**CAPITOLO 13: WEB SERVICE**

*Nel seguente capitolo vengono illustrate le scelte effettuate per la realizzazione di una versione dell’applicazione mediante Web Service di tipo Soap e Rest.*

**13.1 Premessa**

Sempre più spesso un requisito importante nella realizzazione di un’ applicazione è la possibilità di supportare l’interoperabilità tra componenti in esecuzione su piattaforme diverse sulla base di protocolli standard aperti ed universalmente accettati. Per rispondere a tale vincolo, nell’ambito del lavoro di tesi, è stata realizzata una versione dell’applicazione che facesse uso dei Web Service.

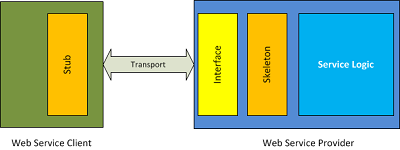
**13.2 Web Service**

Un Web Service, secondo la definizione data da Papazoglou, il cui scopo principale è quello di risolvere le problematiche legate all’interoperabilità tra processi di business spesso eseguiti da organizzazioni indipendenti con differenti piattaforme e linguaggi di programmazione, è:

* *un modulo o componente software, auto-contenuto e auto-descrittivo, accessibile mediante internet in modo indipendente dalla piattaforma*
* *ha lo scopo di svolgere un compito, risolvere un problema o condurre transazioni per conto di un utente o di un’applicazione – ovvero, di incapsulare un “servizio”.*

Nell’ambito del lavoro di tesi sono stati utilizzati sia i Web Service Soap che Rest. In particolare sono state realizzate completamente tutte le funzionalità dell’applicazione mediante Web Service di tipo Soap. I Web Service di tipo Rest sono stati usati per recuperare da Google Maps informazioni utili a contabilizzare il costo di un noleggio e geografiche. Per mostrare che i Web Service sono utili per far colloquiare client diversi è stato realizzato un caso d’uso in cui da un client Swing si richiedono informazioni sulle macchine presenti nel sistema.

**13.2 Web Service Soap**

****

Per realizzare una versione dell’applicazione basata su web service si è scelto un approccio di tipo “bottom-up” o “implementation first”. Nessuno sviluppatore infatti, dovendo realizzare un Web Service, inizierebbe a codificare il WSDL del contratto, in quanto trova più naturale la scrittura del codice di implementazione. Tutti gli ambienti di sviluppo, inoltre, dispongono di strumenti necessari alla pubblicazione di un Web Service a partire dalla logica applicatica, sia essa preesistente o di una realizzazione. Anche se, a rigore, un simile approccio contraddice il modello di progettazione orientato agli oggetti, secondo il quale la definizione delle interfacce è propedeutica all’implementazione della logica applicativa, tuttavia, questa tecnica ci permette di pubblicare agevolmente un Web Service. Il programmatore non si deve confrontare con la complessa realizzazione del WSDL, da cui gli ambienti di sviluppo sono in grado di derivare le classi di stub, endpoint e di trasformazione XML-Java.

Per la realizzazione di tale Web Service è stata usata l’API JAX-WS, acronimo di Java API for XML Web Services, tecnologia dedicata esclusivamente allo sviluppo di Web Service in Java che comunicano utilizzando messaggi SOAP/XML. Usando l’API JAX-WS, la definizione del servizio è semplicemente una classe con alcune annotazioni. Tali annotazioni forniscono pertanto dei metadati sulla classe e sui suoi metodi per precisare il comportamento del web service. L’unica annotazione obbligatoria è @WebService, che indica che una classe implementa un web service.

Per costruzione di tale Web Service è stata una web application di nome car2go-soapws. Come endpoint sono stati esposti le classi RuoloServant, UtenteServant, MacchinaServant e PrenotazioneServant che forniscono dei servizi per ogni tipo di entità dell’applicazione. In tali classi sono state usate le annotazioni @WebService e @WebMethod per indicare l’esposizione di tali oggetti come servizi remoti. Tali classi, per risolvere i loro compiti di business, devono interagire con il database e pertanto necessitano di elementi Dao. Per effettuare in modo corretto l’iniezione delle dipendenze è stata creata una classe Factory, che implementa il pattern Singleton, che ha lo scopo di indicare alle classi esposte come endopoint i riferimenti ai Dao. La Factory ha anche il compito di iniettare ai vari dai il riferimento a un oggetto di tipo Connection per l’interazione con il database.

Terminato di scrivere il codice lato server, per generare il WSDL è stato sufficiente utilizzare l’implementazione di JAX-WS offerta da Glassfish.

Lato client, per generare le risorse necessarie per interagire con il WebService, è stato utilizzato **wsimport**, un tool offerto da JEE. La sua funzione è quella di creare tutte le risorse definite in un file wsdl passato come parametro.

La sintassi da utilizzare è la seguente:

**wsimport** [opzioni] <url file wsdl>

le principali opzioni da tener presente sono:

* **-d <directory>** rappresenta la cartella dove verranno salvati i file di output (.class) generati.
* **-help**mostra la lista completa delle opzioni utilizzabili e la loro descrizione.
* **-p <pkg>** indica il package delle classi prodotte.
* **-s <directory>**indica la cartella dove salvare I file sorgenti (.java) generate.
* **-target <version>**indica la versione dello standard da utilizzare: 2.0 per la specifica JAXWS 2.0.

**13.3 Web Service Rest**

I Web Service di tipo Representational State Transfer (detti anche REST o “lightweight”) sono un’alternativa ai WS SOAP. I Web Service Rest sostengono l’interoperabilità e sono più efficienti rispetto a quelli di tipo SOAP infatti si basano sul protocollo http e richiedono una infrastruttura più leggera. Il concetto alla base dello stile REST è quello di “risorsa”, univocamente identificabile mediante un URI (Uniform Resource Identifier). Le risorse possono essere accedute, create, modificate oppure cancellate mediante messaggi autodescrittivi che invocano operazioni definite in corrispondenza con le operazioni tipiche di HTTP (GET, PUT, POST, DELETE).

Inoltre, i Web Services di tipo REST prevedono la possibilità di scegliere il formato di interscambio più opportuno tra XML, HTML o JSON.

Il caso d’uso realizzato con i Web Service Rest ha riguardato l’esposizione della collezione di macchine presenti nel sistema. Come primo passo, per ogni risorsa è stata definita un URI di base. Successivamente è stata creata la classe esposta come endpoint che ha utilizzato annotazioni del tipo @Path (per specifica l’URI), @GET (per restituire un elemento o una collezione), @POST (per inserire un nuovo elemento nella collezione), @DELETE (per cancellare un elemento), @PUT per aggiornare una risorsa.

Per quanto concerne il client che ha sfruttato i servizi messi a disposizione da Google Maps, le risposte ottenute mediante richieste http sono state interpretate facendo uso della libreria JSON Simple.