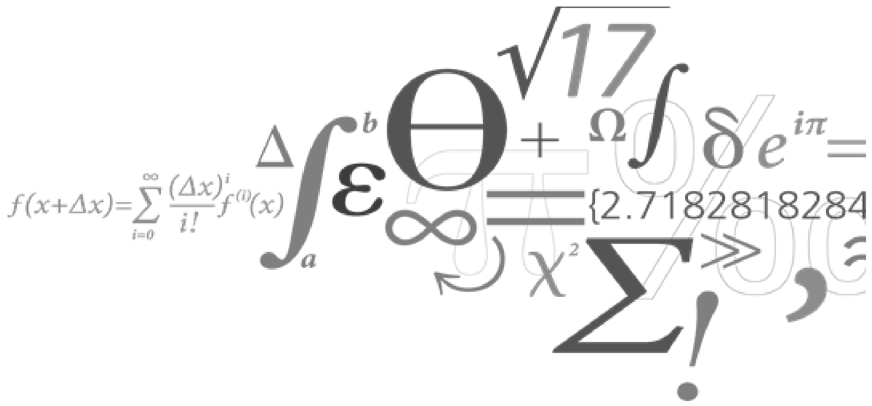
Technical University of Denmark

02267 Software Developement of Web Services

TRAVEL GOOD WEB SERVICE

Group 11:	Paulina Bień	s131107
	Monica Coman	s130919
	Oğuz Demir	s114503
	Audrius Grigaliunas	s141759
	Cæcilie Bach Kjærulf	s093063
	Johannes Sanders	s142249



01. December 2014

1 Introduction

This section should introduce the project and the material covered in the report.

1.a Introduction to Web services

In addition, there should be an introduction to Web services (c.f. 2.1) of about roughly 2 pages.

2 Coordination Protocol

We assume that the clients connecting to TravelGood are stand-alone applications calling our web services. This process starts by sending a customer/client reference identification through the *CreateItinerary* operation, which will create a unique itinerary for the client. In the interaction with the client, the state itinerary changes and this is represented in the state diagram below (see figure 1).

In the state diagram several operation that can change the state of the itinerary are defined. Most of them, such as *AddToItinerary*, *GetHotels/Flights*, *BookItinerary*, *CancelItinerary* are self explanatory and well-defined in the project description. However, we should mention that *TerminateItinerary*, although possible from all states (unbooked, booked, cancelled) has different meanings. The user is able to terminate the itinerary by calling the *CancelItinerary* operation in the unbooked state. Concurrently, in the booked and cancelled states, it might happen that the itinerary is terminated whenever the date of a flight or a hotel has passed.

As soon as an itinerary is terminated, it is removed from TravelGoods record therefore its status can not be called for anymore.

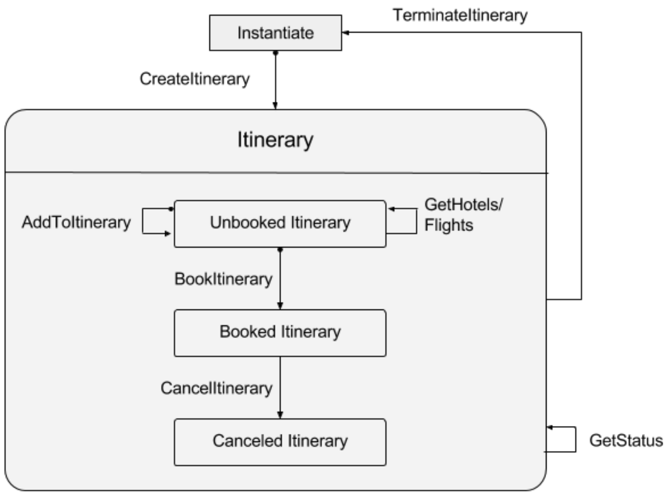


Figure 1: State Diagram for the coordination protocol between the client and TravelGood.

The coordination process between TravelGood, LameDuck and NiceView are similar for both our REST and BPEL implementations. Communication with the client happens only via the TravelGood web service which will afterwards concurrently connect to the external web services (LameDuck and NiceView). Moreover, both LameDuck and NiceView call operations from the FastMoney web service. All the connections between client the and services are portrayed in figure 2.

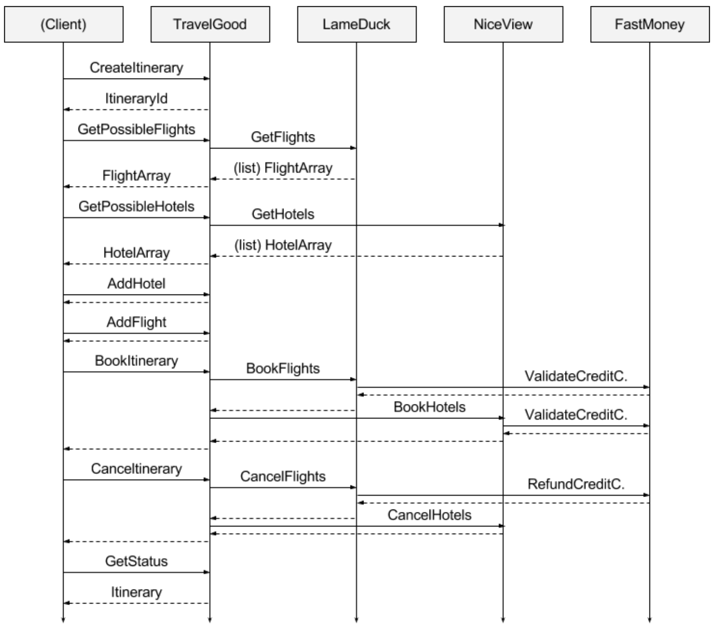


Figure 2: Sequence Diagram for TravelGood, LameDuck and NiceView.

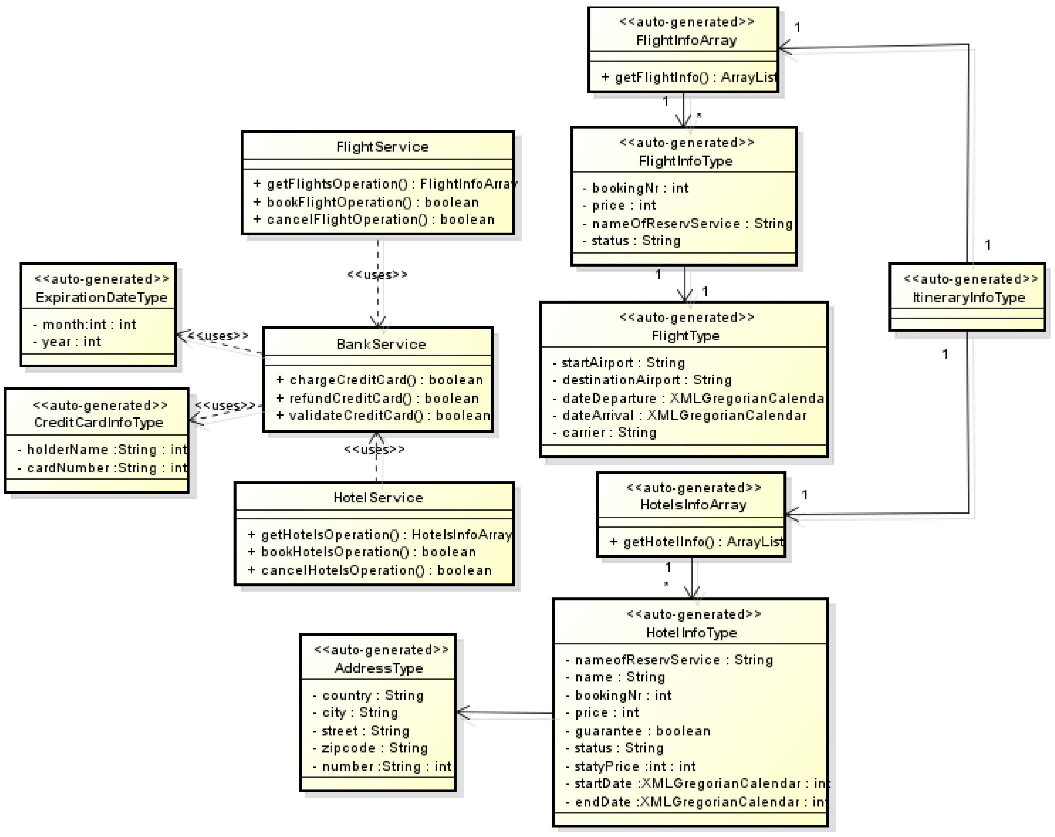
As it can be seen as mentioned above, the client only interacts with the TravelGood service which will afterwards start a new itinerary and return its identifier. In BPEL, TravelGood creates a GUID identification for further communication whereas in REST, a unique string identifier is used. After the itinerary has been created, the client can request available flights and hotels from TravelGood, which will in turn request the info from the LameDuck and NiceView services. The result of *GetFlights* and *GetHotels* operations will be an itinerary which holds two arrays, one for the list of flights and one for the list of hotels.

During the booking (*BookItinerary*) process, the client application is required to also send the bank information along with the itinerary id. This operation can return a fault if booking fails and all exceptions are handled properly (see 3. Web Service Implementations). Similarly, the cancel itinerary operation (*CancelItinerary*) also requires a booking number and credit card info and can return an error if some parts of the cancellation fail.

3 Web Service Implementations

3.a Data Structures

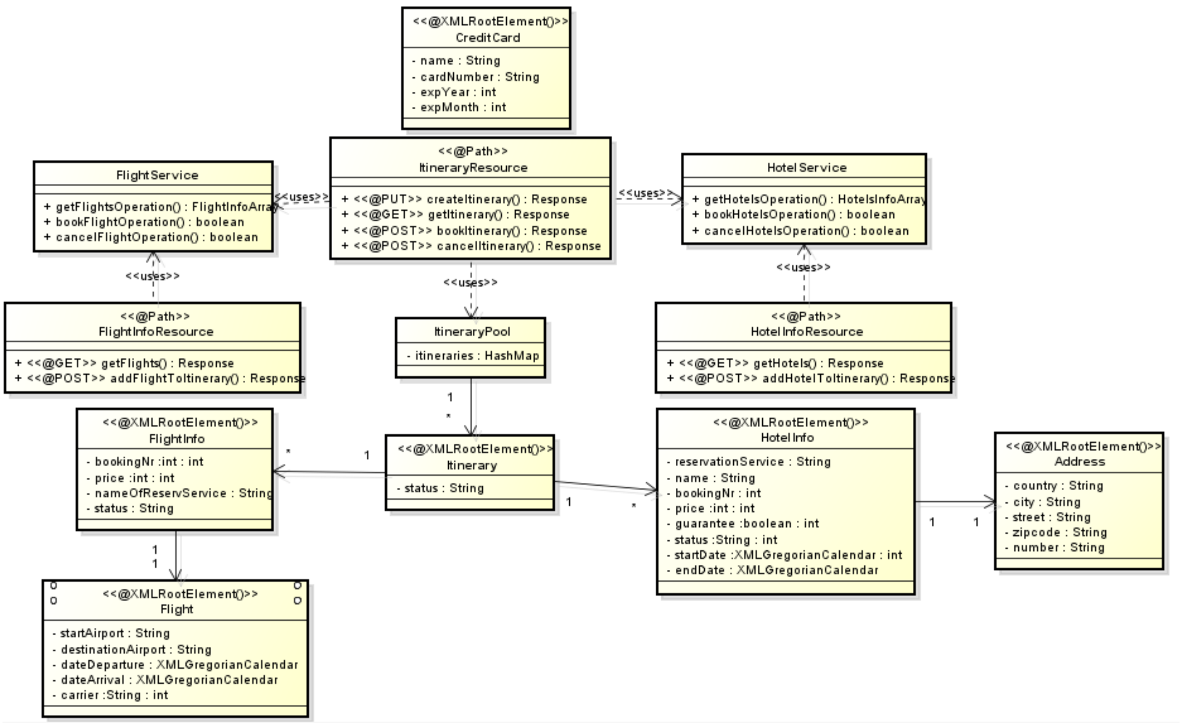
The data structures used within the TravelGood RESTful and SOAP/BPEL are depicted in figure 3 and figure 4. Although describing the exact same process, the two implementations of TravelGood are different and therefore use somewhat different data structures. More details about what are the main differences follow in the discussion below.



<<auto-generated>> FlightInfoArray
+ getFlightInfo() : ArrayList
<<auto-generated>> FlightInfoType
- bookingNr : int
- price : int
- nameOfReservService : String
- status : String
FlightService
+ getFlightsOperation() : FlightInfoArray
+ bookFlightOperation() : boolean
+ cancelFlightOperation() : boolean
<<auto-generated>> ExpirationDateType
- month:int : int
- year : int
<<uses>>
BankService
+ chargeCreditCard() : boolean
+ refundCreditCard() : boolean
+ validateCreditCard() : boolean
<<auto-generated>> CreditCardInfoType
- holderName :String : int
- cardNumber :String : int
<<auto-generated>> FlightType
- startAirport : String
- destinationAirport : String
- dateDeparture : XMLGregorianCalendar
- dateArrival : XMLGregorianCalendar
- carrier : String
<<auto-generated>> ItineraryInfoType
HotelService
+ getHotelsOperation() : HotelsInfoArray
+ bookHotelsOperation() : boolean
+ cancelHotelsOperation() : boolean
<<auto-generated>> HotelsInfoArray
+ getHotelInfo() : ArrayList
<<auto-generated>> AddressType
- country : String
- city : String
- street : String
- zipcode : String
- number :String : int
<<auto-generated>> HotelInfoType
- nameofReservService : String
- name : String
- bookingNr : int
- price : int
- guarantee : boolean
- status : String
- statyPrice :int : int
- startDate :XMLGregorianCalendar : int
- endDate :XMLGregorianCalendar : int
1
*

Figure 3: Class diagram of TravelGood BPEL/SOAP.

Itinerary (ItineraryInfoType) is the main data object that will be manipulated by TravelGood operations. It contains two arrays of *FlightInfo* (FlightInfoType) and *HotelInfo* (HotelInfoType) data objects that will be populated by the user during the planning phase when calling the *addFlightToItinerary()* and *addHotelToItinerary()* operations. Although the *FlightInfo* and *FlightInfoType* classes contain the same attributes (as is the case for *HotelInfo* and *HotelInfoType*), different data structures are created separately for both REST and SOAP/BPEL services since the first one requires classes to be annotated with *XMLRootElement* whereas the latter works with auto-generated classes from the WSDL complex types declarations. Same principle applies for the *CreditCard* (CreditCardInfoType) data object required for communication with *FastMoney* web service.



<<@XMLRootElement()>> CreditCard
- name : String
- cardNumber : String
- expYear : int
- expMonth : int
FlightService
+ getFlightsOperation() : FlightInfoArray
+ bookFlightOperation() : boolean
+ cancelFlightOperation() : boolean
<<uses>>
<<@Path>> ItineraryResource
+ <<@PUT>> createItinerary() : Response
+ <<@GET>> getItinerary() : Response
+ <<@POST>> bookItinerary() : Response
+ <<@POST>> cancelItinerary() : Response
<<uses>>
HotelService
+ getHotelsOperation() : HotelsInfoArray
+ bookHotelsOperation() : boolean
+ cancelHotelsOperation() : boolean
<<uses>>
<<uses>>
<<uses>>
<<@Path>> FlightInfoResource
+ <<@GET>> getFlights() : Response
+ <<@POST>> addFlightToItinerary() : Response
ItineraryPool
- itineraries : HashMap
<<@Path>> HotelInfoResource
+ <<@GET>> getHotels() : Response
+ <<@POST>> addHotelToItinerary() : Response
<<@XMLRootElement()>> FlightInfo
- bookingNr :int : int
- price :int : int
- nameOfReservService : String
- status : String
<<@XMLRootElement()>> Itinerary
- status : String
<<@XMLRootElement()>> HotelInfo
- reservationService : String
- name : String
- bookingNr : int
- price :int : int
- guarantee :boolean : int
- status :String : int
- startDate :XMLGregorianCalendar : int
- endDate : XMLGregorianCalendar
<<@XMLRootElement()>> Address
- country : String
- city : String
- street : String
- zipcode : String
- number : String
<<@XMLRootElement()>> Flight
- startAirport : String
- destinationAirport : String
- dateDeparture : XMLGregorianCalendar
- dateArrival : XMLGregorianCalendar
- carrier :String : int
1
*
1
*
1
*
1
1
1
1

Figure 4: Class diagram of TravelGood REST.

Information about flights and hotels is stored, maintained and manipulated by the external web services, LameDuck and NiceView. Therefore, when calling their operations, both implementations of TravelGood's operations need to make sure that the right input types are used. In BPEL, this consistency is easily ensured by having and XML Schema Document enclosing the declaration of all complex types necessary for LameDuck, NiceView and TravelGood. In REST, however, this is not as straightforward since TravelGood uses *XMLRootElements* such as *FlightInfo*, *HotelInfo*, *CreditCard* which need to be transformed to *FlightInfoType*, *HotelInfoType* and *CreditCardType* in order to be understood by the external web services. This transformations are ensured inside the *FlightService* and *HotelService* classes (see figure 4).

In addition, inside LameDuck and NiceView services, each flight and hotel data is represented with *Flight* (FlightType) and *Hotel* (HotelType) classes where their instances contain static data (name, address, start airport, destination, etc.) of the entity they are representing. When a user requests a flight information, we create the instances of classes *FlightInfo* (FlightInfoType) and *HotelInfo* (HotelInfoType) that contain additional information (such as booking number, price, status) on top of Flight and Hotel. With these additional information, the flight or hotel can be booked and the bookings can be tracked.

Last but not least, REST uses resources (*Path*) on top of all the data objects discussed above which can be accessed by the user via URLs in order to execute different operations (create, book, cancel, etc). This is not the case in BPEL, when only complex types are used for all client-WS or WS-WS interactions. This matter will be discussed more in-depth in the following sections.

3.b Lame Duck and Nice View Web Services

3.c SOAP/BPEL Web Service

3.d RESTful Web Service

URI Path	**Resource Class**	**HTTP Methods**
`/users/userid/itinerary/itineraryid`	ItineraryResource	PUT, GET
`/users/userid/itinerary/itineraryid/book`	ItineraryResource	POST
`/users/userid/itinerary/itineraryid/cancel`	ItineraryResource	POST
`/users/userid/itinerary/itineraryid/flights`	FlightInfoResource	GET
`/users/userid/itinerary/itineraryid/flights/add`	FlightInfoResource	POST
`/users/userid/itinerary/itineraryid/hotels`	HotelInfoResource	GET
`/users/userid/itinerary/itineraryid/hotels/add`	HotelInfoResource	POST

Table 1: API reference for TravelGood RESTful service

URI Path	create-Itinerary	getHotels/getFlights	addHotel	addFlight	book	cancel	get-Itinerary
... (PUT)		x					x
... (GET)		x			x	x	
`.../book`		x*				x	x
`.../cancel`	x**						x
`.../flights`		x		x	x	x	x
`.../flights/add`		x			x	x	x
`.../hotels`		x	x		x	x	x
`.../hotels/add`		x			x	x	x

Table 2: Links returned to client for TravelGood RESTful service
*Only if booking fails, **Only if cancelled in planning phase

4 Web Service Discovery

5 Comparison RESTfull and SOAP/BPEL Web Services

6 Advanced Web Service Technology

7 Conclusion

8 Who did what