Relazione del progetto del Laboratorio di Applicazioni Internet

Federico Mariti

January 6, 2012

1 Descrizione del problema

Un albergo richiede la pubblicazione di due funzionalità: la ricerca di una camera e la prenotazione di una camera. Una camera dell'albergo è caratterizzata dalle seguenti proprietà:

- il numero massimo di persone adulte e di bambini che può ospitare,
- da un codice che identifica la tipologia della stanza, ad esempio matrimoniale, doppia, suite, etc ...,
- il costo per notte,
- un elenco dei servizi disponibili,
- un codice identificativo usato dall'albergo per riferirsi unicamente a tutte quelle stanze che rispettino le proprità precedenti,
- il numero di stanze con stesso identificatore diposonibili nell'albergo.

La ricerca di una stanza può essere effettuata specificando, oltre ad un periodo temporale, il numero di adulti e di bambini; per raffinare la ricerca possono essere aggiunte le altre proprietà di una stanza. La prenotazione di una stanza richiede, oltre che ad un periodo temporale, il codice identificativo con cui l'albergo riferisce alla stanza, e le informazioni sugli ospiti e sulla carta di credito con cui effettuare il pagamento. La prenotazione viene registarata con stanto "pendente" se la stanza specificata è disponibile per il periodo temporale specificato, successivamente con stato "confermato" a pagamento avvenuto. La prenotazione non viene registrata se non viene fornito l'identificatore della stanza, la richiesta è ambigua, o se la stanza specificata non è disponibile nel periodo di tempo specificato.

2 Le specifiche richieste dal progetto

Di seguito viene spiegasto come sono state realizzate le specifiche richieste:

- I due web service richiesti realizzano rispettivamente le funzionalità di ricerca di una stanza e di prenotazione di una stanza, sono descritti nei file WSDL [1, ricerca] e [2, prenotazione]. Per brevità, nel seguito, verranno chiamati *ricerca-ws* e *prenotazione-ws*.
- Di fatto l'approccio con cui si sono implementati entrambi i web service è stato quello *Top-Down*, ovvero per entrambi i web service si è partiti dalla definizione delle descrizioni WSDL, dopo di che si è implementato le funzionalità in Java. Sono stati usati gli strumenti forniti dal framework Apache CXF:
  - è stata usata l'utility `wsdl2java` per creare le classi skeleton del *prenotazione-ws*, l'implementazione della logica applicativa in [4] di tale web service prevede quindi l'uso di oggetti delle classi Java create,

  - il *ricerca-ws* ha tutte le funzionalità realizzate nel metodo `invoke` di una classe Java che implementa l'interfaccia `javax.xml.ws.Provider<SOAPMessage>`, per tanto l'implementazione della logica applicativa prevede la gestione manuale dei messaggi SOAP, ciò è stato realizzato con le sole funzionalità della SAAJ.

- Entrambi i web service interagiscono con il database dell'hotel per l'espletazione delle funzionalità.

- Tutte le comunicazioni utilizzano il protocollo HTTP per il trasporto delle informazioni. In alcune interazioni vengono realizzati dei livelli di sicurezza per mezzo della PKI X.509:

  - tutti i messaggi di risposta inviati dai web service ai client sono firmati,
  - nel messaggio di richista di prenotazione di una stanza le informazioni relative alla carta di credito sono cifrate.

- Il componenete applicativo Proxy è realizzato con una Servlet Java, l'analisi dei messaggi SOAP ricevuti viene effettuata unicamente con le funzionalità della JAX-P. Tale componente realizza anche la validazione del contenuto SOAP e del contenuto appplicativo (se non è cifrato) dei messaggi di richiesta ricevuti dai client.

- È stato realizzato un programma Java che agisce da cliente dell'applicazione e che interagisce con il Proxy. Tale programma è state-less e comandato da agromenti da riga di comando. Viene usato il framework Spring per la specifica WS-S.

2.1 Ulteriori specifiche

- Viene usata una unica macchina dove sono allocate tutte le funzionalità del lato server, ovvero il Proxy, i due web service, il server del database sono eseguiti tutti nello stesso nodo.

3 Messaggi SOAP e schemi XML

Si è adottato l'approccio *Top-Down* per l'implementazione dei servizi, perciò si è defito, prima di tutto, lo schema XML dell'applicazione, quindi si è data la descrizione dei web services, tramite due file WSDL, poi si è realizzata l'implementazione Java.

La definizione dello schema XML dell'applicazione è divisa in tre file XSD: uno di questi è `hotel.xsd` e defisce i tipi di dato generali, usati da entrambi i web service dell'applicazione, gli altri due file XSD sono `ricerca.xsd` e `prenotazione.xsd`, si riferiscono ad uno specifico web service, includono `hotel.xsd` e definiscono i tipi e gli elementi propri del corrispondente web service. Nella figura 1 sono mostrati, in modo informale, i principali tipi di dato definiti nei file XSD.

Per brevità siano:
hotelns = "http://www.hotel.com/hotel/",
ricercans = "http://www.hotel.com/ricerca/",
prenotns = "http://www.hotel.com/prenotazione/".

Il tipo `hoteln:stanza` descrive le proprità di una camera d'albergo come descritto nella sezione 1, non sono presenti le proprietà della stanza proprie dell'hotel, quali l'identificatore e il numero di stanze diponibili. La risposta ad un messaggio di ricerca stanza è descritta dal tipo `ricercans:ricercaStanzaResponse` ed è costituita da una sequenza di elementi di tipo `hotelns:rispostaHotel`, ovvero di `hotelns:stanza` seguito dall'identificatore usato dell'hotel per riferire la stanza stessa, la disponibilità effettiva della stanza nel periodo di tempo indicato nella richiesta, il costo della stanza stessa. Una sequenza di elementi `hotelns:rispostahotel` può essere presente anche nel messaggio di risposta alla prenotazione di una stanza, `prenotazione:prenotazioneResponse`, se la richiesta di prenotazione è ambigua (non presenta l'identificatore dell'hotel). In caso di prenotazione di una stanza registrata con successo viene ritornato l'identificatore con il quale l'hotel ha registrato la prenotazione,

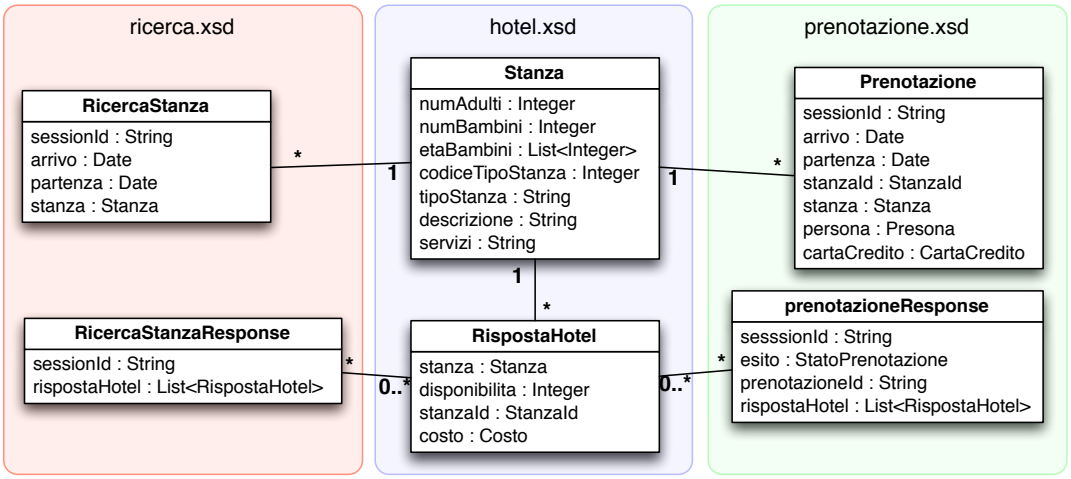

Figure 1: Rappresentazione grafica dei principali tipi definiti nei file XSD

utilizzabile successivamente dal client per verificare lo stato di prenotazione o cancellare la prenotazione.

3.1 Esempi del contenuto XML nei messaggi SOAP

Di seguito vengono presentati alcuni esempi di contenuti XML che verificano i due schemi, ricerca.xsd e prenotazione.xsd.

```
<ricercaStanza xmlns="http://www.hotel.com/ricerca/">
  <arrivo>2011-12-31</arrivo>
  <partenza>2012-01-28</partenza>
  <stanza xmlns:hotel="http://www.hotel.com/">
    <hotel:numAdulti>2</hotel:numAdulti>
    <hotel:numBambini>2</hotel:numBambini>
  </stanza>
</ricercaStanza>
```

Listing 1: ricerca di una stanza

```
<ricercaStanzaResponse xmlns="http://www.hotel.com/ricerca/">
  <sessionId>123456ABCDEFG</sessionId>
  <rispostaHotel xmlns:t="http://www.hotel.com/">
    <t:stanza>
      <t:numAdulti>2</t:numAdulti>
      <t:numBambini>2</t:numBambini>
      <t:codiceTipoStanza>1</t:codiceTipoStanza>
      <t:tipoStanza>suite per famiglia</t:tipoStanza>
      <t:descrizione>Suite per famiglia con matrimoniale e due singoli</
          t:descrizione>
    </t:stanza>
    <t:disponibilita>5</t:disponibilita>
    <t:stanzaId>1A0</t:stanzaId>
    <t:costo>
      <t:valuta>euro</t:valuta>
      <t:perNotte>90</t:perNotte>
      <t:totale>123</t:totale>
    </t:costo>
  </rispostaHotel>
  <rispostaHotel xmlns:t="http://www.hotel.com/">
```

```
    <t:stanza>
      <t:numAdulti>2</t:numAdulti>
      <t:numBambini>2</t:numBambini>
      <t:codiceTipoStanza>1</t:codiceTipoStanza>
      <t:tipoStanza>suite per famiglia</t:tipoStanza>
      <t:descrizione>Suite per famiglia con matrimoniale e due singoli</
          t:descrizione>
    </t:stanza>
    <t:disponibilita>1</t:disponibilita>
    <t:stanzaId>1A1</t:stanzaId>
    <t:costo>
      <t:valuta>euro</t:valuta>
      <t:perNotte>60</t:perNotte>
      <t:totale>123</t:totale>
    </t:costo>
  </rispostaHotel>
  <rispostaHotel xmlns:t="http://www.hotel.com/">
    <t:stanza>
      <t:numAdulti>2</t:numAdulti>
      <t:numBambini>2</t:numBambini>
      <t:codiceTipoStanza>4</t:codiceTipoStanza>
      <t:tipoStanza>suite executive</t:tipoStanza>
    </t:stanza>
    <t:disponibilita>5</t:disponibilita>
    <t:stanzaId>4A0</t:stanzaId>
    <t:costo>
      <t:valuta>euro</t:valuta>
      <t:perNotte>120</t:perNotte>
      <t:totale>123</t:totale>
    </t:costo>
  </rispostaHotel>
</ricercaStanzaResponse>
```

Listing 2: risposta alla ricerca di una stanza

```
<prenotazione xmlns="http://www.hotel.com/prenotazione/">
  <sessionId>123456ABCDEF</sessionId>
  <arrivo>2011-12-30</arrivo>
  <partenza>2012-01-06</partenza>
  <stanzaId>1A0</stanzaId>
  <stanza xmlns:t="http://www.hotel.com/">
    <t:numAdulti>2</t:numAdulti>
    <t:numBambini>2</t:numBambini>
    <t:etaBambini>2 12</t:etaBambini>
    <t:codiceTipoStanza>1</t:codiceTipoStanza>
  </stanza>
  <persona xmlns:t="http://www.hotel.com/">
    <t:codiceFiscale>ASDFGHJ123456789</t:codiceFiscale>
    <t:nome>mario</t:nome>
    <t:cognome>rossi</t:cognome>
    <t:email>mario.rossi@gmail.com</t:email>
    <t:telefono>123456</t:telefono>
  </persona>
  <cartaCredito xmlns:t="http://www.hotel.com/">
    <t:Tipo>visa</t:Tipo>
    <t:Numero>1234</t:Numero>
    <t:Identificatore>1234</t:Identificatore>
    <t:scadenzaAnnoMese>2012-12</t:scadenzaAnnoMese>
  </cartaCredito>
</prenotazione>
```

Listing 3: prenotazione di una stanza

```
<prenotazioneResponse xmlns="http://www.hotel.com/prenotazione/">
  <sessionId>123456ABCDEF</sessionId>
  <esito>PD</esito>
  <prenotazioneId>9876ABCD</prenotazioneId>
</prenotazioneResponse>
```

Listing 4: risposta che conferma, con stato pendente, la prenotazione di una stanza

```
<prenotazioneResponse xmlns="http://www.hotel.com/prenotazione/">
  <sessionId>123456ABCDEF</sessionId>
  <esito>NL</esito>
  <rispostaHotel xmlns:t="http://www.hotel.com/">
    <t:stanza>
      <t:numAdulti>2</t:numAdulti>
      <t:numBambini>2</t:numBambini>
      <t:codiceTipoStanza>1</t:codiceTipoStanza>
      <t:tipoStanza>suite per famiglia</t:tipoStanza>
      <t:descrizione>Suite per famiglia con matrimoniale e due singoli</
          t:descrizione>
    </t:stanza>
    <t:disponibilita>5</t:disponibilita>
    <t:stanzaId>1A0</t:stanzaId>
    <t:costo>
      <t:valuta>euro</t:valuta>
      <t:perNotte>90</t:perNotte>
      <t:totale>123</t:totale>
    </t:costo>
  </rispostaHotel>
  <rispostaHotel xmlns:t="http://www.hotel.com/">
    <t:stanza>
      <t:numAdulti>2</t:numAdulti>
      <t:numBambini>2</t:numBambini>
      <t:codiceTipoStanza>4</t:codiceTipoStanza>
      <t:tipoStanza>suite executive</t:tipoStanza>
    </t:stanza>
    <t:disponibilita>5</t:disponibilita>
    <t:stanzaId>4A0</t:stanzaId>
    <t:costo>
      <t:valuta>euro</t:valuta>
      <t:perNotte>120</t:perNotte>
      <t:totale>1200</t:totale>
    </t:costo>
  </rispostaHotel>
</prenotazioneResponse>
```

Listing 5: altra risposta ad una prenotazione che mostra piu' alternative

Tali esempi sono presenti nella directory `hotelBaseDir/xml_testFiles/` la validazione di questi può venir effettuata con l'esecuzione di

```
$ java com.hotel.test.XmlTest -o test_validateXmlDocument
```

con la variablile CLASSPATH opportuna.

4 Note sulla realizzazione

4.1 Organizzazione dei sorgenti e delle risorse

Per la realizzazione del progetto si è utilizzato l'ambiente di sviluppo *Eclipse*, tutti i sorgenti e le risorse sono in `hotelBaseDir`, la base directory del progetto di eclipse, eccetto le librerie usate (apache-cxf, servlet-api, etc ...).

build.xml file *Ant* con dei target per eseguire programmi clients e di test. Non vi sono target per la compilazione, la compilazine è stata effettuata da Eclipse;

doc doucmentazione e la relazione del progetto;

sources/java i sorgenti java del progetto, sia client side che server side;

sources/resources/etc files di configurazione, in particolare i files properties di configurazione

sources/resources/keystore gli archivi java key store che contengono le chiavi RSA usate per realizzare la specifica WS-I;

WebContent definizioni di schemi XML, descrizioni dei web services ;

WebContent/WEB-INF configurazioni del servlet container e del framework Spring

xml_testFiles files xml per realizzare dei semplici test tramite programmi client quali `curl`.

5 Database

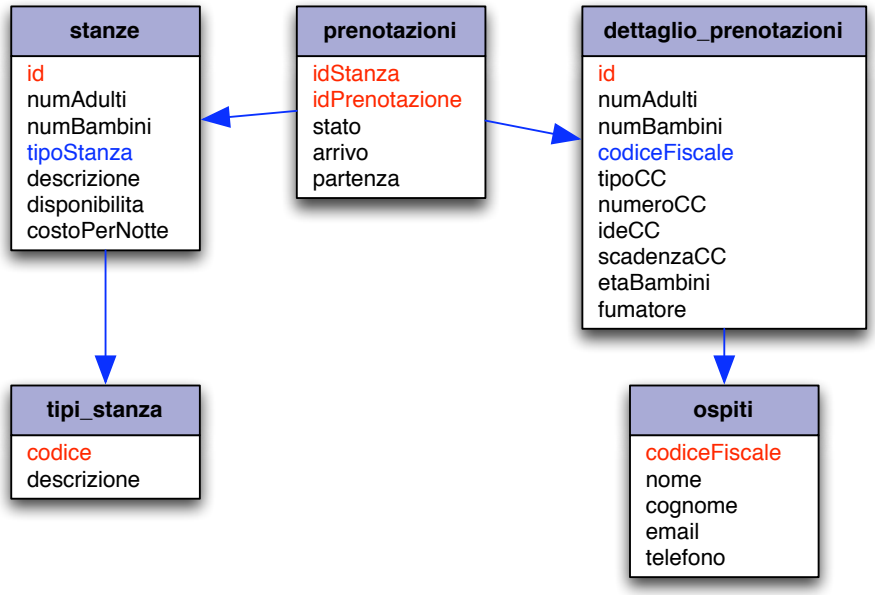

Figure 2: Tabelle del database

5.1 Interazioni con il database

Si sono descritti i tipi di messaggi che vengono scambiati per le operazioni di ricerca e di prenotazione, ...

5.1.1 Ricerca di una stanza

I parametri in ingresso all'operazione di ricerca sono:

1. data di arrivo (es. `2011-12-30`) e di partenza (es. `2012-01-02`),
2. numero di adulti e numero di bambini (es. `2`, `2`).
3. codice del tipo di stanza, non è obbligatorio (es. 1, stà per Suite Famiglia)

L'operazione di ricercaStanza consiste nella ricerca, nella tabella `stanze` del database, delle camere d'albergo che soddifano i parametri in ingresso *2* e *3*. Successivamente, per ogni stanza che soddifa i vincoli imposti, si contano il numero di prenotazioni che hanno un'intersezione non vuota con il periodo temporale specificato nel parametro *1*, tale valore è usato per determinare l'effettiva disponibilità di ogni stanza che soddisfa la richiesta.

Le query eseguite sono:

```
select * from stanze
   where tipoStanza = 1 and numAdulti >= 2 and numBambini >= 2 ;
```

Per ogni riga ritornata, sia ad esempio '1A0' il valore della colonna `id`, si esegue:

```
select count(*) as numPrenotazioni from prenotazioni
  where idStanza = '1A0' and (
    ( arrivo > '2011-12-30' and arrivo < '2012-01-02' ) or
    ( arrivo <= '2011-12-30' and partenza > '2011-12-30' )) ;
```

I risultati di queste query vengono usati per calcolare la disponibilità effettiva di ogni stanza nel periodo di tempo specificato.

5.1.2 Prenotazione di una stanza

I parametri in ingresso all'operazione di prenotazione sono:

1. identificatore della stanza, fornito dall'hotel in una precedente ricerca di stanza (es. `'1A0'`),
2. data di arrivo (es. `'2011-12-30'`) e di partenza (es. `'2012-01-02'`),
3. numero di adulti e numero di bambini (es. `2`, `2`) ed età dei bambini (non necessaria),
4. credenziali del cliente: codice fiscale, nome, cognome, etc... (es. 'ASDFGHJ123456789', 'Federico', 'Mariti', ...),
5. tutte le informazioni sulla carta di credito (es. tipo='visa', numero='123', identificatore='123', scadenza='2013-04').

Se l'identificatore della stanza non viene fornito l'operazione di prenotazione agisce come la ricerca, lo stato di prenotazione ritornato è nullo, `NL`. Altrimenti si interroga il database per ricavare il valore effettivo di disponibilità della stanza nel periodo di tempo specificato.

Vengono eseguite le query per determinare la disponibilità effettiva della stanza identificata da *1*:

```
select disponibilita as disponibilita from stanze
    where id='1A0' ;

select count(*) numPrenotazioni from prenotazioni
    where idStanza = '1A0' and (
    ( arrivo > '2011-12-30' and arrivo < '2012-01-02' ) or
    ( arrivo <= '2011-12-30' and partenza > '2011-12-30' )) ;
```

Se `disponibilita - numPrenotazioni` è maggiore di zero allora si procede con la prenotazione della stanza. Vengono effettuate query in lettura sulle tabelle `ospiti` e `prenotazioni` per determinare se il cliente è già registrato e quale sia l'identificatore dell'ultima prenotazione registrata. Supponendo che il cliente non sia registrato e che l'ultima prenotazione abbia identificatore 'AA0003', vengono eseguite le seguenti query:

```
insert into ospiti (codiceFiscale, nome, cognome)
    values ('ASDFGHJ123456789', 'Federico', 'Mariti');

insert into dettaglio_prenotazioni (id, numAdulti, numBambini,
    codiceFiscale, tipoCC, numeroCC, ideCC, scadenzaCC)
    values ('AA0004', 2, 2, 'ASDFGHJ123456789', 'visa', 123, 123,
        '2013-04');

insert into prenotazioni (idStanza, idPrenotazione, stato, arrivo,
    partenza)
    values ('1A0', 'AA0004', 'PD', '2011-12-30', '2012-01-02');
```

5.2 Attacchi SQL injection

È un tipo di attacco che si svolge invocando le funzionalità del sistema con parametri d'ingresso semanticamente non significativi, ma creati ad arte per scoprire informazioni sulla struttura del database, sul contenuto delle tabelle, ed anche per modificare il contenuto delle tabelle. Tale attacco va a buon fine se le query, descritte precedentemente, sono create come concatenazione di stringhe, ovvero i parametri in ingresso ricevuti dal sistema sono trattati come stringhe e concatenati tra loro per formare la query compessiva. Nell'implementazione Java dei web services, ad ogni interazione parametrica con il database, vengono usati oggetti della classe java.sql.PreparedStatement per costruire le query da eseguire nel database. Tutti i parametri della query, ricevuti in ingresso, vengono interpretati e castati al rispettivo tipo prima di essere compilati nella query tramite l'invocazione dei metodi set dell'oggetto PreparedStatement. Tale modo di procedere nega l'attacco SQL injection.

5.3 Problemi relativi alle transazioni concorrenti

Nelle sezioni precedenti sono state descritte le query eseguite sul database per effettuare le operazioni di ricerca e prenotazione di una stanza. In entrambi i casi viene eseguita una sola transazione costituita dalle query descritte. Dato che il DBMS esegue in modo concorrente le operazioni sottomesse da transazioni diverse, ovvero, non è garantita la proprietà di isolamento delle operazioni di una transazione, senza una adeguata gestione si possono verificare comportamenti non coerenti con quelli attesi.

Si ipotizza che le uniche operazioni eseguite sul database siano quelle di ricerca di una stanza e prenotazione di una stanza.

Si ipotizza che il DBMS realizzi la proprità *read safe transaction*, ovvero che la ricerca delle righe effettuata nelle operazioni di una transazione veda unicamente gli aggiornamenti consolidati, ovvero, effettuati da transazioni che sono state committate.

Analiziamo la transazione della ricerca, prevede una lettura della tabella `stanze` e più letture della tabella `prenotazioni`, tali letture sono riferite a sottoinsiemi disgiunti di righe. L'esecuzione concorrente delle transazioni non provoca particolari problemi. Non è possibile il fenomeno di *letture sporche* in quanto gli aggirnamenti non committati non vengono considerati. È possibile leggere valori non corretti del numero di prenotazioni di una certa stanza se dopo la lettura di tale valore vengono committate una o più operazioni di prenotazione della stessa stanza. Tuttavia i valori letti non vengono usati per discriminare una scelta nella transazione, ma sono restituiti subito all'utente[1].

[1] perciò possono diventare "non corretti" durante l'esecuzione della ricerca come anche in un futuro non precisato. Per tale motivo non si è ritenuto necessario l'accesso in sola lettura alla tabella prenotazioni durante la ricerca.

Analiziamo la transazione della prenotazione, prenvede la lettura delle tabelle `stanze` e `prenotazioni` e la scrittura nelle tabelle `ospiti`, `dettaglio_prenotazione` e `prenotazione`. Le scritture sono effettuate solo se le precedenti letture trovano la stanza richiesta per la prenotazione e se la sua disponibiltà effettiva è maggiore di zero. È possibile il verificarsi della *perdita di aggiornamenti* sulla tabella `prenotazioni` se dopo la lettura del valore del numero di prenotazioni della stanza vengono committate una o più prenotazioni che hanno intersezione non vuota con il periodo temporale della prenotazione. Ne segue che per una certa stanza e per un certo periodo temporale possono venir registrate un numero di prenotazioni maggiore della disponibilità della stanza. Per evitare tale situalzione è necessario imporre un vincolo di mutua esclusione nell'accesso in scrittura alle tabelle coinvolte. Un modo di fare ciò nello standard SQL è quello di imporre come *livello di isolamento* il *Serializzable*.

6 Dettagli sulla realizzazione

6.1 Proxy

Tale componente, realizzato da una Servlet, agisce da nodo intermedio nelle comunicazioni SOAP tra i clients ed i web services. Per ogni web service deve essere noto l'endpoint, la locazione dello schema XML e l'insieme dei nomi che può assumere l'elemento wrapper nel soap:Body (le operazioni). Anche se il problema considerato non prevede un elevato numero di serivzi e di operazioni, si è deciso di risolvere il problema della memorizzazione di tali informazioni in modo generale, al fine di garantire una agevole modificabilità ed estensibilità dell'applicazione. Tali parametri vengono impostati e/o aggiunti nel file `web.xml` senza conoscere (ne modificare) l'implementazione della classe che realizza il proxy. I valori di tali parametri vengono letti all'inizializzazione della Servlet che realizza il proxy, estendendo il metodo `init()`. Nel file `web.xml` esistono, per la servlet `Proxy` tre parametri iniziali: i nomi dei web services, gli endpoints, le locazioni degli schemi; il valore di ogni parametro è composto da più sottostringhe separate dal carattere 'spazio'; la prima sottostringa in endpoints o in schemas è relativa alla prima sottostringa in web services, e così via. Inoltre per ogni web service esiste un parametro il cui valore sono i nomi delle operazioni possibili, separati da 'spazio'.

In `Proxy` è stata definita una static-nested class, `Service`, che descrive un web service con i parametri menzionati precedentemente. Durante l'inizializzazione della servlet viene costruita un lista [2] di `Service`, tale oggetto è usato nel metodo `doPost` della servlet per verificare se l'operazione richiesta dal client è una di quelle accessibili, validare tutto il contenuto della richiesta [3] ed inoltrare il messaggio al web service corriposndente.

6.2 Database

```
for while if else sadf
 Schema |           Name           | Type  | Owner
--------+--------------------------+-------+--------
 public | dettaglio_prenotazioni   | table | mariti
 public | ospiti                   | table | mariti
 public | prenotazioni             | table | mariti
 public | stanze                   | table | mariti
```

Listing 6: Tabelle

```
   Column    |          Type          | Modifiers
-------------+------------------------+-----------
 id          | character varying(255) | not null
 numadulti   | smallint               | not null
```

[2] synchronized

[3] solo se il contenuto della richiesta non è stato cifrato

```
 numbambini     | smallint                | not null
 tipostanza     | smallint                | not null
 descrizione    | character varying(255) |
 disponibilita | smallint                | not null
 costopernotte | double precision        | not null
Indexes:
    ''stanze_pkey'' PRIMARY KEY, btree (id)
Foreign-key constraints:
    ''stanze_tipostanza_fkey'' FOREIGN KEY (tipostanza) REFERENCES tipi_
        stanza(codice)
Referenced by:
    TABLE ''prenotazioni'' CONSTRAINT ''prenotazioni_idstanza_fkey''
        FOREIGN KEY (idstanza) REFERENCES stanze(id)
```

Listing 7: Tabella: Stanze

```
     Column      |          Type           | Modifiers
-----------------+-------------------------+-----------
 idstanza        | character varying(255) | not null
 idprenotazione | character varying(255) | not null
 stato           | character varying(2)   | not null
 arrivo          | character varying(10)  | not null
 partenza        | character varying(10)  | not null
Foreign-key constraints:
    ''prenotazioni_idprenotazione_fkey'' FOREIGN KEY (idprenotazione)
        REFERENCES dettaglio_prenotazioni(id)
    ''prenotazioni_idstanza_fkey'' FOREIGN KEY (idstanza) REFERENCES
        stanze(id)
```

Listing 8: Tabella: prenotazioni

```
     Column      |          Type           | Modifiers
-----------------+-------------------------+-----------
 id              | character varying(255) | not null
 numadulti       | smallint                | not null
 numbambini      | smallint                | not null
 codicefiscale  | character varying(16)  |
 tipocc          | character varying(127) | not null
 numerocc        | smallint                | not null
 idecc           | smallint                | not null
 scadenzacc      | character varying(7)   | not null
 etabambini      | character varying(127) |
 fumatore        | boolean                 |
Indexes:
    ''dettaglio_prenotazioni_pkey'' PRIMARY KEY, btree (id)
Foreign-key constraints:
    ''dettaglio_prenotazioni_codicefiscale_fkey'' FOREIGN KEY (
        codicefiscale) REFERENCES ospiti(codicefiscale)
Referenced by:
    TABLE ''prenotazioni'' CONSTRAINT ''prenotazioni_idprenotazione_fkey
        '' FOREIGN KEY (idprenotazione) REFERENCES dettaglio_prenotazioni
        (id)
```

Listing 9: Tabella: dettaglio_prenotazioni

```
     Column      |          Type           | Modifiers
-----------------+-------------------------+-----------
 codicefiscale  | character varying(16)  | not null
 nome            | character varying(255) |
```

```
 cognome        | character varying(255) |
 email          | character varying(255) |
 telefono       | character varying(255) |
Indexes:
    ''ospiti_pkey'' PRIMARY KEY, btree (codicefiscale)
Referenced by:
    TABLE ''dettaglio_prenotazioni'' CONSTRAINT ''dettaglio_prenotazioni_
        codicefiscale_fkey'' FOREIGN KEY (codicefiscale) REFERENCES
        ospiti(codicefiscale)
```

Listing 10: Tabella: ospiti

6.3 Creazione dei certificati X.509 e dei java key store

```
$ cd <baseDir>/sources/resources/keystore/
$ keytool -genkey -alias hotel -keypass hotelpassw -keystore
hotel-keystore.jks -storepass storepass -dname "cn=Hotel" -keyalg RSA
$ keytool -list -keystore hotel-keystore.jks -storepass storepass  -v

Tipo keystore: jks
Provider keystore: SUN

Il keystore contiene 1 entry

Nome alias: hotel
Data di creazione: 5-gen-2012
Tipo entry: keyEntry
Lunghezza catena certificati: 1
Certificato[1]:
Proprietario: CN=Hotel
Organismo di emissione: CN=Hotel
Numero di serie: 4f056ebd
Valido da Thu Jan 05 10:34:53 CET 2012 a Wed Apr 04 11:34:53 CEST 2012
Impronte digitali certificato:
 MD5: AA:0F:F6:9F:B8:6B:BA:18:EC:E9:02:EB:60:76:CA:7F
 SHA1: 15:B5:FF:BB:A1:23:55:B2:60:82:7E:0A:39:29:5F:D2:47:B3:E4:8A


*******************************************
*******************************************

$ keytool -export -keystore hotel-keystore.jks -storepass storepass
-alias hotel -file hotel.rsa
Il certificato ? memorizzato nel file <hotel.rsa>
$ keytool -import -file hotel.rsa -keystore alice.truststore.jks
-storepass storepass -alias hotel
Proprietario: CN=Hotel
Organismo di emissione: CN=Hotel
Numero di serie: 4f056ebd
Valido da Thu Jan 05 10:34:53 CET 2012 a Wed Apr 04 11:34:53 CEST 2012
Impronte digitali certificato:
 MD5: AA:0F:F6:9F:B8:6B:BA:18:EC:E9:02:EB:60:76:CA:7F
 SHA1: 15:B5:FF:BB:A1:23:55:B2:60:82:7E:0A:39:29:5F:D2:47:B3:E4:8A
Considerare attendibile questo certificato? [no]:  si
Il certificato ? stato aggiunto al keystore
```

7 Esecuzione

7.1 Client

Il Client usa il framework *Spring*, controllato dalla configurazione in `hotelBaseDir/sources/-java/com/hotel/client/beans.xml`; in tale file sono cablati gli indirizzi su cui sono pubblicati i due servizi web. Per il testing dell'applicazione tali indirizzi sono caratterizzati da host = `localhost` e porta = `2020`. Prima di eseguire i processi client, localmente, viene mandato in esecuzione un programma che faccia da proxy (ad esempio TcpMonitor) con i parametri porta locale = `2020`, host remoto = `www.laiserver.com`, porta remota = `8080`; dove l'host e la porta remota fanno parte degli indirizzi in cui i servizi sono effettivamente pubblicati.

7.2 Uso di Ant

Per l'esecuzione con *Ant* deveno essere impostate le seguenti properties in `build.xml`:

- cxf.classpath,
- commons-cli.jar.dir.

Mostra l'aiuto:

```
$ ant -buildfile <hotelBaseDir>/build.xml client -Darg1="--help"
```

Esegue la ricerca:

```
$ ant -buildfile <hotelBaseDir>/build.xml client -Darg1="-o ricerca -b
2011-12-30 -e 2012-01-02 -a 2 -c 2"
```

ovvero:

```
$ ant -buildfile<hotelBaseDir>/build.xml client -Darg1="
> --operation ricerca
> --arrivo 2011-12-39
> --partenza 2012-01-02
> --numAdulti 2
> --numBambini 2"
```

Esegue la prenotazione

```
$ ant -buildfile <hotelBaseDir>/build.xml client -Darg1="-o prenotazione
-s 1A0 -b 2011-12-30 -e 2012-01-02 -a 2 -c 2 -p
ASDFGHJ123456789:Federico:Mariti:federico.mariti@gmail.com:1234 -C
visa:123:456:2014-01"
```

ovvero:

```
$ ant -buildfile build.xml client -Darg1="
> --operation prenotazione
> --stanzaId 1A0
> --arrivo 2011-12-39
> --partenza 2012-01-02
> --numAdulti 2
> --numBambini 2
> --persona
ASDFGHJ123456789:Federico:Mariti:federico.mariti@gmail.com:1234
> --cartaCredito visa:123:456:2014-01"
```

7.3 Invocazione da shell

```
$ CP_MY=build/classes/
$ CP_CXF=/usr/share/java/apache-cxf-2.2.6/lib/
$ CP_CLI=/usr/share/java/apache-commons-cli-1.2/commons-cli-1.2.jar
$ export CLASSPATH=$CP_MY:$CP_CLI:‘allJars.sh $CP_CXF‘
$ java com.hotel.client.Client --help
```

Dove `allJars.sh` è

```
#!/bin/bash
RESULT=
for j in ‘‘$@’’ ; do
    for i in ‘ls ‘‘$j’’‘ ; do
        if [ ‘‘${i##*.}’’ = ‘‘jar’’ ] ; then
            RESULT=’’${RESULT}:${j}/${i}’’
        fi

    done
done
echo ‘‘$RESULT’’
```

References

[1] baseDir/WebContent/ricerca.wsdl

[2] baseDir/WebContent/prenotazione.wsdl

[3] baseDir/sources/java/com/hotel/servlet/Proxy.java

[4] com.hotel.ws.ricerca.RicercaImpl