Problem Description

Course 02267 (Fall 2016)

Hubert Baumeister
huba@dtu.dk

Version 1: 23.10.2016

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 sites (e.g. offered by the travel agency itself) and by client applications, e.g. stand alone PC applications used by the consultants working in the offices of TravelGood.

To cater for different types of clients, TravelGood offers its services both, as SOAP/BPEL based Web services and as REST services. While it would make sense to base one implementation of the Web

services on the other, management has decided to implement REST services and SOAP/BPEL Web services separate from each other. The project is intended to act as a pilot project to find the strengths and weaknesses of each of the types of Web services.

TravelGood works together with an airline reservation agency called LameDuck and a hotel reservation agency called NiceView. Both offer their services only as traditional SOAP based Web services. TravelGood and the other companies use the bank FastMoney for credit card payment and money transfer, which offers its services as SOAP based Web services.

1.1 Main Business Process of TravelGood

TravelGood allows the customer to manage itineraries consisting of flight- and hotel bookings; in particular, clients can plan, book, and possibly cancel the bookings of an itinerary. The itineraies a stored on the server on behalv of the customer. Note that the customer shall be able to plan several itineraries at the same time.

The customer initiates the business process by sending a request to create an itinerary on his behalf to TravelGood. During the planning phase, the customer can repeatedly add flights and hotel bookings to the itinerary and get a list of possible flights and hotels for some destination and time. During planning, i.e. before booking the itinerary, the customer can cancel the planning phase (and thus the whole process) at any time.

For simplicity, flights to and from a destination will be booked separately, and thus appear as two flight bookings in the itinerary. There is no requirements that flights and hotels are booked in order. It should be possible to book the flight home before the hotel booking and the flight to the destination. It should also be possible to create more complex itineraries than just going to a destination, staying there in a hotel, and flying back. For example, flying to a destination, booking one hotel, changing the hotel, flying to another destination, booking a hotel there, and coming back.

To answer the list of possible flights and hotels, the business processes uses the SOAP based Web services from the airline- and hotel reservation agencies (cf. Sect. 2.3.1 and 2.3.2). Note that these services are to be used by both TravelGood versions, i.e. by the BPEL and SOAP based Web services as well as by the REST Web services.

Planning is completed when the customer books the itinerary. After booking has completed successfully, the customer cannot change the itinerary anymore. Booking of hotels and flights is done with the help of the airline- and hotel reservation agencies. Booking, however, may fail. In this case, previous, successful bookings have to be cancelled before the booking operation returns with a fault. When booking has failed, the process stays active until the day of the start of the itinerary (e.g. the day of the first flight or hotel booking). It is not possible to start the planning process again. The user has to start a new process again, if he wants to find alternative flights/hotels.

After an itinerary has been booked, the itinerary can be cancelled. The latest time when a booked itinerary can be cancelled is the time when the first travel starts (either flight booking or hotel booking). This is also the time when the overall process is terminated and the storage for the itinerary at TravelGood is released.

When an error occurs during cancelling a booking, cancelling proceeds with the other bookings, and the cancel operation shall return with a fault.

Regardless whether cancelling of the bookings fails or not, the process waits until the time of the first travel before the process terminates. This allows the user to query for the itinerary to see which of the bookings could not be cancelled.

At each time during the process, the customer can ask for the current itinerary which then returns an itinerary of flight and hotel bookings including their respective status (unconfirmed, confirmed, cancelled).

If you have any questions regarding this business process, please contact the TravelGood customer representative for clarification. His e-mail is `huba@dtu.dk`.

2 Tasks

The project has six major parts.

1. A short introduction to Web services as part of the report.

2. The coordination protocol between client and travel agency as a state machine as part of the report
3. An implementation of the services and operations of TravelGood, LameDuck, and NiceView as ordinary Web services, REST Web services, and BPEL
4. WSIL files usable for service discovery
5. A comparison between REST and SOAP-based Web services as part of the report, and
6. A discussion on the use of advanced Web service technology as part of the report.

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, Web service coordination, and REST services. 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 Coordination Protocol: Client and Travel Agency

Include a state machine in the report showing the interactions between the client and the travel agency. Make sure that is consistent with both of your implementations (i.e. BPEL and REST). If it is not, explain the discrepancies.

2.3 Implement the following Web services

2.3.1 Airline Reservation Service

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

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. It returns a list of flight informations (possibly empty), where each flight information consists of a booking number, the price of the flight, the name of the 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 cancelling 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).

2.3.2 Hotel Reservation Service

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

getHotels The getHotels operation takes a city and the arrival date and the departure date. It 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, 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. The bookHotel 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.

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 cancelling of the hotel fails.

2.3.3 BPEL/SOAP Version of the TravelAgency

Implement the business process described in Sec. 1.1 as a BPEL process. What are the appropriate port types, what are the operations within these port types? How is the data represented?

2.3.4 REST Version of the TravelAgency

Implement the business process described in 1.1 using REST Web services. What are the appropriate resources to use? How do the HTTP verbs, e.g., GET, PUT, POST, DELETE, map to the required operations on these resources? Implement the business logic by returning links to next actions in addition to representations of resources. Choose an appropriate representation for your resources and explain your choice.

Include the design of your REST service in the report in a form of a table showing the resource URL's, the HTTP verb, and the action being performed. Here is an example for the student registration used in the lecture.

URI Path = Resource	HTTP Method	Function
/institute	GET	Return the institute object
/institute	PUT	Update the institute object
/institute/name	GET	Return the institute name
/institute/name	PUT	Upadte the institute name
/institute/students?name={n}	GET	Find a student matching name n
/institute/students	POST	Register a student
/institute/students/{id}	GET	Get the student with id id
/institute/students/{id}	PUT	Update the student with id id
/institute/students/{id}	DELETE	Dergister udent with id id

2.3.5 General remarks

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

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 regarding this, feel free to send an e-mail to `huba@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-manipulation, to use some helper services in BPEL processes. Note however, it is not allowed that these helper services call other Web services.

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 group that is calling the service.

2.5 Test your Web services

For both, your BPEL implementation and your REST implementation, you need to implement the following test scenarios as a Java application using JUnit. There should be two separate projects with the required tests for the REST implementation and for the BPEL implementation.[1]

The test scenarios test the basic functionality of the travel good services and thus serve as acceptance tests for these Web services.

Make sure that each of the mentioned test cases is implemented as one test method of the JUnit test with the names testP1, testP2, testP3a, ... Using these names makes it easier for me to check your test implementations.

Please also follow the notes on using JUnit tests on the home page of the course.

P1 (planning and booking) 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. Ask for the itinerary and check that it is correct using JUnit's assert statements – i.e. assertEquals, assertTrue, ... – in particular, that the booking status for each item is `unconfirmed`.

Book the itinerary and ask again for the itinerary. Check that each booking status is now `confirmed`

P2 (cancel planning) Plan a trip by first getting a list of flights and then adding a flight to the itinerary. Then cancel planning without booking the itinerary first.

B (booking fails) Plan an itinerary with three bookings (mixed flights amd hotels). Get the itinerary and make sure that the booking status is `unconfirmed` for each entry. Then book the itinerary. During booking, the second booking should fail. Check that the result of the bookTrip operation records a failure (either an exception or returns a failure HTTP status code). Get the itinerary and check that the returned itinerary has `cancelled` as the booking status of the first booking and `unconfirmed` for the status of the second and third booking.

C1 (cancel booking) Create an itinerary with three bookings (mixed flights and hotels) and book it. Get the itinerary and make sure that the booking status is `confirmed` for each entry. Cancel the trip and check that now the booking status is `cancelled` for all bookings of the itinerary.

C2 (cancelling fails) Create an itinerary with three bookings and book it. Make sure that the booking status is `confirmed` for each entry. During cancelling of the trip, the cancellation of the second

[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.

booking should fail. Check that the cancelling resulted in an error condition (e.g. an exception or a failure HTTP status code). Get the itinerary and check that the returned itinerary has `cancelled` as the first and third booking and `confirmed` for the second booking.

Note that, it is a good development strategy to use the above tests as a guideline for the development of your Web services. Also, it makes sense to add tests testing a *smaller* set of functionalities from the above test cases. 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.

2.6 Web service discovery

Create three WSIL files, one for each company (TravelGood, LameDuck, 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. 2.3.

2.7 Comparison REST and SOAP/BPEL Web services

Describe your experiences of developing a REST service and SOAP/BPEL Web service implementation for the same business logic. What have you learned about the similarities and the differences between the two approaches? Use the following lists of aspects as a starting point for your discussion:

- ease of implementation
- ease of understanding the implementation
- ease of changing the business process later to new or changed requirements
- scalability. Are both approaches suitable for large amount of planning and booking requests?
- ...

2.8 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. Coordination Protocol
3. Web service implementations
   (a) A section on the data structures used
   (b) A section for the airline- and hotel reservation services
   (c) A section for the BPEL implementation
   (d) A section for the REST implementation
4. Web service discovery
5. Comparison REST and SOAP/BPEL Web Services
6. Advanced Web service technology
7. Conclusion
8. Who did what

Note that the report may not have appendices.

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 Coordination Protocol

This section should show and explain the state machine for the coordination protocol between the client and the travel agency – i.e. how the client is allowed to interact with the services (planning, booking, and cancelling) (c.f. 2.2).

3.4 Web service implementations

The first subsection of this section should explain the data structures used and contain class diagrams providing an overview over for the used data structures for both REST and SOAP/BPEL Web services

The airline- and hotel reservation services mentioned in Sect. 2.3.1 and 2.3.2 should have a short description of what they do and how they are implemented. What were the design decisions involved? For example, which binding style was used and why (e.g. document/literal)?

The next section should explain the BPEL process and how it works. It should be possible to understand the implementation of the BPEL process from your text. Explain the design decisions you have made, e.g. what are the port types, operations, which binding style was used and why, ...

The last section should explain your REST implementation. In particular

- What have you chosen to be represented as resources and why
- How are activities of the business process mapped to HTTP verbs, like GET, PUT, POST, DELETE, ...
- How the business logic is implemented
- Which representations you have chosen and why

Please include the design of your REST service as a table containing resource URL's, HTTP verb and function as shown in Sect. 2.3.4.

3.5 Web service discovery

This section should refer to, and explain the XML documents generated according to the task description in 2.6.

3.6 Advanced Web service technology

This section should discuss the points mentioned in Sect. 2.8. This section should be at most 2 pages.

3.7 Comparison REST and SOAP/BPEL Web services

This section should discuss the points mentioned in Sect. 2.7. This section should be at most 2 pages.

3.8 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!)

3.9 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 needs to have an author *as well as all Java files (or other programming language file) used for implementing the simple Web services.* In addition, you should list which author contributed to which section and which file in the section *Who did what* in the report.

Make sure that each member of the group does something of each task! In particular, but not exclusively, make sure that all of you work on the REST part as well as on the SOAP/BPEL part.

3.10 A Note on Correct Citation

Make sure that you properly create references for any ideas you use from other sources or text passages you copy from other sources. With respect to regarding verbatim copy of text passages, they need to be put in quotes and need to contain a reference to where they have been published before. Failure of doing so will be understood as cheating.

This is from section 3.4 of the study handbook (`http://sdb.dtu.dk/2016/25/467`):

> "[..]The general rules regarding quotations and references in connection with written assignments state that direct quotes from other peoples work or own work must be indicated with quotation marks at the beginning and end of the quotation, and a precise reference to the source of the quote must be made either in parenthesis or in a note, including the pages on which the quote is found. If the quote is not rendered word by word, but derives from a specific source, the source must also be indicated in parenthesis or in a note with reference to the relevant page numbers. Sources must be listed in the bibliography.
>
> Stop plagiarism (`www.stopplagiat.nu`) is a web tutorial for students on plagiarism. Here you can find further guidelines on quotes and source references.
>
> **3.4.1. Procedure in case of cheating at exams** The department must notify the Office for Study Programmes and Student Affair at eksamenssnyd@adm.dtu.dk if there is suspicion of violation of the exam rules. Violation may lead to sanctions. See chapter 8.1 of these rules and regulations. [..]"

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` ... — a zip file needs to be submitted.

The zip-file should be called `application_xx.zip`, contains the source files for the Web services (i.e. in form of NetBeans projects), and should include the projects for the test described in Sect. 2.5 as JUnit tests. *Please make sure that I am able to run your projects, possibly after refreshing the Web service client stubs. It is also a good idea to test your projects on a freshly installed computer in the E-bar.*

Please make sure that you submit all the files with the correct name and case (i.e. ***all*** *lowercase). Following these instructions helps me a lot with organizing your contributions.*

The two files should be submitted through the assignment module on CampusNet. It is not necessary to deliver a printed copy of the report.

Version 1

- Initial version