TravelGood

Final Project

Luca Cambiaghi, s161162
Anders Holmgaard Opstrup, s160148
Vitali Parolia, s130661
Andreas Rømer Hjorth, s134835

Technical University of Denmark
Department of Applied Mathematics and Computer Science
02267 Software Development of Web Services
November 29, 2016

1 Introduction

A web service can be described as a computing device which offers a range of services over the Internet. The web services communicate by using standard message protocols *HTTP requests, XML and SOAP messages*, among several others. The client initiates the communication to a web service and after that a cascade of communication with different services could happen depending on the complexity of the system. Figure 1 illustrates how a communication flow could be with a client and multiple services.

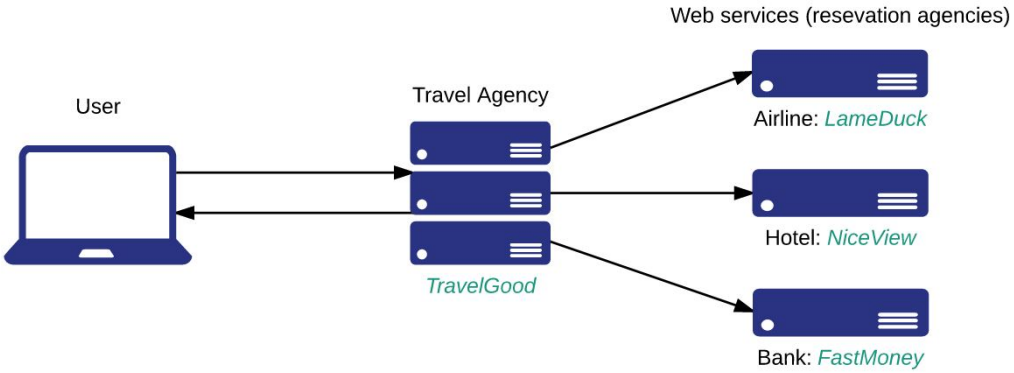

Figure 1: The orchestration layer.

An important distinction to make is between web service and web application: a web service is only the communication between two devices over the Internet where a web application includes a GUI part as well.

1.1 Data protocol introduction

Well known protocols are often used for communication between web services and clients. In this section two of the most common protocols will be described.

1.1.1 JSON

Java**S**cript **O**bject **N**otation is a very popular protocol for transmitting data objects among web services. It is most commonly used in the browser/mobile server communication, which in most places replaces existing XML communication.

The JSON-objects consists of a list of key value pairs, which makes it very readable and less prone to mistakes when creating the objects.

1.1.2 XML

E**x**tensible **M**arkup **L**anguage is another commonly used data protocol for web service communication. XML is often used in conjunction with the SOAP protocol to create SOAP Messages. SOAP messages will be discussed further in some later chapters of the report.

1.2 SOAP introduction

Simple **O**bject **A**ccess **P**rotocol is a protocol for exchanging structured information between web services. The protocol works in conjunction with the XML protocol as described earlier in the report. SOAP is independent which means it doesn't rely on any specific OS or programming environment. The SOAP protocol is also neutral about the transport layer of the communication between the services, which means it can function on any transport layer such as HTTP, SMTP og TCP. This makes the protocol very flexible. The SOAP message protocol consists of three parts:

Envelope

The envelope is the wrapper for the entire message. It contains information about which version of the SOAP protocol is used for formatting the message.

Header

The header part of a message is not required for the SOAP message to be valid. The header contains blocks of information which describes how the message is to be processed. Examples of information in the header could be information about type of encryption, target node for the message, representation of data or any other important information for the receiving web service for interpreting the message.

Body

The body of the message contains all the information intended for the receiving node, sent from the client/web service. In addition to the information sent the body can also contain a SOAP fault as child element of the body.

1.3 REST introduction

Repr**E**sentational **S**tate **T**ransfer is a stateless architecture design of web services, meaning that the server does not hold any state regarding a session: the user needs to provide all the information in the requests to the web service. REST web services rely on the application layer of the network communication to be HTTP. The restrictions to the application layer means that the REST application can only communicate with the standard HTTP keywords *(GET, PUT, POST and DELETE etc.)*.
REST applications are easy to conceptually understand, because of the above statements. The services a REST application offers are represented as resources, which the user can interact with. The resources are represented as web pages on the server, which means that the interaction with the services is through the path to the file on the server.

Example of interaction with a REST web service:

- POST - uri: facebook/userId/profile/photo [creates a new profile photo]
- PUT - uri: facebook/userId/status [updates the status of the user]
- GET - uri: facebook/userId [gets the user information]
- DELETE - uri: facebook/userId/profile/photo/photoId [deletes the profile photo]

1.4 BPEL introduction

Business Process Execution Language (BPEL) is a composition language used to specify the behaviour of a business process utilizing Web Services. A business process is a set of logically related tasks defined by business rules in order to obtain a specified outcome.
BPEL promotes a service-oriented architecture when designing a business process and provides a more abstract tool for modelling the orchestration between Web Services and the actual BPEL process.
The basic setup for a business process using BPEL has a main process which communicates with other services through what is called a *PartnerLink*. Partnerlinks are WSDL-defined services and a BPEL process will always have at least one client partnerlink that can instantiate an instance of the BPEL process. Besides at least one client partnerlink, the BPEL process can have invoked partnerlinks that it can utilize through the invoke command.

An example of a BPEL process is shown in the following flow chart.

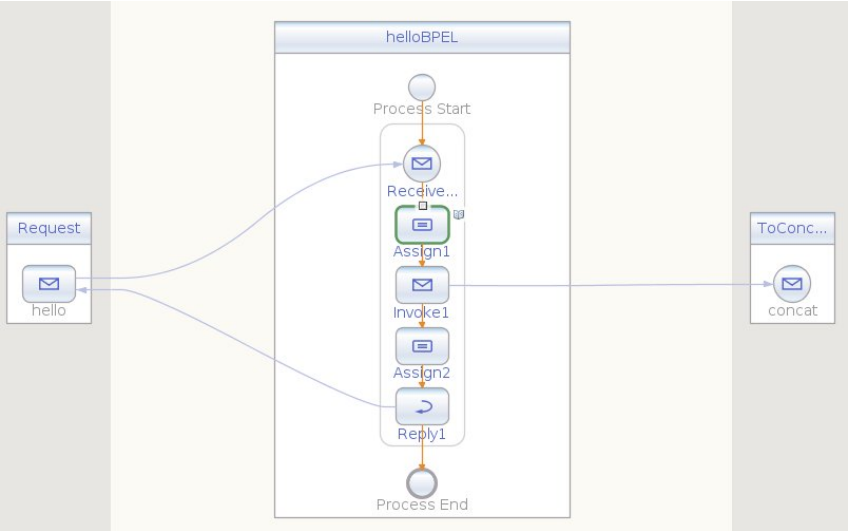

Figure 2: helloBPEL process designed in the openESB IDE

In the example on Figure 2, the Request service act as the client partnerlink and can, by the defined *hello* operation, instantiate an instance of the helloBPEL process, which is handled in the Receive activity of the process. The BPEL process then uses an invoke activity to call the web service ToConcat and finally uses the reply activity to "answer" the client with the business process result.
The following XML-code shows how this BPEL process is described in XML.

```
- <process name="helloBPEL" targetNamespace="urn:hello">
    <import namespace="urn:hello" location="hello.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="urn:concat" location="concat.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/concatWrapper" location="concatWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
  - <partnerLinks>
      <partnerLink name="ToConcat" partnerLinkType="tns:concatLinkType" partnerRole="concatRole"/>
      <partnerLink name="Request" partnerLinkType="tns:hello" myRole="helloPortTypeRole"/>
    </partnerLinks>
  - <variables>
      <variable name="ConcatOut" messageType="tns:concatResponse"/>
      <variable name="ConcatIn" messageType="tns:concatRequest"/>
      <variable name="HelloIn" messageType="tns:helloRequest"/>
      <variable name="HelloOut" messageType="tns:helloResponse"/>
    </variables>
  - <sequence>
      <receive name="ReceiveRequest" createInstance="yes" partnerLink="Request" operation="hello"
      portType="tns:helloPortType" variable="HelloIn"/>
    - <assign name="Assign1">
        + <copy></copy>
        + <copy></copy>
      </assign>
      <invoke name="Invoke1" partnerLink="ToConcat" operation="concat" portType="tns:concatPortType"
      inputVariable="ConcatIn" outputVariable="ConcatOut"/>
    + <assign name="Assign2"></assign>
      <reply name="Reply1" partnerLink="Request" operation="hello" portType="tns:helloPortType"
      variable="HelloOut"/>
    </sequence>
  </process>
```

Figure 3: The XML definition of the helloBPEL process

2 Coordination

In the following state machine a diagram of the client interaction for each created itinerary is displayed. When the createItinerary transition has been taken, the diagram shows the possible transitions and states relevant for that itinerary, thereby omitting the fact that a client would always be able to query for other itineraries.

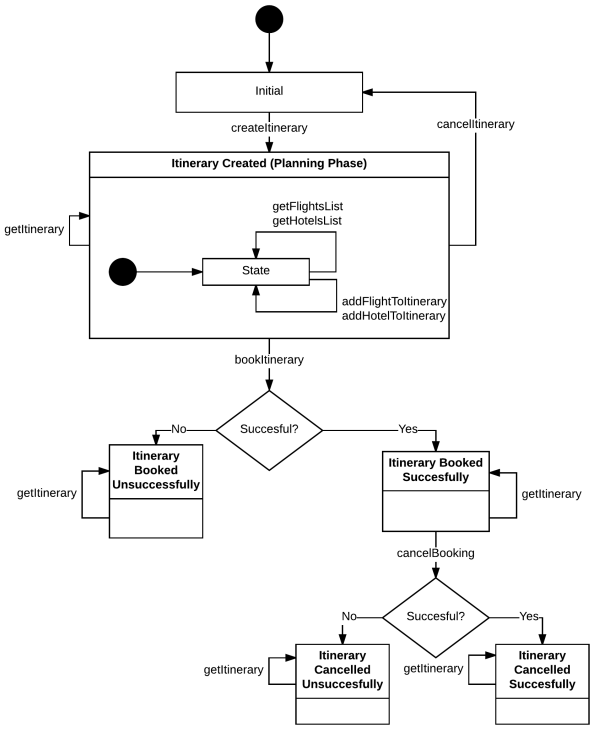

Figure 4: State machine for a single itinerary

3 Implementation

3.1 Data structures

Our implementation of the different web services rely on the following essential data structures.

3.1.1 Flight

Each Flight object has an ID and holds information that tells about the route, time and price of the trip.

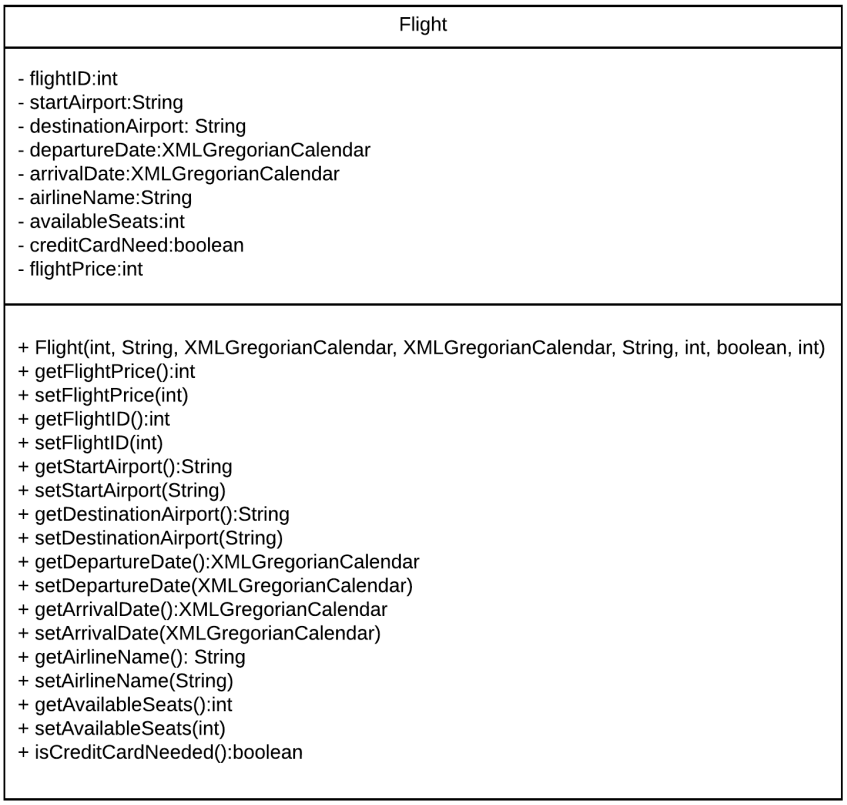

Figure 5: Class diagram for the Flight data structure

3.1.2 FlightInformation

The FlightInformation ckass has been implemented with the purpose of creating a link between a booking number and a Flight object. The FlightInformation object also tells whether or not a booking of the flight has been successfully achieved or not.

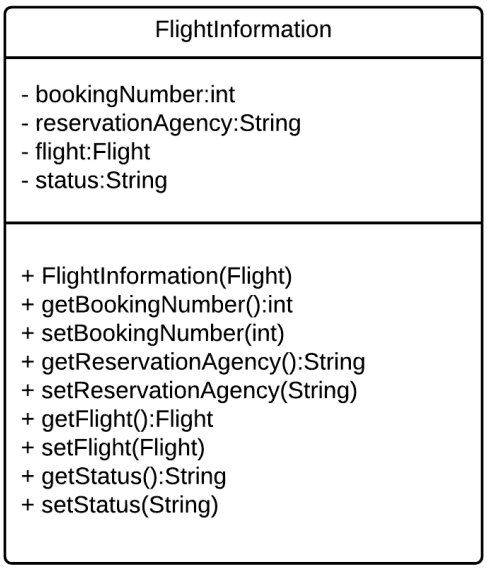

Figure 6: Class diagram for the FlightInformation data structure

3.1.3 Hotel

An instance of the Hotel class holds informations about hotel name, the price of a stay and the address of the hotel.

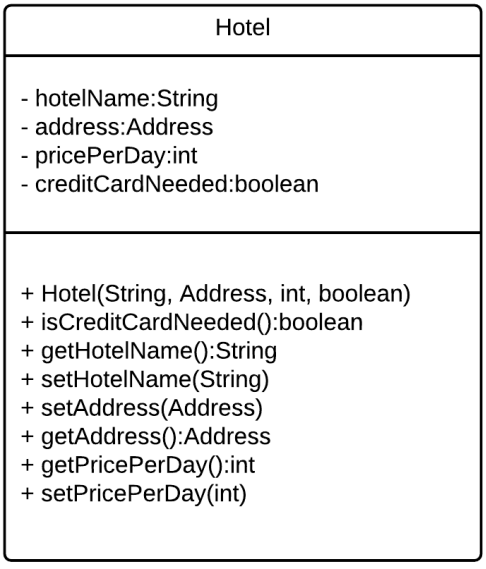

Figure 7: Class diagram for the Hotel data structure

3.1.4 Address

Address is the type used by the Hotel class explaining street and city.

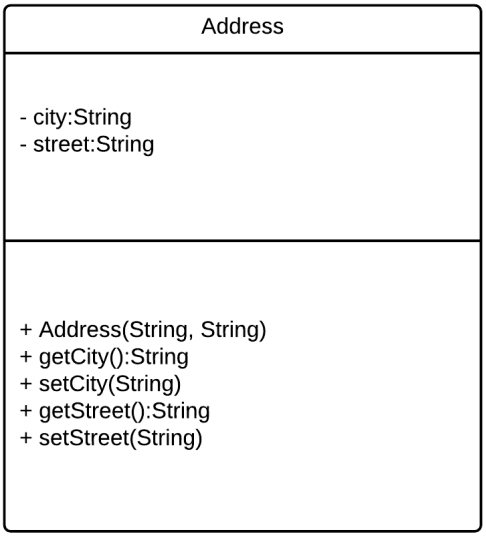

Figure 8: Class diagram for the Flight data structure

3.1.5 HotelInformation

The HotelInformation data structure acts as a link between a booking number and an actual instance of the Hotel class. If a Hotel is booked or cancelled, the status of the HotelInformation will reflect the changes.

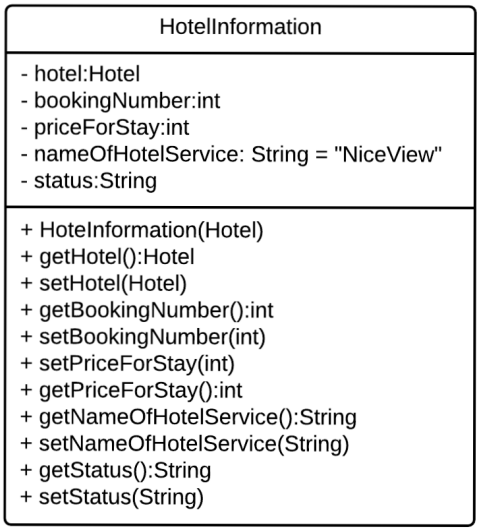

Figure 9: Class diagram for the HotelInformation data structure

3.1.6 Itinerary

Each instance of the Itinerary class has a unique id. The object holds information about the booking status of the whole itinerary and a list for both HotelInformation and FlightInformation representing the planned travels in the itinerary.

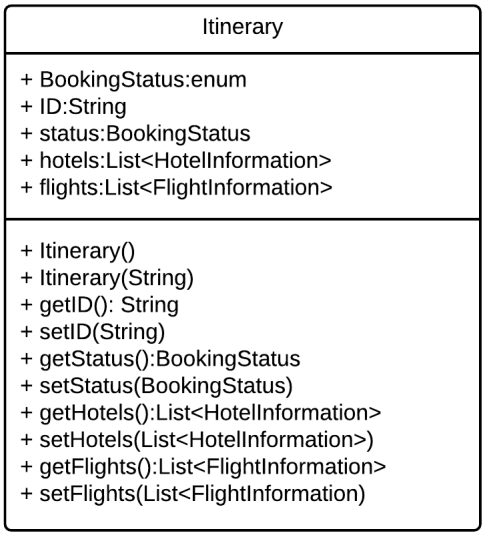

Figure 10: Class diagram for the Itinerary data structure

3.2 Airline and Hotel SOAP services

In this project we have implemented our SOAP web services using a bottom up approach and implemented it by writing the java code.
The basic structure for both the airline web service, Lameduck, and the hotel web service, Niceview, is that they have a controller class that makes use of the @javax.jws.WebService annotation such that the public methods are offered as web services. These 3 methods call methods implemented in the AirlineModel and HotelModel classes, in order to hide from the client the implementation logic. The methods in the two controllers correspond to the requirements specified in the problem description.

3.2.1 AirlineModel and Hotelmodel

The AirlineModel and HotelModel classes each contains three different methods, one for each of the available methods in the respective controller class.
For the AirlineModel class they are:

- *getFlights(startAirport, destinationAirport, departureDate)*
- *bookFlight(bookingNumber, ccInfo)*
- *cancelFlight(bookingNumber, ccInfo)*

For the HotelModel class, instead:

- *getHotels(city, arrivalDate, departureDate)*
- *bookHotel(bookingNumber, ccInfo*
- *cancelHotel(bookingNumber)*

Furthermore, each of the two model classes hold two lists intended as databases. One is the list of the flights/hotels available to the services. The other one is a list of the flights/hotels the client searched for. This list gets populated every time the user looks for flights or hotels.
This is done in order to add to the itinerary the objects the clients has previously searched for. Both lists get scanned and a match of the ID is used to retrieve the correct object.

The getFlights/getHotels methods are implemented to find flights/hotels of interest by doing a search of the flights/hotels available in flight/hotel list and checking if the input parameters can be matched with an object from the list. If an object is relevant for the specified criteria a FlightInformation/HotelInformation object will be created. The FlightInformation/HotelInformation contains the Flight/Hotel object, a randomly generated booking number and is stored in the list of FlightInformation/HotelInformation objects. When the getFlights/getHotels method has been called and FlightInformation/HotelInformation objects are available in the *flightsInformationDB*/*hotelInformationDB* the bookFlight/bookHotel method will be able to succeed. The bookFlight/bookHotel methods iterates over the list of FlightInformation/HotelInformation objects and checking if the parameter *bookingNumber* is matching the bookingNumber in one of the objects.

If so, the bookFlights will first try to validate the credit card information and then charge the correct amount from the account is done by the use of the Fastmoney Bankservice provided to us.
Whereas the bookHotel method will only try to validate the credit card information. If a booking is successful the status of the FlightInformation/HotelInformation object will be changed to "Confirmed".

The cancelBooking method finds the FlightInformation/HotelInformation with the booking number corresponding to the input parameter bookingNumber. If a FlightInformation has the status "Confirmed" the cancelFlight method will call the refundCreditCard method of the Fastmoney Bankservice and refund half of the price of the booking, whereas the cancelHotel method will just try to set the status to "Cancelled", because no money was charged when doing the booking.

To ensure that we can test a booking/cancellation failing we have implemented it such that if the start airport of the Flight object or the hotel name of the Hotel object is set to "BookingFailure" or "CancelingFailure", the method will throw an exception. We have defined four different exceptions to precisely handle all situations: Flight/HotelBookException, Flight/HotelCancelException.

3.3 BPEL

3.3.1 Complex data structures

We have defined some complex data types, in order to mirror the parameters and return elements of the operations provided by the two SOAP web services. These data types are:

- flightType
- hotelType
- flightsListType
- hotelsListType
- itineraryType

The flightType holds information about the flights available for booking, and the information needed to book the actual flight, such as flightID, bookingNumber. Similarly, hotelType.
The flightsListType is a sequence of one or more objects of the type flightType and is used when returning the response from the getflights call. Similarly, hotelsListType.
The itineraryType holds the ID of the itinerary and the status, telling whether the itinerary is booked or not. It also features two lists, of types flightsListType and hotelsListType, to indicate the bookings belonging to that itinerary.
Two SOAPFaults should have also been defined, to indicate the failure of a booking and the failure of a cancellation. These faults should have been thrown in correspondence of these two events in the business process.

3.3.2 Correlation sets

In order to have multiple instances of the BPEL process running on the BPEL engine we had to create a correlation set. We do this by defining a property, meant to represent the ID of the itinerary traversing the whole process. This property needs to have an alias in each message(receive and response) exchanged between the client and the server, to identify the correct itinerary.
Every receive and reply operation following the first ones, related to the creation of an itinerary, needs to be included in the same correlation set as the first two messages. It is important to note that the correlation set is instantiated on the first reply message, when the itineraryID is generated by the server using the GUID(Generate Unique ID) function.

3.3.3 Data flow

In this section we describe the data flow we would have liked to have implemented. We unfortunately could not do so due to several bugs that presented to us during the development of the BPEL module. Eventually, none of us was anymore able to reliably deploy and therefore test the BPEL composite application. Though, we feel that we have understood the core theoretical concepts and learned how to model a business process.

The BPEL process starts with the creation of an itinerary. After this, a loop on a pick operator is performed. This loop is meant to represent the whole process and the ability of the user to call whatever method at any time.
In this pick event, seven lanes are present, corresponding to the seven operations the user can do: getFlights(), getHotels(), addFlightToItinerary(), addHotelToItinerary(), cancelPlannedItinerary(), bookItinerary(), cancelBookedItinerary(). An eight additional lane is present, with an On Alarm operator, representing the end of the process when the time of the first booking comes.

The lanes representing the getFlights()/Hotels() have the same logic: they receive the request from the client, they invoke the operation, they use a ForEach operator to loop on the list of objects returned, assigning them to another list variable the process returns to the client.
Each of this two lanes has the same structure: an invoke is called on the desired method of the needed partnerlink and a for each loop is applied to the list of flight/hotel information the operations provide.

The lanes representing the addFlights/HotelsToItinerary() are also similar and simple. After receiving the message, the process adds the desired object to the itinerary thanks to an assign operator. Also the lane representing cancelPlannedItinerary() follows this simple logic, only setting the itinerary status to be cancelled.

The lanes representing the bookItinerary() and cancelItinerary() are the most complicated, including for each operators, combined with assign operators as we have seen in the getFlights()/Hotels() operations. They must also include fault handlers to front the events of booking/cancellation failures. Moreover, they have to include a rollback mechanism to revert the previous successful booking in case of failure of a following booking.

3.4 REST

This section will describe our implementation of the REST version of the TravelGood web service.

3.4.1 Resources

Itineraries
The itinerary resource is used for keeping track of the itineraries in the planning phase. There is a database storing all the itineraries created by the user. The user can create a new itinerary, retrieve it to check its status and add flights/hotels to the itinerary. It is important to note that an user can only create an itinerary using a POST word, in order for the ID to be handled by the web service rather than the client.

Booked
The booked resource is used when the user decides to book the planned itinerary. This is done to make a clear separation between the booked itineraries and the unbooked itineraries. Doing this, we also avoid the use of operations in the URI, like for example *http://travelgood.ws/itineraries/[itineraryID]/book*. On this resource the user can book an itinerary and cancel it, along with retrieving the status of the booked itinerary.

Airline and Hotel
The only purpose of these two resources is to search for flights and hotels to add to the itineraries in the planning phase. On these resource the user can therefore only call the *getFlights/Hotel()* methods.

3.4.2 Table of endpoints

URI Path = Resource	HTTP Method	Function
/itinerary	GET	Retrieves a list of itineraries
/itinerary	PUT	Creates a new itinerary
/itinerary/{itineraryId}	GET	Retrieves the itinerary
/itinerary/{itineraryId}	DELETE	Deletes the itinerary
/itinerary/{itineraryId}/hotel/{bookingNumber}	PUT	Adds a hotel to the itinerary
/itinerary/{itineraryId}/flights/{bookingNumber}	PUT	Adds a flight to the itinerary
/itinerary/reset	PUT	Clears the itinerary (test)
/bookings/{itineraryId}?name={n}&number={n}&expMonth={n}&expYear={n}	POST	Books the itinerary
/bookings/{itineraryId}?name={n}&number={n}&expMonth={n}&expYear={n}	DELETE	Cancels the booking
/bookings/reset	PUT	Clears the bookings (test)
/hotels/{city}?start={n}&end={n}	GET	Retrieves hotels for the itinerary
/flights/{startAirport}/{endAirport}?departureDate={n}	GET	Retrieves flights for the itinerary

Figure 11: Endpoints for the TravelGoodREST web service

3.5 Representation

To represent the itineraries, we created an `ItineraryRepresentation` class, extending the `Representation` class. This is done in order to implement the business logic providing links to the client. For every representation, we can set a URI and a Relation. So, for example, once the user creates an itinerary, he gets back a representation with the links to the planning operations like *getFlights()* and *getHotels()*, along with *getItinerary()*. We also specify a custom MIME type "application/itinerary+json" to be accepted in the header of the HTTP requests, to retrieve the representation with the links.

4 Web service discovery

In this section of the report, some of the different technologies for web service discovery are discussed further.

4.1 WSDL

Web **S**ervices **D**escription **L**anguage is a standard used for describing the web services. It describes their locations, operations they provide and what data structures they return. A WSDL file is a machine generated XML document. The WSDL is used in combination with SOAP and an XML Schema to provide Web services over the Internet. A client program connecting to a Web service can read the WSDL file to determine what operations are available on the server. Any special datatypes used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the operations listed in the WSDL file, using for example XML over HTTP.
The WSDL file contains different parts, such as *<types/>*, *<portType/>*, *<binding/>*, *<message/>* and *<service/>*, among others. Each of these parts describe the web service and its functionality. Depending on which version of WSDL the parts can have different names.

4.2 WSIL

The **W**eb **S**ervice **I**nspection **L**anguage is an XML based document that can help to explore web service definitions. It consists of different tags that can be used to describe the web service, where its specific definition is located and relevant links to explore other WSIL files relevant for the web service. Our WSIL documents for the NiceView, Lameduck and Travelgood services are located in the project zip-file in the root folder called WSIL.

4.3 WADL

Web **A**pplication **D**escription **L**anguage is the RESTful equivalent of the WSDL files used in the SOAP web service development. It is a platform and language independent description of the web service, promoted by the Sun Microsystems. The technology hasn't really caught any wide adoptions.
The WADL file contains different parts such as *<resources/>*, *<resource/>*, *<method/>* and *<param/>* among others.

5 Comparison REST and SOAP/BPEL Web services

In this section we will look further into the advantages and disadvantages of a RESTful approach and a SOAP/BPEL approach to web service development, to compare the two technologies.

5.1 REST advantages

One of the clear advantages of the REST architecture is its statelessness, which makes the implementation of the service very simple to debug. The fact that the technology is restricted to the HTTP application protocol also makes easy to understand how the communication is working in the service. Even though the application layer is restricted, the way of representing data is very flexible. The REST architecture supports a wide range of technologies for representing data, among them are JSON, XML, binary files, etc.
In web development where mobile first is a big focus, REST web services has a very good advantage of supporting JSON, which gets used extensively in the JavaScript/Mobile world of software development.

5.2 REST disadvantages

With the design of stateless web service, creating long running processes can be very complicated. The architecture of REST is not designed for asynchronous processing and invocation of long running processes, but rather simpler operations such as fetching data and representing it to the client.
Since the REST web services are not described by WSDL files, creating client testa for a REST web services can be quit cumbersome since there is no way of generating the data structures used involved. A manual way of creating stubs for the tests is needed, which is not very practical.
As mentioned, a REST web service implementation doesn't use a WSDL file, this means that web discovery can also be quit problematic. Some technologies have been created to solve this problem like the **W**eb **A**pplication **D**escription **L**anguage technology, but this technology hasn't seen any wide spread adoption.

5.3 SOAP/BPEL advantages

While REST is restricted to the HTTP application layer, SOAP/BPEL is very flexible. The application layer doesn't matter, which is a big plus for more complicated applications and internal communications between web services. If the application needs contextual information and conversational state management, then SOAP has the additional specification in the WS* structure to support those things *(Security, Transactions, Coordination, etc)*. Instead, the RESTful approach to these problems has to manually be implemented.
The use of WSDL files is also a clear advantage with SOAP/BPEL development: the WSDL files makes the web discovery much easier compared to the REST approach. The WSDL files are also used to auto generate stubs for the data structures used in the web service, which makes the testing of the web service easier.

5.4 SOAP/BPEL disadvantages

Some of the great strengths of SOAP/BPEL can also be some of its weaknesses. The auto generated files from the WSDL can sometimes lead to bugs in the code or mismatch between different packages. In bigger projects, the auto generation of sources files can also make the development process slow.

5.4.1 Conclusion of the comparison

Both technologies have strengths and weaknesses, none of them are better than the other, it all depends on the requirements of the implementation.
BPEL/SOAP would be a good solution if the requirements comprise internal communication within SOAP/BPEL services, where asynchronous processing or invocation of long running processes is needed. REST would be a good solution if the requirements comprise a web service which communicates with a browser/client, situation where JSON is preferred.

6 Advanced Web service technology

Eventually it would be beneficially to use all of the advance WS technologies. For the credit transfer we want to have security this is essential for the bank service. The policies would be good to use, but as long you are the only one whom uses them, then it wouldn't make sense using them. Addressing and reliability in web services is essential to improved the communication too ensure services sustainability.

6.1 WS-Addressing

WS-Addressing provides a light way specification of addressing endpoint information, a transport-neutral mechanisms to address web services and messages. Specifically, this specification defines XML elements. It defines a schema for a portable address construct reference. Main purpose of WS-Addressing is to incorporate message addressing information into web services messages. This enables the description of complex message path in the SOAP message header. Also in BPLE we are using addressing. It is used for automatic creating correlation sets, that dynamically addressing the invocation. To deliver the address information to the partner link the endpoint address follows the WS-Addressing schema as mentioned. On the other hand looking back on the problem with SOAP message is that it does not contain information of who sent a message and to whom it addressed, this is however solved with the independent encoding in the SOAP header. Where SOAP benefit using addressing, to identify message source.

6.2 WS-Reliable Messaging

With SOAP we basically achieving message exchange between endpoints over a unreliable communication channel of employment by different transfer protocols. The message might even be transformed through different protocols, or it might be modified or delegated for use by another web service before reaching its destination. This addresses the problem that the message could be lost during the transmission. The robustness of SOAP messaging does not guarantee that the message is being received by the receiver, therefore it's a clear need to have a realigning messaging routing in SOAP. With reliable messaging protocol, it can ensure that the message is arrived in sequences and it will be retransmitted until it has been acknowledged. Also the message can be detected in the order it has been sent. With the protocol which defines and supports a number of delivery assurances we can specify how to send and request acknowledgments, but which type to use is not a part of the protocol. It is obvious an advantage to use WS-reliable messaging it offers many advantages, but it might also introduce some disadvantage. This could be the system could get slower, since the processes would need to wait for acks before continuing with the flow. We also introducing sates to the system, which could be a disadvantage.

6.3 WS-Security

We adapt security into a web service by need to transport messages or packages in a secure network environment. SOAP messages are being transported throng HTTP that is in general unsafe. An intruder could easy intercept communication between two services and begin to mislead the information by either reading and changing the message. This is especially important aspect when we are handling the bank service, transporting vital data which we don't want to share with other which must be secured. Generally, we should secure all the communication, and establish a secure channel when data is being send in a network. To establish security, we need to identify the recipient, by an authentication proses throng the whole transport way. The HTTPS protocol might be used for the purpose, but this does not ensure confidentiality of the channel, nor the integrity or authenticity. The message can be secured, by several means by hashing, encrypted with different types or keys (privet or public) same as symmetric or asymmetric encryption. Message could be signed to make it more secure. The bit-length of the encryption are significant factor for how secure the encryption would be. In web services, we can apply security in SOAP header, by reusing existing XLM standards. For example, the namespaces and their prefixes or signing the body would be an advantage.

7 Conclusion

In this project a travel agency reservation service has been implemented using different web service technologies like SOAP, BPEL and REST.
For each service an analysis was performed, from which a clear view of the implementation emerges, highlighting the advantages and disadvantages of them. The report outlines basic principles of web service theory as well the practical aspects of the implementation, i.e. how these services communicate to each other, the involved protocols, the peculiarities of the different technologies.

The team as a whole agrees that this project has been a very interesting way to deeply learn two completely different approaches to web services: SOAP and REST. The project was of challenging size, requiring a lot of time to reach this state. However, the main problem is that most of this time has not been spent on the implementation itself, rather on fixing bugs of OpenESB/Glassfish server/Standalone server.
The topics are very interesting but it is much frustrating not to be able to create a working environment from the start to the end of the project. One member of the team was forced to reinstall the operative system on his computer two times in order to be able to deploy its projects, losing at least three full days of work due to some unidentifiable bugs, not even with the help of the professor or teaching assistants.
We would have also appreciated more focus in the course on the now widely used REST rather than on BPEL having such a central part in the project. Speaking of BPEL, it has given uncountable troubles to each member of the team: we arrived to a point where we had to give up on it, even if it was the last thing we wanted to do.

8 Who did what in the project

8.1 Code

- TravelGoodREST
  - AirlineResource - Anders
  - BookedResource - Luca
  - HotelResource - Andreas
  - ItineraryResource - Andreas
  - Representation - Luca
  - Tests - All
- SOAP Airline - Anders
- SOAP Hotel - Andreas
- BPEL - Luca

8.2 Report

- Introduction - All
- Coordination - All
- Implementation - Andreas, Luca and Anders
- Web service discovery Andreas and Anders
- REST vs SOAP/BPEL - Anders
- Advanced technologies - Vitali
- Conclusion - All