# Problem Description

Course 02267 (Fall 2010)

Hubert Baumeister
hub@imm.dtu.dk

Version 1: 25.10.2010

## Contents



# 1 Introduction

TravelGood is a travel agency providing a set of Web services to plan and book trips, which are composed of flights and hotel bookings. These Web services are used by Web applications (e.g. offered by the travel agency itself) and by client applications, used by the consultants working in the offices of TravelGood.

TravelGood works together with two airline reservation agencies called LameDuck and CheapBird and with one hotel reservation agency called NiceView.

TravelGood and the other companies use the bank FastMoney for credit card payment and money transfer.

# 2 Tasks

The project has four main parts. The first part consists of a short introduction to Web services. The second part is the implementation of the services and operations of TravelGood, LameDuck, CheapBird, and NiceView as Web services (some of them have to be implemented in BPEL, others may be implemented in BPEL or another programming language). The third part is the creation of WSIL files usable for service discovery, and the fourth part is a discussion on the use of advanced Web service technology.

## 2.1 Introduction to Web services

Write a section in the introduction of the report introducing the basic concepts of Web services, such as service orientation, basic service technologies, description of Web services, Web service discovery, Web service composition and Web service coordination. This section should paint the big picture and highlight the important aspects of each of these topics. What do you think is important and why? The size of the section should be around 2 pages.

## 2.2 Implement the following Web services

### 2.2.1 Travel Agency Services

The process of booking consists of first creating an itinerary using the PlanTrip service. The itinerary contains the information of flights and hotels of the trip with their respective dates and times. After the itinerary is created and returned to the client of the travel agency, the client then uses the BookTrip service to actually book the flights and reserve the hotels. If, after having booked the itinerary, it should be necessary to cancel the bookings, the CancelTrip service can be used.

**Plan Trip** The PlanTrip service is used to create and plan an itinerary. The startPlanning operation creates an empty itinerary. The getFlights operation returns a list of flights for the client to choose from. The client uses the operation addFlightToItinerary to add the flight the itinerary. The operations getHotels and addHotelToItinerary do the same for hotels. Note that it should be possible to plan more than one flight and one hotel. Note that it also should be possible to ask for the list of flights and hotels several time in one planning session. Finally, the operation getItinerary is used to return the completed itinerary to the client. The operations flightList and hotelList are used by the AirlineReservation and HotelReservation services to return the search results for flights and hotels. Figure 1 shows an example of the use of the PlanTrip service.

**startPlanning** The startPlanning operation takes no arguments, creates an itinerary object, and returns a session id associating the following operations with the same session.

**getFlights** The getFlights operation takes as argument information on where the flight should start and the final destination of the flight. Furthermore, the getFlights service takes the date of the flight. The getFlight operation returns with a set of flight informations, where each flight information consists of a booking number, the price of the flight, the name of the airline reservation service, and a flight containing of start airport, destination airport, date and time of lift-off and date and time of landing, the carrier operating the flight, e.t.c..

The getFlights operation is implemented by calling the getFlights operation of the LameDuck and CheapBird AirlineReservation services. LameDuck and CheapBird call the flightList operation of the travel agency to return their search results asynchronously. The travel agency waits for 5 seconds for an answer of either (or both) of the AirlineReservation services. It then returns the result of the services whose answers were received within 5 seconds.

**addFlightToItinerary** The addFlightToItinerary operation takes a flight (i.e. a booking number, the name of the AirlineReservation service, ...) and adds the flight to the itinerary.

**getHotels** The getHotels operation takes a city and the arrival date and the departure date. The operation returns a list of hotels with the information containing the name of the hotel, the address of the hotel, a booking number, the price for the whole stay at the hotel, start- and end date of the stay, whether a credit card guarantee is required or not, the name of the HotelReservation service, e.t.c..

The getHotels operation is implemented by calling the getHotels operation of the NiceView HotelReservation service. The services calls hotelList to return its result. The getHotels operation of TravelGood should return the empty list if NiceView does not answer withing 5 seconds.

**addHotelToItinerary** The addHotelToItinerary operation takes a hotel (i.e. booking number, the name of the HotelReservation service, ...) and adds it to the itinerary.

**getItinerary** The operation getItinerary returns an itinerary with all the necessary information on flights and hotels, like booking numbers, prices, addresses, etc. For each of the items, the itinerary should have a field that says whether the flight/hotel is booked or not (i.e. `confirmed` or `unconfirmed`) or the booking is `cancelled`. After doing the planning process, the booking status field should be `unconfirmed` for all entries. After successful booking of the itinerary, the booking status field should be `confirmed` for all entries. When the intinerary is cancelled, the booking status should contain `cancelled`.

**flightList** The flightList operation is a one-way operation and takes as argument a set of flight informations as described in Sect. 2.2.2 for the operation getFlights.

**hotelList** The hotelList operation is a one-way operation and takes as argument a set of hotel informations as described in Sect. 2.2.3 for the operation getHotels.

**Remarks** Note that it is not necessary to implement business logic related to the validity of flights/hotels, e.g. that flights are not overlapping or that the hotel is booked in the destination country/city of the flight.

**Book Trip** The BookTrip service offers an operation bookItinerary which goes through all flights/hotels in an itinerary and books them with the respective AirlineReservation and HotelReservation services.

The bookItinerary operation takes an itinerary and customer information (including credit card information) and books all the flights and hotels in the itinerary by calling the respective bookFlight and bookHotel operations of the respective AirlineReservation and HotelReservation service. It returns the itinerary where the status of each flight and hotel reservation is set to `confirmed` together with a boolean status field which is set to `true` (cf. Fig. 2).

If any of the bookings fail then any successful bookings done so far should be cancelled using either the cancelFlight or cancelHotel operations. BookTrip returns a status field with value `false` and an itinerary where all items that were booked and then cancelled have the booking status `canceled` and all items that are not booked have the booking status `unconfirmed`..

**Cancel Trip** The CancelTrip service offers an operation cancelItinerary which takes as argument an itinerary that has been booked and cancels all the reservations by calling the respective operations cancelHotel and cancelFlight of the AirlineReservation and HotelReservation services (cf. Fig. 3). After cancelling, the booking status field should say cancelled. In case there is an error when cancelling one of the flights/hotels, the process should continue cancelling the remaining flights/hotels. CancelTrip returns an itinerary and a status field. With successfull canceling the booking status of each item should be `canceled` and the status should be `true`. In case an error occured during cancelation, the booking status is `confirmed` for the itinerary item that had an error and `canceled` for the others.

### 2.2.2 Airline Reservation Service

The AirlineReservation service offers operations getFlights, bookFlight, and cancelFlight.

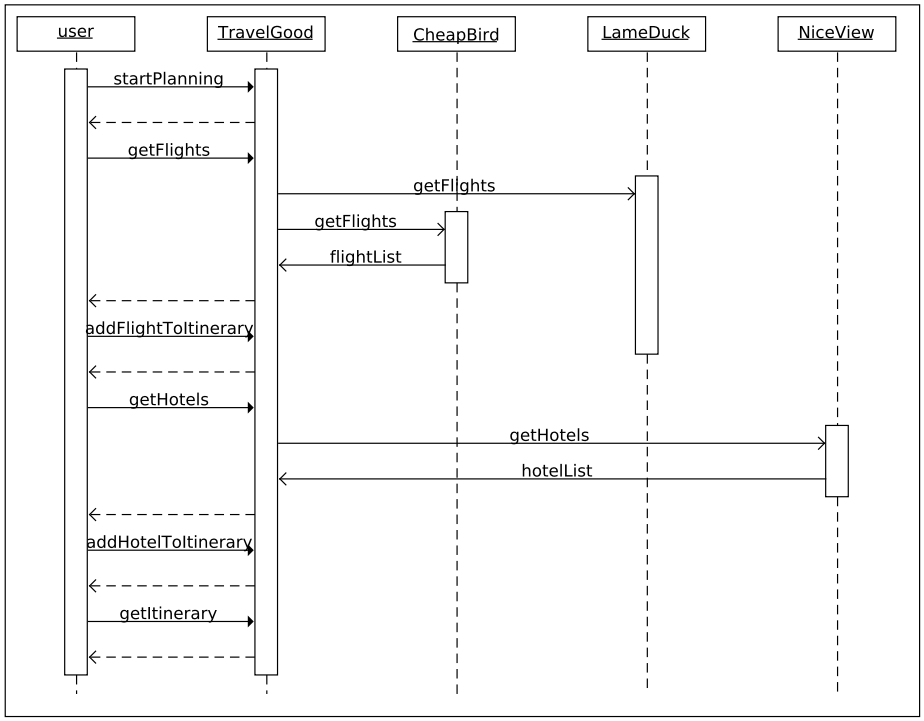


Figure 1: An example of a successful planning session. Filled arrow heads denote synchronous communication where an arrow with a brocken line denotes the return from a synchronous call and open arrow heads asynchronous communication. Note that LameDuck didn't return a flight list in time.

**getFlights** The getFlights operation takes as argument information on where the flight should start and the final destination of the flight. Furthermore, the getFlights operation takes as arguments the date of the flight. The operation is a one-way operation and calls the one-way operation flightList of the TravelAgency service with a set of flight informations. Each flight information consists of a booking number, the price of the flight, the name airline reservation service, and one flight, and each flight contains start airport, destination airport, date and time of lift-off and date and time of landing, and the carrier operating the flight.

**bookFlight** The bookFlight operation takes a booking number and credit card information and permanently books the flight after first having charged the credit card for the flight using the chargeCreditCard of the bank. The bookFlight operation returns true, if the booking was successful and returns a fault (i.e. throws an exception) if the credit card information was not valid, there was not enough money on the client account, or if for other reasons the booking fails.

**cancelFlight** The cancelFlight operation takes a booking number of a booked flight, its price, and credit card information and cancels the flight by refunding 50% of the price of the flight using the refundCreditCard operation of the Bank service. It throws an exception when for whatever reasons the canceling of the flight fails.

Note that for simplicity, the cancelFlight operation does not check if cancellation is actually possible (e.g. that the booking number exists, the price is correct, and that the date of the cancellation is before the date of the first flight of the booking).

### 2.2.3 Hotel Reservation Service

The HotelReservation service offers operations getHotels, bookHotel, and cancelHotel.

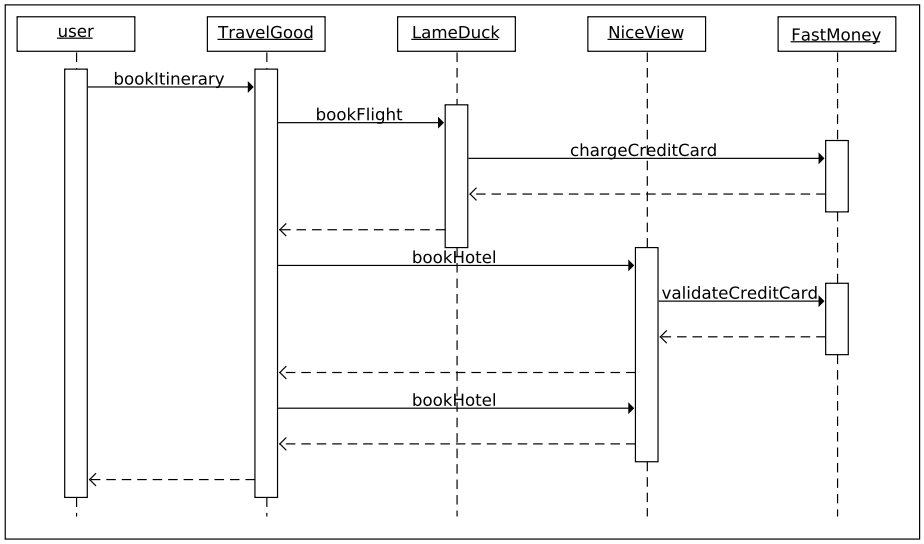


Figure 2: An example of booking an itinerary having one flight and two hotel reservations (one of them requiring a credit card guarantee).

**getHotels** The getHotels operation takes a city and the arrival date and the departure date. The operation is a one-way operation and calls the one-way operation hotelList of the TravelAgency service with a list of hotels with the information containing the name of the hotel, the address of the hotel, a booking number, the price for the whole stay at the hotel, whether a credit card guarantee is required or not, and the name of the HotelReservation service.

**bookHotel** The bookHotel operation takes a booking number and credit card information (depending on whether a credit card guarantee is required or not) and books the hotel. If a guarantee is required, the operation validateCreditCard from the Bank service is called.

**cancelHotel** The cancelHotel operation takes a booking number of a hotel reservation and cancels the hotel reservation. It throws an exception when for whatever reasons the cancelign of the hotel fails.

### 2.2.4 Bank Service

The Bank service offers operations to validateCreditCard, chargeCreditCard, and refundCreditCard and will be provided.

The validateCreditCard operation takes information about a credit card (holder name, 8 digit number, and expiration date) and the amount to be guaranteed and returns true if the credit card is valid and the amount is guaranteed, and returns a fault otherwise.

The chargeCreditCard operation takes a credit card information and an amount in Danish crowns and charges the account with the amount. The operation returns true if the credit card could be charged, it returns a fault otherwise.

The refundCreditCard operation takes a credit card information, an amount and an account and transfers the money from the account to the credit card account.

The bank services will be provided at `http://fastmoney.imm.dtu.dk:8080`. There you can find a WSIL file providing detailed information of the provided services.

Note that for logging purposes all the operations take as an additional argument the group number of the project that is calling the service.

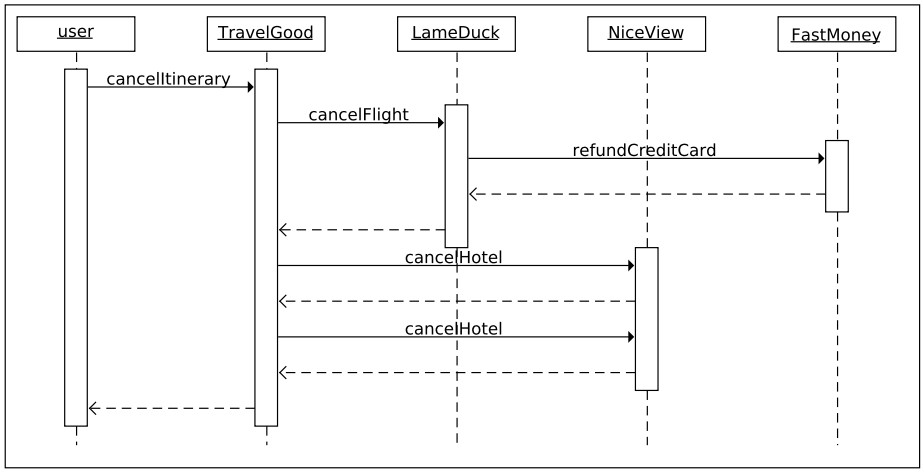


Figure 3: An example of cancelling the itinerary booked in Fig. 2.

### 2.2.5 General remarks

Note that despite the use of the word service in TravelAgency service, AirlineReservation service, HotelReservation service, and Bank service it is possible that these companies offer several port types for implementing the operations, e.g. the TravelAgency can have a port type for all the operations related to planning a trip, a port type for all operations related to booking the trip e.t.c.

The implementation of simple Web services can be done using Java with NetBeans but also using any other Web service technology, e.g. .Net.

All Web services offered by the TravelAgency as desribed in Sect. 2.2.1 shall be implemented as BPEL processes. The Web services of the Hotel Reservation Service and the Airline Reservation Services can be implemented using BPEL, Java, or another programming language.

Note that it is important to implement the correct logic of the Web services as described in this problem description. However, it is possible to hard-code answers for, e.g. the hotel lists or flight lists, for a fixed set of requests. If you have questions regardings this, fell free to send an e-mail to hub@imm.dtu.dk.

All your Web services should be running and have a client application exercising them (preferably using a JUnit — or xUnit for other programming languages, e.g NUnit for .Net — similar to the example Web services from the lectures). Note that during the project presentations you may be asked to show the running system by executing the clients.

It might be helpful, e.g. for data-manimpulation, to use some helper services in BPEL processes. Note however, it is not allowed that these helper services call other Web services.

## 2.3 Test your Web services

The following test scenarios test the basic functionality of the travel good services and thus serve as acceptance tests for these Web services. The tests should be implemented in JUnit and submitted as a (NetBeans) project as part of the file `application_xx.zip` (cf. Sect. 4). Make sure that each of the mentioned test cases corresponds to one test method of the JUnit test.[1]

1. Planning

   P1 Plan a trip by first planning a flight (i.e. getting a list of flights and then adding a flight to the itinerary), then by planning a hotel, another flight, a third flight, and finally a hotel. Check the resulting itinerary for correctness; in particular, that the booking status for each item is

[1] If you are using a different programming language for implementing the project, please use that corresponding xUnit frame work (e.g. NUnit if you use C# or another language based on Microsofts CLR). If such a framework does not exists, you have to write the tests using JUnit.

`unconfirmed`. Have at least one getFlighs operation return flights from both LameDuck (at least two) and CheapBird (at least 2) and check that both lists are correctly merged.

P2a Plan a trip by first getting a list of flights and then adding a flight to the itinerary and return the itinerary. LameDuck does not send its flightList operation within 5 seconds. Check that the correct list of flights is returned and that the itinerary contains one flight with booking status `unconfirmed`.

P2b Plan a trip by first getting a list of flights and then adding a flight to the itinerary and return the itinerary. CheapBird does not send its flightList operation within 5 seconds. Check that the correct list of flights is returned and that the itinerary contains one flight with booking status `unconfirmed`.

P3 Plan a trip by first getting a list of hotels and return the itinerary. The hotelList operation is not being sent within 5 seconds. Check that the list of hotels is empty and that the itinerary is empty.

2. Booking

B1 Create an itinerary with two flight bookings and two hotel bookings (one requiring a credit card validation). Make sure that the booking status is `unconfirmed` for each entry. Check that the itinerary returned by the bookTrip operation returns the same itinerary where each booking has the booking status `confirmed`.

B2 Create an itinerary with a flight booking, followed by a hotel booking, followed by a flight booking. Make sure that the booking status is `unconfirmed` for each entry. During booking, the hotel booking should fail. Check that the result of the bookTrip operations records a failure and that the returned itinerary has `cancelled` as the booking status of the first flight and `unconfirmed` for the status of the hotel– and the second flight booking

3. Cancel

C1 Create an itinerary with two flight bookings and one hotel booking. Make sure that the booking status is `confirmed` for each entry. Check that the itinerary returned by the cancelTrip operation returns the same itinerary where the each booking has the booking status `cancelled`.

C2 Create an itinerary with a flight booking, followed by a hotel booking, followed by a flight booking. Make sure that the booking status is `confirmed` for each entry. During cancelTrip, the cancellation of the hotel booking should fail. Check that the result of the cancelTrip operation records a failure and that the returned itinerary has `cancelled` as the first and second flight booking and `unconfirmed` for the hotel booking

Note that, if you use the tests to guide the development of your Web services, then it makes sense to add tests testing a smaller set of functionalities in one go. For example, one could only test planning an itinerary containing one flight. Then an itinerary containing one hotel booking, and so on.

In addition, it is also possible to add tests for situations that you think are important, but which I haven't covered with my test cases.

In all cases, however, put the required tests into one file separate from the other tests and have the test methods contain the test number as part of their name. This makes it easier for me to check for the presence of the required test methods and their correctness.

## 2.4 Web service discovery

Create four WSIL files, one for each company (TravelGood, LameDuck, CheapBird, and NiceView). Each WSIL file should link to each other WSIL file. Each WSIL should contain references and descriptions of the Web services offered by each company as defined in Sect. 2.2.

### 2.5 Advanced Web service technology

Discuss the use of the following Web standards for your services:

- WS-Addressing
- WS-Reliable Messaging
- WS-Security
- WS-Policy

Which services will benefit of the use of one or all of these standards and which don't? Explain why or why not? The size of the section should be at most 2 pages.

## 3 Reporting

### 3.1 Structure of the report

The report should have the following structure:

1. Introduction
   (a) Introduction to Web services
2. Web service implementations
   (a) A section on the data structures used
   (b) A section for each service
3. Web service discovery
4. Advanced Web service technology
5. Conclusion

Note that the report should *not* have appendices for any the material mentioned above. It may have appendices for material not required by this problem description. The reason is that I expect the material I require in the project report to be at a certain place.

### 3.2 Introduction

This section should introduce the project and the material covered in the report. In addition, there should be an introduction to Web services (c.f. 2.1) of about roughly 2 pages.

### 3.3 Web service implementations

The first subsection of this section should explain the datastructures used and contain class diagrams providing an overview over for the used data structures.

Each Web service mentioned in Sect. 2.2 should have a short description of what it does and how it is implemented. What were the design decisions involved? For example, which binding style was used and why (e.g. document/literal)?

In particular for BPEL, it should be possible to understand the implementation of the BPEL process from your text.

### 3.4 Web service discovery

This section should refer to and explain the XML documents generated according to the task description in 2.4.

### 3.5 Advanced Web service technology

This section should discuss the points mentioned in Sect. 2.5. This section should be at most 2 pages.

### 3.6 Conclusion

This section should summarise the report and contain the experiences with the project. For example, what was learned, what are the things you can improve next time, and what did made good this time. (Won't be graded!)

### 3.7 Important: Who did what in the project

It is important that with each section/subsection it is marked who is responsible for that part of the text. There can only be one responsible person for each part of the text. In addition, each WSDL, XSD, and BPEL file (c.f. Sect. 4) needs have an author *as well as all Java files (or other programming language file) used for implementing the simple Web services.* Who is author of which file should be listed in `author.txt` file in `ws_xx.zip` (c.f. Sect. 4). *Make sure that each member of the group does something of each task!*

## 4 What needs to be delivered?

In addition to the report itself — should be called `report_xx.pdf`, where `xx` is the group number, e.g. `report_01.pdf, report_02.pdf` ... — two zip files need to be submitted (that is, uploaded to the CampusNet of the project group)..

- The first zip file, which should be called `ws_xx.zip`, contains all WSDL, BPEL, XSD, WSIL, and XML files describing the Web services in Sect. 2.2 and 2.4, such that they can be easily opened using Netbeans tools (e.g. if WSDL include XSD files or other WSDL files, then the tool should be able to find them)..
- The second zip file, which should be called `application_xx.zip`, contains the source files for the Web services (e.g. in form of NetBeans projects) and should include the projects for the test described in Sect. 2.3 as JUnit tests. *Make sure that I am able to run your projects.*
- The third zip file, called `deploy_xx.zip`, which contains the files contained in the dist directory of the different Netbeans projects (i.e. war files for Java Web services, zip files for BPEL composite applications, and jar files + lib directory for the JUnit tests). This will allow me to deploy automatically all your projects and test them against the JUnit tests you have provided. More information and an example zip file can be found on the course Web page.

All four files should be uploaded by **Tuesday 30.10.2010 before 8:00 (morning)** to the corresponding subgroup on CampusNet. It is not necessary to deliver a printed copy of the report.

In particular, `ws_xx.zip` should contain the following:

- For each simple Web service, e.g. a Web service that does not call another Web service:
  - the WSDL (and if needed XSD) file
- For each complex Web service, e.g. a Web service that calls another Web service or is represented as a BPEL process:
  - all WSDL (and if needed XSD) files of all the participating services
  - the .bpel file of the BPEL process
- the five WSIL files mentioned in Sect. 2.4
- a file `author.txt` showing who was responsible for each of the files (c.f. Sect. 3.7)

# A Change Log

**Version 1**

- Initial version