KTH V17P01 Programming Web Services: Homework 2

Fannar Magnusson (fannar@kth.se)
Thorsteinn Thorri Sigurdsson (ttsi@kth.se)

February 7, 2017

1 Introduction

This assignment focused on giving hands on experience developing web services and web services clients. We used the JAX-WS technology to develop the web services, using both a top-down approach and a bottom-up approach. We also worked with SOAP processing and used the SOAP header to carry an authentication token from server to client and client to server.

2 Implementation

2.1 Authorization service

We did two implementations of the Authorization service. One where we used the bottom-up approach (Java to WSDL) and another where we used the top-down approach (WSDL to Java).

2.1.1 Bottom-up approach

When we developed the Authorization service using a bottom-up approach, we begin code first and created a `AuthorizationService` class tagged with `@WebService` and implemented a `authorizeUser @WebMethod`. We used `wsgen` to generate JAX-WS artifacts for the service and `wsimport` to generate objects for the client side. We then implemented a test client that used some of the generated objects to test the authorize user web service method.

2.1.2 Top-down approach

When we developed the Authorization service using a top-down approach, we began by creating a WSDL document that describes the service. Particularly, it describes: the schema, the messages, the port information and the SOAP binding for the service, along with some general information, such as the location of the service.

When the WSDL document was done, we used the `wsimport` tool to generate the server side interface and other required objects from the document. We could then use the generated interface to implement the Authorization service.

We used the `wsimport` tool to generate the client side artifacts which we then used to develop a client to test out the Authorization service's authorize user method.

2.2 Itinerary service

We developed the Itinerary service using a bottom-up approach. We began by creating a `ItineraryService` class tagged with `@WebService` and implemented a `@WebMethod` that takes in a request from a client looking for a flight between two places and returns a list of possible itineraries. We used `jgrapht` to find paths between two destinations (using KShortestPaths, giving the 10 shortest paths available) [1].

We wrote a script that compiles the service and the beans and then uses the `wsgen` tool to generate JAX-WS artifacts for the service, such as an WSDL file. We used `wsimport` to generate artifacts for the client sidde. We could then use the generated artifacts to help us create a client to test out the itinerary service. The test client creates a mock bean object that he passes to the itinerary service and then prints out the received response.

2.3 Ticket service

We developed the Ticket service also with a bottom-up approach, in a way much similar to the Authorization service and the Itinerary service. The Ticket service implements two `@WebMethods`: one that takes in a list of itineraries and a date and returns a list of bookable itineraries with information about their price and number of available tickets. The other method takes in a bookable itinerary and books, issues and returns tickets for all the flights in the itinerary. As for the other bottom-up services, we used `wsgen` and `wsimport` to generate the artifacts and then implemented a client to test the web methods.

2.4 Authentication token

The authorization service returns an authentication token. The clients then use this token to authenticate with the other services.

We put the authentication token in the SOAP header of all outgoing requests from clients to a web service that requires authentication (i.e. all services except the authorization service itself).

[1]`http://jgrapht.org`

All services that require authentication then look for the authentication token in the SOAP header and check it against a known token before granting access.

We did this by implementing `SOAPHandler` classes that we inject in the SOAP handler chains of the clients and services. For all outgoing messages from clients we add the authentication token. For all incoming messages to services we check the authentication token, and return a `401 Unauthorized` if the token is invalid or missing.

The SOAP header with authentication token can be seen in figure 1.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
      <X:Security xmlns:X="http://pws-hw2-security.se">
         <X:BinarySecurityToken>thisisanauthenticationtokenthatshouldbebinary</X:BinarySecurityToken>
      </X:Security>
   </SOAP-ENV:Header>
   <S:Body>...</S:Body>
</S:Envelope>
```

Figure 1: Authentication token in SOAP header

2.5 Main client

In addition to the individual test clients developed for each service, we created a main client that goes through the entire process. It begins by talking to the AuthorizationService and getting an authentication token. Then it searches for available itineraries (flight paths) between two cities (Reykjavik and Riga) by talking to the ItineraryService. If there are available itineraries, it talks to the TicketService to get bookable itineraries for a given date, using the itineraries it got from the ItineraryService. The bookable itineraries contain the date, total price for the itinerary and the number of available tickets for that itinerary. It then books the cheapest bookable itinerary available.

Note that the main client does not have any user interface, rather it is just a test script that goes through the entire process, talking to each of the web services.