request: un elemento *request* è usato per descrivere gli input del metodo. L'elemento request contiene uno o più elementi *param* che sono usati er definire i parametri input previsti per effettuare la richiesta HTTP.

- Response: l'elemento *response* viene utilizzato per descrivere gli output della risposta HTTP.

- Representation: sia per la richiesta che per la risposta HTTP è previsto un elemento *representation* che specifica il body degli input/output della richiesta/risposta HTTP. Viene utilizzato anche per specificare il mediaType previsto per tali richieste/risposte.

Listing 2: Esempio file WADL description for Yahoo News Search [12]

```
<?xml version="1.0"?>
 <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
   xmlns:tns="urn:yahoo:yn"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:yn="urn:yahoo:yn"
   xmlns:ya="urn:yahoo:api"
   xmlns="http://wadl.dev.java.net/2009/02">
    <grammars>
     <include
       href="NewsSearchResponse.xsd"/>
     <include
       href="Error.xsd"/>
   </grammars>

   <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
     <resource path="newsSearch">
       <method name="GET" id="search">
         <request>
           <param name="appid" type="xsd:string"
             style="query" required="true"/>
           <param name="query" type="xsd:string"
             style="query" required="true"/>
           <param name="type" style="query" default="all">
             <option value="all"/>
             <option value="any"/>
             <option value="phrase"/>
           </param>
           <param name="results" style="query" type="xsd:int" default="10"/>
           <param name="start" style="query" type="xsd:int" default="1"/>
           <param name="sort" style="query" default="rank">
             <option value="rank"/>
             <option value="date"/>
           </param>
           <param name="language" style="query" type="xsd:string"/>
         </request>
         <response status="200">
           <representation mediaType="application/xml"
             element="yn:ResultSet"/>
         </response>
         <response status="400">
           <representation mediaType="application/xml"
             element="ya:Error"/>
         </response>
      </method>
     </resource>
   </resources>
</application>
```

Come è stato fatto per i soap grounding, è stato necessario anche in questo caso fare una piccola introduzione a WADL per poter poi potere

iniziare a parlare nello specifico del RESTful grounding.

Per quanto riguarda il servizio REST abbiamo appena finito di descrivere un linguaggio che è paragonabile al WSDL, ma vogliamo anche annotare questo nuovo linguaggio chiamato WADL, con annotazioni semantiche, purtroppo però i concetti ontologici che descrivono il modo con cui si interagisce con il servizio, non sono gli stessi per un servizio SOAP e REST, quindi abbiamo qualche difficoltà nell'utilizzare solo l'ontologia classica OWL-S anche per annotare un WADL.

Per questo motivo si è studiato l'estensione di OWL-S soprattutto per le istanze del supports che definisce le modalità di iterazione di un servizio REST.

Questa nuova estensione è stata definita dai professori Otávio Freitas Ferreira Filho e Maria Alice Grigas Varella Ferreira dell'università di São Paulo e descritta nell'articolo "Semantic Web Service: A Restful Approach". [11]

Un Grounding OWL-S/WADL estende uno strato astratto composto da due classi OWL-S, chiamate Grounding e AtomicProcessGrounding.

La figura sottostante è un diagramma delle classi UML che rappresenta questa estensione, e visualizza che le tre ontologie, Service,Profile e Process, che sono definite nella classica OWL-S, non sono state modificate ma rimangono sempre le stesse, e la figura vuole rafforzare l'idea che la nuova ontologia RESTfulGrounding non sostituisce il Grounding ma ne estende le potenzialità.

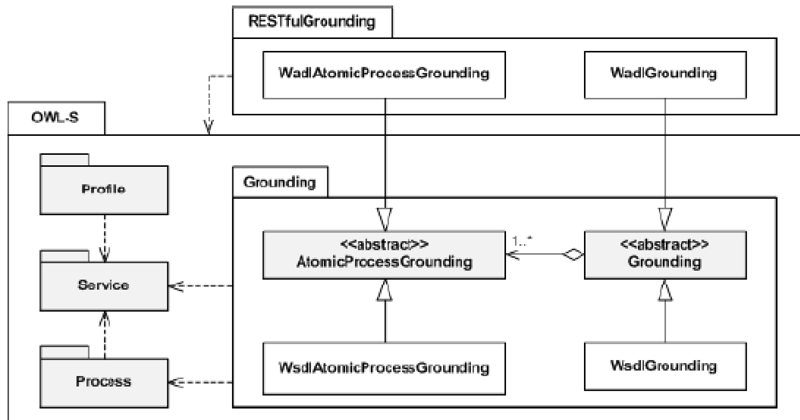

Figura 19: Estensione di OWL-S per OWL-S/WADL Grounding

Andiamo a trattare, anche in questo caso, in particolare le varie entità:

- *WadlGrounding*: è l'entità che estende la classe astratta Grounding.
- *WadlAtomicProcessGrounding*: questa entità è responsabile del mapping tra l'atomic process di OWL-S e la coppia WADL composta dalla risorsa richiesta e il metodo HTTP utilizzato. E' permessa una sola coppia risorsa/metodo per ogni atomic process.
- *WadlResourceMethodRef*: con questa classe si definiscono esattamente la risorsa e il metodo HTTP utilizzato, grazie alle due proprietà *resource* e *method*.
- *WadlRequestParam*: questa classe riferisce un parametro input "owl-s" a un parametro di richiesta WADL. Questa classe estende due classi: *WadlMessageParamMap* e *InputMessageMap*.
- *WadlResponseParamMap*: si occupa di mappare i message parameters delle risposte HTTP.

Listing 3: Esempio file Grounding RESTful per Yahoo News Search [12]

```
<!-- WADL Grounding -->

<restful:WadlGrounding rdf:ID="YNS-Grounding">
  <grounding:hasAtomicProcessGrounding
                                  rdf:resource="#YNS-AtomicProcessGrounding"/>
</restful:WadlGrounding>


<!-- WADL Atomic Process Grounding -->

<restful:WadlAtomicProcessGrounding rdf:ID="YNS-AtomicProcessGrounding">

  <grounding:owlsProcess rdf:resource="&ypc;#YNS-AtomicProcess"/>

  <restful:wadlResourceMethod>
    <restful:WadlResourceMethodRef>
      <restful:resource rdf:datatype="&xsd;#anyURI">
                &wadl;#YNS-Resource
      </restful:resource>
      <restful:method rdf:datatype="&xsd;#anyURI">
                &wadl;#YNS-Method-GET
        </restful:method>
    </restful:WadlResourceMethodRef>
  </restful:wadlResourceMethod>

  <restful:wadlVersion rdf:datatype="&xsd;#anyURI">
    https://wadl.dev.java.net/wadl20090202.xsd
  </restful:wadlVersion>

  <restful:wadlDocument rdf:datatype="&xsd;#anyURI">
    http://search.yahooapis.com/NewsSearchService/V1/YahooNewsSearch.wadl
  </restful:wadlDocument>


  <!-- Request Parameters -->

  <restful:wadlRequestParam>
    <restful:WadlRequestParamMap>
      <grounding:owlsParameter rdf:resource="&ypc;#YNS-AppID"/>
      <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">
                &wadl;#appid
      </restful:wadlMessageParam>
    </restful:WadlRequestParamMap>
  </restful:wadlRequestParam>
  ...
  <!-- Response Parameters -->

  <restful:wadlResponseParam>
    <restful:WadResponseParamMap>
      <grounding:owlsParameter rdf:resource="&ypc;#YNS-Out-Success"/>
      <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">
                &wadl;#YNS-Rep-Success
      </restful:wadlMessageParam>
    </restful:WadResponseParamMap>
  </restful:wadlResponseParam>
    ...

</restful:WadlAtomicProcessGrounding>
```

CAPITOLO 4

Progettazione & Implementazione

4 PROGETTAZIONE & IMPLEMENTAZIONE

La base del semantic web è quello di creare delle ontologie, quindi una descrizione semantica, per tutti quelli che sono i servizi del web, per permettere che essi siano machine-readable, e quindi che possano essere interpretati e riconosciuti direttamente dalle macchine.

Una volta create le ontologie però, è necessario che queste vengano lette e interrogate per essere sfruttate nella loro intera funzionalità.

4.1 Ontologies Manager

Ontologies Manager è il nome della classe principale di tale tesi. Questo manager, come si può dedurre dal nome stesso, si occupa di caricare le ontologie in un triple store, per potere effettuare quindi delle interrogazioni sullo stesso al fine di individuare poi il servizio adatto da richiamare. In questa tesi si trattano solo servizi con atomic process.

L'obiettivo è quindi di individuare il servizio adatto alla richiesta del client, e poterlo poi richiamare in maniera trasparente.

Si possono individuare quattro punti focali di tale manager che sono:

- ricavare il modello RDF di quelli che sono: il servizio, il processo e il profilo del servizio;
- ricavare la descrizione dal profilo del servizio preso in esame ed esaminare se è adatto alla richiesta del client;
- ricavare tutti i dati utili del servizio;
- ricavare dal service i grounding supportati per poter poi richiamare il servizio.

Ontologies Manager è una classe che ha un costruttore che richiede come parametri i modelli RDF del profilo, del servizio e del processo.

Listing 4: Costruttore Ontologies Manager

```
//costruttore
public OntologiesManager(Model profile, Model service, Model process){
        this.profile = profile;
        this.process = process;
        this.service = service;
}
```

I modelli RDF sono formati da tre elementi: risorse, proprietà e affermazioni. Le risorse sono entità riferite attraverso URI. Le proprietà sono relazioni binarie tra risorse e/o valori atomici ti tipo primitivo. Le affermazioni specificano il valore di una certa proprietà relativa a una risorsa.

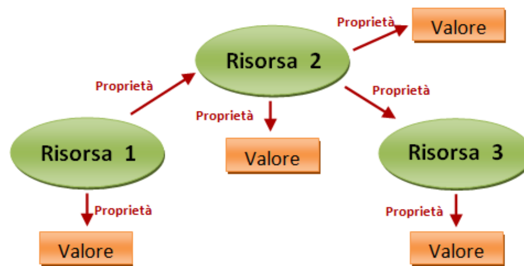

Figura 20: RDF Data Model

E' necessario che tali ontologie vengano salvate all'interno di questi triple stores (modelli), per potere effettuare su di essi delle interrogazioni attraverso query SPARQL.

Listing 5: Costruttore Ontologies Manager

```
Model profile = FileManager.get().loadModel("Profile.owl");
Model service = FileManager.get().loadModel("Service.owl");
Model process = FileManager.get().loadModel("Process.owl");
```

SPARQL è un linguaggio di interrogazione per dataset RDF, si basa su un costrutto detto "triple pattern", che ricalca la configurazione a triple delle asserzioni RDF:

?soggetto ?predicato ?oggetto

Figura 21: SPARQL triple pattern

Il primo punto di tale classe è quello quindi di controllare la compatibilità del servizio preso in esame e la richiesta del client.

Questo è possibile attraverso un metodo chiamato *"checkService"* che si occupa di controllare la compatibilità che è presente tra la descrizione presente nel *Profile* del servizio stesso e la richiesta.

Viene richiamato il model del Profile e viene interrogato attraverso una query SPARQL per ricavarne la descrizione.

Listing 6: Query SPARQL - Descrizione Servizio

```
String queryProfileString =
        "PREFIX Profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>"+
        "SELECT * WHERE { "+
        "?profile Profile:textDescription ?x ."+
        "}";

Query queryProfile = QueryFactory.create(queryProfileString);
QueryExecution qexec = QueryExecutionFactory.create(queryProfile, profile);
Literal descrizioneResource = null;
try {

        ResultSet descrizione = qexec.execSelect();
        while( descrizione.hasNext() ){
                QuerySolution sol = descrizione.nextSolution();
                descrizioneResource = sol.getLiteral("x");
        }

} finally {
        qexec.close();
}
```

Una volta che è stato ricavata la descrizione del servizio, ne viene controllata la compatibilità.

Questo passaggio viene effettuato in maniera molto semplice, facendo un controllo tra le parole inserite dal client nella richiesta di servizio e le parole presenti nella descrizione del servizio.

Una volta controllata e verificata la compatibilità, viene ricavato il nome del servizio di interesse, sempre eseguendo una query sparql che interroga il model Profile.

Il nome del servizio è necessario per poter individuare, in seguito, l'*atomic process* specifico da dover richiamare.

Listing 7: Query SPARQL - Nome Servizio

```
//se c'è riscontro tra richiesta e descrizione
while(matcher.find()) {

        //creo la query sparql che mi ritorna il nome del servizio
        String queryServiceString =
        "PREFIX Profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>"+
                "SELECT * WHERE { "+
                "?profile Profile:serviceName ?x."+
                "}";

        [...]

        //una volta eseguita la query, prendo il valore dell'oggetto "?x"
        QuerySolution sol = nameService . nextSolution ();
        Literal servicename = sol . getLiteral ("x");

}
```

Una volta ricavato il servizio e, nel caso della nostra tesi, avendo solo un atomic process, il processo da richiamare, è necessario ricavarne gli input necessari.

Questo viene effettuato utilizzando il model del *"Process"* del servizio.

Viene interrogato il Process, attraverso query SPARQL, individuando quelli che sono gli oggetti del predicato *"hasInput"* del soggetto process.

Passaggio necessario per ricavare gli input da richiedere al client, per poter poi invocare il processo.

Listing 8: Query SPARQL - Input Processo

```
//creo e gestisco le query SPARQL
//prende gli input del servizio
String queryInputsString =
        "PREFIX Process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>"+
        "SELECT * WHERE { "+
        "?process Process:hasInput ?x ."+
        "}";

[...]

//una volta eseguita la query, prendo il valore dell'oggetto "?x"
//esso rappresenta i miei input.
QuerySolution sol = inputs.nextSolution();
Resource inputsResource = sol.getResource("x");
```

Una volta ricavate tutte le informazioni base per il processo (precondizioni verificate, input) si passa alla parte più importante per richiamare il servizio: controllare i grounding presenti per il servizio.

Per controllare se sono presenti grounding e quanti ne sono presenti per tale servizio, è necessario interrogare il model del *Service*. E' necessario individuare gli oggetti che sono connessi al soggetto *service* attraverso il predicato *"supports"* (in quanto un servizio può supportare - supports - uno o più grounding).

Questo controllo viene effettuato in un metodo chiamato *"checkGrounding"* dell'Ontologies Manager.

Listing 9: Query SPARQL - Grounding Supportati

```
//controllo i grounding supportati
String queryGroundingString =
        "PREFIX Service: <http://www.daml.org/services/owl-s/1.2/Service.owl#>"+
        "SELECT * WHERE { "+
        "?service Service:supports ?x ."+
        "}";

[...]

//una volta eseguita la query, prendo il valore dell'oggetto "?x"
//esso rappresenta i miei input.
QuerySolution sol = inputs.nextSolution();
Resource groundingResource = sol.getResource("x");
```

Una volta individuati il/i grounding supportati dal Service, si procede alla gestione degli stessi.

Come già anticipato nell'introduzione, questa tesi si pone di poter trattare due tipi di grounding: restful e soap.

Viene creato quindi un metodo chiamato *"manageGrounding"* che si occupa di istanziare e gestire oggetti Grounding, ricavando le informazioni necessarie per richiamare il servizio dall'ontologia relativa al/ai grounding ricavati.

Tale metodo viene richiamato all'interno del metodo *"checkGrounding"*.

Possiamo immaginarci i Grounding implementati sulla base di una gerarchia di classi descritta dalla figura sottostante.

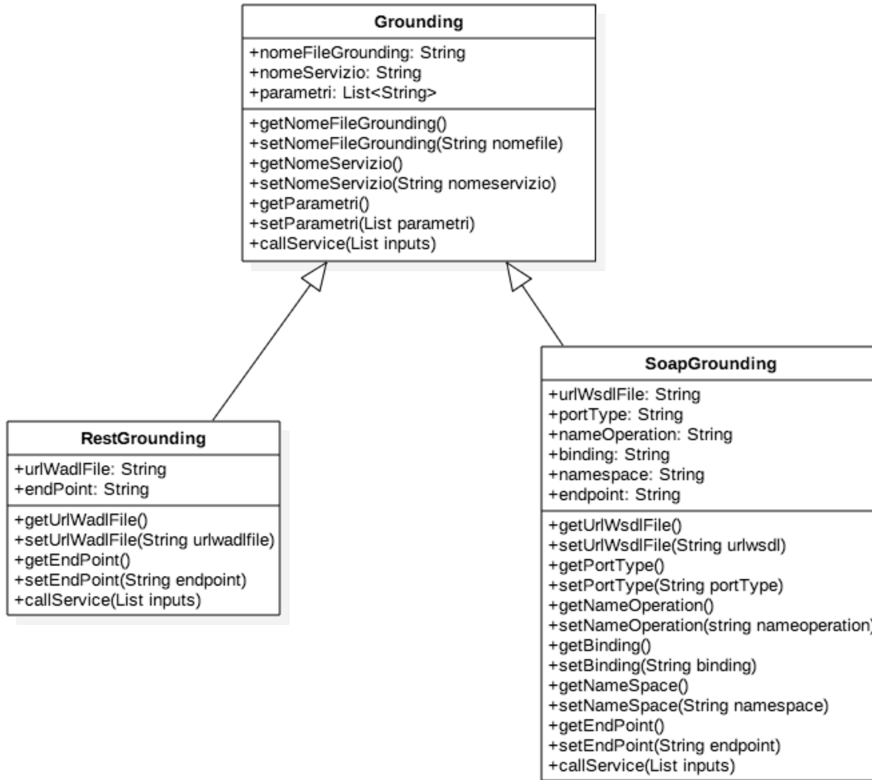

Figura 22: Gerarchia Grounding

Questo tipo di struttura ci permette di implementare uno degli obiettivi che ci siamo posti all'inizio di tale tesi: invocare un tipo di servizio, che sia REST o SOAP, in maniera trasparente.

La superclasse *Grounding* viene estesa dalle due classi *GroundingSoap* e *GroundingRest*.

Nelle due sottoclassi, di particolare importanza, viene sovrascritto il metodo principale: *"callService"*.

Questo metodo prende come input una lista che si riferisce agli input ricavati dal client che viene sovrascritto da entrambe le classi, in modo tale da avere la stessa interfaccia, ma che effettua una chiamata diversa (SOAP o REST) in base al tipo di oggetto che la invoca.
Ciò, permette di rispettare il principio di trasparenza che ci eravamo imposti.

Ritornando alla classe Ontologies Manager e in particolare al metodo *manageGrounding*, il primo compito è quello di individuare che tipo di grounding sono quelli che vengono supportati dal service.

Viene quindi interrogato il file owl grounding preso in esame, ricavato dal service.

Viene controllato se è stato importata l'ontologia che rappresenta i grounding rest, nel caso positivo, viene creato un oggetto grounding di tipo restful; nel caso negativo, viene creato un oggetto grounding di tipo soap.

Listing 10: Query SPARQL - Controllo Tipologia Grounding

```
String queryGroundingString =
        "PREFIX owl: <http://www.w3.org/2002/07/owl#>"+
        "SELECT * WHERE { "+
        "?x owl:imports
               <http://fullsemanticweb.com/ontology/RESTfulGrounding/
                                                v1.0/RESTfulGrounding.owl>."+
        "}";

 [...]

//eseguo query
ResultSet grounding = qexec.execSelect();

//una volta eseguita la query, controllo se è presente un risultato
if( grounding.hasNext() ){

        //se è presente si crea grounding rest
        RestfulGrounding restfulgrounding = new RestfulGrounding();

        [...]

} else {

        //se non è presente, si crea grounding soap
        SoapGrounding soapgrounding = new SoapGrounding();

        [...]
}
```

Una volta creato l'oggetto, vengono estrapolati dall'ontologia tutti i dati fondamentali per la chiamata del servizio quali:

- per il servizio REST:
  - URL del file WADL;
  - URL dell'endpoint.
- per il servizio SOAP:
  - URL del file WSLD;
  - il nome del port type;
  - il nome dell'operazione da richiamare;
  - il nome che rappresenta il binding;
  - il name space;
  - URL dell'endpoint.

Tali dati vengono ricavati interrogando le ontologie attraverso query SPARQL.

Questo metodo ritorna quindi una lista di oggetti Grounding, con i loro relativi dati settati.

Richiamare il servizio sarà quindi ora molto semplice, si potrà utilizzare tali oggetti per poter richiamare il relativo servizio attraverso il metodo *"callService"*.

Essendo un metodo che viene sovrascritto sia nella classe GroundingSoap che GroundingRest, richiamerà il servizio nel modo corretto in base a che tipo di oggetto è presente nella lista dei grounding ritornati.

Listing 11: Esempio CallService()

```
//controllo se e quanti grounding ci sono
List<Grounding> lg = manager.checkGrounding();

String result = "";
int i=0;

//ciclo i grounding da richiamare in base a se il servizio ha riportato
//un risultato che sia diverso da vuoto e finchè sono presenti altri grounding
while(result.equals("") && i<lg.size())
{
        result = lg.get(i).callService(input_client);
        i++;
}

System.out.println("--------- Il risultato è: "+result+" ---------");
```

In conclusione, questo manager permette una lettura e una manipolazione semplice di quelle che sono le ontologie dei servizi e di conseguenza anche la loro chiamata.

4.2 Servizi

Per poter effettuare un lavoro completo, in questa tesi mi sono posta di esaminare e creare un caso completo: mi sono occupata quindi di creare anche i servizi e le ontologie relative, per poter testare il manager creato.

Mi sono occupata di creare due servizi di stessa natura, cioè che svolgessero lo stesso compito, ma con interfaccia diversa: SOAP e REST.

I due servizi creati sono molto semplici e basilari in quanto lo scopo con cui sono stati creati è stato quello di testare il manager.

Questi servizi si chiamano rispettivamente: *CalculatorRest* e *CalculatorSoap*. Come si può dedurre dal nome, si tratta di servizi che si occupano di effettuare calcoli.

Come input vengono richiesti:

- il tipo di operazione da effettuare - somma/divisione/sottrazione/-moltiplicazione
- il primo numero
- il secondo numero

I due servizi sono stati implementati attraverso le due librerie JAX-WS e JAX-RS, rispettivamente per quanto riguarda il servizio SOAP e il servizio RESTful.

4.2.1 JAX-WS - Calculator SOAP service

Iniziamo dal servizio SOAP.

JAX-WS è una Java API per lo sviluppo veloce di applicazioni SOAP. Permette agli sviluppatori di scrivere messaggi RPC-oriented. Una volta creato il servizio in Java viene effettuato il deploy del suo file .war sul server Glassfish. Una volta effettuato il deploy, verrà generato, attraverso annotazioni inserite nel codice, il documento descrittivo WSDL del servizio stesso.

Dal punto di vista Server, gli sviluppatori:

- definiscono le operazioni del web service attraverso metodi all'interno di un'interfaccia Java;
- creano classi che implementano tali interfacce e quindi di conseguenza tali operazioni.

Dal punto di vista del Client:

- viene creato un proxy, oggetto locale che rappresenta il servizio;
- viene invocato il metodo.

Come prima cosa quindi è stata creata l'interfaccia del servizio SOAP:

Listing 12: Interfaccia CalculatorSoap

```
public interface CalculatorInterface {
        public float doOperation(String operation, int numUno, int numDue);
}
```

Il secondo passo è stato quello di implementare all'interno di una classe l'interfaccia.

Java API for XML-Based Web Services (JAX-WS) si basa sull'uso di annotazioni utilizzate per specificare i metadati associati all'implementazione del web service e per semplificarne lo sviluppo.

Le annotazioni descrivono come un'implementazione server-side può essere acceduta come web service oppure come una classe Java client-side può accedere al web service stesso.

Usando le annotazioni all'interno di codice Java, si semplifica lo sviluppo e il deploy di un servizio web, definendo alcune informazioni aggiuntive che sono solitamente ottenute da un file di descrittivo, file WSDL, o mappando metadata da XML e WSDL all'interno di artefatti.

Le annotazioni sono usate per configurare il binding, settare il nome del portType, del servizio e di altri parametri WSDL. Sono usate per mappare attraverso Java lo schema WSDL, e a runtime per controllare come JAX-WS processa e risponde alle invocazioni del web service.

In particolare quindi, il codice di tale classe deve contenere tali annotazioni JAX-WS, necessarie per fornire informazioni utili per la costruzione del file WSDL:

- Il target namespace del servizio;
- La classe che racchiude il messaggio di richiesta;
- La classe che racchiude il messaggio di risposta;
- Se l'operazione è di tipo One-way;
- Il tipo di stile di Binding;
- Il namespace dei tipi usati dal servizio.

Le due annotazioni principali utilizzate sono:

- *@WebService* (richiesto), che definisce che il metodo a seguire rappresenta l'end point del servizio (SEI). All'interno di esso è stato definito il nome del servizio (*serviceName*).
- *@SOAPBinding*, indica il tipo di binding; di default il tipo di binding è Wrapped. Nel nostro caso era necessario che non fosse di tipo Wrapped quindi sono stati definiti lo stile RPC (*style*) e lo stile dei parametri *Bare* (*parameterStyle*).

Listing 13: Implementazione classe CalculatorSoap

```
@WebService(serviceName="CalculatorService")
@SOAPBinding(style=Style.RPC, parameterStyle=SOAPBinding.ParameterStyle.BARE)
public class Calculator implements CalculatorInterface{
        public Calculator(){

        }

        [...]
}
```

Una volta settato il SEI (Service Endpoint Implementation), il passo successivo è quello di definire il metodo del servizio ed aggiungergli le relative annotazioni.

@WebMethod, è l'annotazione che definisce il metodo che rappresenta l'operazione del servizio. E' stato definito il nome dell'operazione (operationName)

Dovendo noi avere i parametri in input del metodo ben definiti con determinati nomi nel documento WSDL, nel metodo Java essi sono stati delineati attraverso l'annotazione *@WebParam*.

Tale annotazione personalizza il mapping di un parametro al "part" del web service message.

Listing 14: Metodo doOperation - CalculatorSoap

```
@WebMethod(operationName="doOperation")
public float doOperation(
        @WebParam(name="operation", partName="OperationType")
                                                String operation,
        @WebParam(name="num1", partName="NumeroUno") int numUno,
        @WebParam(name="num2", partName="NumeroDue") int numDue){

        [...]
}
```

Una volta effettuato il deploy di tale servizio sul server glassfish, il web service sarà utilizzabile da un eventuale client e sarà generato il relativo WSDL.

Il WSDL generato è il seguente:

Listing 15: WSDL CalculatorSoap

```
<definitions
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://calculator.service/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://calculator.service/"
name="CalculatorService">

<types/>

<!-- MESSAGE -->
<message name="doOperation">
        <part name="OperationType" type="xsd:string"/>
        <part name="NumeroUno" type="xsd:int"/>
        <part name="NumeroDue" type="xsd:int"/>
</message>

<message name="doOperationResponse">
        <part name="return" type="xsd:float"/>
</message>

<!-- PORTTYPE -->
<portType name="Calculator">
  <operation name="doOperation"
     parameterOrder="OperationType NumeroUno NumeroDue">
  <input
    wsam:Action="http://calculator.service/Calculator/doOperationRequest"
    message="tns:doOperation"/>
   <output
     wsam:Action="http://calculator.service/Calculator/doOperationResponse"
     message="tns:doOperationResponse"/>
    </operation>
</portType>

<!-- BINDING -->
<binding name="CalculatorPortBinding" type="tns:Calculator">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
   <operation name="doOperation">
   <soap:operation soapAction=""/>

   <input>
   <soap:body use="literal" namespace="http://calculator.service/"/>
   </input>

   <output>
   <soap:body use="literal" namespace="http://calculator.service/"/>
   </output>
   </operation>
</binding>

<service name="CalculatorService">
   <port name="CalculatorPort" binding="tns:CalculatorPortBinding">
   <soap:address
   location="http://macbook-di-mavi.local:8080/CalculatorSoap/CalculatorService"/>
   </port>
</service>

</definitions>
```

Una volta creato il servizio SOAP e il relativo documento WSDL, è possibile creare il client.

Per effettuare una chiamata trasparente, è stato necessario effettuare una chiamata del servizio dinamica.

La chiamata del servizio in modo dinamico è stata implementata utilizzando sempre le librerie di JAX-WS. Essa si può delineare nei seguenti passaggi:

1. Creare un servizio ed aggiungerne almeno un endpoint. L'end point viene ricavato dal grounding del servizio preso in esame dal manager.

Listing 16: WSDL CalculatorSoap

```
Service service = Service.create(serviceQN);
service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpoint);
```

2. Creare un'istanza di dispatch dal servizio, delineando sempre quella che è la porta di riferimento.

Listing 17: WSDL CalculatorSoap

```
Dispatch<SOAPMessage> dispatch =
service.createDispatch(portName, SOAPMessage.class, Service.Mode.MESSAGE);
```

3. Creare la richiesta SOAPMessage. La richiesta viene effettuata creando il suo SOAPEnvelope e il relativo SOAPBody.

Listing 18: WSDL CalculatorSoap

```
// comporre un messaggio di richiesta
MessageFactory mf =
   MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

// Creare un messaggio.
SOAPMessage request = mf.createMessage();
SOAPPart part = request.getSOAPPart();

// Ottenere il SOAPEnvelope e gli elementi intestazione e corpo.
SOAPEnvelope env = part.getEnvelope();
SOAPBody body = env.getBody();

// Creare il payload di messaggio.
SOAPElement operation =
    body.addChildElement(nameOperation, "ns2", namespace);

for(int i=0; i<inputs.size(); i++){
        SOAPElement value = operation.addChildElement(parameters.get(i));
        value.addTextNode(inputs.get(i));
}

request.saveChanges();
```

4. Gestire ed elaborare la risposta del server.

Listing 19: WSDL CalculatorSoap

```
/** Elaborare la risposta. **/
SOAPMessage response = dispatch.invoke(request);
```

4.2.2 JAX-RS - Calculator REST Service

Come è stato creato il servizio SOAP, è stato necessario creare anche lo stesso servizio ma con interfaccia REST.

JAX-RS è una collezione di interfacce e annotazioni Java che semplifica lo sviluppo server-side di applicazioni REST.
Usando la tecnologia JAX-RS, i web services REST sono più facilmente sviluppabili e più facilmente utilizzabili.

JAX-RS funziona in modo similare a JAX-WS, una volta effettuato il deploy del file .war sul server glassfish, grazie alle annotazioni apportate sul codice del servizio, viene automaticamente creato il file descrittivo WADL.

Bisogna rispettare una certa procedura per sviluppare un web service con JAX-RS, che segue tali passaggi:

1. Creare la classe "root" del web service. Deve presentare l'annotazione *@ApplicationPath*, che viene utilizzata per effettuare un mapping dell'URL del web service. Rappresenterà l'URI base per tutte le risorse descritte.

Listing 20: Classe root

```
import javax.ws.rs.core.Application;
import javax.ws.rs.ApplicationPath;

@ApplicationPath("resources")
public class RESTApplication extends Application {
        [...]
}
```

2. Creare la classe Java utilizzata per rappresentare la risorsa. Una volta creata, deve essere annotata con l'annotazione *@Path* del pacchetto javax.ws.rs.Path. Tale annotazione definisce l'URI che la andrà ad identificare.

Listing 21: Classe che rappresenta la risorsa

```
@Path("calculator")
public class Calculator {

        [...]

}
```

Potremo quindi individuare la risorsa al path:
http://localhost:8080/CalculatorRest/resources/calculator". (In questo caso si tratta del localhost:8080 in quanto viene effettuato il deploy sul server glassfish).

3. Una volta creata e definita correttamente la classe che rappresenta la risorsa, bisogna definire il metodo che rappresenterà quella che è l'operazione del servizio, tale deve essere annotato correttamente.

Listing 22: Metodo

```
@GET
@Produces("text/plain")
public float doOperation(
                @QueryParam("OperationType") String operation,
                @QueryParam("NumeroUno") int numeroUno,
                @QueryParam("NumeroDue") int numeroDue) {

        [...]

}
```

Nel codice mostrato, vengono apportate due tipi di annotazioni.
@GET, rappresenta il metodo HTTP utilizzato nella comunicazione; *@Produces*, usata per specificare il media type MIME, della risorsa che viene prodotta e restituita al client.

Ogni parametro in input deve essere specificato e nel nostro caso sono stati definiti come QueryParameters. L'annotazione *@QueryParam*, specifica i nomi dei parametri che possono essere estratti dall'URL della richiesta HTTP e utilizzati nel metodo.

Una volta definito tutto il web service, inserendo le relative annotazioni, deve essere effettuato il deploy del file .war sul server Glassfish.

Una volta effettuato il deploy, verrà creato automaticamente il file WADL relativo al servizio.

Possiamo vedere il file WADL del nostro servizio nella sezione sottostante:

Listing 23: File WADL - CalculatorRest

```
<application xmlns="http://wadl.dev.java.net/2009/02">

<doc  xmlns:jersey="http://jersey.java.net/"
jersey:generatedBy="Jersey: 2.21 2015-08-14 21:41:51"/>

<doc xmlns:jersey="http://jersey.java.net/"
jersey:hint="This is simplified WADL with user and core resources only.
To get full WADL with extended resources use the query parameter detail.
Link:
http://localhost:8080/CalculatorRest/resources/application.wadl?detail=true"/>

<grammars/>

<resources base="http://localhost:8080/CalculatorRest/resources/">
<resource path="calculator">

<method id="doOperation" name="GET">

<request>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema"
        name="OperationType" style="query" type="xs:string"/>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema"
        name="NumeroUno" style="query" type="xs:int"/>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema"
        name="NumeroDue" style="query" type="xs:int"/>
</request>

<response>
<representation mediaType="text/plain"/>
</response>

</method>

</resource>
</resources>

</application>
```

Una volta creato il servizio e il relativo documento WADL, è stato possibile creare un Client che potesse effettuare una chiamata dinamica al

servizio.

Questo client è stato implementato all'interno della classe "*Restful-Grounding*" all'interno del metodo *callService*.

Il client è stato creato sempre sfruttando le librerie JAX-RS.

Come primo passo deve essere creato il un oggetto della classe *javax.ws.rs.client.Client*. Una volta creato l'oggetto Client, è necessario creare il Web Target, cioè gli viene impostato quello che è l'end point del servizio. L'end point viene ricavato dal grounding del servizio attraverso il manager.

Listing 24: File WADL - CalculatorRest

```
//creiamo il Client
Client client = ClientBuilder.newClient();

//settiamo il web target
WebTarget myResource = client.target(endPoint);
```

Una volta effettuato ciò, vengono passati impostati i parametri e i relativi valori.

Nel nostro caso abbiamo passato i parametri come query parameters.

Essendo una chiamata dinamica e non sapendo a priori il numero dei parametri richiesti, essi vengono settati in modo dinamico all'interno di un ciclo.

Questi parametri, come anche l'end point, vengono ricavati dal grounding analizzato nel manager.

Listing 25: File WADL - CalculatorRest

```
for(int i=0; i<inputs.size(); i++){

        //impostiamo i parametri in input e i relativi valori
        myResource = myResource.queryParam(parameters.get(i), inputs.get(i));
}
```

Vediamo come, nell'esempio sopra, *parameters* rappresenta una lista dei nomi dei parametri richiesti in input e *inputs* rappresenta una lista dei re-

lativi valori.

Una volta impostati tutti i valori necessari, bisogna effettuare quella che è la vera richiesta al server.

Listing 26: File WADL - CalculatorRest

```
String response = myResource.request(MediaType.TEXT_PLAIN).get(String.class);
```

Il metodo *request* viene utilizzato per effettuare la richiesta. Viene passato in input il media type richiesto per la richiesta.

Il metodo *get* si occupa invece di gestire quella che è la risposta alla richiesta. Viene passato in input il tipo di risposta richiesta.

La variabile *response* conterrà il valore della risposta del server. Nel caso del nostro esempio, la risposta sarà il risultato dell'operazione effettuata.

4.3 Ontologie Calculator - OWL

Per poter utilizzare il manager è stato necessario creare anche le ontologie relative ai servizi creati.

Essendo due servizi che svolgono lo stesso compito è stato possibile creare un unico "Service", un unico "Profile" e un unico "Process".

Basandosi su protocolli diversi, è stato necessario creare due grounding diversi: uno SOAP e uno REST.

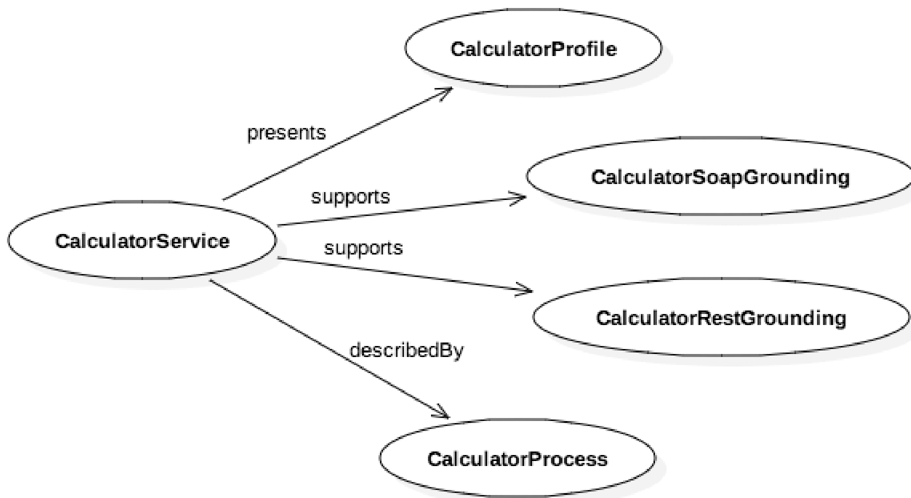

Figura 23: Calculator OWL-S

Queste ontologie sono state create con il linguaggio OWL-S.

4.3.1 Calculator Profile

Come prima cosa definiamo il Profile del Calculator, quindi ciò che realmente esegue il servizio.

Viene creata un'istanza profile, subclasse di Profile, chiamata CalculatorProfile.

In seguito, si definisce quello che è l'aspetto funzionale del servizio.

Come prima cosa definiamo qual è il nome e qual è la descrizione del processo che stiamo prendendo in esame.

Ogni Profile, utilizza tali object property per descrivere il servizio:

- *serviceName*: object property attraverso il quale si descrive il nome del servizio;
- *textDescription*: object property che permette di trascrivere quella che è la descrizione del servizio;
- *has_process*: object property che permette di connettere il profilo al processo preso in esame.

Una volta effettuate tali descrizioni, è necessario descrivere quelli che sono gli input, gli output, le precondizioni e gli effetti del processo preso in esame. Tutto ciò viene effettuato attraverso tali object properties:

- *hasInput*: descrive gli input del processo;
- *hasOutput*: descrive gli output del processo;
- *hasResult*: che può essere visto come riferimento a una coppia di output ed effetti;
- *hasPrecondition*: descrive le precondizioni del servizio.

In particolare nel nostro caso, il *CalculatorProfile* sarà così designato:

Listing 27: CalculatorProfile

```
<owl:NamedIndividual rdf:about="&calculatorprofile;#CalculatorProfile">
    <rdf:type rdf:resource="&profile;#Profile"/>

    <Profile:serviceName>
        CalculatorService
    </Profile:serviceName>

    <Profile:textDescription>
        Questo servizio permette di effettuare le operazioni di
        somma, sottrazione, moltiplicazione e divisione.
    </Profile:textDescription>

    <Profile:has_process
        rdf:resource="&calculatorprocess;#DoOperation"/>

    <Process:hasInput
     rdf:resource="&calculatorprocess;#NumeroDue"/>
    <Process:hasInput
     rdf:resource="&calculatorprocess;#NumeroUno"/>
    <Process:hasInput
     rdf:resource="&calculatorprocess;#OperationType"/>

    <Process:hasOutput
     rdf:resource="&calculatorprocess;#CalculatorOutput"/>

    <Process:hasResult
     rdf:resource="&calculatorprocess;#CalculatorNegativeResult"/>
    <Process:hasResult
     rdf:resource="&calculatorprocess;#CalculatorPositiveResult"/>

</owl:NamedIndividual>
```

4.3.2 Calculator Process

Una volta definito cosa fa un servizio è necessario esplicitare come esso lavora. Questo viene effettuato nel "Process".

Anche in questo caso viene creata un'istanza di Process, di preciso un AtomicProcess, chiamata *"DoOperation"* che rappresenta il metodo del nostro servizio, quindi il processo che prendiamo in esame.

Come prima cosa devono essere istanziate le classi e gli individual necessari per poter descrivere il processo.
Nel nostro process vediamo istanziate le seguenti classi:

- *CalculatorOutput*: che rappresenta quelli che sono gli output del calculator;
- *OperationType*: rappresenta il primo input del processo, nonchè il tipo di operazione da effettuare. Infatti, in relazione a tale, sono state create anche le classi:
  - *Moltiplicazione*
  - *Divisione*
  - *Somma*
  - *Sottrazione*
- *NumeroUno*: rappresenta il primo numero da inserire in input nel processo;
- *NumeroDue*: rappresenta il secondo numero da inserire in input nel processo.

Listing 28: CalculatorProcess - Classi

```
<!-- CalculatorOutput -->
<owl:Class rdf:about="&calculatorprocess;#CalculatorOutput"/>

<!-- Divisione -->
<owl:Class rdf:about="&calculatorprocess;#Divisione"/>

<!-- Moltiplicazione -->
<owl:Class rdf:about="&calculatorprocess;#Moltiplicazione"/>

<!-- Somma -->
<owl:Class rdf:about="&calculatorprocess;#Somma"/>

<!-- Sottrazione -->
 <owl:Class rdf:about="&calculatorprocess;#Sottrazione"/>

<!-- NumeroDue -->
<owl:Class rdf:about="&calculatorprocess;#NumeroDue"/>

<!-- NumeroUno -->
<owl:Class rdf:about="&calculatorprocess;#NumeroUno"/>

<!-- OperationType -->
<owl:Class rdf:about="&calculatorprocess;#OperationType">
        <owl:equivalentClass>
        <owl:Class>
        <owl:oneOf rdf:parseType="Collection">
            <rdf:Description rdf:about="&calculatorprocess;#Divisione"/>
            <rdf:Description rdf:about="&calculatorprocess;#Moltiplicazione"/>
            <rdf:Description rdf:about="&calculatorprocess;#Somma"/>
            <rdf:Description rdf:about="&calculatorprocess;#Sottrazione"/>
        </owl:oneOf>
        </owl:Class>
        </owl:equivalentClass>
</owl:Class>
```

Una volta create le classi si procede a creare gli individuals, nonchè istanze di tali classi, per poter definire il processo stesso.

Una volta definito lo scheletro di quello che è il processo, si inizia a definire le proprietà che collegano al process gli individuals, cioè quindi vengono definiti gli input, gli output e gli effetti del processo.

- *hasInput*: come in profile, tale object property definisce quelli che sono gli input del process;
- *hasOutput*: come in profile, tale object property definisce quelli che sono gli output del process;
- *hasResultt*: come in profile, tale object property definisce quelli che sono i risultati del process.

Listing 29: CalculatorProcess - DoOperation

```
<owl:NamedIndividual rdf:about="&calculatorprocess;#DoOperation">
<rdf:type rdf:resource="&process#AtomicProcess"/>

<rdfs:comment>
Questo processo permette di effettuare un'operazione tra:
- somma
- divisione
- moltiplicazione
- sottrazione
</rdfs:comment>

<Process:hasInput rdf:resource="&calculatorprocess;#NumeroDue"/>
<Process:hasInput rdf:resource="&calculatorprocess;#NumeroUno"/>
<Process:hasInput rdf:resource="&calculatorprocess;#OperationType"/>

<Process:hasOutput rdf:resource="&calculatorprocess;#CalculatorOutput"/>

<Process:hasResult rdf:resource="&calculatorprocess;#CalculatorNegativeResult"/>
<Process:hasResult rdf:resource="&calculatorprocess;#CalculatorPositiveResult"/>

</owl:NamedIndividual>
```

4.3.3 CalculatorSoapGrounding

Una volta definito cosa fa un servizio e come lo fa, bisogna definire i dettagli riguardanti le modalità di accesso al servizio.

Nel caso del Grounding SOAP bisogna definire le modalità di accesso al servizio riferendosi al documento WSDL.

Il nostro CalculatorSoapGrounding, è un'istanza di *WsdlGrounding*.

Il nostro grounding, però, farà riferimento a quello che è l'atomic process preso in considerazione.

Verrà quindi definita l'object property *hasAtomicProcessGrounding* che farà riferimento al nostro processo.

Listing 30: CalculatorSoapGrounding

```
<owl:NamedIndividual
    rdf:about="&calculatorsoapgrounding;#CalculatorServiceGrounding">

    <rdf:type rdf:resource="&grounding;#WsdlGrounding"/>

    <grounding:hasAtomicProcessGrounding
        rdf:resource="&calculatorsoapgrounding;#CalculatorGrounding"/>

</owl:NamedIndividual>
```

Una volta definita l'istanza di grounding, deve essere definito il grounding relativo al processo preso in esame.

Viene quindi creata un'istanza di *WsdlAtomicProcessGrounding*, e gli vengono definite tutte le proprietà relative:

- *owlsProcess*: si occupa di effettuare il riferimento all'individuals del processo che è preso in esame;
- *wsdlDocument*: attraverso tale object property viene evidenziato il path del file WSDL.
- *wsdlInput*: proprietà che si occupa di evidenziare quelli che sono gli input del processo: è formata dalla coppia *owlsParameter* e *wsdlMessagePart* che si riferiscono rispettivamente all'input definito nel file

owl CalculatorProcess e all'input definito nel part del message del file WSDL.

- *wsdlInputMessage*: proprietà che fa riferimento al nome del message del file WSDL.
- *wsdlOperation*: proprietà che fa riferimento all'*operation* del file WSDL. Si compone di altri elementi quali:
  - *owlnameSpace*: fa riferimento al namespace definito nel file WSDL;
  - *owlendPoint*: fa riferimento all'endpoint definito nel file WSDL;
  - *owloperation*: fa riferimento al nome dell'operation definita nel file WSDL;
  - *owlportType*: fa riferimento al portType definito nel file WSDL;
  - *owlbinding*: fa riferimento al binding definito nel file WSDL.
- *owlOutputMessage*: proprietà che fa riferimento al nome del message di risposta del file WSDL.

Nel nostro CalculatorSoapGrounding, la struttura che si è andata a creare è riportata nella sezione sottostante.

Listing 31: CalculatorSoapGrounding

```
<owl:NamedIndividual
rdf:about="&calculatorsoapgrounding;#CalculatorGrounding">

<rdf:type rdf:resource="&grounding#WsdlAtomicProcessGrounding"/>

<grounding:owlsProcess rdf:resource="&calculatorprocess;#DoOperation"/>

<grounding:wsdlDocument rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
            http://localhost:8080/CalculatorSoap/CalculatorService?WSDL
</grounding:wsdlDocument>

<grounding:wsdlInput>
<rdf:Description>
<rdf:type rdf:resource="&owls;WsdlInputMessageMap"/>

<grounding:owlsParameter rdf:resource="&calculatorprocess;#OperationType"/>

<grounding:wsdlMessagePart rdf:datatype="&xmlSchema#anyURI">
http://localhost:8080/CalculatorSoap/CalculatorService?WSDL#OperationType
</grounding:wsdlMessagePart>

</rdf:Description>
</grounding:wsdlInput>
```

```
<grounding:wsdlInput>
<rdf:Description>
<rdf:type rdf:resource="&owls;WsdlInputMessageMap"/>

<grounding:owlsParameter rdf:resource="&calculatorprocess;#NumeroUno"/>

<grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://localhost:8080/CalculatorSoap/CalculatorService?WSDL#NumeroUno
</grounding:wsdlMessagePart>

</rdf:Description>
</grounding:wsdlInput>

<grounding:wsdlInput>
<rdf:Description>
<rdf:type rdf:resource="&owls;WsdlInputMessageMap"/>

<grounding:owlsParameter rdf:resource="&calculatorprocess;#NumeroDue"/>

<grounding:wsdlMessagePart rdf:datatype="&xmlSchema;#anyURI">
http://localhost:8080/CalculatorSoap/CalculatorService?WSDL#NumeroDue
</grounding:wsdlMessagePart>

</rdf:Description>
</grounding:wsdlInput>

<grounding:wsdlInputMessage rdf:datatype="&xmlSchema;#anyURI">
http://localhost:8080/CalculatorSoap/CalculatorService?WSDL#doOperation
</owl-s:Grounding.owlwsdlInputMessage>

<grounding:wsdlOperation>
<rdf:Description>

<rdf:type rdf:resource="&grounding;WsdlOperationRef"/>

<grounding:nameSpace rdf:datatype="&xmlSchema;#anyURI">
http://calculator.service/
</grounding:nameSpace>

<grounding:endPoint rdf:datatype="&xmlSchema;#anyURI">
 http://localhost:8080/CalculatorSoap/CalculatorService
</grounding:endPoint>

<grounding:operation rdf:datatype="&xmlSchema;#anyURI">
http://localhost:8080/CalculatorSoap/CalculatorService?WSDL#doOperation
</grounding:operation>

<grounding:portType rdf:datatype="&xmlSchema;#anyURI">
http://localhost:8080/CalculatorSoap/CalculatorService?WSDL#Calculator
</grounding:portType>

<grounding:binding rdf:datatype="&xmlSchema;#anyURI">
http://localhost:8080/CalculatorSoap/CalculatorService?WSDL#CalculatorPortBinding
</grounding:binding >

</rdf:Description>

</grounding:wsdlOperation>

<grounding:wsdlOutputMessage rdf:datatype="&xmlSchema;#anyURI">
http://localhost:8080/CalculatorSoap/CalculatorService?WSDL#doOperationResponse
</grounding:wsdlOutputMessage>

</owl:NamedIndividual>
```

Nei riferimenti al file WSDL, devono essere segnati i *"name"* delle varie parti del WSDL.

Quindi prendendo ad esempio il binding del Calculator, il "name" del binding è *"CalculatorPortBinding"*.

4.3.4 CalculatorRestGrounding

Prendendo in esame due servizi di natura uguale, ma con protocollo diverso, è stato necessario creare due grounding diversi: uno REST e uno SOAP.

Come già detto in precedenza, a differenza dal protocollo SOAP, i grounding per servizi REST non si basano ancora su uno standard preciso. Questo grounding RESTful è stato redatto secondo le specifiche date dall'articolo "Semantic Web Services: A Restful Approach" [11].

Vediamo come viene creato un grounding RESTful.

Come per il grounding SOAP, viene definito prima un'istanza della classe *WadlGrounding*, al quale viene associato il processo atomico preso in esame (*hasAtomicProcessGrounding*).

Listing 32: CalculatorRestGrounding - hasAtomicProcessGrounding

```
<owl:NamedIndividual
    rdf:about="&calculatorrestgrounding;#CalculatorRestServiceGrounding">

<rdf:type
    rdf:resource=
    "http://fullsemanticweb.com/ontology/RESTfulGrounding/v1.0/RESTfulGrounding.owl
    #WadlGrounding"/>

<restful:hasAtomicProcessGrounding
    rdf:resource="&calculatorrestgrounding;#CalculatorGrounding"/>

</owl:NamedIndividual>
```

Una volta definito il grounding, bisogna definire l'atomic process WADL.

In questa sezione vengono definiti i riferimenti alla risorsa (*resource*), al metodo della richiesta HTTP e all'endpoint dichiarati nel file WADL.

Listing 33: CalculatorRestGrounding - definizione WADL

```
<restful:WadlAtomicProcessGrounding rdf:ID="CalculatorRestGrounding">

<grounding:owlsProcess rdf:resource="&calculatorprocess;#DoOperation"/>

<restful:wadlResourceMethod>
<restful:WadlResourceMethodRef>
<restful:resource rdf:datatype="&xsd;#anyURI">
   http://localhost:8080/CalculatorRest/resources/application.wadl#Calculator
</restful:resource>

<restful:endpoint rdf:datatype="&xsd;#anyURI">
http://localhost:8080/CalculatorRest/resources/calculator
</restful:endpoint>

<restful:method rdf:datatype="&xsd;#anyURI">
http://localhost:8080/CalculatorRest/resources/application.wadl#GET
</restful:method>
</restful:WadlResourceMethodRef>
</restful:wadlResourceMethod>


<restful:wadlDocument rdf:datatype="&xsd;#anyURI">
http://localhost:8080/CalculatorRest/resources/application.wadl
</restful:wadlDocument>
```

Vediamo come vengono definiti all'interno del *WadlResourceMethodRef*, le object properties:

- *endpoint*: fa riferimento all'endpoint del servizio
- *method*: fa riferimento al metodo HTTP utilizzato nella richiesta
- *resource*: fa riferimento alla risorsa (*resource*) del file WADL

Viene inoltre definito il collegamento al file WADL, attraverso l'object property *wadlDocument*.

Una volta definito ciò, si passa a definire i messaggi di richiesta e i messaggi di risposta, quindi quelli che saranno i parametri di input e output.

Listing 34: CalculatorRestGrounding - Messaggi Richiesta/Risposta

```
 <!-- Request Parameters -->

<restful:wadlRequestParam>
<restful:WadlRequestParamMap>
   <grounding:owlsParameter rdf:resource="&calculatorprocess;#OperationType"/>
   <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">
   &wadl;#OperationType
   </restful:wadlMessageParam>
</restful:WadlRequestParamMap>
</restful:wadlRequestParam>

<restful:wadlRequestParam>
<restful:WadlRequestParamMap>
   <grounding:owlsParameter rdf:resource="&calculatorprocess;#NumeroUno"/>
   <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">
   &wadl;#NumeroUno
   </restful:wadlMessageParam>
</restful:WadlRequestParamMap>
</restful:wadlRequestParam>

<restful:wadlRequestParam>
<restful:WadlRequestParamMap>
   <grounding:owlsParameter rdf:resource="&calculatorprocess;#NumeroDue"/>
   <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">
   &wadl;#NumeroDue
   </restful:wadlMessageParam>
</restful:WadlRequestParamMap>
</restful:wadlRequestParam>

<!-- Response Parameters -->

<restful:wadlResponseParam>
<restful:WadResponseParamMap>
   <grounding:owlsParameter rdf:resource="&calculatorprocess;#CalculatorOutput"/>
   <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">
   http://localhost:8080/CalculatorRest/resources/application.wadl
   </restful:wadlMessageParam>
</restful:WadResponseParamMap>
```

Possiamo vedere come per ogni parametro viene fatto un collegamento tra il parametro definito nel file owl (*owlsParameter*), e quello definito nel file wadl (*wadlMessageParam*).

Come nel grounding SOAP, anche in questo caso, per quanto riguarda il file WADL, si fa sempre riferimento al "name" degli elementi presi in considerazione.

4.3.5 Calculator Service

Come ultimo passo, "mettiamo insieme" tutti i pezzi all'interno del Service.

Nel service andremo a definire tutte le relazioni che ci sono tra il Service stesso e gli altri elementi dell'ontologia.

Listing 35: CalculatorService

```
<owl:NamedIndividual rdf:about="&calculatorservice;#CalculatorService">
<rdf:type rdf:resource="&service;#Service"/>

   <Service:describedBy
        rdf:resource="&calculatorprocess#DoOperation"/>

   <Service:presents
        rdf:resource="&calculatorprofile;#CalculatorProfile"/>

   <Service:supports
        rdf:resource="&calculatorrestgrounding;#CalculatorRestGrounding"/>

   <Service:supports
        rdf:resource="&calculatorsoapgrounding;#CalculatorGrounding"/>

</owl:NamedIndividual>
```

Viene quindi delineata la struttura di una ontologia di servizi:

- Un Servizio è descritto da (**describedBy**) un Processo;
- Un Servizio presenta (**presents**) un Profilo;
- Un Servizio supporta (**supports**) due Grounding: uno REST e uno SOAP.

CAPITOLO 5

Conclusioni

5 CONCLUSIONI

L'obiettivo principale di tale tesi è stato quello di poter creare descrizioni semantiche di servizi con diversi protocolli di comunicazione e implementare un manager che permettesse di poterli gestire e richiamare in maniera trasparente.

Tale obiettivo è stato raggiunto grazie alla creazione di web service sia di tipo SOAP che di tipo REST, che sono stati utilizzati come "esempi" per poter vedere come sfruttare le possibilità del manager.

Inoltre, creando due servizi di tipo diverso, è stato possibile verificare la possibilità di creare un'unica descrizione semantica, che permettesse poi di avere però due grounding diversi in base al tipo di servizio da richiamare.

Molto interessante, e uno dei principali punti, è stato implementare il grounding RESTful, ancora poco utilizzato, nonostante i web service di tipo REST stiano prendendo sempre più piede, grazie alla loro facilità di utilizzo.

In particolare è stato interessante vedere l'implementazione vera e propria di un grounding RESTful, seguendo la guida trascritta nell'articolo "Semantic Web Services: A RESTful Approach". [11]

Per quanto riguarda il Manager si può riassumere quindi in quattro funzionalità principali:

1. Verificare compatibilità richiesta client e servizio preso in esame;
2. Verificare elementi necessari per effettuare la richiesta al servizio (parametri in input ...);
3. Verificare i grounding presenti e inizializzarne i valori necessari per la richiesta al server;
4. Chiamata ai servizi disponibili.

Possiamo vedere riassunte tali funzionalità, in un diagramma di attività che illustra quindi quelli che sono i passi da seguire per l'utilizzo del Manager stesso.

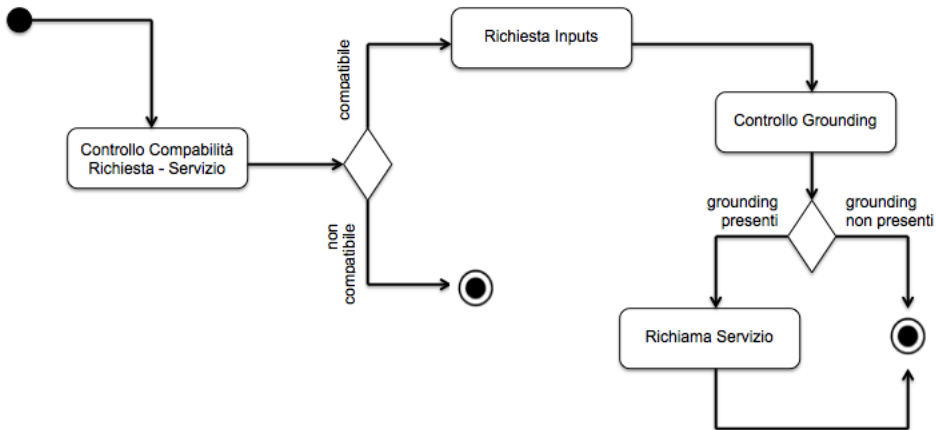

Figura 24: Diagramma Di Attività

Il Manager elaborato in tale tesi può essere sfruttato per un continuo sviluppo futuro che possa approfondirne l'utilizzo. Il manager può essere esteso, in modo tale che possa manipolare e richiamare servizi più complessi: servizi che siano formati da più atomic process, quindi da composite process.

Questo lavoro può essere effettuato in un modo non troppo complesso, cioè inserendo nella classe OntologiesManager, metodi che permettano la manipolazione, attraverso query SPARQL più complesse, di process composite.

Durante lo svolgimento di tale tesi, mi sono accorta che non ci sono molte descrizioni semantiche OWL o molti articoli ed esempi che si riferiscono a servizi con protocollo REST, quindi capirne e sfruttarne il funzionamento è stato molto gratificante.

5.1 Tecnologie Utilizzate

Per lo svolgimento di tale tesi ho sfruttato diverse tecnologie.

Come ambienti di sviluppo ho utilizzato:

- Eclipse, per lo sviluppo del manager e per l'implementazione dei web service SOAP e REST;
- Protégé, per l'implementazione delle ontologie OWL-S.

Come librerie principali ho utilizzato:

- Jena, per lo sviluppo del manager. Libreria che consente la manipolazione e la lettura delle ontologie.
- JAX-RS e JAX-WS, per quanto riguarda lo sviluppo dei due web service.

I Web Service sono stati fatti girare sul server Glassfish.

Ringraziamenti

In primo luogo vorrei ringraziare il professore Davide Rossi per aver svolto il ruolo da Relatore della mia tesi e per essere stato sempre molto cordiale e disponibile durante lo svolgimento del progetto, di avermi suggerito questo argomento di tesi e di avermi dispensato sempre ottimi consigli a riguardo.

Ringrazio inoltre il Dott. Francesco Poggi, per il ruolo di correlatore della tesi per essere stato spesso presente e di avermi aiutato in momenti di difficoltà.

Ringrazio i miei genitori, mio padre e mia madre, per avermi dato la grande opportunità di continuare i miei studi permettendomi di seguire questo corso universitario. Per essermi sempre stati vicino nei momenti di difficoltà maggiore e di aver sempre creduto in me.

Ringrazio infine mia sorella, colei che mi ha sempre supportato in ogni momento difficile, colei che mi ha sempre dato la forza di andare avanti e colei che mi ha fatto sempre credere in me stessa. Ma soprattutto mia grande compagna di studi.

Riferimenti bibliografici

[1] *W3C Working Group. Web services architecture. http://www.w3.org/TR/ws-arch/.*

[2] *TED - Tim Berners-Lee e il Web prossimo venturo.* `https://www.ted.com/talks/tim_berners_lee_on_the_next_web?language=it#t-648636`.

[3] *Evolution of world wide web: from web 1.0 to web 4.0. Sareh Aghaei, Mohammad Ali Nematbakhsh and Hadi Khosravi Farsani.*

[4] *"A Fundamental Component of the Semantic Web", Bulletin of the American Society for Information Science and Technology. ane, Greenberg & Stuart, Sutton & D. Grant, Campbell*

[5] *RDF Schema 1.1. https://www.w3.org/TR/rdf-schema.*

[6] *Ontology Development 101: A Guide To Creating Your First Ontology. atalya F. Noy and Deborah L. McGuinness , Stanford University, Stanford, CA, 94305. noy@smi.stanford.edu and dlm@ksl.stanford.edu*

[7] *Differenze tra Web service REST e SOAP. TML.it*

[8] *OWL-S: Semantic Markup for Web Services.*

[9] *Bringing Semantics to Web Services with OWL-S avid Martin, Mark Burstein, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Katia Sycara, Deborah L. McGuinness, Evren Sirin, Naveen Srinivasan*

[10] *A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools Edition 1.3 atthew Horridge*

[11] *Semantic Web Services: A Restful Approach. tàvio Freitas Ferreira Filho, Maria Alice Grigas Varella Ferreira*

[12] *Yahoo! Web APIs. Technical report, Yahoo!, 2005. Guarda http://developer.yahoo.net/.*