Technical University of Denmark

Project Report Software Development of Web Services

(Group 10)

Amandeep Dhir ,s094744

Fariha Nazmul, s094747

Khurram Bashir, s091370

Sameer KC, s094746

1 Introduction

Author -Amandeep Dhir

1.1 Report Layout

The report has been prepared according to the guidelines given in the problem description for the exam project as uploaded on the Campusnet. The work has been equally divided among all the four group members as specified in the instructions. The focus of the report is to outline both theoretical and implementation aspects of the exam project. We covered topics such as system design, developing different types of web services, interaction between the different services and solving problems encountered during the successful completion of the allocated task. The objective of this report is to understand concepts of web service technology, implement a complex solution, resolve problems faced while realising the solution practically and clearly documenting the various learning experiences during the course and exam project as a whole. The testing of the system under different scenarios is presented while results are analyzed.

1.2 Introduction to Web services

Author -Amandeep Dhir

Web service is defined as an Application Programming Interface (API) that can be accessed through Hypertext Transfer Protocol (HTTP) and executed on a remote machine which is hosting the requested service [Reference]. Web services are based on the service oriented architecture (SOA) which is latest evolution of distributed computing and web services architecture. SOA enables different software components, objects, application functions, and processes from different systems, to be exposed as services. The World Wide Web Consortium (W3C) has defined the web services as “a software system designed for supporting interoperable machine-to-machine interactions over a network”. It has an interface represented in a machine process able format called as Web Services Description Language (WSDL). Other systems communicate with Web service according to the specifications of the SOAP messages. The SOAP messages are conveyed through HTTP, XML and other web related standards.

Web services facilitate different applications from different sources to setup communication with each other. Moreover, the communication is established with less coding due to the use of XML. Web services are not tied with the programming language or operating system such as Unix application communicate easily with the Windows application. The same applies to the services written in different languages such as Java, C#, Perl, etc.

Overall, Web services expose any business logic in the form of services over Internet. They use programming interface having standard protocols for finding, subscribing and invoking these services. They are based on XML standards so they can be developed as loosely coupled application components which are independent to programming language, platform and protocol so developer has the complete freedom to choose any of them for development purposes. They are self describing and fit well for developing modular business applications. Moreover, they facilitates in delivering business applications as a service which is accessible anytime, anywhere and using any platform.

1.3 Basic Service Technologies

Web service technologies are majorly implemented through open standards and technologies taking advantage from XML. Web services refers to the standardized method for integrating Web-based applications using Extensible Mark-up Language (XML), Simple Object Access Protocol (SOAP), Web service description language (WSDL), Universal Description, Discovery, and Integration (UDDI) like open standards over Internet. These open standards (i.e. SOAP, UDDI, WSDL, etc) are also referred as building blocks of Web services. XML helps to tag data while SOAP is used for transferring data. WSDL helps in describing the available services and UDDI lists the available services.

- XML- It is a W3C recommended standard which is designed for carrying data rather than display data like other mark-up languages like HTML. It is self descriptive and due to this XML tags are never predefined. The new Web 2.0 XML can interface between two different environments. It can be used for describing both data and Web Services.
- 2. SOAP- It is a standard specification which is a W3C recommendation. It is used for exchanging information between different Web services and clients in a structured form. SOAP messages are used in the Web services for establishing communication between different applications. The benefits of using SOAP are platform - language independence and extensibility. SOAP messages are similar to XML. It also uses different application layer protocols such as RPC for invoking, negotiating and communicating data to different sections of the system.
- 3. WSDL- It consists of all kind of basic information about operations, messages, bindings and port type services that help in defining the Web service functionality and working. It also helps in locating Web services.
- BPEL- It is explained below in the (??)

1.4 Description of Web Services

Adding to the Introduction as we have discussed above - Web services are easily combined in loosely coupled manner so as to achieve complex operations. They are described as standard to establish interoperability (due to the extensibility and machine process able description) between different software applications making use of different frameworks and platforms. The messages (SOAP messages) are encoded using XML and standard protocols like HTTP, SMTP are used for transporting the messages. They support both synchronous and asynchronous messaging where the former provides one-way without acknowledgement and latter provides request/response structure.

1.5 Web Service Discovery

Web service discovery can be achieved through two ways -

- Web Service Inspection Language (WSIL) -It is a specification that uses XML schema for informing existence and location of the Web services.
- Central Repository -Service provider registers the service with a central repository or database so the clients can access this central repository for getting information about different services who have registered themselves with the server. Example UDDI is

defined as a method of discovering Web services and it is interoperable with SOAP and WSDL.

1.6 Web Service Composition

Today, Businesses are quickly transforming their operation to support customer needs and changing market demands so it required that our solution must be flexible both internally and externally. It is very difficult to attain this objective it companies do not work over a common set of standard rather than projecting their own proprietary protocols. Web Service Composition provides an open, standard based approach to connect different web services together so as to construct high level business processes. These standards can reduce the complexity required for composing web services for increasing efficiency and reduce cost, time, etc. There are several ways/standards of performing Web Service Composition such as WSCI(Web Services Choreography Interface), BPEL(Business Process Execution Language), BPML(Business Process Management Language), etc but we used BPEL in our implementation so we will explain it as follows-

BPEL -It is an OASIS standard executable language which is also a service composition language used for specifying the interactions with different Web services. A real-time business process consists of sequence of steps which are performed systematically hence these processes require interactions with each other so BPEL import and export information through Web service interfaces. The interactions discussed above can be categorized as Abstract Business Processes (ABP) and Executable Business Processes (EBP). ABP are partially specified which are not intended to be executed so it is possible that they are hiding some required concrete operational details where as EBP are close to real-time practical interactions as ongoing in any business application. Their aim is to model the real or actual behaviour of the entities participating in any business interaction.

1.7 Web Service Coordination

The operations in Web services can be defined through an operational model consisting of service provider, service registry and service requestor. These three are also referred as actors in Web service operational model. They can be described as below –

- Service Provider- It is responsible for defining, developing, deploying and finally publishing the Web services through service registry.
- Service Requestor- It is responsible for service registration and Web service discovery. It is also referred commonly as service broker but we will use service registry throughout this report. It is also responsible for listing various service descriptions, types and their locations as it will help service requester in locating and subscribing them.
- Service Registry- It is responsible for the service invocation where it can locate Web service using service registry as explained above. Overall, requestor will invoke the required services and execute them through the service provider.

1.8 Importance and why?

Web applications based on the client-Server model only provides interaction between the Web site and end user but Web services provide application to application interaction over the internet. The differences between Web services and traditional client-server Web page/Web server can be listed as -

- Web services do not provide any Graphical User Interface (GUI)
- Web services share business logic, processes and data through a programmatic interface across the network.
- Web services do not require HTML or browser
- Web service can be added to a GUI of a webpage or any executable application for providing specific functionalities to the end user.

Web services are service oriented, easily accessible and provide interoperability in the heterogeneous environment consisting of different applications and devices. This shows that Web services provide cross-platform and cross-language solution using XML messaging. The major benefit of using Web services is that it allows different organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall through XML based RPC mechanisms. Moreover, Web services also ensure scalability by simple application integration using a lightweight infrastructure.

2 Implementation of web services

TravelGood is a travel agency that provides different Web Services for planning and booking trips. It provides the user with flight and hotel booking. TravelGood works together with two airline reservation agencies i.e. “CheapBird” and “Lameduck” and one hotel reservation agency called “NiceView”. TravelGood and 3 reservation agencies mentioned above are using “FastMoney” as banking service for performing money transfer through credit card and other financial operations. Figure 1 below provides a high level view of the project implementation –

2.1 LameDuck Web Service

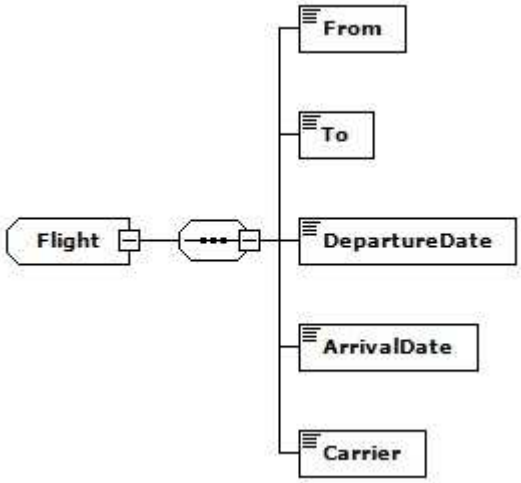
LameDuck is one of the airline reservation services in the Travel Good agency. The operations performed by the reservation service, the data structures used and the implementation details are mentioned in the sections below.

2.1.1 Data Structures

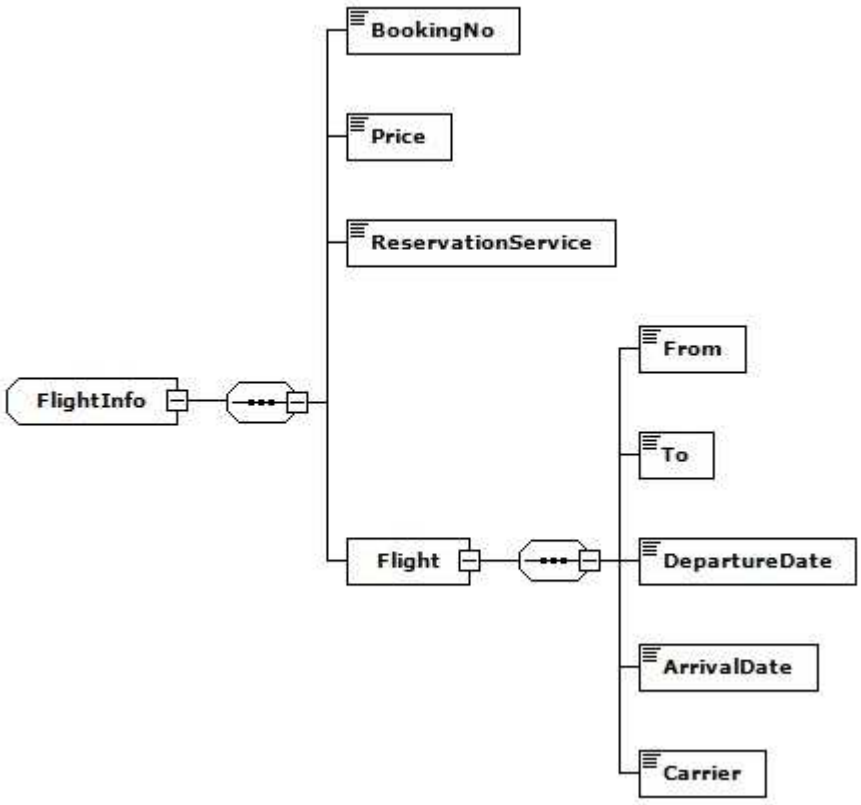
The implementation of the LameDuck is based on the following data structure:

- **Flight-**
  It is a complex data type that holds the information of a single flight. It has the following parts:
  i. **From** - It has a type string and contains the departure place of the flight.
  ii. **To** - It is also of type string and holds the destination of the flight.

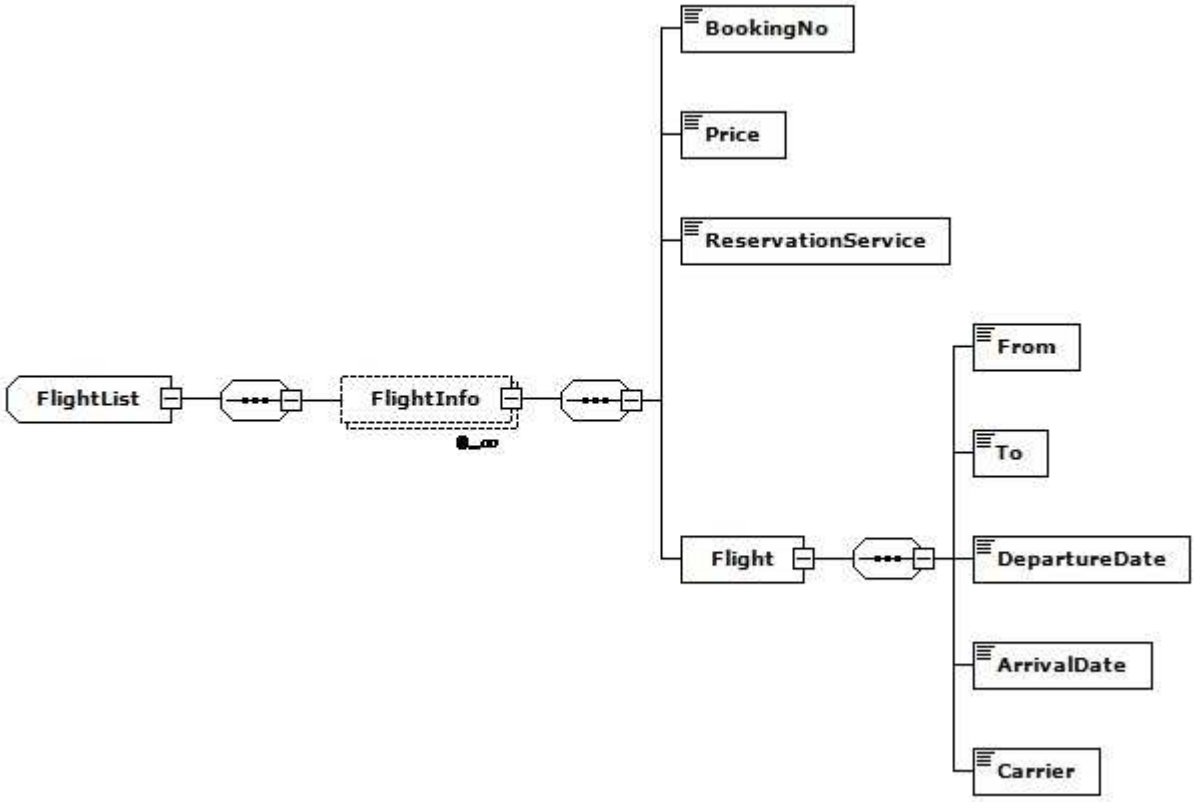
iii. **DepartureDate-** It has the type dateTime and keeps the departure date and time.
iv. **ArrivalDate** - It is also similar to departuedate but it holds the arrival information
v. **Carrier-** It has the type string and keeps the information about the flight carrier.



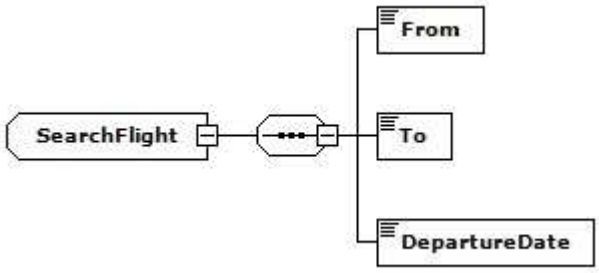
- FlightInfo
  I. **BookingNo** – It is of type string and it holds the uniqe booking number associated with the flight.
  II. **Price –** It is of type float and keeps the price of a flight.
  III. **ReservationService-** It is of type string and it determines the reservation service name i.e. LameDuck and CheapDuck.
  IV. **Flight** – It is of type Flight and keeps information about arrival and departure.



- **FlightList** - It contains the list of the flightInfo type.



- SeachFlight- It is the type of the parameter that are used to search for a particular flight.
  I. **From-** It is of type string and holds the departure place information.
  II. **To –** It is of type string and keeps information of arrival place information.
  III. **DepartureDate –** It is of type dateTime and maintains the information of the date and time of departure of the file.



2.1.2 Services offered

LameDuck is one of the airline reservation services that offers operations mentioned below -

• **getFlights** -This operation will take argument information such as departure place, final destination of the flight and date of the journey.It is a one way operation that calls the one way operation flightList of the TravelAgency service having the list of flight informations. Overall the every flight information contains - booking number, price of the flight, name of airline reservation

service, start airport, destination airport, data and time of departure and arrival at destination and carrier of the flight.

• **bookFlight** - This operation will take booking number and credit card information from the user and permanently books the flight after successful credit card charging(It will call "chargeCreditCard" service of the "FastMoney" to verify as if the user has sufficient funds available in his or her credit card account). It has boolean as the return type where "True" is returned for successful booking and "False" if any fault happens such as credit card information is invalid or due to any other reasons of booking failure.

• **cancelFlight** - This operation takes a booking number of a booked flight, price of the flight, credit card details and cancels the airline booking. It will refund 50% of the total price of flight (It will call "refundCreditCard" of the "FastMoney"). It has boolean as the return type where "True" is returned for successful cancelling and "False" if any fault happens due to any other reasons of booking failure. As instructed in our exam project, cancelFlight do not check if the given information such as booking number, price of the ticket, date of cancellation, etc is correct. In our implementation, it will only take "Booking number" and "Credit card information" as input the proceed with the refund.

2.1.3 Implementation

Message:

We have used the following message types for LameDuck Airline Reservation Service.

- **getFlightsRequest**
  This message is used as the request for searching flights. This is the input parameter for the one way function **getFlights.** The parts of the message are:
  I. **SearchFlight of type SearchFlight**
  SearchFlight includes the information for searching a flight.
  II. **SessionID of type string**
  Session ID is the unique id that is assigned to each client.

- **bookFlightRequest**
  This message is the information passed as input parameter to the **bookFlight** operation.
  i. **BookingNo of type string**

  It has the unique booking number associated to each flight.

  ii. **CreditCardInfo of type creditCardInfoType**

  It is the type for holding the creditcard information of the client.

- **bookFlightResponse**
  This message is used as the output of the operation **bookFlight.**

  Status of type boolean

- **cancelFlightRequest**
  This message is similar to the bookFlightRequest and have the same type.
  I. **BookingNo of type string**
  II. **CreditCardInfo of type creditCardInfoType**
- **cancelFlightResponse**
  This is the message that is used to return the status of cancelling flight.

  Status of type boolean

PortType

We have used one port type for all the operations provided by the LameDuck web service and the name of the port type is **LameDuckPortType** This port type has the following operations with the mentioned input output message types>

- **getFlights**
  It is the operation to perform the search flight.

  input: getFlightsRequest

- **bookFlight**
  It is the operation to perform the service of booking flight.

  input: bookFlightRequest

  output: bookFlightResponse

- **cancelFlight**
  It is the operation for cancelling the flight.

  input: cancelFlightRequest

  output: cancelFlightResponse

Binding Type

We have used the RPC-literal type binding in this web service.

Database Handling:

We have handled the data manipulations using static data in the web service.

2.2 CheapBird Service

CheapBird is the another airline reservation service in the Travel Good agency. The operations offered by this service is exactly similar to that of LameDuck service. We have used the same data structure and WSDL messages and operations for CheapBird service that we have used for LameDuck service. The implementation of the CheapBird and LameDuck is exactly the same. We have just used different static data for each of the services.

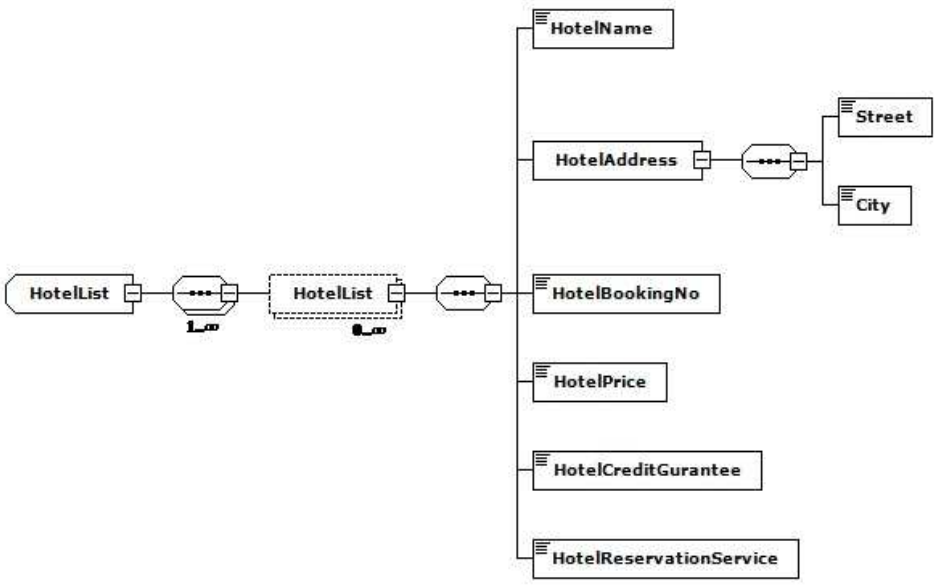
2.3 NiceView Web Service

This is our HotelReservation service that offers the different operations related to hotel reservation such as **getHotels**, **bookHotel** and **cancelHotel**. NiceView is the name given to this hotel reservation service which will provide us the list of available hotels matching the customer (TravelGood clients) preferences. The customers have to input the arrival and departure dates of a particular city where they want to book the hotel. Apart from these basic features, Niceview also provides functionalities such as - cancel the reservation after booking the hotel, book the list of selected hotels at a given time, etc.

2.3.1 Data Structures

The functionalities provided by the NiceView make use of several data structures which are explained as follows –

- **HotelList** - It contains the entire list of the hotels which matches to the user input criteria such as arrival and departure date, city of arrival. The value of the HotelList is returned to TravelGood service by calling it. The class diagram can be shown as below. The structure of the HotelList contains complex type having multiple instances of the hotel.



- **Hotel**- It refers to the individual hotels that match the user input criteria. The diagram shown in the Figure represents data structure called Hotel. The Hotel structure contains 7 different fields such as-

  i. **HotelName** - It has a data type as string containing name of the Hotel.

  ii. **HotelAddress** - It has completes data type which contains two parts

    a. Street – It has the type string and contains the hotel street name.

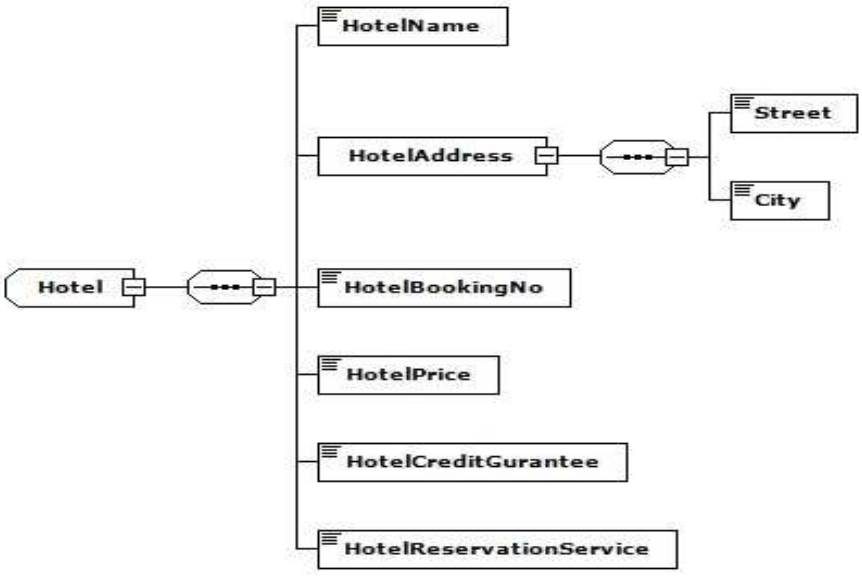
    b. City – It is a string containing the city the hotel is situated.

  iii. **HotelPrice** - It has a data type float containing price for the complete stay or duration given by the user.

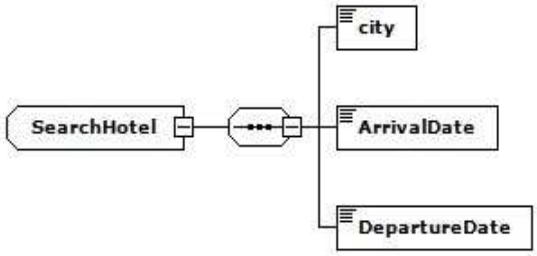
  iv. **HotelBookingNo**- It has a data type string where it contains the booking number for the reservation in case hotel is booked.

  v. **HotelCreditGuarantee** - It has data type Boolean where it determines whether the credit card guarantee is required or not for booking the hotel.

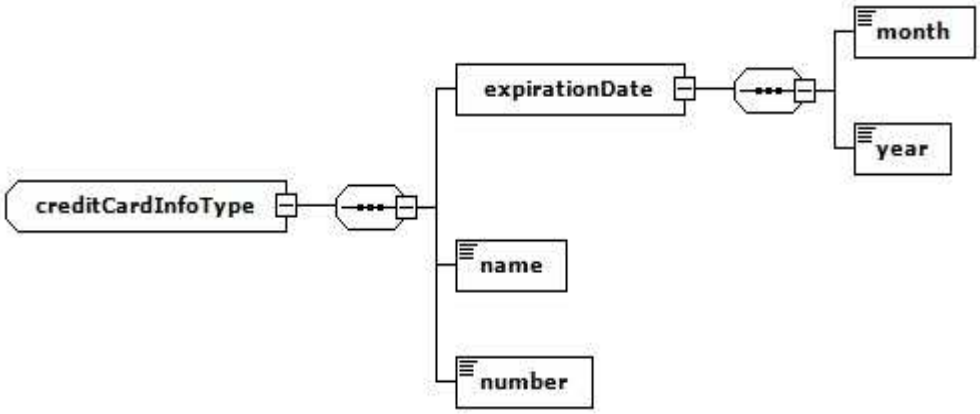
  vi. **HotelReservationService** - It has a data type string and contains the name of the agency through which the hotel was reserved like we have NiceView in our implementation.



- **SearchHotel** – It a structure containing information such as ArrivalDate, DepartureDate and **City**. **ArrivalDate** and **DepartureDate**- It has a data type “dateTime” that represents the starting date from when the hotel needs to be booked.



- **crediCardInfoType-** This data structure is used from the bankservice.xsd.



2.3.2 Services offered

The services or functionalities offered by the NiceView can be explained as follows -

- **getHotels** - This operation will ask the user to enter the city where they plan to book hotel with the arrival and departure dates. It actually takes a single parameter of complex type Searchinfo that contains all the above information. This is one way operation hence no value is returned where the list of available hotels is generated and returned by using the callback function hotelList of Plantrip service of the TravelAgency. The hotelList is an operation of the TravelAgency service which contain vital information such as names of the hotel, list of hotels, addresss of hotel, price or traffic, booking number.

- **bookHotel** - This operation will take the credit card information and booking number of the reservation made by the user and books the hotel. This service also calls the validateCreditCard operation of the bank service if gurantee is required. It returns a boolean expression having value “True” for successful booking and “False” showing failure. If forexample, the credit card validation fails then bookHotel operation will not succeed.

- **cancelHotel**- This operation will take the booking number of the hotel reservation and cancels the hotel reservation. It will also have boolean as a return type meaning “True” for successful operation and “False” if due to any of the reasons cancel process for hotel reservation fails then it will throw some exception

2.3.3 Implementation

The NiceView Hotel Reservation Service is implemented as a Java Web Service in Netbeans in top down approarch. The NiceView.wsdl is the WSDL file defined for this web service. The different components of the WSDL file are presented here:

Messages

We have defined the following message types for NiceView Hotel Reservation Service:

- **getHotelsRequest -** This message is used as the request for searching hotels. This message is used as an input parameter for the one way function **getHotels.** This message has two parts. They are:
    i. **searchInfo of type SearchHotel-** SeachInfo includes the information for searching a flight or hotel and it consists of the city of choice, arrival date and departure date.
    ii. **sessionID of type string -** SessionID stores the session id that is created for each client.
- **bookHotelRequest-** This message is the information that is needed as input parameter by the **bookHotel** operation of the service. It has the following parts
    i. **bookingNum of type string-** It contains the unique booking number associated to each hotel and the client uses this number to book or cancel a hotel.
    ii. **creditCardInfo of type of creditCardInfoType-** It has the credit card information of the client to provide the guarantee to the hotel.
- **bookHotelResponse**- This message part is used as the output of the **bookHotel** operation and it has only one part.
    i. **bookingStatus of type of Boolean-** bookingStatus holds true if the booking is successful and false otherwise.
- **cancelHotelRequest**- This is the message used to cancel a hotel and it is similar to the structure of **bookHotelRequest**.
    i. **bookingNum of type string**
- **cancelHotelResponse**- This is similar to the bookingStatus and contains the status of cancelling a flight.
    i. **cancelStatus of type Boolean**

PortType: We have used a single port type to offer all the operations supported by NiceView service. The name of the NiceViewPortType is it has the following operations:

- **getHotels-** It is the implementation of the getHotels operation of the hotel reservation service and it is a one way function. It takes the following message as input parameter:

Input: getHotelsRequest

- **bookHotel-** It implements the bookHotel functionality of the service. The input and output message type of this operation is as below:

  input: bookHotelRequest

  output: bookHotelResponse

- **cancelHotel-** It is functionality of cancelling the hotel based on booking number.

  input: cancelHotelRequest

  output: cancelHotelResponse

Binding Type- We have used the RPC-literal type binding in this web service. The benefits of using this binding type are it is WS-I compliant and the WSDL is straight forward.

2.3.4 Database Handling

To handle the data manipulations of the service we are using static data in the web service.

2.2 LameDuck Web Service

2.2.1 Data Structures

2.2.2 BPEL process and Class Diagram

2.2.3 Description

2.3 CheapBird Web Service

2.3.1 CheapBird Reservation Service

2.3.2 Data Structures

2.3.3 Services offered

LameDuck is one of the airline reservation service that offers services mentioned below -

- getFlights -This operation will take argument information such as departure place, final destination of the flight and date of the journey. It is a one way operation that calls the one way operation flightList of the TravelAgency service having the list of flight information. Overall the every flight information contains - booking number, price of the flight, name of airline reservation service, start airport, destination airport, data and time of departure and arrival at destination and carrier of the flight.

- bookFlight - This operation will take booking number and credit card information from the user and permanently books the flight after successful credit card charging(It will call "chargeCreditCard" service of the "FastMoney" to verify as if the user has sufficient funds available in his or her credit card account). It has Boolean as the return type where "True" is returned for successful booking and "False" if any fault happens such as credit card information is invalid or due to any other reasons of booking failure.

- cancelFlight - This operation takes a booking number of a booked flight, price of the flight, credit card details and cancels the airline booking. It will refund 50% of the total price of flight (It will call "refundCreditCard" of the "FastMoney"). It has Boolean as the return type where "True" is returned for successful cancelling and "False" if any fault happens due to any other reasons of booking failure. As instructed in our exam project, cancelFlight do not check if the given information such as booking number, price of the ticket, date of cancellation, etc is correct. In our implementation, it will only take "Booking number" and "Credit card information" as input and proceed with the refund.

2.3.4 Implementation

2.3.5 Test Data

2.4 TravelGood Web Service

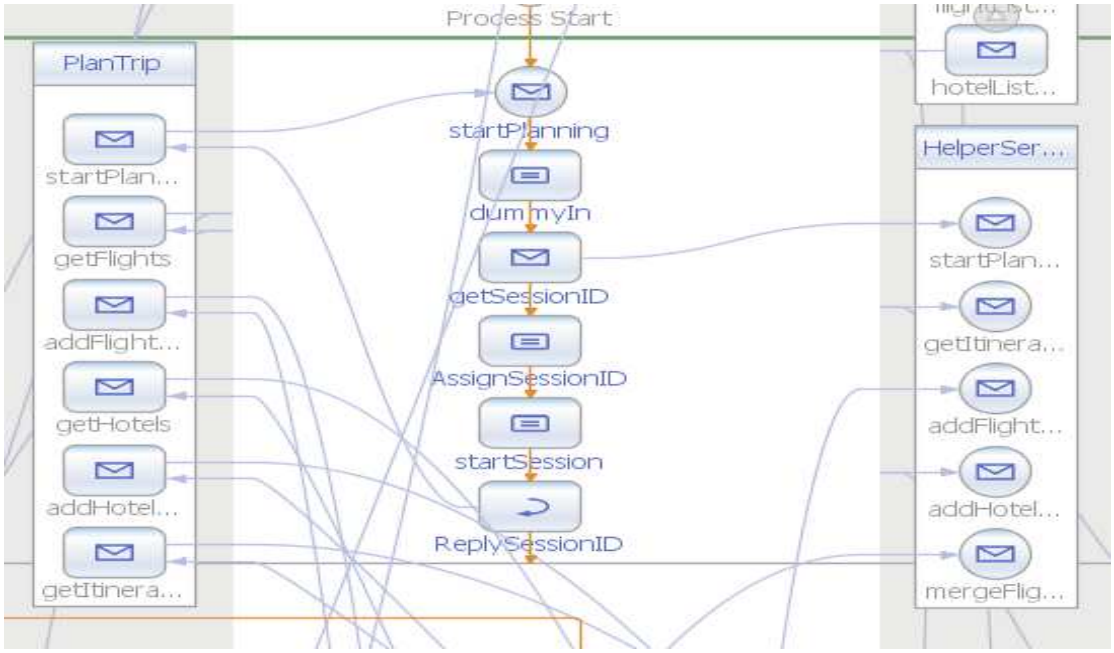
The process of booking consists of - first creating itinerary using the PlanTrip service which is returned to the user and booking and reserving purposes. If after booking the itinerary, it can cancel the booking using the CancelTrip service. TravelGood Web service provides three different services that will help in operations related to flight and hotel reservations. The services are listed below as -

2.4.1 PlanTrip

Plan trip is used to create and plan an itinerary. The user creates an empty itinerary with **startPlanning** operation. A session id is generated and assigned to the user. Then user can use **getFlights** operation to get a list of flights and getHotels operation to get a list of hotels. The user can use **addFlightToItinerary** and **addHotelToItinerary** to add a flight and hotel to the itinerary.

The **getItinerary** operation takes session id of the user as parameter and returns the itinerary bound to that particular session id. These operations are explained as below-

- **startPlanning**- It takes a dummy string as input to create an empty itinerary and returns a session id (type string) to the user. This session id is later used to create property so that all other operations can correlate with this session id. As seen in the following diagram, the PlanTrip WSDL calls **startPlanning** and assigns a dummy input and invokes the **startPlanning** which return the session id. If no user interaction is received for the 15 minutes then a timer is used for ending the current session i.e. the user have to start from **startPlanning** again.



Before we go any further with the operations of Plan Trip, we would like to introduce another service called ***Helper Service***. It's written in plain java. Helper service has the functions for **startPlanning, addHotelToItinerary, addFlightToItinerary, mergeFlightList and getItinerary**.

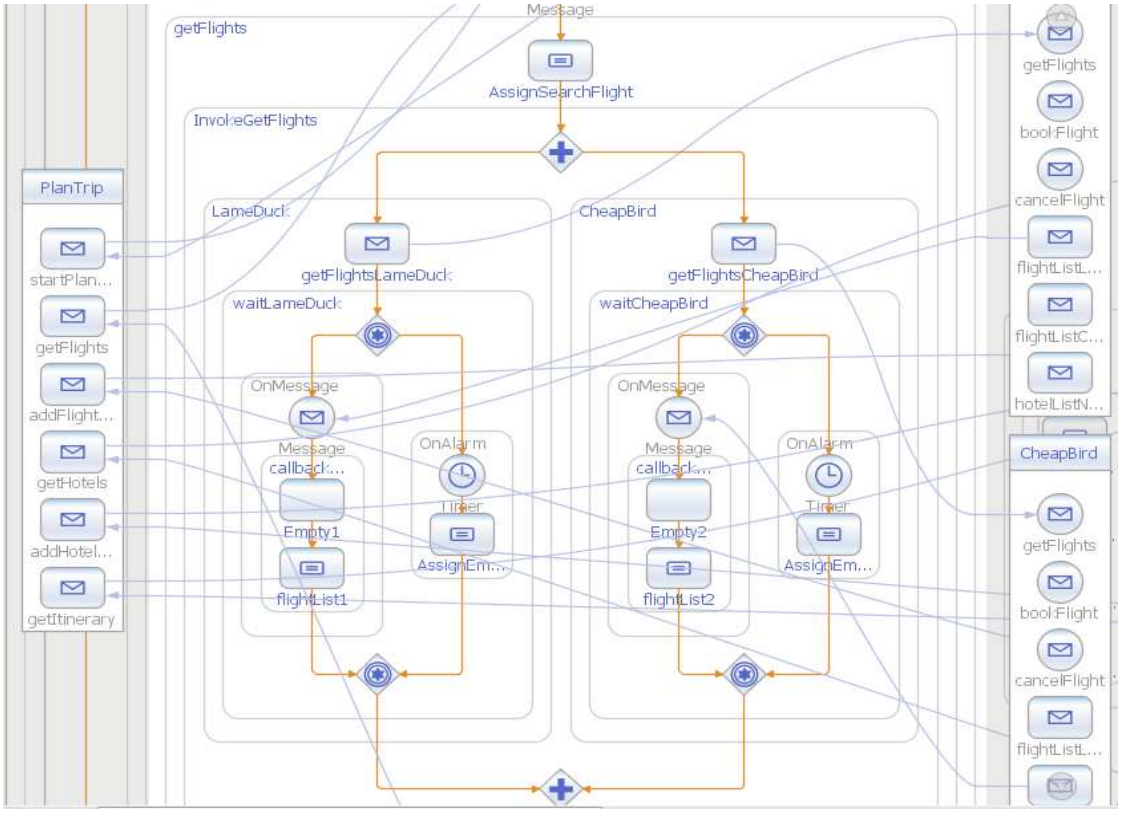
Continuing with Plan Trip, following are the rest operations:

- **getFlights** : this operation takes the place where the flight should start, destination and date of the flight as arguments. It will return list of flights available where every flight information contains - booking number, price of the flight, name of airline reservation service, start airport, destination airport, data and time of departure and arrival at destination and carrier of the flight. This operation is implemented by first calling the **getFlights** operation of airline reservation services namely LameDuck and CheapDuck which in turn calls back the **flightList** operation of Travel Good to return the list of available list of flights.

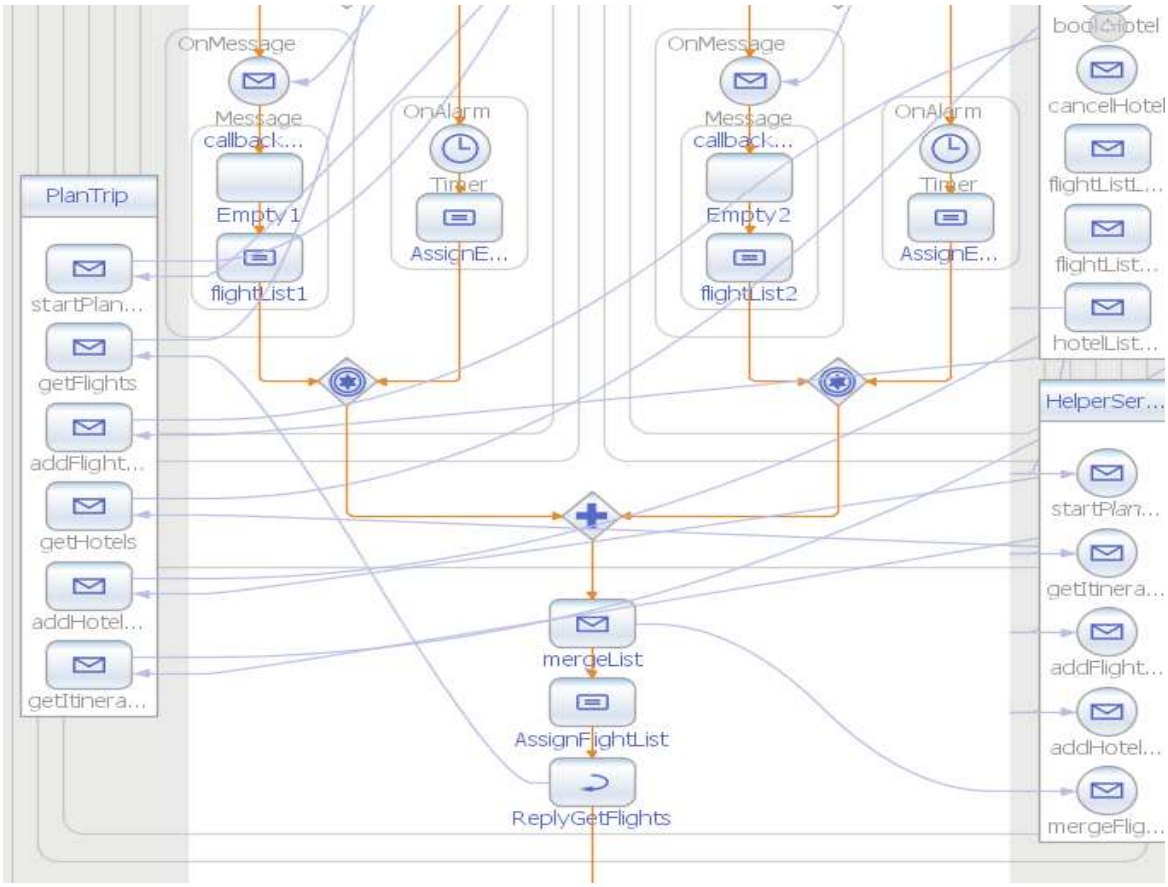
- Since there are two airlines reservation services the getFlight operation calls for both the airlines. The travel agency then waits for 5 seconds for the answer which is achieved by using timer and pick activity. Meanwhile, both the airline services try to get data by calling flightList operation of the Travel Good Agency. The results from the both airline reservation services are then merged together by calling **mergeFlightList** operation of **HelperService**.

- **flightList and hotelList** operations are implemented on a different port called CallBackPlanTripPortTYpe whereas rest of the Plan Trip operations are implemented in another port type.

- The following figure shows the pick activity and alarm on the two airline service and the calling of getFlights and response from flightList.



The below diagram shows the results obtained by merging result from the two airline reservation services -



- **getHotels**- It takes arrival date, departure date and city as arguments. The operation will return list of hotel with all necessary information (such as name, address, booking number, price of the hotel, start and end of stay, etc). This service is implemented by first calling getHotels operation of the NiceView Hotel reservation service which in turns call backs the **flightList** operation of Travel Good and finally returns the list of available hotels.

  The following diagram shows the BPEL process of this operation. This also waits for 5 seconds for the answer from the NiceView and if no answer is received then it assigns none to the parameters in the search result. The figure also shows it calling the getHotels operation of NiceView and the response receiving from **flightList** operation.

2.4.2 Book Trip

This service is used to book an itinerary the user has created in a session. It takes session id, **creditCardInfoType** and itinerary as arguments. It then calls the **bookFlights** operation of the two airline reservation service and bookHotel of NiceView. If booking is successful then status of each flight and hotel will be marked as booked. But if during the booking process if any of the flight or hotel couldn't get booked then the status of so far booked things will be cancelled. To cancel the booking **CancelTrip** service is called. The following diagram shows the process calling **bookFlight**, bookHotel and **cancelItinerary.**

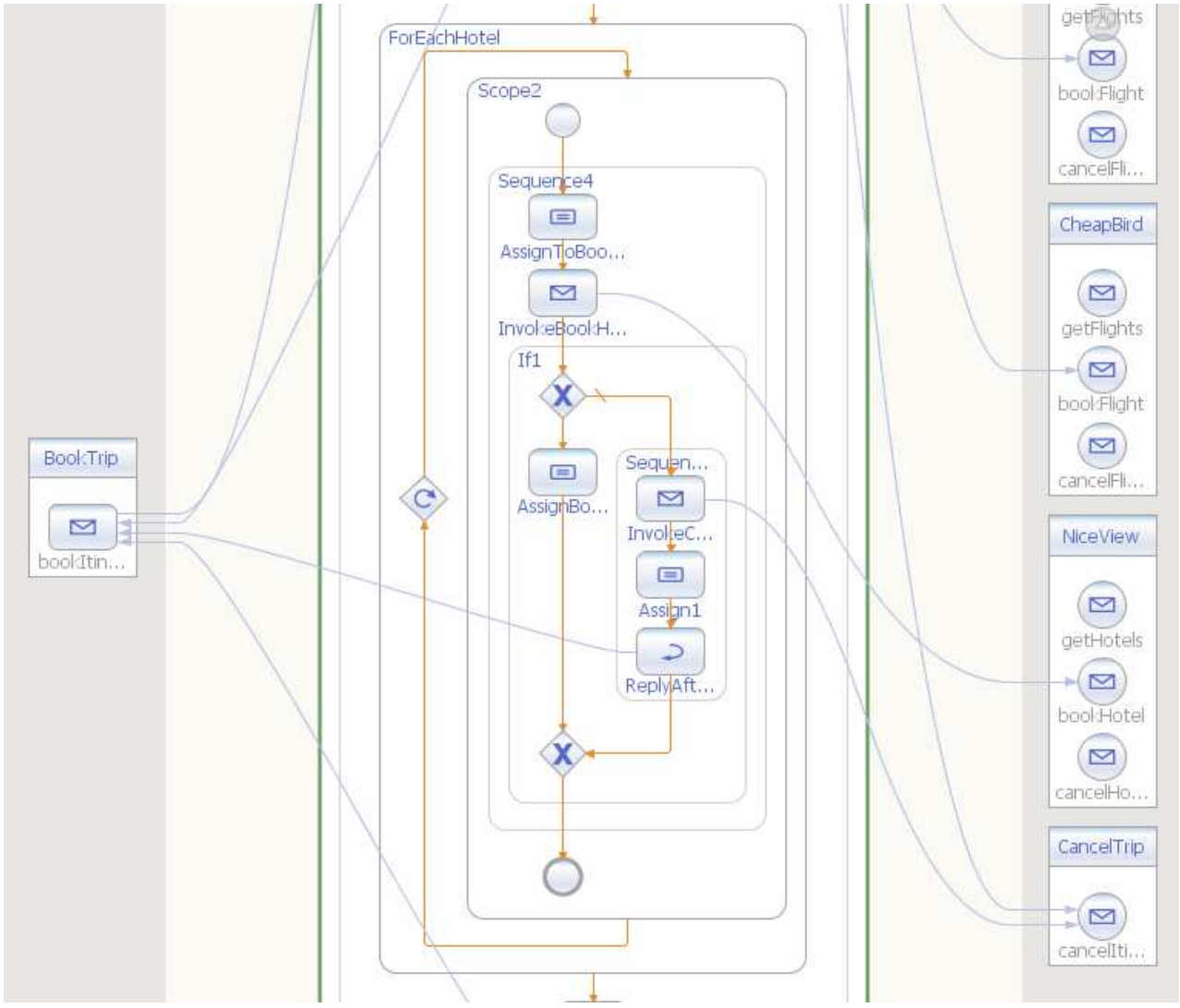
Steps in BPEL for book trip:

1. Process starts with three inputs in receive of type creditCardInfo, session id and Itinerary.

2. A for each loop is placed to loop through hotel list and for each instance of hotel we call the book flight operation of the two airline reservation service. We assigned the input to inputItineraryIn. We also assigned the received input to a variable named CancelItineraryIn later for cancelling purpose.

3. If booking goes successful then we go through the same process i.e. in a for each loop for the hotel list. Whenever booking is successful we change the field Confirmation to confirmed and we also update this in CancelItineraryIn.

4. In both loop i.e. in flightlist and hotellist, if one instance of booking is failed then CancelTrip is called and CancelItineraryIn is passed as argument.

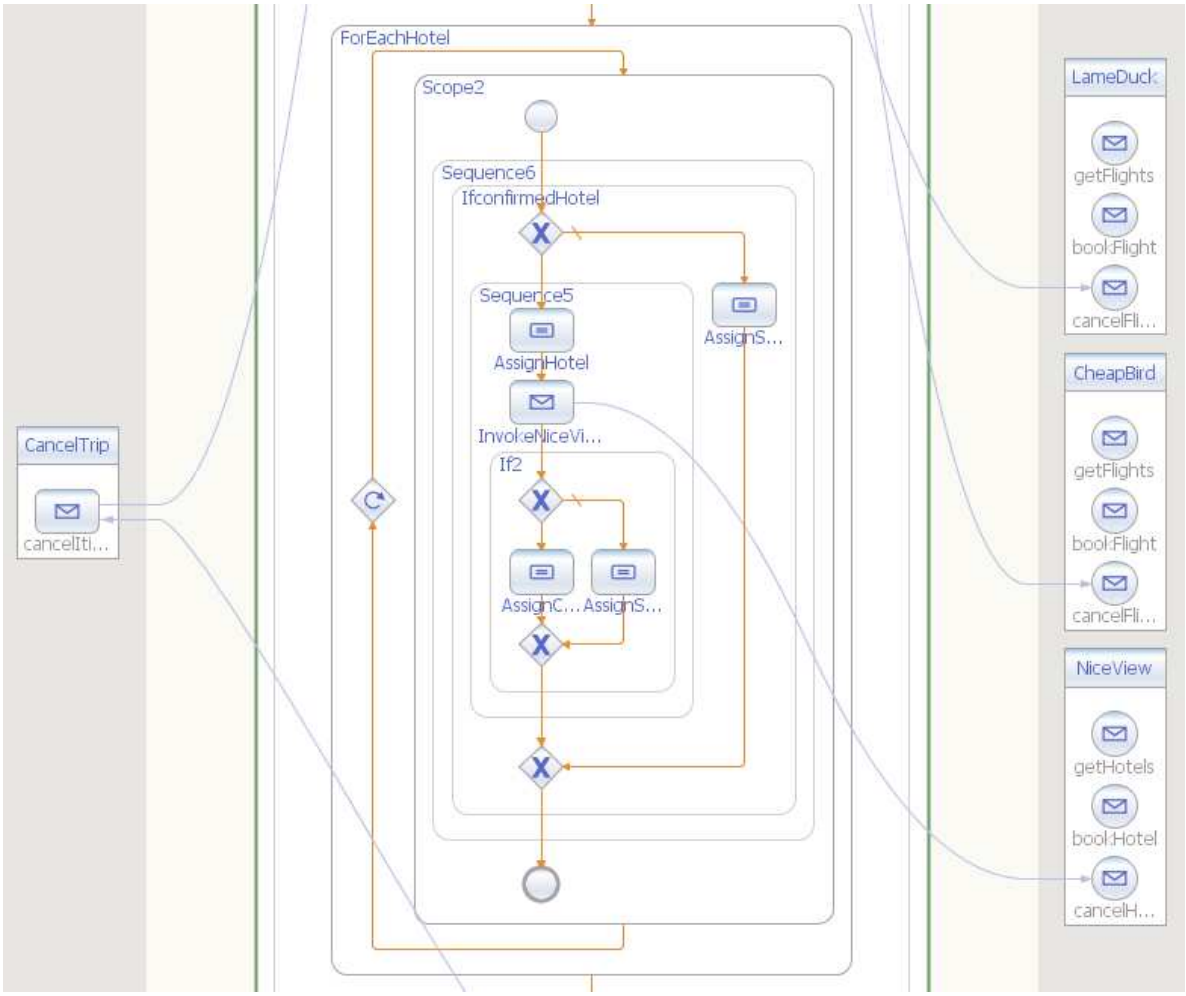


2.4.3 Cancel Trip

Cancel Trip takes an itinerary as an argument and cancels all the flights and hotels in that itinerary. It calls the **cancelFlight** and **cancelHotel operations** of the two airline services and the NiceView respectively. If something went wrong while cancelling then the booking status for the itinerary where error occurred is kept as confirmed while others status is set to cancelled.

Following steps occur in this BPEL process:

1. Similar to book trip we place for each loop for both flight and hotel.
2. At first we check if the field Confirmation is confirmed then we invoke cancelFlight and cancelHotel and assign the Confirmation field to cancelled and if the Confirmation field is not confirmed then we leave the field as it is.
3. Then if the canelFlight/cancelHotel succeeds then we proceed and if something went wrong then we leave the confirmation field to confirmed.



2.x Helper Service

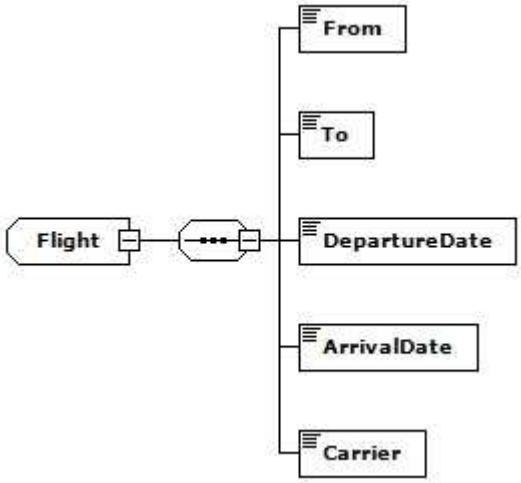
Helper Service is written in plain java to simplify some operations. It includes following operations which have been described before:

- addFlightToItinerary
- addHotelToItinerary
- getItinerary
- mergeFlightList

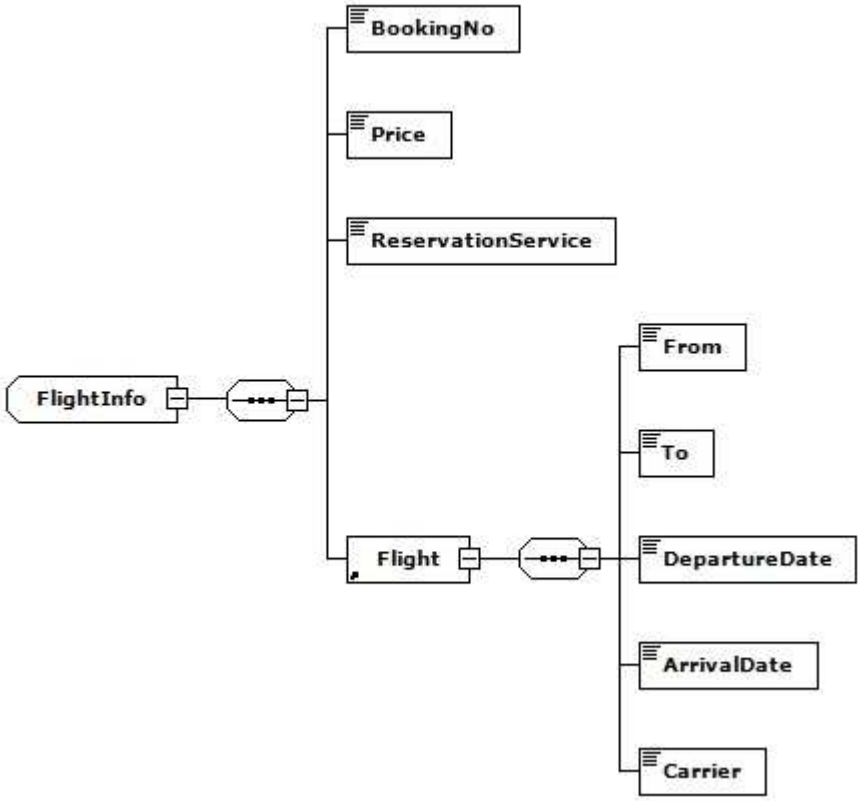
- startPlanning

Here we present the data structure of the Itinerary schema on which the above mentioned operations are working. We define following complex types in Itinerary.xsd:

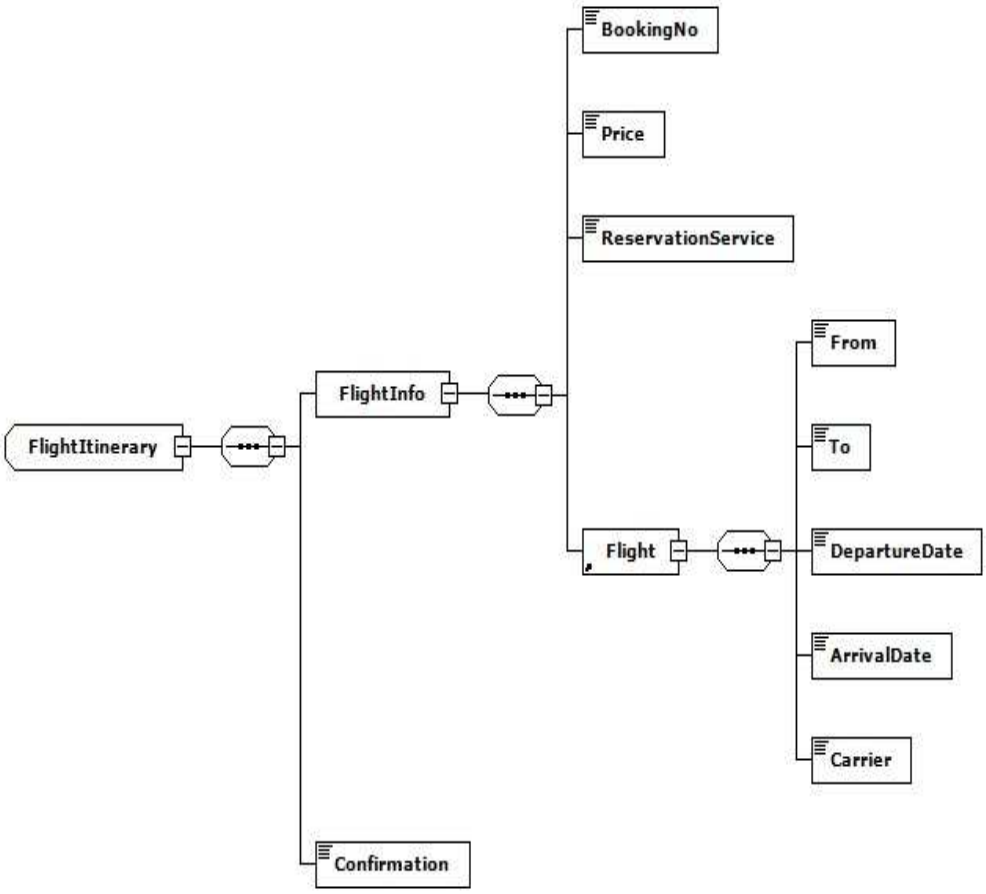
- **Flight** : It contains the elements From (string), To (string), DepartureDate (datetime), ArrivalDate (datetime) and Carrier (string).



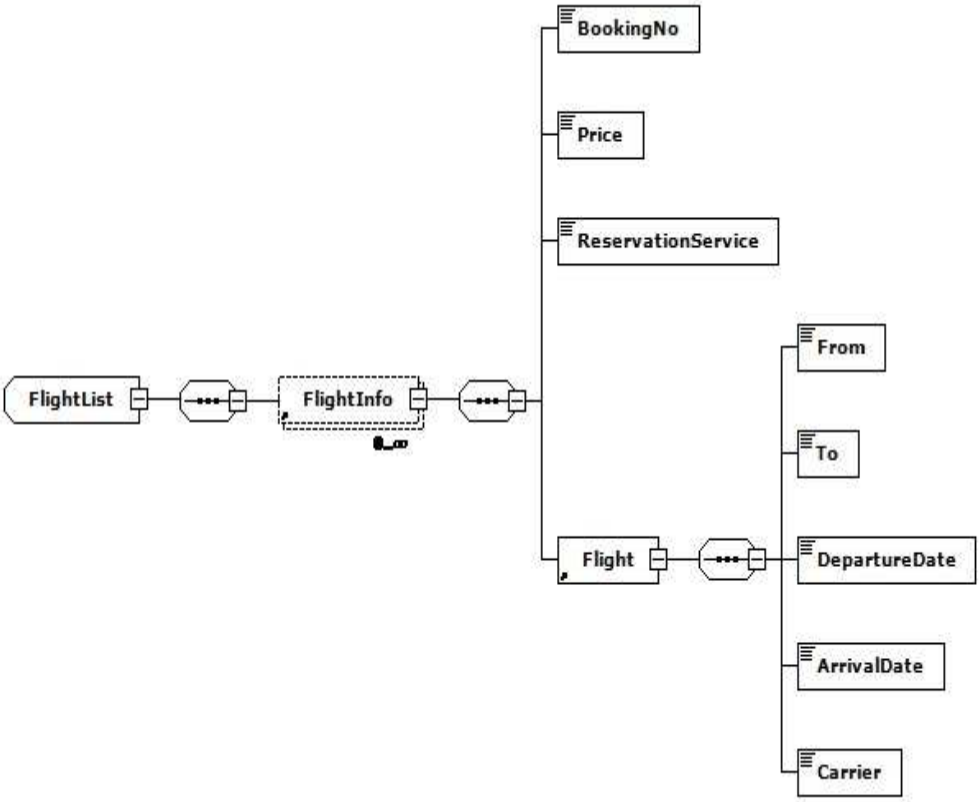
- **FlightInfo** : this contains the Flight type mentioned above along with a flight Booking No (string), price of the flight (string) and Reservation Service (string).



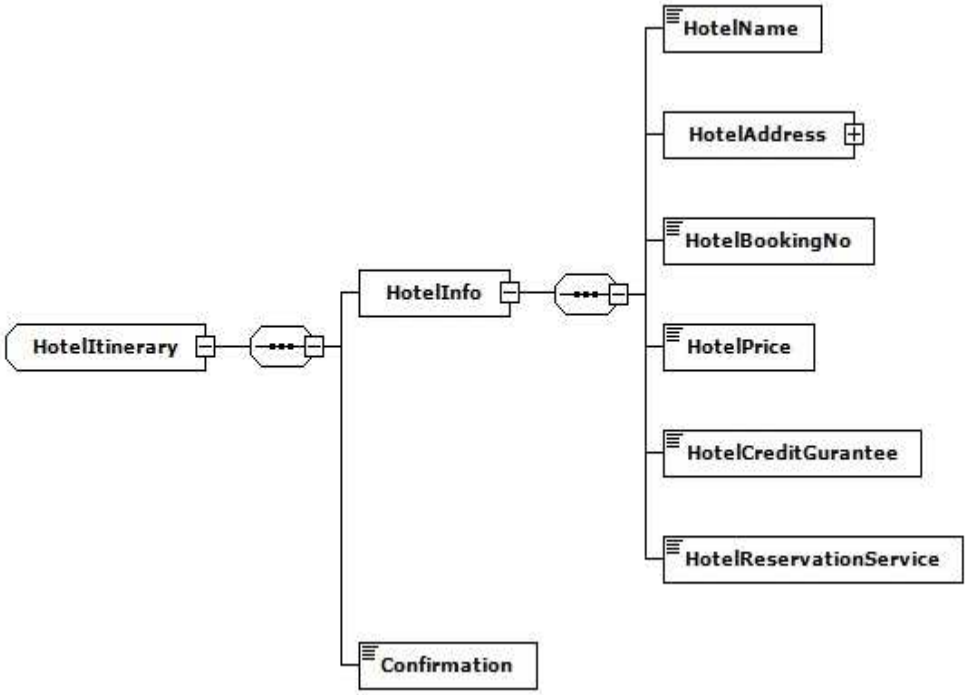
- **FlightItinerary** : this contains the FlightInfo with another simple element Confirmation (string). Confirmation can be booked, cancelled or unconfirmed.



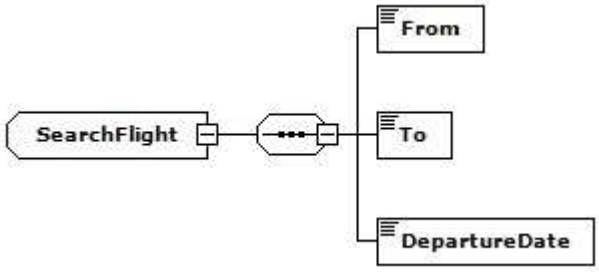
- **FlightList** : this is an unbounded sequence that contains a complex element of type FlightInfo



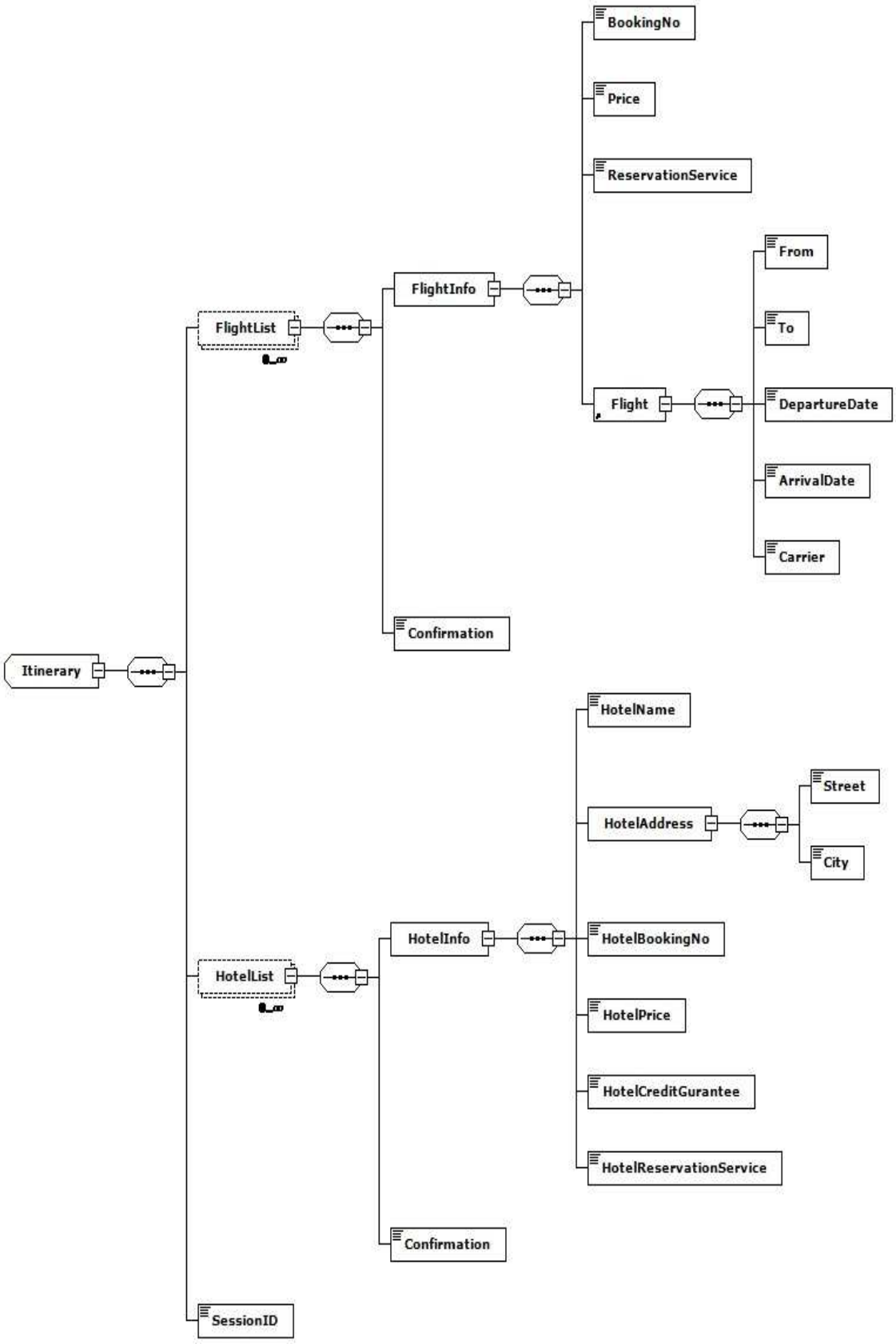
- **HotelItinerary** : this contains the complex type HotelInfo and a simple element Confirmation. Confirmation can be unconfirmed, booked or cancelled.



- **SearchFlight** : this contains three simple types From (string), To (string) and DepartureDate (datetime). It is the search parameter for getFlight operation



- **Itinerary:** this has three elements hotellist, flightlist and session id



3 Web Service Discovery

4 Advanced Web Service Technology

Author -Amandeep Dhir

There are hundreds of standards flooded over internet pointing some of the short-comings of the simple WS model but we will discuss only four of them in this section. These are referred as advanced Web services Technology. As we know that Web Services are based on the exchanged messages and there are few problem associated with this message exchange. We can list down these associated issues such that - WS-Addressing, WS-Policy, WS-Reliable Messaging and WS-Security. In our implementation the “FastMoney” web service has provided us an opportunity to make use of the advanced web service technology

4.1 WS-Addressing

Author -Khurram Bashir

SOAP messages do not contain information about "from whom" and "to whom" a message is sent. This information is the part of the transport protocol. This problem can be solved if we can encode this missing information into the SOAP headers. WS-Addressing is a specification that deals with solving the problem faced by the naive SOAP messages so it will provide transport functionality inside SOAP instead of the network protocols. WS-Address schema is used to provide information about the end points by encapsulating this information inside the SOAP message headers.

The specification provides two interoperable constructs that will work similar to messaging and transport protocols. This will normalize the underlying information into a uniform format that helps in processing independent of the transport protocol. In real time projects, WS-Addressing simples the communications in the web services so its saves time, effort and cost for supporting different platforms. WS-addressing specified constructs i.e. message information headers and endpoint references that can be explained as follows - first includes tags like “To” and “From” that helps in communicating information such as source endpoint, message ID, message destination URI, reply end point etc

4.1.1 Significance to Our Implementation

In our implementation, we made the user of WS-Addressing to reduce and avoid session management overheads and issues like message identities. WS-Address constructs discussed above (message information headers and end point references) contain information that can be used to identify the sessions between the service-service and client-services.

4.2 WS-Reliable Messaging

Message reliability ensures that messages are successfully delivered at any desired destination which is considered important for any kind of system. In case of Web services, SOAP messages are exchanged without any focus on the reliability in the delivery. WS-Reliable Messaging provides the functionality for ensuring that the SOAP messages are successfully delivered to the end point. We can also customize the type of message communication through optional types provided by this specification. The naive form of Web services used TCP sequence numbering having

acknowledgement to provide reliable message delivery. WS-Reliable Messaging consists of set of SOAP messages - CreateSequence and TerminateSequence. It adds the header information to the SOAP messages to identify messages in a sequence, to acknowledge the receipt of a sequence (SequenceAcknowledgement) and to request an acknowledgement (AckRequested). First step is to create sequence of messages that need to be received or sent successfully. Second Step - Send the messages so that data messages are exchanged between two entities. Third step, terminate sequence is sent to close the session, once the data is transmitted. Overall, it ensures that SOAP messages are not lost during the transmission. It also ensures that messages are retransmitted until get acknowledged. Moreover, the message must arrive in a sequence. Messing also includes optional types such as AtMostOnce, AtLeastOnce, ExactlyOne and Inorder depending upon when the messages must be resent if lost.

4.2.1 Significance to Our Implementation

The functionalities provided by the WS- Reliable messaging can make different services implemented by efficient and robust. Reliable message is very important for any implementation and companies pay due attention to this kind of activities for example if a user input the information from the client and those request message do not proceed further then user will stop using the service hence it can affect the company reputation and revenue. In our implementation, sensitive information such as passwords, credit card transactions, itinerary information exchanges can be lost so WS-Reliability Messaging will check that a transaction or account authentication operation has successfully completed.

4.3 WS-Security

Author -Amandeep Dhir

Current form of the web services implementation as we carried out for the exam project is vulnerable to any type of attack. Typically Web services are considered insecure in their native form and third parties can easily read, intercept and change the SOAP messages exchanged if proper security mechanisms are not adopted. Generally it is assumed the intra-level communication or intranet doesn’t pose any major threat but outside communication i.e. both Business to Business (B2B) and internet are vulnerable to potential threats. If an attack is successful then it may lead to Un-authorised access to information which is treated sensitive or restricted for example an attacker can book a flight without paying any money to the ticket booking website.

The counter argument to this vulnerability can be given that web service model is secure as it is implemented internally for a company so it is free from any possible external attack but we should also keep in mind the interactions carried with the clients. In the event of interaction between the clients and web service model then traffic generated is exposed in the middle network which is out of company's control.

So overall, we can establish a secure communication in the un-trusted environment through several ways such as Hypertext Transfer Protocol Secure (HTTPS). However, HTTPS also posses few security concerns such as it can provide confidentiality but does not guarantee integrity and

authenticity. Digital signatures are also one way-out but they only guarantee authentic and integrity but not privacy. Combination of Public Key encryption and Digital signature can solve the security challenges to an extent. To solve these problems, advanced technologies such as WS-Security are used for secure communication. WS-Security is a means to apply security to the Web services. It specifies procedures to add integrity, authentication and confidentiality features to the SOAP protocol that is also referred as message level security. WS-Security adds wsse:Security header block which enhances the security of the messages. It includes timestamps, security tokens, signatures, reference to security tokens and encrypted information. Oasis-Open released WS-Security 1.0 April 19, 2004. Oasis-Open released version 1.1 on February 17, 2006.

4.3.1 Significance to Our Implementation

Security of the system is very important especially when the system is using information which is regarded as "very sensitive" for example in our case when a user starts booking of flights or hotels then at the final stage he or she has to enter his or her credit card information which is a critical information that must not be revealed to an outsider at any cost. Particular to our project the services "Lameduck", "CheapBird" and "Niceview" should be encrypted so that user's vital information is secured from outside. All kinds of operations in these services must provide integrity, authentication and confidentiality.

Overall WS-Security is treated as a base standard for ensuring message level security through security tokens (to know sender of the message), signed SOAP messages(ensure integrity and message authentication) and encrypted SOAP messages for ensuring privacy.

4.4 WS-Policy

WS-Policy refers to a framework that defines rules that any Web services have to abide. The policies supported by the Web-services can be listed as quality of service (QoS) policies, security policies, etc. Practical examples can be policy related to WS-Security and WS-Addressing, Policy as which part of the message is signed and encrypted. In our project implementation the services such as - “LameDuck”, “CheapBird” and “NiceView” communicate with “FastMoney” service “exactly one”. Our approach for “exactly one” ensures confidentiality because it will enable these services to implement encryption mechanism. Similarly, when client provides sensitive and confidential information to the “TravelGood” then policy should be in such as way that client can only able to provide confidential information only if application supports encryption and ensure secure transmission of the data to the target entity i.e. “TravelGood” in our implementation.

4.4.1 Significance to Our Implementation

In our implementation, we can attach different policies to the operations of the Web Service. This step will ensure that operations that involve sensitive information exchange are properly handled in our implementation.

5 Conclusion

Author -Amandeep Dhir

In this report, we focused to discuss our learning and project experience throughout the software development in web services course. The team consisted of four members and as per the project specification we have equally divided the sections in the exam project. The enthusiasm and motivation of team members was enough to complete the exam project on time. The fruitful discussion during group meetings helps each one of us to solve the individual assigned work and refine the written report as a whole. There is always a room for further improvements and extensions but we all firmly believe that this exam project has provided us basis to build our career in the area of web services in general. The implementation tools particularly Net Beans and Glassfish were quite unstable due to which it took much more time than anticipated. During the middle of the exam project some of us even started feeling that objective of course is bug correction but somehow we were able to rectify the problem faced due to group effort and strong desire. Overall we believe the developing functionality didn't take much of time compared to the issues faced because of Net Beans.

Our targets for this exam project was to maximum utilize our learning and develop a practical system as mentioned in the project description. We face many problems through the course specially related with the implementation due to unstable tools so we would request the organizer of this course to use stable tools instead of Net Beans. Moreover, exam project should be given more time with a possible section of extension where students are free to extend the given project to show their learning and implementation skills.

Overall the course is quite good, teaching is clear and support during the lab sessions is very effective. We are happy that after completing this exam project we take with us knowledge and experiences gained during 13 weeks of this course.

Reference

[1]. http://www.webopedia.com/TERM/W/Web_services.html

[2]. NetBeans IDE, http://netbeans.org/

[3] GlassFish Sever, http://glassfish.java.net/

[4] Lecture Notes by Hubert Baumeister, course on Software Development of Web Services.

[5]. Michael P. Papazoglou, Web services: Principles and Technology, Pearson Education Limited, 2008

Appendix A

Testing

In this project, most of the operations have been tested by creating the java client applications. This helped us to check the funcationlity of the operations and assure that there is not any problem while extracting data. Moreover, some of the bpel process files like plan trip, book trip and cancel trip have been tested by creating the test cases through wizard. In this way, we have checked that the proper responce has been generated in the resultant xml file.

Testing by java Client Application

This type of testing helped us to find out the basic errors while calling the operations. When we call these operations then we need to pass the proper xml data type to see if the web service is responding with the correct result.

Testing by Test Cases

Testing by creating the test cases within the TravelGoodAgency helps us catching those errors that cannot be figured out by Client Applications. For example, there was a problem in the soap address and it was not figured out by the client application but when we created the test case then it was giving error 404 which means that the glass fish server could not find the file. So, these kind of errors were resolved by creating the test cases.

Testing by Junit 4.5

We have also tried making client application and with the help of Junit testing we have created the test file for the plan trip where we can see the input and output of the operations. In the unit testing, we have figured out that there were many minor problems in the project. For example, the loop for the database was not working properly and we were not been able to get the flight data. So, we tried giving the static data and make it work. Further, we had the same problem with the NiceView and we could not get the hotel data. So, we provided the static data in the testing class and make it work.