SAPIENZA
UNIVERSITÀ DI ROMA

COMPOSIZIONE ED ORCHESTRAZIONE REMOTA DI WEB SERVICES

Tesina del corso "Seminari di Ingegneria del Software"

Professore: Giuseppe De Giacomo
Professore: Massimo Mecella

Studenti:

- Constantin Moldovanu
- Emanuele Tatti

Indice generale

Introduzione

Il principio alla base della composizione dei servizi web è uno dei principi che si trova anche alla base dello sviluppo di software: la riusabilità. L'obiettivo è quindi di creare un nuovo servizio sfruttando eventualmente altri servizi esistenti che già implementano alcune o tutte le funzionalità necessarie. Il nuovo servizio dovrà comunque essere a sua volta un web service normale, al quale accedere come a tutti gli altri e che possa essere, a sua volta, oggetto di composizione

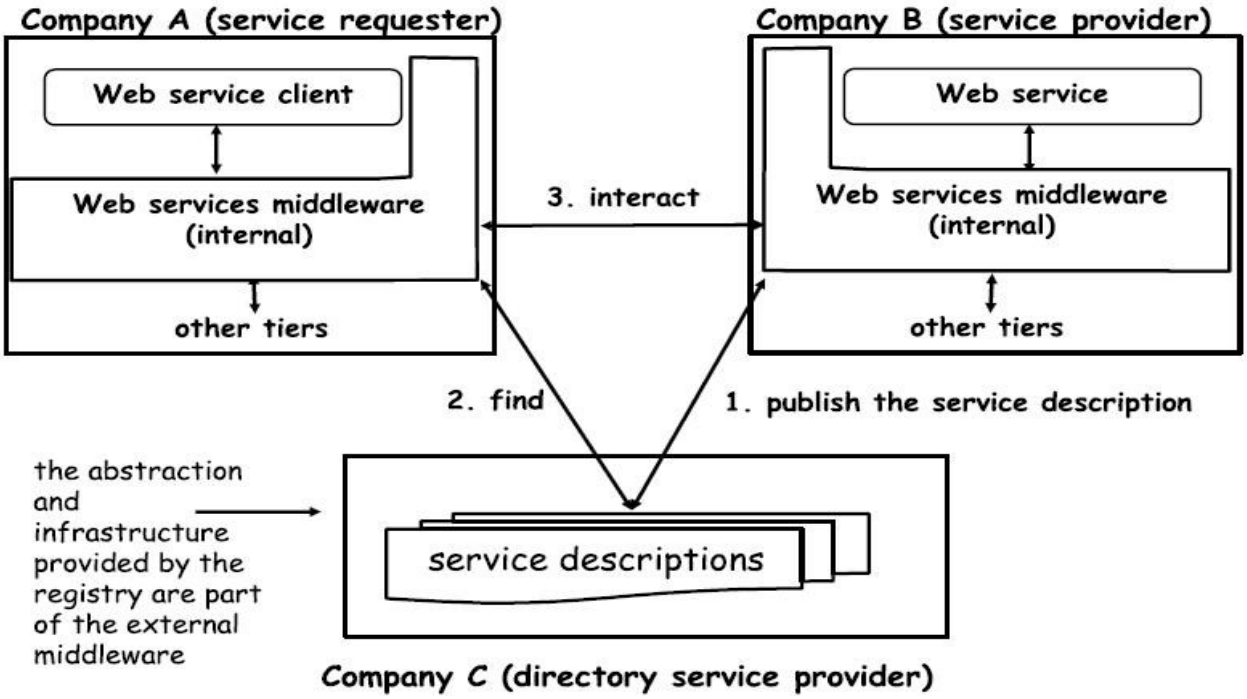
C'è ovviamente una grande differenza tra l'importare una libreria sviluppata da qualcun altro nel proprio progetto e utilizzare invece un web service remoto all'interno del proprio servizio; c'è dietro un processo di orchestrazione che deve capire quando e quale servizio chiamare.
Nel primo caso, importare una libreria porterebbe conseguenze pesanti per il progetto in causa, perché le librerie potrebbero essere scritte in un altro linguaggio oppure utilizzare a loro volta altre librerie che non sono compatibili oppure comunque necessitano di ulteriore tempo dedicato per la comprensione ed eventuali test di integrazione. D'altro canto, un Web Service fornisce le sue funzionalità utilizzando un'interfaccia standardizzata, in un linguaggio di uso comune (XML) e, inoltre, fornisce le informazioni richieste lavorando in contesto separato, svincolato da quello del cliente del servizio. Oltre a svincolarsi completamente dall'implementazione del servizio, il cliente non deve preoccuparsi nemmeno delle risorse necessarie per la produzione del servizio.
La composizione ed orchestrazione di servizi per crearne uno virtuale è stata oggetto di tanti progetti, che sono arrivati a risultati più o meno rilevanti. Il nostro obiettivo è quello di rendere il processo che porta dal deploy dei servizi della community alla creazione di un servizio target finale attuabile in modalità remota. I vari componenti necessari per i vari passi (composizione, generazione files, deploy) dovranno integrarsi il più possibile e collaborare in maniera ordinata per eseguire le loro operazioni in modo trasparente, ove possibile, ed efficiente.

Il problema della composizione e dell'orchestrazione

Web services

Nel paradigma degli applicativi client-server, un Web Service (WS) è un sistema progettato per supportare in modo trasparente l'interoperabilità tra diversi sistemi. Il server offre un insieme di operazioni, accessibili mediante un linguaggio standard, ai suoi clienti. Il vero punto di forza dei WS è che riescono ad offrire interazioni client-server astraendo completamente dal linguaggio e protocolli utilizzati dalle varie macchine, sfruttando canali comuni quali l'HTTP, sui quali far viaggiare messaggi in formato XML.

I web services vengono proposti in varie tipologie, quali REST o SOAP. I servizi REST pongono meno enfasi sulla forte tipizzazione e sulla dichiarazione esplicita di dati ed operazioni, puntando sulla velocità, semplicità ed efficienza.
I servizi SOAP invece puntano alla forte tipizzazione e dichiarazione esplicita, utilizzando formati prestabiliti (SOAP) che possono essere correttamente interpretati da ogni attore del sistema. In questo modo un servizio SOAP riesce ad avere quella trasparenza nell'accesso che garantisce la completa interoperabilità tra le diverse piattaforme.
L'interazione che nasce si allontana quindi dal paradigma client-server e si basa sull'architettura SOA (service oriented architecture).



L'interazione tra i vari attori avviene mediante un middleware esterno, che integra le definizioni dei vari servizi, fornendo un punto d'accesso unico, che può essere invocato con strumenti standard quali SOAP. I vari client (e server) possono a loro volta avere middleware interni che implementino l'accesso ai propri sottosistemi in relazione alle invocazioni SOAP.

I servizi vengono descritti utilizzando lo stesso linguaggio XML, che ne definisce la struttura e comportamento: WSDL (web service description language). Tali WSDL sono

contenuti all'interno del middleware esterno, messi a disposizione dai vari server e invocati dai client interessati.
Le chiamate avvengono attraverso il protocollo SOAP (), che impachetta richieste, risposte e tipi di dato in XML in una busta SOAP, corredata di un header SOAP che contiene informazioni aggiuntive.

Riusabilità

Supponiamo che ci venga richiesto di sviluppare un servizio particolare, che chiameremo *Target,* le cui funzionalità sono già implementate da altri esistenti in *Available Service*. Quello che banalmente potremmo fare è costruire il servizio da zero, ma l'operazione sarebbe costosa in termini di tempo o magari addirittura le nostre conoscenze potrebbero non essere sufficienti per l'implementazione di una delle operazioni richieste o comunque l'operazione da noi sviluppata potrebbe essere peggiore di quella già esistente. Si sta quindi estendendo ai servizi web un principio fondamentale nello sviluppo software tradizionale.

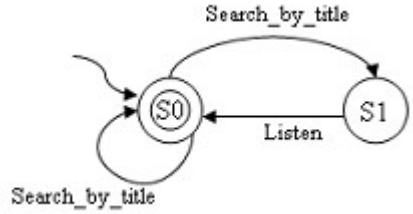
Rappresentazione astratta dei servizi

I servizi possono essere realizzati in linguaggi differenti ed essere implementati su piattaforme diverse, d'altronde il linguaggio WSDL, che invece è uno standard, fornisce una descrizione dell'interfaccia di un servizio ma ancora non dice nulla riguardo il suo comportamento.
Per questo nasce l'esigenza di una rappresentazione che astragga dall'implementazione e mostri semplicemente questo comportamento. Questo si può fare mediante una rappresentazione sotto forma di stati e transizioni: i transition system

Un Transition System TS è una tupla T = < A, S, So, δ, F> dove

– A è l'insieme delle azioni
– S è l'insieme degli stati
– So □ S è l'insieme degli stati iniziali
– δ □ S x A x S è l'insieme delle transizioni
- F □ S è l'insieme degli stati finali

Si può così ad esempio rappresentare un servizio che permette di ascoltare una canzone dato l'artista nel seguente modo.



A = {Search_by_title,Listen}
S = {S0,S1}

So={S0}
δ = {(S0, Search_by_title,S1),(S1,Listen,S0),(S0, Search_by_title,S0)}
F = {S0}

Il servizio target

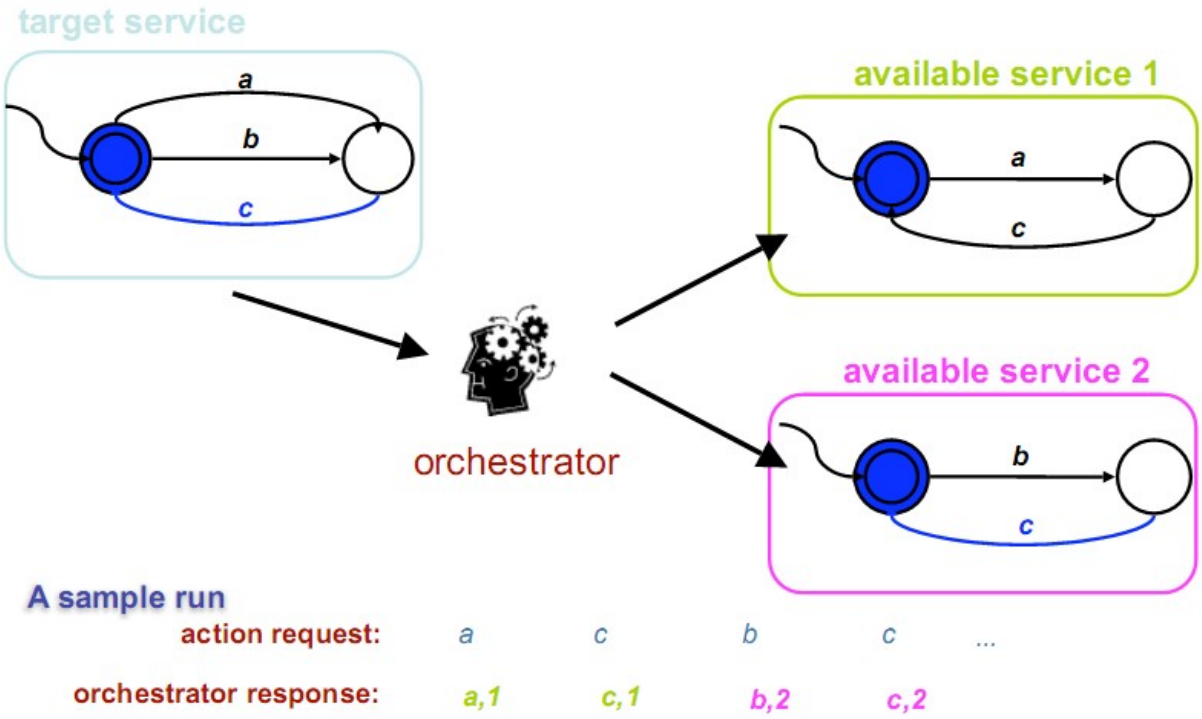
Nel corso del presente documento definiremo servizio target quel web service che è risultato della composizione di un insieme di servizi già presenti, chiamati available services. Il servizio target rappresenta il servizio virtuale che il cliente è interessato ad avere e la cui creazione deve essergli trasparente. Come ogni altro servizio, anche il target è descrivibile con un transition system <A, S, S_0, δ, F> ma deve avere le seguenti caratteristiche:

- deve avere un unico stato iniziale S_0;
- deve avere transizioni deterministiche.

Su tale servizio risultante il cliente ha pienamente controllo e può, ovviamente, essere oggetto di future composizioni in qualità di available service. Non si pongono alcune restrizioni sul determinismo degli available services: essi possono anche avere transizioni non deterministiche (trattasi di nondeterministic – devilish – available services) e possono essere composti ed orchestrati.

Composizione ed orchestrazione

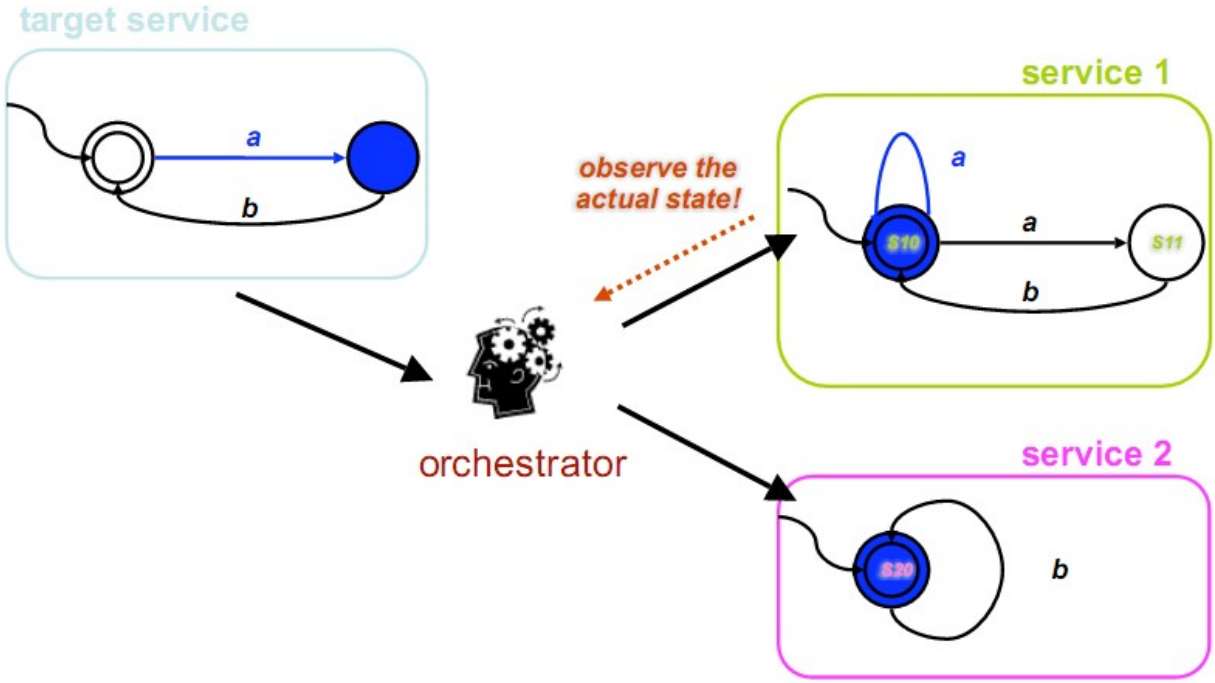
La composizione del target significa che ad ogni transizione richiesta al servizio target debba corrispondere un servizio che effettui l'operazione richiesta. La richiesta viene quindi eseguita dal servizio delegato che restituisce il risultato al target che, a sua volta, risponde al suo cliente. Sorge quindi la necessità di avere un'entità in grado di gestire i servizi invocati, nel giusto ordine. Tale entità si chiama Orchestratore.



L'orchestratore ha come unico scopo organizzare le richieste e delegarle ai giusti servizi disponibili, in modo completamente trasparente per il client del servizio target.
L'orchestratore può, quindi, essere considerato come una funzione
P(h, a) = i
che, data una history delle chiamate precedenti h ed un'azione da compiere a, individua un servizio i che è in grado di effettuare l'operazione. Nel caso dei servizi deterministici, inoltre, l'orchestratore può ricostruire lo stato dei vari servizi a partire dalla loro history:

- il servizio target ha lo stato dato dall'insieme delle azioni eseguite;
- gli available services deterministici possono avere il proprio stato descritto dalla tupla: $<a_k, P([a_1\ a_2\ ..\ a_{k-1}], a_k)$.

Nel caso di available services non deterministici (devilish), l'orchestratore non è più in grado di ricavare lo stato dei servizi in base alla semplice history ed è, quindi, costretto ad interrogare il loro stato prima di decidere.



La funzione dell'orchestratore P(h, a) = i riceve quindi una history che indica lo stato di ogni servizio in seguito ad ogni azione: $<a_k, (s_k^1, s_k^2, \ldots s_k^n, e_k)>$.

Tecniche per la composizione

Dati i transition systems dei servizi nella community e del target, si può decidere se il target può essere definito come composizione di un insieme finito di servizi esistenti utilizzando due approcci: logica dinamica preposizionale (PDL) e simulazione.

PDL (propositional dynamic logic) consente di esprimere la correttezza parziale di alcune asserzioni o di predicati generalmente validi. Il concetto fondamentale da tradurre in logica PLD è che l'orchestratore deve essere in grado di eseguire una transizione del target attraverso una transizione di un available service. Creata la formula PDL è sufficiente determinarne la soddisfacibilità, operazione che è eseguibile in PDL in EXPTIME e spazio polinomiale rispetto all'input. Questa modalità può essere utilizzata per comporre servizi deterministici e non deterministici, modificando opportunamente la formula PDL in modo che descriva le varie transizioni (con un incremento delle dimensioni dello spazio delle soluzioni nel caso non deterministico).

La **simulazione** d'altro canto è una tecnica che si propone di dedurre se un servizio target può essere riprodotto da una cerca community, dato che se un servizio è riprodotto da altri allora è simulato. Ne consegue che una comunità C simula un target service C se è in grado di riprodurre il suo comportamento in ogni istante.
Dal punto di vista matematico, dati i due transition systems

- $T = <A, S^T, S_0^T, \delta^T, F^T>$,
- $C = <A, S^C, S_0^C, \delta^C, F^C>$,

allora una relazione di simulazione R è quella relazione binaria che associa ad ogni stato di T uno stato di C tale che se $<S^T, S^C> \in$ a R allora S^T finale implica che S^C è finale e, per ogni azione a, se $s^T \rightarrow_a s^{T'}$ allora $\exists\ s^{C'} \mid s^C \rightarrow_a s^{C'}$ e $(s^{T'}, s^{C'}) \in$ R.

Ne consegue che fanno parte della relazione solo quelle coppie dove:

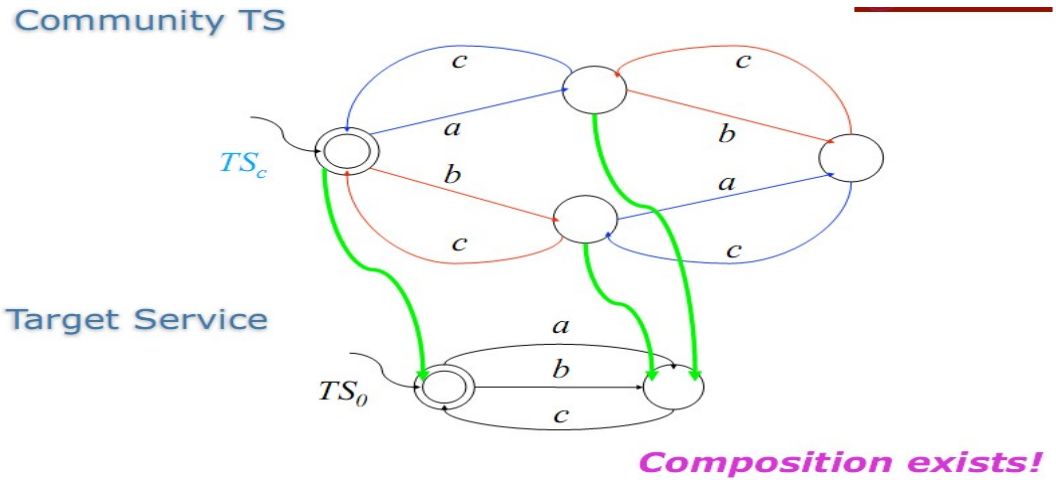
- se il primo elemento è finale lo è anche il secondo;
- se il primo elemento, attraverso un'azione, arriva da uno stato ad un altro allora anche il secondo elemento deve arrivare, mediante la stessa azione, in uno stato consistente ed i due stati risultanti devono essere lo stesso una coppia in R.

Il progetto corrente, come conseguenza dei progetti che lo precedono, utilizzerà la tecnica della simulazione.

Esempio: data una community di servizi descritti dai loro transition systems:

Available Services

la community degli available services deve essere almeno in grado di riprodurre i passi del servizio target:



Composizione tramite simulazione

TEOREMA: La composizione esiste se e solo se la community può simulare il target.

Tale teorema ci sottolinea la stretta correlazione che esiste tra composizione e smulazione. Un algoritmo per il calcolo della composizione può calcolare il prodotto cartesiano degli stati dei transition systems della community e del servizio target ed eliminare le coppie che violano le regole esposte nel paragrafo precedente. Questo algoritmo procede per esclusione e calcola quindi la simulazione col numero più grande di servizi (largest simulation):

Algorithm ComputingSimulation
Input: transition system $T = \langle A, T, t^0, \delta_T, F_T \rangle$ and
transition system $\mathcal{C} = \langle A, S, s_{\mathcal{C}}^0, \delta_{\mathcal{C}}, F_{\mathcal{C}} \rangle$
Output: the **simulated-by** relation (the largest simulation)

Body
$R = \emptyset$
$R' = T \times S - \{(t,s) \mid t \in F_t \wedge \neg(s \in F_{\mathcal{C}})\}$
while $(R \neq R')$ {
$R := R'$
$R' := R' - \{(t,s) \mid \exists t', a.\ t \rightarrow_a t' \wedge \neg \exists s' .\ s \rightarrow_a s' \wedge (t',s') \in R'\}$
}
return R'
Ydob

Questo algoritmo lavora in spazio polinomiale rispetto all'input, in particolare in |T| * |C| ed è esponenziale nel tempo di esecuzione.

Come esposto in precedenza, l'orchestratore può essere considerato come una funzione che seleziona il servizio disponibile da invocare in base allo stato e all'azione corrente. Questa funzione è, pertanto, svincolata dai servizi disponibili e dal target e può essere costruita in base a delle regole. La costruzione di questa funzione è delegata ad un'entità chiamata OrchestratorGenerator (chiamato anche OG nel corso di questo documento), descritto nel seguente riquadro:

Def: **OG** = $< A, [1,\ldots,n], S_r, s_r^0, {}_r, {}_r, F_r >$ with
- A : the **actions** shared by the community
- $[1,\ldots,n]$: the **identifiers** of the available services in the community
- $S_r = S_t \times S_1 \times \ldots \times S_n$: the **states** of the orchestrator program
- $s_r^0 = (s^0_t, s^0_1, \ldots, s^0_m)$: the **initial state** of the orchestrator program
- $F_r \subseteq \{ (s_t, s_1, \ldots, s_n) \mid s_t \in F_t$: the **final states** of the orchestrator program
- ω_r: $S_r \times A_r \rightarrow [1,\ldots,n]$: the **service selection function**, defined as follows:
  - If $s_t \rightarrow_a s'_t$ then **choose** k s.t. $\exists s_k'.\ s_k \rightarrow_a s_k' \wedge (s_t', (s_1, \ldots, s'_k, \ldots, s_n)) \in S$
- $\delta_r \subseteq S_r \times A_r \times [1,\ldots,n] \rightarrow S_r$: the **state transition function**, defined as follows:
  - Let $\omega_r(s_t, s_1, \ldots, s_k, \ldots, s_n, a) = k$ then
    $(s_t, s_1, \ldots, s_k, \ldots, s_n) \rightarrow_{a,k} (s_t', s_1, \ldots, s'_k, \ldots, s_n)$ where $s_k \rightarrow_a s'_k$

L'entità orchestratore ha quindi le seguenti proprietà:

- la lista delle azioni
- dei servizi disponibili
- degli stati dell'orchestratore, iniziale e finali,
- delle funzioni di selezione del servizio a cui delegare l'azione dato lo stato dell'orchestratore
- delle funzioni di selezione del prossimo stato dell'orchestratore dopo aver applicato una funzione di selezione del servizio ω_r

È, quindi, possibile descrivere l’orchestratore attraverso un oggetto ben strutturato, in grado di effettuare le sue operazioni (simulazioni) in modo standardizzato ed in grado di poter essere gestito da sistemi di persistenza in contesti in cui le richieste di orchestrazione possono arrivare da vari client diversi, in momenti diversi.

Strumenti per la composizione automatica

Web services in Axis

Axis è un progetto Apache, scritto in Java come applicazione web da deployare in un web server J2EE come, per esempio, Tomcat. Axis consente di deployare ed eseguire web services, aggiungendo features che consentono un facile sviluppo in Java. L'applicazione si comporta come una SOAP library ed un SOAP listener che traduce richieste client, in SOAP, in chiamate a metodi nel proprio classpath.
Per deployare un web service in Axis ci sono due possibilità:

- JWS: deploy di file Java dento Axis
    - creare una classe Java, con alcuni metodi pubblici da invocare, che possono accedere a varie classi e librerie;
    - copiare il file sorgente della classe nella root della web application axis, cambiandole l'estensione da .java a .jws: verrà creato un nuovo servizio, che avrà il nome della classe;

Esempio:

```
// file axis/HelloWS.jws
package it.uniroma1.ws;
public class HelloWS {
        public String hail(String name) {
                return "Hail,  " + name + "!";
        }
}
```

- WSDL: deploy di classi Java con descrizioni WSDL:
    - creare una classe Java come interfaccia verso classi e librerie presenti nel classpath di Axis: costituiranno il backend del servizio;
    - creare il deployment descriptor: nomeClasse.wsdd, in XML, indicando il nome del servizio, il nome e package della classe java, la lista dei metodi offerti, eventuali tipi complessi, la tipologia di servizio (RPC, RMI, ecc), la tipologia di conversazione con il client (Request – stateless, Session – Stateful, Application);
    - effettuare il deploy utilizzando Axis, che utilizzerà le classi ed il WSDD per pubblicare il WSDL.

Esempio:

```
// file it/uniroma1/ws/HelloWS.java
package it.uniroma1.ws;
public class HelloWS {
        public String hail(String name) {
                return "Hail,  " + name + "!";
        }
}
```

compilato in it.uniroma1.ws.HelloWS

WSDD:

<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
	<service name="urn:HelloWS" provider="java:RPC">
	<parameter name="className" value="it.uniroma1.it.ws.HelloWS"/>
	<parameter name="allowedMethods" value="hail"/>
	<parameter name="scope" value="Request"/>
</service>
</deployment>

deploy del WSDD:

- invocando Axis: java org.apache.axis.client.AdminClient <file>.wsdd
- programmaticamente: org.apache.axis.client.AdminClient.main(<file>.wsdd)

Qualora il servizio dovesse utilizzare tipi diversi da quelli Java standard, chiamati Complex Types, è necessario che vengano dichiarati all'interno del WSDD. I complex types devono far riferimento a classi java che sono nel classpath di axis e che rispettano le specifiche dei Java Beans (hanno un costruttore senza argomenti, implementano java.io.Serializable, definiscono metodi getter e setter per ogni variabile pubblica) cfr. java beans

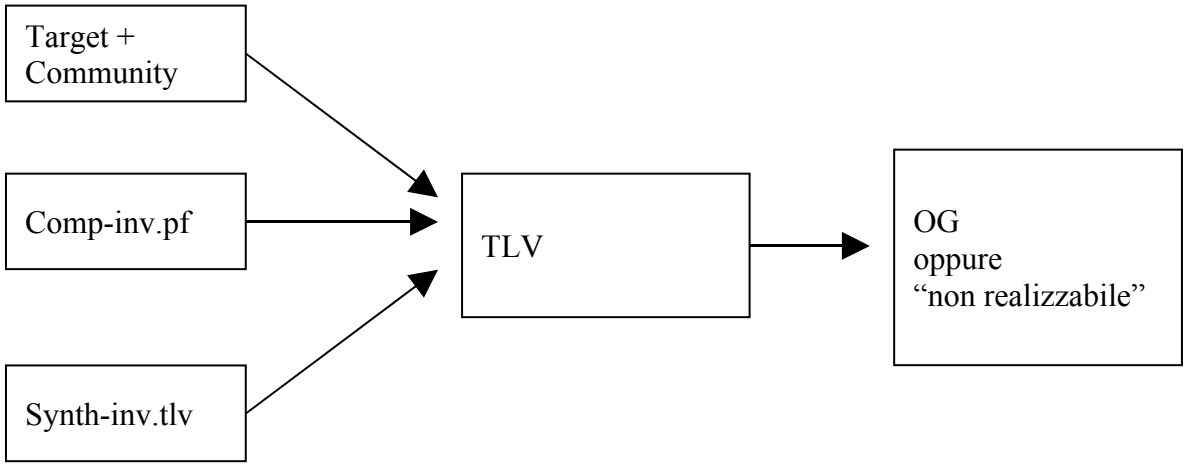
Esempio: supponendo di avere una classe bean Person, dovrà essere indicata nel WSDD come segue:
<beanMapping qname="myNS:Person" xmlns:myNS="urn:PersonList" languageSpecificType="java:package.Person" />
In questo modo verrà creato un qualfied name nel namespace myNS che descrive la classe specificata.

Tesine precedenti

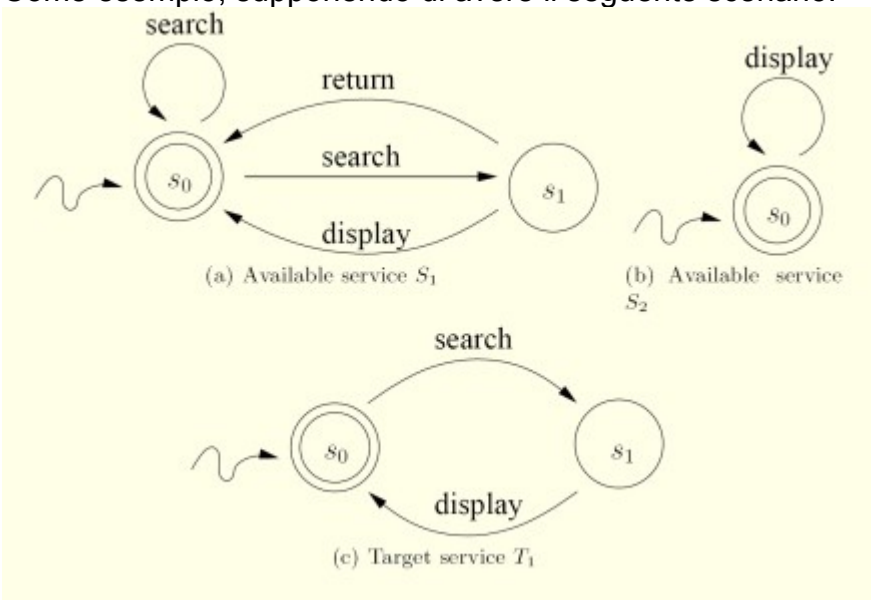
Questa tesina segue il lavoro svolto da due precedenti tesine, WSCE e Composizione ed Orchestrazione di servizi Web non deterministici.

WSCE ha come scopo generare automaticamente la logica temporale in SMV da utilizzare in seguito con TLV per generare le strutture dati richieste dall'orchestratore.

TLV è uno strumento basato su LTL in grado di calcolare l'orchestrator generator se la composizione esiste oppure in alternativa decidere che non è realizzabile. TLV si aspetta in input la descrizione della community attuale e del servizio target, da orchestrare, descritti in logica temporale in un file SMV. In particolare, il programma SMV è un insieme di moduli interconnessi, che descrivono i vari servizi e le interazioni tra essi.



Come esempio, supponendo di avere il seguente scenario:



in cui la community è composta da due available services (S1, S2) ed il servizio target è T1. Ne consegue che il file SMV necessario alle computazioni di TLV dovrà avere i seguenti moduli:

- un modulo mSi per ogni servizio nella community:

```
MODULE mS1(index,action)
    VAR
       loc : 0..1;
    ASSIGN
       init(loc) := 0;
       next(loc) :=
          case
             index != 1                : loc;
             loc=0 & action in {search} : {0,1};
             loc=1 & action in {display,return} : {0};
             TRUE                      : loc;
          esac;
    DEFINE
       failure :=
          index = 1 &
          !(
             (loc = 0 & action in {search} )|
             (loc = 1 & action in {display, return})
          );
       final := (loc = 0);
```

```
MODULE mS2(index,action)
   DEFINE
      failure :=
         index = 2 & !(action in {display});
      final := TRUE;
```

- un modulo mT1 per il target:

```
MODULE mT1(act)
VAR
  loc : 0..1;
ASSIGN
  init(loc) := 0;
  init(act) := nil;
  next(loc) :=
              case
                       loc = 0 & act = search  : 1;
                       loc = 1 & act = display : 0;
                       TRUE                    : loc;
              esac;
  next(act) :=
              case
                       act = nil               : {search};
                       loc = 0 & act = search  : {display};
                       loc = 1 & act = display : {search};
                       TRUE                    : {act};
              esac;
DEFINE
       final := (loc = 0);
```

- un modulo Input che definisce le interazioni tra i vari servizi e in particolare costruisce la variabile failure:

```
MODULE Input(index)
VAR
  action  : {nil,search,display,return};
  T1 : mT1(action);
  S1 : mS1(index,action);
 S2 : mS2(index,action);
DEFINE
failure := (S1.failure | S2.failure) |
           !(T1.final -> (S1.final & S2.final));
```

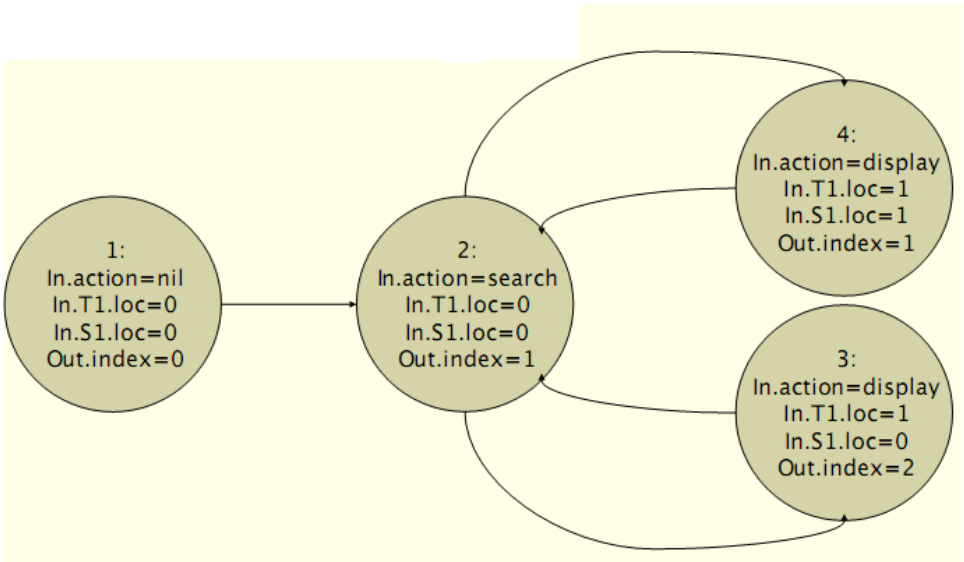
- un modulo Output che rappresenta la selezione del servizio:

```
MODULE Output
   VAR
      index:0..2;
   ASSIGN
      init(index) := 0;
      next(index) := 1..2;
```

il modulo Main che include Input e Output:

```
MODULE main
   VAR
      In: system Input(Out.index);
      Out: system Output;
   DEFINE
      good := !In.failure;
```

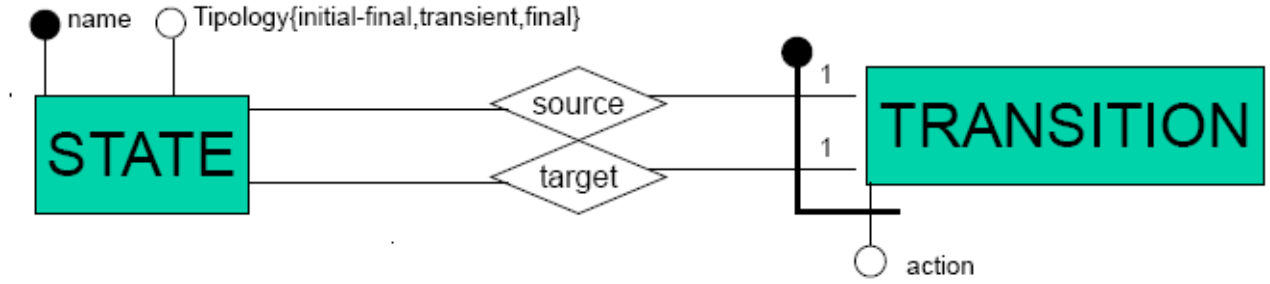
A questo punto TLV è in grado di capire se la composizione è realizzabile e produce come output il seguente orchestrator generator:



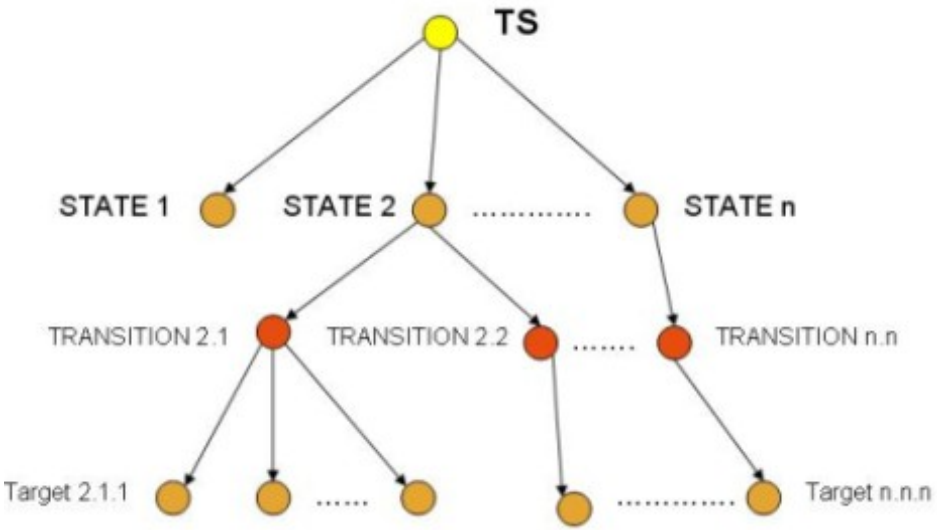
Per ogni stato dell'OG sono indicate l'azione da effettuare, lo stato dei servizi ed il servizio scelto per svolgere l'azione. L'output di TLV è testuale ed il lavoro di automatizzazione dell'orchestrazione viene continuato dalla tesina WSCE, che effettua il parsing di tale output in un insieme di strutture dati più utili.

WSCE

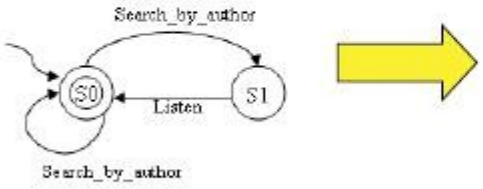
WSCE (a Web Service automatic Composition Engine) è un'applicazione sviluppata in Java che ha come scopo preparare l'input SMV per TLV e gestire il suo output. In particolare, WSCE prende in input il TS di un insieme di available services e del servizio target e, mediante chiamate a TLV, genera la tabella SQL che sta alla base dell'orchestrator generator. Il progetto è stato sviluppato da Valerio Colaianni come progetto universitario.
Dato che l'input da passare a TLV dev'essere una rappresentazione temporale dei TS dei vari servizi, è stato necessario creare un'ulteriore linguaggio che descriva il comportamento dei servizi. Questo linguaggio è basato su XML e si chiama WSTSL (web service transition system language) e può essere descritto utilizzando la seguente astrazione:



Il dominio di un TS è un insieme di stati e transizioni, legati tra di loro da relazioni source – target. È possibile navigare questa struttura, partendo dagli stati, vedendo le transizioni di cui sono sorgenti e, per ogni transizione, vedere l'insieme dei possibili stati target.:



Dalla navigazione ad albero consegue la descrizione in XML, dove ogni transizione avrà al suo interno una lista di riferimenti agli stati target.
Corollario: un TS deterministico avrà un unico target per ogni transizione.




```
<TS
    xmlns='http://www.dis.uniroma1.it/WS-TSL'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.dis.uniroma1.it/WS-TSL

   service="SearchByAuthorND">

    <STATE name="S0" tipology="initial-final">
        <TRANSITION action="SearchByAuthor">
            <TARGET state="S1"></TARGET>
            <TARGET state="S0"></TARGET>
        </TRANSITION>
    </STATE>

    <STATE name="S1" tipology="transient">
         <TRANSITION action="Listen">
            <TARGET state="S0"></TARGET>
        </TRANSITION>
    </STATE>

</TS>
```

Funzionamento di WSCE:

- prende in input l'insieme di descrizioni WSTSL degli available services nella community e del servizio target;
- costruisce i grafi TS corrispondenti;
- costruisce il file SMV associato;
- esegue TLV per valutare se la composizione esiste;
- se la composizione esiste, effettua il parsing del output testuale di TLV per ricavare la tabella SQL che rappresenta l'orghestrator generator.

Output di WSCE:
La tabella SQL prodotta consente di rappresentare lo stato corrente dell'orchestrator generator e le sue possibili azioni future.

Esempio: prendendo come esempio il servizio composto di ricerca e ascolto musica, la tabella generata sarebbe la seguente:

STATE	IN_ACT	TargetState	State_S1	State_S2	State_S3	OUT_ACT
S1	Login	0	0	0	0	1
S2	searchByAuthor	1	1	0	0	2
S3	searchByTitle	1	1	0	0	3
S4	Listen	2	1	1	0	2
S5	Listen	2	1	0	1	3

Il significato delle colonne è:

- STATE – stato dell'orchestrator generator
- IN_ACT – azione da seguire;
- TargetState – stato del servizio target;
- State_Si – stato dei vari available services;
- OUT_ACT – indice del servizio della community in grado di eseguire l'azione corrispondente alla transazione;

Questa tabella è coadiuvata da una seconda, anch'essa generata dinamicamente, che indica per ogni stato STATE quali sono i possibili stati successivi NEXT_STATE. Questa tabella, insieme all'informazione riguardo l'ultima azione eseguita, ci permette di conoscere con esattezza lo stato del target dopo ogni invocazione.

Composizione ed Orchestrazione di Web Services non deterministici

Questo progetto universitario, che sta alla base di questa tesina,è stato sviluppato da Flavio Palandri Antonelli e Alessandro Porfiri, con lo scopo di ultimare l'automatizzazione nell'orchestrazione di web services, partendo dalla tabella SQL generata da WSCE per mantenere lo stato dell'orchestrator generator e arrivando fino al deploy del target service generato. Il progetto ruota attorno alla realizzazione di due componenti software:

- TargetGenerator – prende la tabella SQL generata da WSCE e le descrizioni delle interfacce degli available web services (WSDL) per generare la clase java del target service e effettuare il suo deploy come nuovo web service;
- Orchestrator – gestisce l'orchestrazione di servizi: di fronte alle richieste del target, verifica le tabelle SQL per poter invocare il giusto servizio.

TargetGenerator:
Il compito di questo componente è sintetizzare il target service e pubblicarlo su axis come nuovo servizio.
Il funzionamento prevede i seguenti passi:

- prende in input il nome del target, la tabella SQL generata da WSCE, i WSDL dei servizi della community;
- crea il database completo per le operazioni dell'orchestrator generator;
- crea e compila la classe java del servizio target;
- crea e compila le classi java dei vari complex types presenti;
- crea il WSDL del servizio target;
- effettua il deploy del target service, ricavando l'indirizzo del web service appena pubblicato.

Orchestrator:
Il pattern di funzionamente prevede i seguenti passi:

- il target service accede a questo orchestrator per gestire le richieste dei client;
- l'orchestratore accede alle tabelle SQL dell'orchestrator generator per individuare il servizio da invocare;
- l'orchestratore invoca l'operazione richiesta e restituisce la riposta al target service;
- il target service risponde al client.

L'orchestrator è, inoltre, indipendente dalle varie composizioni e le sue istanze possono essere utilizzate in tutti i contesti. Questo suo basso accoppiamento con il resto della logica di composizione lo rende riusabile per tutti i processi. La sua presenza è trasparente al cliente esterno che percepisce il target service come unica entità.

Concorrenza e stato dei servizi

L'orchestrazione di target deterministici a partire da available services non necessariamente deterministici implica la conoscenza dello stato dei vari servizi in gioco, che deve essere utilizzato dall'orchestratore per poter decidere qual è il prossimo servizio da invocare.
Dato che il target service è deterministico, il suo stato può essere sempre ricavato dalle tabelle dell'orchestrator generator, data l'ultima azione richiesta. Questo però non è più vero per gli available services, il quale stato di possibile indeterminismo rende impossibile dedurre il comportamento in base alle azioni passate. Ne consegue che ogni available service nella community deve essere in grado di specificare il proprio stato ed offrire un metodo (int getStatus()) che lo descriva.
Questo implica che i vari servizi attualmente presenti nel mondo dovrebbero prima implementare questo metodo per restituire il proprio stato prima di poter essere composti ed orchestrati automaticamente.

Per poter gestire la concorrenza nell'accesso di vari client a vari target services è necessario che ogni client abbia associato il suo target service (e di conseguenza l'Orchestrator) e che lo possa utilizzare per tutta la durata della sessione. Il collegamento del target service alla sessione client avviene mediante la specifica del parametro scope, nel WSDD del target service:
<parameter name="scope" value="session" />

Inoltre l'Orchestrator deve mantenere aperte sessioni diverse con i vari available services, onde poter accedere ai loro status per decidere l'orchestrazione. La concorrenza è coerente rispetto all'accesso di più client ed è, quindi, garantita dal vincolo della sessione che rappresenta di fatto il correlation-id associato ad ogni client.

Limitazioni nell'accesso agli available services nella community

Per poter beneficiare della composizione ed orchestrazione automatica, gli available services devono soddisfare i seguenti vincoli:

- devono essere stateful e quindi definire uno scope di session;
- devono esporre un metodo int getStatus() che indichi lo stato corrente;
- i metodi offerti devono essere di tipo RPC (remote procedure call), in quanto RMI (remote method invocation) non è supportato;
- i complex types devono essere dichiarati nel WSDD prima di poter essere riferiti da eventuali altri complex types.

Problemi legati all'implementazione

L'implementazione dei componenti TargetGenerator ed Orchestrator effettua molte delle sue operazioni sfruttando processi separati, creando ritardi e a volte anche possibili deadlock, come evidenziato anche nella loro stessa documentazione. Questi problemi

verranno affrontati nel corso della nostra tesina, come parte di ottimizzazione e miglioramento strutturale.
In particolare si riscontrano chiamate a processi esterni nei seguenti casi:

- invocazione di WSCE;
- compilazione del codice java generato;
- esecuzione dell'AdminService per il deploy del target service.

Come descritto in seguito, abbiamo trovato un modo per evitare la creazione di processi esterni, riducendo l'overhead complessivo e rimuovendo anche gran parte della configurazione iniziale richiesta all'applicazione.

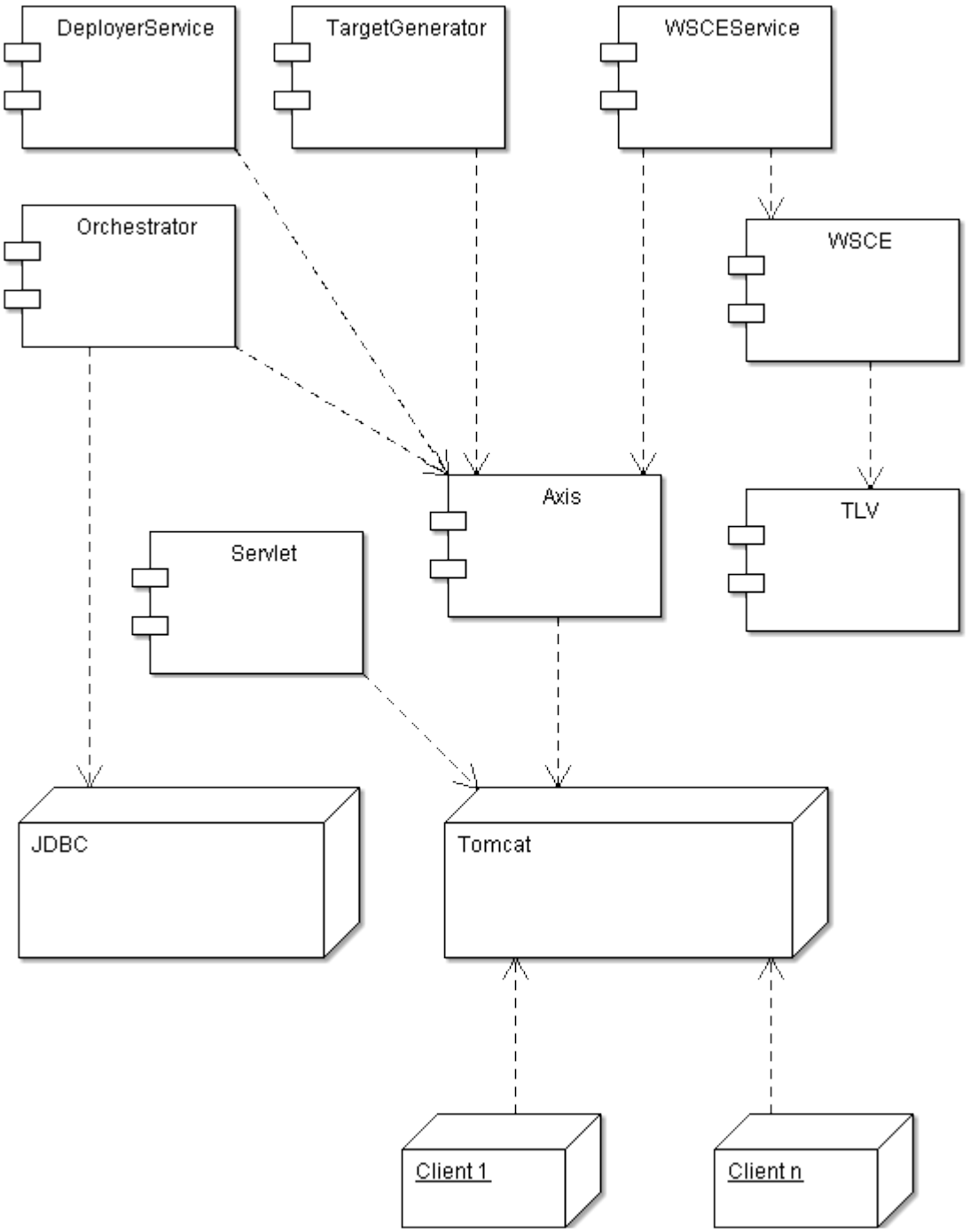
Composizione ed Orchestrazione remota di Web Services

Il progetto di Composizione ed Orchestrazione remota, argomento di questa tesina, si propone di portare avanti il lavoro svolto nei precedenti progetti per creare un insieme di servizi, deployabili in Axis, che offrano interfacce facili da usare ed automatizzino gran parte del lavoro di chi dovrà comporre ed orchestrare servizi.
La tesina precedente, pur svolgendo i suoi incarichi in modo sufficiente, risultava impossibile da usare per un utente remoto e anche difficile da gestire per un utente con accesso alle risorse locali.

Parte del lavoro svolto si è incentrato sulla comprensione ed integrazione dei progetti esistenti, onde migliorare la loro integrazione in modo più efficiente, in un unico sistema in grado di offrire le suddette funzionalità di composizione ed orchestrazione. In questa parte rientrano anche le varie modifiche a codice ed al sistema di configurazione iniziale, reso più modulare e disaccoppiato.
La seconda parte del lavoro si è incentrata sullo sviluppo di quelle funzionalità aggiuntive che rendono il prodotto finale più coeso e semplice da gestire. Abbiamo creato nuovi servizi, servlet integrative, ottimizzato vari aspetti dell'applicazione nel suo complesso, introdotto delle interfacce di log e standardizzato l'output del sistema. Inoltre abbiamo aggiunto un'interfaccia grafica di accesso e gestione (utilizzando la teconologia Adobe® Flex) e una per la creazione e gestione grafica di WSTSL e WSDD (utilizzando la tecnologia Adobe® Air).

Struttura dell'applicazione

In questa sezione verrà descritta la struttura dei componenti dell'applicazione.



Passi di sviluppo

Abbiamo visto la necessità di creare due nuovi servizi che completano la parte di web services necessaria per rendere l'applicazione utilizzabile in remoto:

- il **DeployerService**, che è in grado di effettuare il deploy di un nuovo servizio;
- **WSCEService**, che è in grado di invocare WSCE a partire da un WSTSL presente sul server.

Questi servizi sono corredati da due servlet, che sono complementari nel far gestire tutte le operazioni in remoto:

- **UploadServlet**: consente di effettuare l'upload di WSTSL, WSDL, WSDD e librerie JAR esterni;
- **CompositionDirectoryCreatorServlet**: dato un parametro workingDir, crea sul server la struttura dei folder necessaria ai vari componenti;

La servlet di upload utilizza le librerie Apache commons-upload e commons-io, da distribuire tra le librerie a runtime di Axis.

I nuovi servizi sono stati creati per ovviare al problema di dover gestire manualmente le varie fasi di generazione e deploy dei files intermedi prima dell'orchestrazione automatica. Il DeployerService può effettuare il deploy di un nuovo servizio, dati i suoi WSDD e directory di lavoro. Il WSDD e la directory devono essere presenti sul server e, una volta passati come parametro al servizio, esso chiama org.apache.axis.client.AdminClient per fargli effettuare il deploy. La chiamata ad AdminClient avviene utilizzando le librerie Axis presente all'interno della stessa web-application, ottenendo quindi un minimo overhead di accesso.
WSCEService prende come parametro il nome della directory di lavoro del servizio da generare, nome che è identico per convenzione al suo WSTSL, ed invoca WSCE, offrendo all'utente-gestore dei messaggi standard in output, descrittivi del risultato dell'invocazione. Come indicato nel paragrafo successivo, WSCE non viene più invocato in un processo separato. Inoltre, WSCEService fornisce un metodo per avere l'insieme di tutti i target ancora da generare presenti sul server (funzionalità utile per la gestione degli stessi) ed un metodo che forza il ricaricamento dell'applicazione axis.

Questi due nuovi servizi assumono che i files necessari siano tutti già presenti sul server di deploy. Per avere quindi un accesso remoto è necessario fornire dei mezzi per caricare le risorse necessarie, prima di poter effettuare la generazione e l'orchestrazione. Le due servlet aggiuntive consentono di espletare questa funzione.
La servlet di upload, UploadServlet, consente di caricare varie risorse:

- JAR: possono contenere le classi che definiscono il comportamento dei servizi da gestire nella community interna; il JAR va deployato tra le librerie dell'applicazione, entrando a far parte del suo classpath;
- WSDD, WSDL, WSTSL: data la working directory, che è dove il nuovo servizio dovrà essere generato, i nuovi files vengono scritti nelle corrispondenti sottocartelle (wsdd, wsdl, wstsl), pronti per essere utilizzati dai componenti di generazione ed orchestrazione.

Ovviamente l'upload di questi files deve trovare pronta la struttura delle directory, ed è compito di CompositionDirectoryCreatorServlet creare la directory di lavoro e le sue sottodirectory (wsdd, wsdl, wstsl), dato il nome del servizio da generare (che andrà a diventare nome della working directory). Se un servizio con quel nome è già presente, la servlet restituirà un errore.

Oltre ai servizi web e alle suddette servlet è stata creata una pagina "AxisAdmin.html" nella root della web application per chiamare velocemente questi servizi remoti.
E' stata inoltre creata un'applicazione per aiutare nella generazione di file WSTSL e file WSDD: WSGenerator.

Guide d'uso sono disponibili alla fine di questo documento.

Integrazione di WSCE

Per motivi di efficienza e facilità di distribuzione abbiamo deciso di distribuire WSCE integrato all'interno dell'applicazione. Questa operazione non è stata immediata, causa alcune particolarità nella struttura della libreria WSCE stessa.
La versione di WSCE distribuita precedentemente era costruita in modo da funzionare utilizzando esclusivamente il layout proposto nella sua distribuzione. In dettaglio, utilizzava una cartella Composizioni come sua sottocartella, dove si aspettava di trovare le cartelle contenenti i servizi da generare. Oltre al fatto che la cartella Composizioni era riferita solamente con un path contestuale, il suo nome ("composizioni") era embeddato come stringa in più punti nel codice sorgente, come ha dimostrato un'attenta fase di decompilazione.

E' stato quindi necessario trovare il codice sorgente di tale libreria e modificarla, onde renderla più portabile e compatibile con eventuali altre integrazioni ed aggiunte e svincolarla il più possibile da parametri troppo restrittivi. In particolare abbiamo dovuto inserire una classe singleton per definire una repository unica delle variabili comuni, come, per esempio, il nome della directory composizioni. Il nome di tale directory, inoltre, ora viene preso come parametro in input, utilizza un path assoluto e quindi può essere svincolata dall'ubicazione della libreria WSCE.

Ulteriori modifiche migliorano la portabilità della libreria su vari sistemi operativi (utilizzo di costanti per la gestione dei separatori di path) e le modalità di accesso alla libreria (metodi statici di ingresso). Per invocare i metodi della libreria non è più necessario passare per l'invocazione da riga di comando di un processo esterno ma è ora possibile utilizzare un metodo statico, TestComp.executeWSCE(). I parametri da passare a tale metodo sono:

- path completo directory composizioni;
- nome directory di lavoro, contenente i WSTSL;
- riga di comando: comando per eseguire processi esterni ("cmd.exe /c" per Windows NT o "bash" per Unix/GNU-Linux), necessario per poter invocare TLV; la riga di comando è necessaria per poter eseguire ed interpretare i risultati di TLV.

La nuova libreria WSCE è distribuita insieme all'applicazione, completa di codice sorgente modificato.

Ottimizzazioni e miglioramenti

L'esigenza di creare un'applicazione unica che offra servizi di generazione ed orchestrazione servizi porta a modifiche strutturali che includono l'unificazione del connection manager, l'integrazione di WSCE, miglioramento del sistema di configurazione e la rinuncia di affidarsi a processi separati per l'esecuzione degli altri componenti.

La classe ConnectionManager, che prima, per esigenze di progetto, era duplicata per il TargetGenerator e per l'Orchestrator, ora è diventata unica e fornisce le connessioni al database ad entrambi i componenti.

WSCE, come indicato nel paragrafo precedente, è opportunamente modificato e distribuito integrato nell'applicazione. WSCE è l'unico componente che utilizza processi esterni, in particolare TLV.
I componenti che prima erano invocati in processi separati, quali la compilazione delle classi generate, l'esecuzione dell'AdminClient di Axis, l'esecuzione di WSCE, ora vengono invocati utilizzando le loro API java, ottenendo un incremento delle prestazioni e dell'efficienza nell'utilizzo degli stessi. Scompare, così, il possibile problema del deadlock menzionato nella tesina precedente, in quanto tutti i componenti lavorano all'interno dello stesso spazio di memoria occupato da Axis.
I messaggi dei vari componenti, di informazione o errore, non vengono più lanciati verso il System.out ma utilizzano l'interfaccia di log offerta dalle librerie commons-logging, facilmente integrabile con quelle di Tomcat e con eventuali altri files esterni di log. In questo modo l'applicazione risulta più consona agli standard empiricamente affermatisi nel mondo delle applicazioni web Java EE.

Particolare attenzione è stata dedicata alla riscrittura del sistema di configurazione dell'applicazione. La tesina precedente proponeva un insieme di costanti statiche definite all'interno di un file Java, la modifica delle quali richiedeva la compilazione ed il nuovo deploy. Correntemente le proprietà di configurazione sono definite in un file di testo esterno al codice e alle classi, in formato standard properties (chiave – valore). Questo file viene in seguito letto dall'applicazione. È, quindi, possibile modificare le proprietà di questo file senza impattare sul codice sorgente e senza richiedere la nuova compilazione. Eventuali modifiche richiedono solamente il ricaricamento del contesto dell'applicazione axis, che può avvenire utilizzando il metodo definito ad-hoc nel servizio WSCEService oppure comodamente dalla console di Tomcat.
Come descritto nella documentazione dell'applicazione, è necessario modificare il file di configurazione per meglio descrivere il sistema host del server. Il file ha la seguente struttura:

- application.classpath – deve indicare il percorso di tools.jar, la directory classes dell'applicazione, il percorso di jaxrpc.jar;
- application.composizioni – nome della directory composizioni (Composizioni);
- application.tomcat – percorso della directory classes dell'applicazione deployata in Tomcat;
- axis.contextRoot – nome dell'applicazione (axis);
- axis.port – port del webserver Tomcat (8080);
- axis.prefix – percorso dell'applicazione deployata in Tomcat;
- db.compositionFile – nome del file generato SQL per le composizioni (composition.sql);
- db.driver – driver da utilizzare (com.mysql.jdbc.Driver in caso di MySQL, che si trova all'interno del jar connector);
- db.password – password di accesso, come definita nello script di creazione del database;
- db.schema – nome dello schema SQL da utilizzare (compositiondb);
- db.url – URL di accesso al database (per MySQL è jdbc:mysql://127.0.0.1:3306/compositiondb?autoReconnect=true);
- db.username – username di accesso, come definito nello script di creazione del database;
- shell.command - comando di accesso alla shell di sistema, per l'esecuzione di TLV (cmd.exe per Windows NT);

- shell.parameters - parametri aggiuntivi per la shell di sistema (/c per Windows NT).

Esempio di utilizzo

Il workflow della creazione di un servizio target usando gli strumenti messi a disposizione è costituito dai seguenti 10 passi (per informazioni sui singoli tool consultare le guide in appendice); ovviamente la sequenza viene modificata se ci sono dei servizi della community già dispiegati sul server o disponibili solo in remoto, si da inoltre per scontato che siano già disponibili le librerie compilate dei servizi della community:

1. Creazione di una nuova directory di lavoro tramite AxisAdmin
2. Creazione dei file WSTSL dei servizi della community e del servizio target usando WSGenerator
3. Upload dei suddetti file nella relativa directory di lavoro tramite AxisAdmin
4. Chiamata al Web Service “WSCE” sulla directory di lavoro tramite AxisAdmin per creare la composizione.
5. Creazione dei file WSDD dei servizi che devono essere eventualmente dispiegati sul server locale tramite WSGenerator
6. Upload dei suddetti file e eventualmente delle librerie java compilate (file jar) tramite WSGenerator
7. Chiamata al Web Service “Deployer” per ogni servizio da dispiegare in locale.
8. Upload di tutti i file wsdl dei servizi in locale e remoti che fanno parte della community tramite AxisAdmin
9. Chiamata al Web Service “Target Generator”, che, nel caso sia andato tutto a buon fine, resituirà l'URL del servizio target.
10. Godersi un meritato caffè.

Conclusioni e sviluppi futuri

Lo scopo che abbiamo cercato di raggiungere in questa tesina è stato quello di rendere il processo di creazione di un servizio target semplice e lineare, oltre ovviamente a rendere tutto ciò accessibile non solo localmente ma anche in remoto.
Alcuni spunti sono stati presi dalla sezione “Future Works” della tesina di Alessandro Porfiri e Flavio Palandri Antonelli come quello di creare un tool grafico per creare file WSTSL o quello di creare uno strumento di upload.
Pensiamo di aver fatto molti passi avanti, ma sicuramente ce se sono altri da fare.
Per esempio sarebbe opportuno creare una community centralizzata con un repository di web services liberamente utilizzabili, ognuno con il suo wsdl e il suo wstsl, in modo da renderli facilmente integrabili in una composizione.
Per raggiungere questo scopo è fondamentale la distribuzione di un'interfaccia, come già veniva ipotizzato nella suddetta tesina, con le funzioni da implementare obbligatoriamente (allo stato attuale solo getStatus),in tal modo si potrebbe definire uno standard da seguire per i vari fornitori di WS.

Riferimenti

Nella documentazione sono presenti vari riferimenti alle slide del corso di seminari di ignegneria del software: http://www.dis.uniroma1.it/~degiacom/didattica/semingsoft/

- TS-ServiceComposition-DeGiacomo-08-2up.pdf
- NondeterministicBehaviorsIJCAI07-2up.pdf
- Web service composition via TLV-08-2up.pdf
- PracticalServicesMecellaSIS2008-4up.pdf
- Berardi_etal@AISC2006.pdf
- ijfcs07-berardi-etal-final.pdf

Sono presenti anche riferimenti al libro:

Web Services - Alonso, Casati, Kuno, Machiraju

Vari spunti e codice sono stati ripresi dalla tesina di Alessandro Porfiri e Flavio Palandri Antonelli e da documentazione in essa contenuta.

Axis:
- http://ws.apache.org/axis/

Flex/Air
- www.adobe.com/products/**flex**/

Appendice A – Guida all'installazione

Ambiente e server

L'applicazione si appoggia su Axis 1.4 (http://ws.apache.org/axis/), un'applicazione Java EE che implementa il protocollo SOAP per la comunicazione client / server. Come suggerito nella documentazione di deploy di Axis, l'applicazione è stata costruita attorno alle classi e librerie di Axis e, quindi, deployata nella stessa applicazione web; ne consegue che l'applicazione distribuita include Axis.
L'applicazione utilizza TLV, che deve essere installato separamente. Per l'installazione e configurazione di TLV si faccia riferimento alla propria documentazione. In seguito si assume che TLV sia stato correttamente installato.
Per l'accesso al database si è utilizzato il DBMS locale MySQL 5.0, la cui configurazione verrà descritta in seguito. E', ovviamente, possibile utilizzare un qualsivoglia DBMS, fornendogli la giusta configurazione.

Per il deploy è stato utilizzato un server web Tomcat 6.0, mentre per la compilazione ed esecuzione dei servizi è stata utilizzata la versione 6 di Java. Prima di procedere con l'installazione è necessario seguire i seguenti passi:

- impostare le variabili d'ambiente JAVA_HOME e CATALINA_HOME in modo che indichino il path completo delle directory di installazione di Java e, rispettivamente, Tomcat;
- aggiungere la directory JAVA_HOME/bin nella path di sistema;
- per poter usufruire della compilazione automatica delle classi dei servizi, copiare la libreria tools.jar tra le librerie di Tomcat (copiare JAVA_HOME/lib/tools.jar in CATALINA_HOME/lib);
- per accedere a MySQL è necessario scaricare il connector MySQL per Tomcat, correntemente nella versione 5.1 (http://www.mysql.com/products/connector/j/) e copiare mysql-connector-{version}-bin.jar nella directory CATALINA_HOME/lib.

Deploy

A questo punto è possibile effettuare il deploy dell'applicazione. Scompattare l'archivio fornito all'interno della directory webapps di Tomcat: si verrà a creare una cartella *axis*, con il seguente percorso:

CATALINA_HOME/webapps/axis

La cartella axis contiene sia le librerie e i servizi di Axis sia quelli dell'applicazione. E' possibile ora avviare tomcat (eseguendo CATALINA_HOME/bin/tomcat6) ed accedere all'applicazione utilizzando il browser web preferito con il seguente indirizzo:

http://localhost:8080/axis/

se tutto va bene, è possibile verificare la correttezza dell'installazione di Axis dal seguente indirizzo:

http://localhost:8080/axis/happyaxis.jsp

Creazione database

Si assume che sia stato installato e configurato un DBMS ed una sua libreria connector sia stata fornita a Tomcat. Al primo avvio è necessario creare il database da utilizzare, con il seguente script:

```
set @@autocommit=0;
drop database compositiondb;
create database compositiondb default character set 'utf8' collate 'utf8_general_ci';
grant all on compositiondb.* to 'user' identified by 'pass';
commit;
use compositiondb;
set @@autocommit=1;
```

Questo script crea il database compositiondb ed un utente user (con password pass) da utilizzare con il suddetto database.

Configurazione applicazione

Si è cercato di rendere più facile possibile la parte di configurazione dell'applicazione. A tal scopo la fase di configurazione ruota attorno ad un'unico file editabile, in formato chiave-valore, *axis.properties*. E' possibile trovare questo file nel seguente percorso:

```
CATALINA_HOME/axis/WEB-INF/classes/axis.properties
```

E' consigliato rieffettuare il deploy dell'applicazione ogni volta che questo file viene modificato. Per rieffettuare il deploy andare nel pannello manager di tomcat (http://localhost:8080) oppure, semplicemente, cambiare il timestamp del file web.xml in modo che Tomcat rifaccia il deploy dell'applicazione (basta aprirlo in un editor semplice di testi, aggiungere o togliere uno spazio vuoto e salvare):

```
CATALINA_HOME/axis/WEB-INF/web.xml
```

axis.properties

Questo file contiene le proprietà configurabili dell'applicazione; la modifica di queste proprietà non richiede la ricompilazione dell'applicazione. Questo file assume che le variabili di sistema JAVA_HOME e CATALINA_HOME siano state configurate correttamente.

Prendendo come esempio il file già riempito è, quindi, necessario configurare le seguenti proprietà:

- application.prefix - con il percorso completo della directory deployata axis;
- axis.tomcat - con il percorso completo della directory classes, all'interno di axis (axis/WEB-INF/classes);

E' consigliato modificare i parametri di accesso al database:

- db.driver - driver da utilizzare (com.mysql.jdbc.Driver in caso di MySQL, che si trova all'interno del jar connector);
- db.password - password di accesso, come definita nello script di creazione del

database;
- db.username - username di accesso, come definito nello script di creazione del database;

Queste modifiche dovrebbero concludere la parte di deploy e sarà possibile utilizzare l'applicazione, come descritto nel resto del documento.

Installazione WSGenerator

WSGenerator è un'applicazione sviluppata con le sdk AIR/Flex e necessita pertanto del runtime AIR installato sul computer; è possibile scaricarla per tutti i maggiori sistemi operativi sul download center Adobe: http://get.adobe.com/air

Per l'installazione è sufficiente eseguire il file WSGenerator.air e partirà un semplice wizard.

Appendice B – Guida all'uso di WSGenerator e AxisAdmin

WSgenerator

L'applicazione WSGenerator può generare file di tipo WSTSL, necessari a WSCE per generare la composizione, e WSDD, necessari ad Axis per effettuare il deploy di nuovi servizi web.

Generazione file WSTSL

Per cominciare a costruire un file WSTSL è necessario selezionare il primo tab in alto a sinistra chiamato "WSTSL", selezionato di default all'apertura dell'applicazione.
La creazione è costituita da 3 diverse fasi: l'inserimento del nome del servizio, l'inserimento degli stati, l'inserimento delle transizioni tra gli stati.
L'inserimento del nome del servizio si effettua semplicemente scrivendolo nel campo di testo "Service name" in alto.
Gli stati si inseriscono immediatamente più in basso, nella sezione "1 - Create states"; per ognuno dei singoli stati è necessario inserire il nome nel campo di testo "State name", selezionare la tipologia dello stato stesso dal menu a tendina, infine premere il pulsante "Add".
Si verrà man mano a creare una lista di stati inseriti, da cui è possibile effettuare rimozioni semplicemente premendo "Remove" in corrispondenza dello stato che si vuole eliminare.
Una volta completata la lista degli stati è possibile passare alla fase della creazione delle transizioni selezionando in basso il pulsante "2 - Create transitions", che aprirà la sezione corrispondente.
Sulla sinistra saranno presenti gli stati aggiunti precedentemente, per default sarà selezionato il primo; sulla destra invece la lista delle transizioni associate allo stato selezionato, ovviamente all'inizio quest'ultima sarà vuota.
Una volta selezionato lo stato di cui si vogliono creare le transizioni basterà premere il pulsante "Add", che creerà, nella lista alla sua sinistra, una transazione la cui azione associata sarà "actionName"; quest'ultima è modificabile tramite l'apposito campo di testo.
Alla destra del nome ogni transizione avrà il pulsante "Edit targets", la cui pressione farà aprire una popup composta da due liste: a sinistra tutti gli stati disponibili, a destra i target associati all'action (lista inizialmente vuota). Per aggiungere uno stato target per l'action selezionata, basterà trascinare con il mouse uno stato dalla colonna di sinistra alla colonna di destra. I target possono essere rimossi premendo l'apposito pulsante "Remove". Premendo invece "Ok" la finestra popup viene chiusa.
Per rimuovere una transazione è necessario selezionarla e premere il pulsante "Remove" posizionato sotto "Add".
Nel caso in cui si vogliano effettuare delle rimozioni o aggiunte agli stati è possibile tornare alla sezione "1 - Create states"; nel caso in cui venga rimosso uno stato, andranno perse anche le transazioni associate.
Una volta terminate le suddette tre fasi, sarà possibile generare il file premendo il pulsante "Generate WSTSL" e scegliendo il percorso dove salvarlo (ricordandosi di usare l'estensione *.wstsl).

Generazione file WSDD

E' possibile effettuare la costruzione dei file WSDD accedendo al secondo tab in alto chiamato “WSDD”.
Ci si troverà di fronte un form da riempire con i dati che andranno a costituire il file WSDD finale.
Il primo campo di testo è il nome del servizio, cioè quello usato per accedere al servizio stesso in remoto e che sarà poi visibile nella lista dei servizi attivi di Axis.
Il secondo campo, “Fully Qualified Class name” è il nome della classe principale, cioè il punto di accesso del servizio, comprensivo del package java di appartenenza (es. “java.lang.String”).
Di seguito è presente un menu a tendina per scegliere lo “Style” del servizio in Axis. “RPC”, quello di default, utilizza le regole di encoding SOAP RPC; “Document” non utilizza alcuna codifica ma effettua comunque databinding Java<->XML; “Wrapper” è simile allo style “Document” con alcune differenze alla struttura del body SOAP; “Message” utilizza XML arbitrario. Per maggiori informazioni consultare la User's Guide di Axis.
Il check box “Enable Remote Administration” abilita richieste di amministrazione da host diversi dalla macchina in cui risiede Axis, è valido comunque solo per i servizi di amministrazione.
La sezione “Allowed methods” è formato da un check box chiamato “All” e una lista inizialmente vuota. Selezionando il check box si abilitano nel web service tutti i metodi pubblici presenti nella classe java specificata in precedenza. Se si vuole invece abilitare solo alcuni metodi è necessario deselezionare il check box e aggiungere alla lista sottostante i metodi desiderati tramite il pulsante “Add”. Se si vuole utilizzare il servizio in una composizione è necessario abilitare la funzione getStatus.
E' necessario poi specificare lo scope delle chiamate remote al servizio con l'ultimo menu a tendina. Perché il servizio possa funzionare in una community lo scope selezionato deve essere “Session”.
Infine è possibile specificare dei bean mapping per quei tipi non primitivi che Axis non supporta per il data binding; è sufficiente aggiungere le classi (che dovranno essere complete di setter e getter) nella lista “Bean mappings” non tralasciando il nome del package.
Una volta riempiti tutti i campi, sarà possibile generare il file premendo il pulsante "Generate WSDD" e scegliendo il percorso dove salvarlo (ricordandosi di usare l'estensione *.wsdd).

AxisAdmin

Nella root della web application di Axis è disponibile una pagina html AxisAdmin.html
Questa pagina contiene delle utility per rendere il processo di composizione completamente remoto senza la necessità di avere accesso al file system del server.
Prima di procedere con ogni azione è necessario controllare che i parametri impostati di default nei campi “Host name”, “Port number”, “Context root” siano corretti per il server corrente, viceversa è necessario modificarli.

Creazione directory di lavoro

Prima di procedere con una composizione è necessario creare una directory di lavoro nel quale tutti i file saranno creati/caricati. E' sufficiente scrivere il nome nel campo “Create new working directory” e premere il tasto “Create”. Dove verrà poi effettivamente creata questa directory dipende dalla configurazione interna del web server, che deve essere comunque del tutto trasparente all'utente esterno.

Upload dei file

Avendo a disposizione una directory di lavoro è possibile caricare i file per la composizione e generazione del target sul server, senza accedere direttamente al suo filesystem. Dal menu a tendina è possibile scegliere il tipo di file. In particolare il tipo “JAR library” consente di effettuare l'upload della libreria di un proprio servizio direttamente nel classpath della web application.
Nel campo “Working dir” bisogna riportare il nome della directory di lavoro per il quale si vuole effettuare l'upload.

Chiamare un Web Service di amministrazione

Nella sezione accessibile selezionando il tab “Call WS” è possibile effettuare chiamate SOAP ai web services che effettuano il deploy di nuovi servizi, creano la composizione, creano il servizio target.
Per effettuare il deploy di un nuovo servizio è sufficiente selezionare “Deployer” nel primo menu a tendina “Web Service”, specificare la working directory e infine specificare il nome del file wsdd che deve essere stato precedentemente caricato sul server relativamente sempre alla stessa working directory; premere infine il tasto “Call”, il risultato sarà mostrato nell'area di testo sottostante.
Se si vuole invece effettuare una nuova composizione bisogna aver prima effettuato l'upload di tutti i file **wstsl** dei servizi della community e poi sarà possibile selezionare “WSCE”, specificare la working directory, e infine premere “Call”.
E' possibile infine generare il servizio target, bisogna ovviamente aver già creato la composizione e inoltre aver caricato tutti i **wsdl** dei servizi della community, sia di quelli locali che di quelli remoti (specificando in questo caso nel wsdl stesso l'host remoto). Basterà quindi scegliere “Target Generator”, specificare la working directory e premere “Call”, nel caso la generazione va a buon fine, nel campo “Result” comparirà l'url del servizio web generato.