

A semantic choreography-driven Frequent Flyer Program

José-Manuel López-Cobo, Alejandro López-Pérez, James Scicluna

Atos Origin SAE, Albarracin 25, Madrid, Spain
{ jose.lopez, alejandro.lopez }@atosorigin.com
University of Innsbruck, Digital Enterprise Research Institute, Innsbruck, Austria
james.scicluna@deri.org

Abstract. Frequent Flyer Programs reward their members with Travel Services from affiliated Service Providers. In this paper, we present a semantic application performing a Frequent Flyer Program in which the customers can create and reuse travel packages. The application is built upon a Service Oriented Architecture, accessing, discovering, composing and invoking Semantic Web Services for the management of the Travel Packages. The composition of semantic services is driven by Choreography, using the Web Service Modelling Ontology (WSMO) as a framework to describe both the service capability and the Service behaviour.

1. Introduction

Loyalty marketing has become a key element in many marketing companies, because it has been proven to be measurable, and considerably effective. In fact, many companies have changed their marketing strategy from mass-market advertising to different direct-marketing strategies, like loyalty-marketing.

Frequent flyer programs (FFP) are the most visible and the most highly developed of the award programs. They are in fact a subset of a larger class of related marketing approaches known as frequency marketing, relationship marketing or loyalty marketing. Frequent flyer programs are loyalty-marketing programs [1]. STREAM Airlines is a fictitious airline company of the fictitious STREAM Group. The STREAM Group has a FFP called STREAM Flows! System (SFS in short). The customers of the SFS (by means of a membership) can obtain travel points from purchasing services of the STREAM Group. These services might be an airline ticket, a hotel booking, a car rental or a shuttle booking (amongst others). There are several companies affiliated to SFS. Each service of the program has its counterpart in points and these points can be consolidated and exchanged with one or more services.

With respect to the cost of the FFP, a division can be made from the IT perspective and the rest. In particular, for a consulted FFP of a medium-size European airline company the possible costs might include:

- IT costs such as software development, human resources, hardware and facilities: 3.5 million €/per year

2 José-Manuel López-Cobo, Alejandro López-Pérez, James Scicluna

- Communication such as to create and send material to the customers and retain their loyalty: 1 million €per year
- Organization and Marketing such as the cost for maintaining the business and all the relationships with the partners of the program: 1.5 million €per year
- Point cost such as those stored in the FFP. In that sense, the FFP acts as a bank, a voucher for the points from the customers: 36 millions of €per year.

For us it's quite clear that solutions like SFS can allow FFPs to decrease their IT cost, making them more profitable due to the increasing incomes from advertisement and marketing.

SFS allows customer and Service Providers to collaborate with each other to reduce the cost of the system and share information. Using this paradigm, the share of knowledge and the understanding from the machine point of view arises as a necessity. The Semantic Web comes at this point as one possibility to reduce the gaps between the ability of the customer and the operability of Service Providers. SFS provides a catalogue to their members where they can search and browse travel services. It also provides mechanisms to create packages of travel services (packages which can be stored and reused). Each travel service contracted is paid using the points gathered in previous transactions with the SFS.

Within the EU funded INFRAWEBs¹ project, we have faced the challenge to build a Semantic application performing a Frequent Flyer Program for a fictitious airline company. The architecture of the application follows the guidelines of the project and it is based on a Service Oriented Architecture over a P2P paradigm as shown in Fig. 1. In this architecture, each peer has the same infrastructure and implements the same set of components (SWS Editor, SWS Composer, SWS Executor, Organisational Memory, Semantic Information Router, and Distributed Repository among others). Semantic interoperability has been sought in this application. For that reason we have used the framework proposed by WSMO [5], paying special attention to the WSMO choreography and orchestration [6] to show the interaction between the customer and the Service Provider and within a Service Provider. The INFRAWEBs architecture is based on two different stages: Design and Runtime. Fig. 1 shows the INFRAWEBs overall architecture organized by stages. Some of the components belong to more than one stage, because they offer features that are used in more than one stage. Consequently, we have put them at the boundary of both stages, thus making them shared.

At Design/Development stage all the activities offer components for annotating, designing and composing Semantic Web Services. At the Runtime stage there are the execution, monitoring and runtime-discovery components to represent the core group.

¹ <http://www.infrawebs.org> For a complete overview of the project, please see [4].

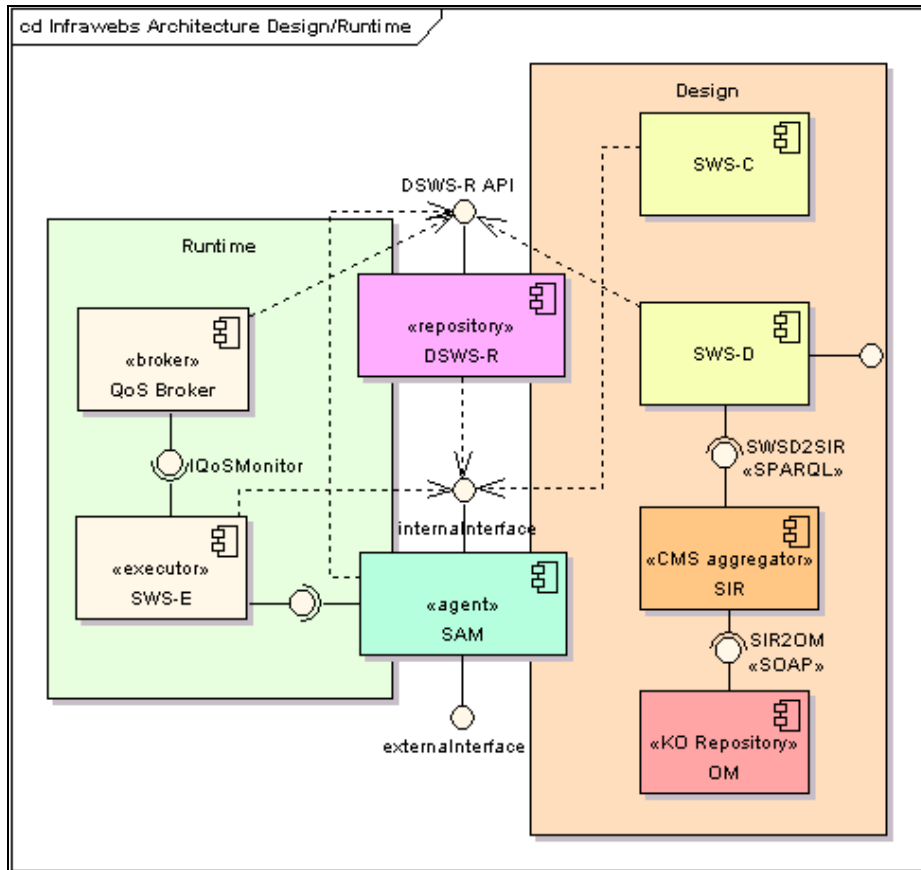


Fig. 1 INFRAWEB architecture

Some of these components appearing on INFRAWEB platform need to interact with each other. The way to achieve this is by means of a set of interfaces exposed by these components. Each of these interfaces describes the set of messages that the component will accept and process. Fig. 1 also includes some of these interfaces used for interactivity purposes. All of these components are briefly described below.

The Functional Architecture of the Designer is a set of functions organized semantically as modules, which are needed for designing a semantic web service using a selected framework for service description. According to the above mentioned basic design principles, a semantic service created by INFRAWEB SWS Designer will be described in WSMO.

SWS-Executor is responsible for executing Semantic Web service descriptions based on WSMO. At the heart of this module are the choreography and orchestration engines which execute the rules defined in the interface descriptions. On top of the executor is also a Quality of Service Broker responsible for monitoring and calculating metric data of the Web services that are being executed.

FCS-OM (Fuzzy Concept Set Organizational Memory) will be responsible for the collection, organisation, refinement and distribution of the entity specific knowledge and will be designed on self-organising memory approaches.

SIR is a meta-data based content management and aggregation platform, that will have a SPARQL query interface, which will be used by the SWS-D to query for service description meta-data, and the UDDI interface integration with UDDI based service repository. It will also include an interface to the OM component, based on SOAP which is described in the following section.

DSWS-R Repository is designed to enable efficient storage and retrieval of WSMO SWS descriptions. It is realized as an extension of the wsmo4j and ORDI components, which already provide basic WSMO software infrastructure. The DSWS-R API defines the interface methods that are available to the other INFRAWEBS components.

SAM (Service Access Middleware) provides a retrieval and execution interface for advertised SW services. The user/customer mandates a user interface agent for fulfilling the service demand. The user/interface agent gives recommendations based on the user's query.

The rest of the paper is structured as follows: in Section 2 a scenario will be described for the creation of a package using the application, in Section 3 we will describe the WSMO elements we have needed for this application, putting the stress in the Ontologies needed for describe the domain and the application. In section 4 and 5 we will show how this application faces the discovery and composition of travel services, highlighting the orchestration and choreography needed both from the requester and provider point of view. Finally we will draw some conclusions and sketch next steps in our work.

2. Scenario Description

In this section we will describe a typical scenario in which a customer uses the SFS portal for the creation of a package of Travel Services, find suitable services for each slot of the package and, finally, contract the proposed services for booking a flight, a hotel room and the renting of a car.

1. The customer logs in the application. We will define a scenario in which a customer "*Jane Doe*", having 2500 points in her account wants to go to Madrid from Brussels and rent a car in Madrid Airport (Barajas) to go to Toledo and book a room in a 4 stars hotel in Toledo".
2. The customer defines her desires on the "Voyage Planner". Within the planner, Jane can choose a stored package (flight + hotel + rent-a-car with standard restrictions) or create a new one. Jane prefers to create her own package. A box will appear and she will drag & drop icons from the left (one for the flight, one for the hotel and one for the car). Each slot in the box represents an abstract service (abstract in terms of desired behaviour but not fully specified [2]). The definition of the constraints may be based on time, distance, accumulated amount of points and whatever other property with an ordinal and quantitative metric. Any of these properties can be graphically defined as connections

between the slots. For her own package she defines the *following set of constraints*:

- The location parking of the rent-a-car service has to be within 3 km of the destination airport.
 - The rent-a-car service must be open at the scheduled landing time.
 - The hotel check-in date has to be later than the scheduled time for landing.
 - The accumulated amount of points for the three services can not be greater than 2200 points.
3. Once she has defined overall constraints for the package, she may want to define *specific constraints for each service*. She will have a form in which she can detail which specific features for each service she wants to obtain and also impose constraints to these features. These constraints can be seen (from a WSMO perspective) as part of the postcondition of the goal and can be easily translated into logical expressions. Some of *these constraints* can be summarised here:
 - The selected hotel must be rated (at least) as a 4 star.
 - The location of the hotel will be some place within a 10 km range from Toledo.
 - The flight destination will be Madrid.
 - She wants a car with air conditioning system.
 4. Once she has the package with the three slots filled (each slot representing an abstract service with constraints to observe. The abstract services are defined by goal templates in which part of the logical expressions are identified with variables. The values for these variables will be substituted with the input from the customer), a goal (in terms of WSMO goals) is created using the templates and the values obtained from the customer interaction with the interface. These goals are sent to the Discovery Component and the Discovery searches into the SFS Catalogue which services are available for each slot.
 5. The selection phase not only performs selection task but also checks the coherence between the services chosen (the set of services chosen, as a whole, have to fulfil the constraints for the package imposed by the customer).
 6. The customer invokes the services which fulfil her package and each Service Provider will contact SFS for the payment of the service (the payment, from the customer perspective is made with points, but from the Service Provider perspective is made by the SFS). This arrangement between SFS and Service Providers is out of the scope of this paper.

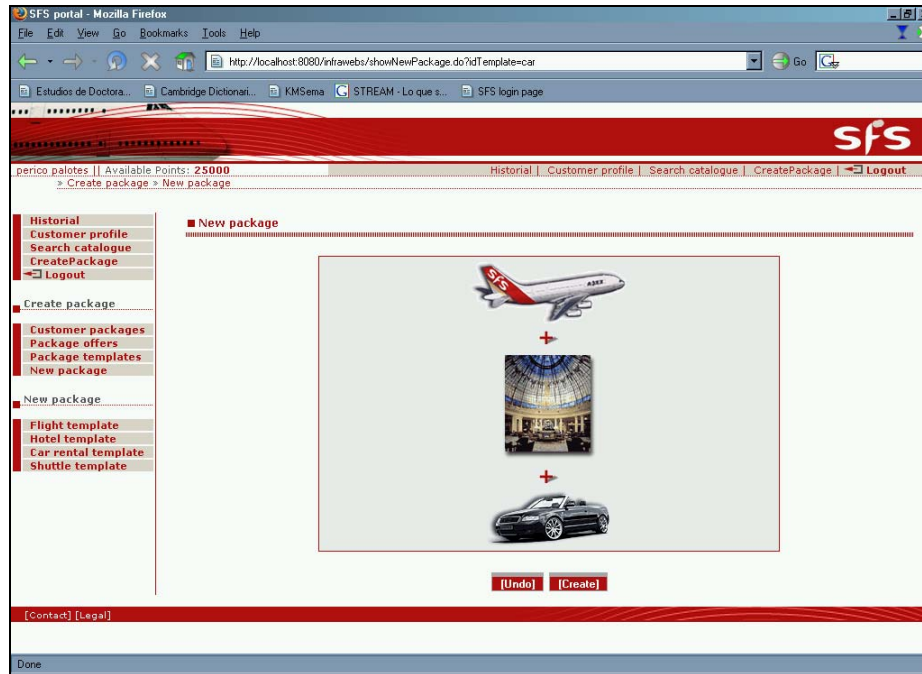


Fig. 2 Screenshot of the creation of a package by means of SFS

3. Ontological structure of the SFS

In this section we describe briefly the SFS ontologies that model the domain presented in our scenario, and which will be used by the Semantic Web services developed by the Service Providers. These ontologies have been written in WSML [7]. According to the classifications of Van Heijst et al. in [9] and Mizoguchi et al. in [8], we can distinguish between the following types of ontologies:

- *General ontologies*, which represent common sense knowledge reusable across domains. In this group we can include our ontologies about customers, date & time and locations. Some of these ontologies have been defined by other people and we only import them.
- *Domain ontologies*, which represent reusable knowledge in a specific domain. In this group we can include our ontologies about travel facilities, such as flight, hotel room, car rental and shuttle.
- *Application ontologies*, which represent the application-dependent knowledge needed. In this group we can include our ontologies about booking travel services and the knowledge needed to build a package inside SFS. The fact that we classify them under this group does not restrict their reusability in other ontology-based applications; rather, it means that we have not designed them taking into account such an objective.

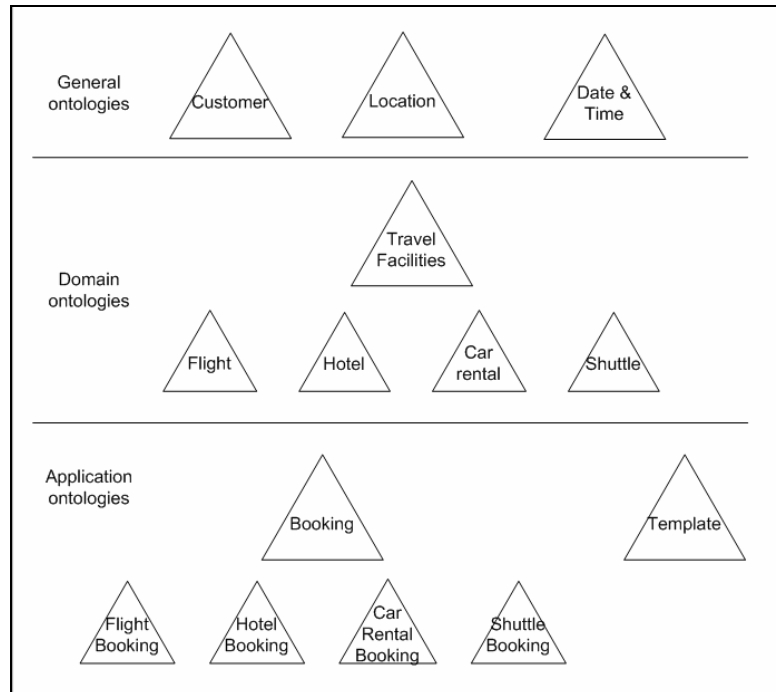


Fig. 3 SFS ontologies

We are going, now, to describe in more detail some of these ontologies, paying more attention to those which are related to the templates and packages. In the following sections we will use them to explain how SFS makes use of the template and the packages for discovering, selecting, composing and invoking Semantic Web Services. Some of the ontologies used in this paper will be only named. For a complete overview of the SFS Ontologies, please consult [3].

Domain Ontologies

These ontologies describe some domain concepts used in the SFS application. Among them we can encounter Tourism Service Providers and the Service's Ontologies (flight, hotel, rent-a-car, shuttle). The service providers will offer different flights, hotel rooms, cars, shuttles and they will have to store the reservations already made in order to be able to make future bookings. The ontologies also include taxonomies such as the rooms of a hotel.

Template and package ontologies

This ontology describes the template and package concepts. It also describes the package constraints. The ontology will be used to find the necessary goal templates for building the package. Logically, the template and the package template are stored

in the SFS as instances of these concepts. Each instance will have a file (for a template) or many (for a package) which represent the abstract capability that the requester wants to achieve. The system will build the goal (for a service or for “many” services) using these templates and the inputs from the customer (using a rich graphical interface or even natural language in future versions). The example below describes a goal template for booking a hotel room.

Listing 1. Goal template for booking a hotel room

```
goal _ "http://www.wsmo.org/goals/sfs/hotelRoomBookingGoalTemplate"

capability hotelRoomBookingGoalTemplateCapability
  nonFunctionalProperties
    dc#description hasValue "This goal template is used to find a hotel booking web service."
  endNonFunctionalProperties

  postcondition
    nonFunctionalProperties
      dc#description hasValue "For such a goal template, the client must define a postcondition
        with a set of hotel stay constraints. Given this is a template, the variables must be replaced
        with concrete values within the postcondition"
    endNonFunctionalProperties
  definedBy
    ?HotelBooking[
      hotelStay hasValue ?BookingRequest,
      buyer hasValue ?Buyer
    ] memberOf hb#hotelRoomBooking and
    ?BookingRequest[
      checkIn hasValue ?CheckIn,
      checkOut hasValue ?CheckOut,
      numberOfPersons hasValue ?Amount,
      hotelStars hasValue ?Stars,
      numberOfBeds hasValue ?NumberOfBeds,
      smoking hasValue ?Smoking
      memberOf hb#hotelStay and
    ?Buyer[
      email hasValue ?Email,
      name hasValue ?Name,
      surname hasValue ?Surname
    ] memberOf cu#customer
```

Booking ontologies

The booking ontology describes the domain of booking services (which is a specialization of the purchase of an item) within a B2C scenario.

The main constructs of the booking ontology are: **Booking**: the overall construct that holds all aspects for a booking where a buyer reserves a service from a service provider, **Booking Partners**: the parties involved in a booking, i.e. buyer and seller, **Payment**: specifies notions of payment, **Delivery**: specifies delivery methods for delivering a booking ticket from the seller to the buyer

4. Discovery related aspects of templates and packages

As one of the main features of the new SFS application, the ability to build packages of Travel Services is at the heart of the business plan of SFS. The whole idea of the system is to provide the customers the ability of build packages to pack services into a whole service for reusing and pricing purposes, reducing the interaction with humans in the Provider side of the program, letting the customer the responsibility of the decision.

Abstract services vs. Concrete services

The SFS administrator will publish templates in the Catalogue. These templates are ASD (Abstract Service Description) [2]. An ASD is the scaffolding used by the requester to build the goal which represents the desire of the customer when she wants to obtain some Service (or some of them) to perform her need. When the customer chooses a template the abstract service description linked to the template is incomplete. The customer will fill the ASD with constraints. These constraints will form part of the axioms and functions of the requested capability. Once she has the package with the slots filled, a WSMO Goal is created using the templates and the values obtained from the customer interaction with the interface. These goals are sent to the Discovery Component which searches into the SFS Catalogue for available services that can fulfil each slot. The following goal Template is selected when the customer wants to book a room in a hotel

Desire: The booking of a room in a hotel and the delivery of the ticket to the buyer via email.

Postcondition: a hotel room booking for a buyer (the SFS customer), the payment method chosen will be by points.

Effect: delivery of the hotel room booking ticket via email.

Discovery & selection

The customer will send a query to the Discovery module to obtain the set of Services which better suits the requirements of the Customer. For each slot belonging to the package, the Discovery module will be asked to find a set of Services that performs the desired behaviour, maintaining the integrity with respect to the constraints of the slot and checking that the package constraints are satisfied. The process for the selection is as follows (the interaction of the customer with the Discovery has been depicted as a choreography interface in the Fig. 4):

- The user receives a list of possible results for each slot (these services have been discovered taking in account not only the constraints for this slot and the desired behaviour, but also the package constraints). She can *manually select one service or she can be advised by the system* (using QoS metrics or other customer preferences defined previously).
- The selected service is asked for final offers which can fulfil the customer preferences and constraints (the service may have more than one possible

realisation of the service for a concrete goal). These offers are specific instantiations of the service. If an offer is finally accepted by the customer, the concrete invocation endpoint of the service is stored until all the services are chosen. The way in which the choreography of this selection is performed is explained in more detail in the following section.

- Once a concrete offer is chosen, the set of possible results for the other slots have to be checked again due to package constraints which impose some restrictions from one slot to another. As a consequence of this revision the set of possible results *may be reduced*. If the set of results for a slot is not empty, then the customer may repeat the previous steps for each slot. If the set is empty, then the constraints have to be relaxed or the customer has to choose another offer, or another service. The mechanism that drives this process is based on “backtracking”.
- When all the slots are concretized with final offers, then the package is ready to be enacted. Since all the services have agreed with the overall constraints, the order for the enactment is not an issue and they can be contracted in whatsoever order and time (even in parallel).

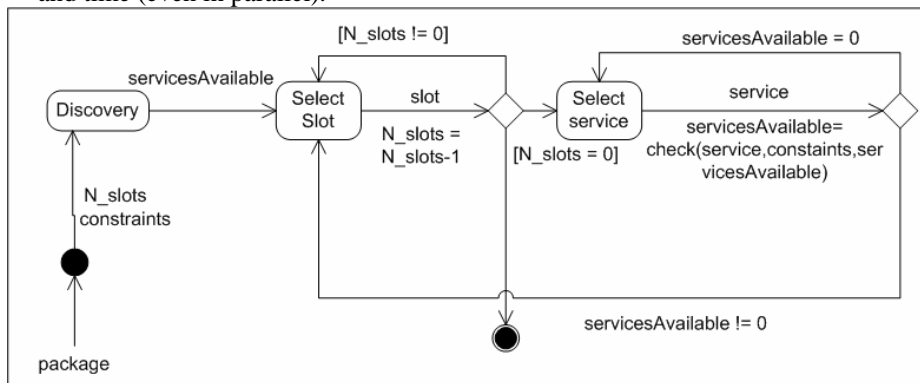


Fig. 4 ASM showing the choreography and backtracking process of the selection phase

5. Composition and enactment of services

We have defined Web Services for booking a flight, a hotel room or renting a car. These Web Services can be composed of other Web Services. This setting allows modelling all notions of a WSMO Web Service description: a Capability of the end-user service and its Choreography for user-service interaction, as well as the orchestration which incorporates the aggregated Web Services. WSMO Choreography deals with interactions of the Web service from the client's perspective. We base the description of the behaviour of a single service exposed to its client on the basic ASM models: (1) they are **state-based**, (2) they represent a state by a **signature**, and (3) it models state changes by **transition rules** that change the values of functions and

relations defined by the signature of the algebra. Taking the ASM methodology as a starting point, a WSMO choreography consists of the following elements:

- State: A state is described in terms of an ontology.
- Guarded transitions: Transition rules that express changes of states by changing the set of instances. These rules are expressions take the form of: **if *Cond* then *Updates***.

The Orchestration part of a service interface defines how the overall functionality of the service is achieved in terms of the cooperation with other service providers with the actual implementation. It describes how the service works from the provider's perspective (i.e. how a service makes use of other WSMO services or goals in order to achieve its capability, see Fig. 5)

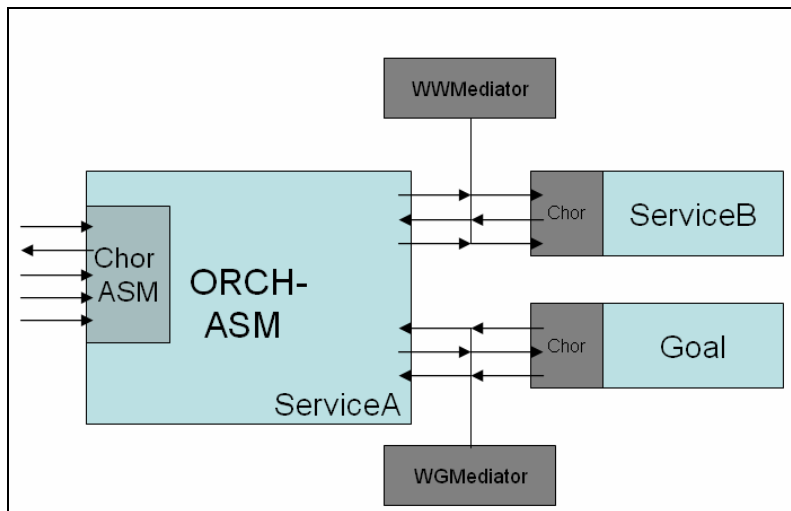


Fig. 5 WSMO Orchestration

We will detail the elements that define a Web Service, highlighting the interface of the Web Service and, especially the Ontologies used for describing the choreography and the orchestration. We use the example of the Hotel Room Booking Web Service. This web service is provided by MH hotel service provider. It offers 4 stars hotels in Spain.

Functionality: A booking for a hotel room will be created and the reservation ticket will be sent to the buyer's email.

Capability

- *Precondition:* The required inputs for this service are
 - 1) The buyer contact information, including the e-mail address
 - 2) the buyer hotel room booking preferences
- *Assumption:* The required payment method is defined. In this case it will be by the subtraction of points from the FFP membership.
- *Postcondition:* A hotel room booking for a buyer is performed and the payment for this room will be made with points. The hotel provider is identified.

- *Effect*: The delivery of the hotel room reservation is done by email to the buyer.

Interface

- *Choreography*
 - When a *hotelBookingRequest* is received, if it fulfils the service provider constraints, all the hotel-room combinations will be returned to the customer (using the *findHotelStayMediator*).
 - Once the customer selects the best combination, and the *hotelStayInfo* is checked, a *hotelBookingInstance* is completed with all the attributes filled in.
- *Orchestration*
 - When a *hotelBookingRequest* is received, the service must make use of another service which provides the *HotelStay* (this internal Web Service looks into the information space of the provider and returns instances of the concept *HotelStay*)

We can see the behaviour of the interface in the next figure:

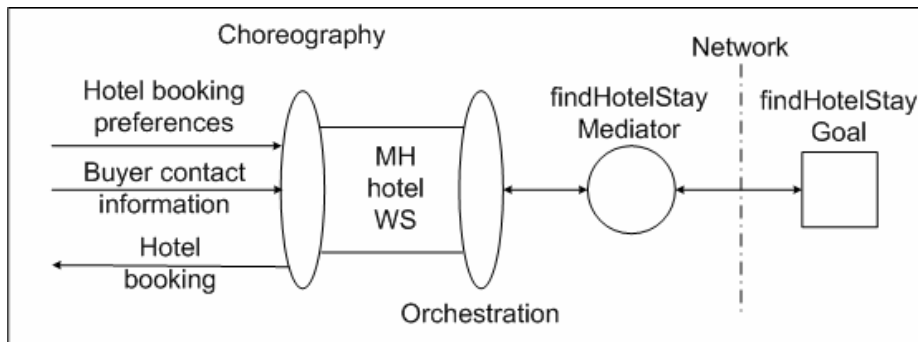


Fig. 6. MH Hotel Web Service accepts the Hotel Booking preferences and the buyer contact information and returns the booking to the customer.

The WSMO Web Service described in WSML is listed here (we only show capability and the interface, without the namespaces and the necessary imports):

Listing 2. The MH Hotel Room Booking Web Service

```

webService http://www.wsmo.org/webservices/sfs/MHHotelRoomBookingWS

  capability MHotelWSCapability

    sharedVariables{?BookingRequest, ?Buyer}

    precondition
      nonFunctionalProperties
        dc:description hasValue "Required input is:
          [1] Hotel Stay Information
          [2] Contact Information
        As precondition the buyer must have an existing email address
        and he has to indicate his preferences of a hotel room booking,
    
```

```

such as: the check-in and check-out date,
desired number of room and beds and if he needs an smoking
room or not, and the minimum hotel stars."
endNonFunctionalProperties
definedBy
  ?BookingRequest memberOf hb#hotelStay and
  ?Buyer memberOf cu#customer.

postcondition
  nonFunctionalProperties
    dc#description hasValue "The output of the service is a NH hotel room
    booking. An instance of a MH hotel room booking have been created.
    This booking has an identifier, a ticket, and the hotel stay info
    has been obtained from the service providers data base
    and satisfy the booking buyer preferences.
    The hotel stay info has:
      Check-in and check-out date,
      room or rooms,
      the hotel."
  endNonFunctionalProperties
  definedBy
    ?BookingRequest[
      checkIn hasValue ?CheckIn,
      checkOut hasValue ?CheckOut,
      numberOfPersons hasValue ?Amount,
      hotelStars hasValue ?Stars,
      numberOfBeds hasValue ?NumberOfBeds,
      smoking hasValue ?Smoking
    ] memberOf hb#hotelStay and
    ?Buyer[
      email hasValue ?Email
    ] memberOf cu#customer implies
      exists { ?HotelBooking } (
        ?HotelBooking[
          hotelStay hasValue ?BookingRequest,
          buyer hasValue ?Buyer
        ] memberOf hb#hotelRoomBooking and
        ?BookingRequest[
          checkIn hasValue ?CheckIn,
          checkOut hasValue ?CheckOut,
          numberOfPersons hasValue ?Amount,
          hotelStars hasValue ?Stars,
          numberOfBeds hasValue ?NumberOfBeds,
          smoking hasValue ?Smoking
        ] memberOf hb#hotelRoomBooking and
        ?Buyer[
          email hasValue ?Email
        ] memberOf cu#customer).

effect
  nonFunctionalProperties
    dc#description hasValue "The booking ticket is delivered to the buyer via email

```

```

        from the service provider to the buyer emailAddress.
        The booking ticket will include the hotel stay info and the buyer info"
    endNonFunctionalProperties
    definedBy
        ?HotelRoomBooking [
            bookingIdentifier hasValue ?BookingIdentifier,
            bookingTicket hasValue ?HotelRoomBookingTicket,
            buyer hasValue ?Buyer,
            seller hasValue ?MHotelServiceProvider
        ] memberOf hb#hotelRoomBooking implies
        exists { ?OnlineDelivery }
        (?OnlineDelivery [
            deliveryItem hasValue { ?HotelRoomBookingTicket },
            sender hasValue ?MHotelServiceProvider,
            receiver hasValue ?Buyer,
            onlineDeliveryMethod hasValue "email"
        ] memberOf bo#onlineDelivery).

interface MHotelWSInterface
    nonFunctionalProperties
        dc#description hasValue "describes the Interface of the Web Service"
    endNonFunctionalProperties
    choreography
        stateSignature
            importsOntology{
                _"http://www.wsmo.org/ontologies/sfs/hotelBooking",
                _"http://example.org/customer"}

            in
                hb#hotelStay withGrounding
                _"http://example.org/MHotelWS#wsdl.interfaceMessageReference(MHotelServicePortType/H
otelRoomBooking/In)"
                cu#customer withGrounding
                _"http://example.org/MHotelWS#wsdl.interfaceMessageReference(MHotelServicePortType/H
otelRoomBooking/In)"

                out
                hb#hotelRoomBooking withGrounding
                _"http://example.org/MHotelWS#wsdl.interfaceMessageReference(MHotelServicePortType/H
otelRoomBooking/Out)"

            transitionRules

                if(?HotelRoombookingRequest
                [
                    hotelCity hasValue ?city,
                    checkIn hasValue ?checkIn,
                    checkOut hasValue ?checkOut,
                    hotelStars hasValue ?stars,
                    allowPets hasValue ?allowPets,
                    allowGroupsReservation hasValue ?allowGroupsReservation
                ] memberOf hb#hotelRoombookingRequest)

```

```

and (
  exists ?MHotelServiceProvider
    (?MHotelServiceProvider
      [
        hasCompanyID hasValue "MH Hotels",
        availableCities hasValue ?city,
        allowPets hasValue _boolean("false"),
        allowGroupsReservation hasValue _boolean("true"),
        minStars hasValue 4,
        hotels hasValue { ?Hotel}
      ] memberOf hb#HotelServiceProvider
    )
)
  then
  add(_# memberOf hb#hotelBooking)
endIf

if(exists { ?BookingRequest, ?Buyer}
  (?BookingRequest[
    checkIn hasValue ?CheckIn,
    checkOut hasValue ?CheckOut,
    numberOfPersons hasValue ?Amount,
    hotelStars hasValue ?Stars,
    numberOfBeds hasValue ?NumberOfBeds,
    smoking hasValue ?Smoking
  ] memberOf hb#hotelStay and
  ?Buyer[
    email hasValue ?Email
  ] memberOf cu#customer)) then
  add(_#[
    hotelStay hasValue ?BookingRequest,
    buyer hasValue ?Buyer
  ] memberOf hb#hotelBooking)
endIf

```

6. Conclusions and further steps

In this paper we have presented an application for a Frequent Flyer Program. It takes advantages of the Semantic Web, particularly of Semantic Web Services. Goal templates ease the usage of the system for the customer since they hide the complexity of the logical descriptions of WSMO. A choreography-driven selection phase has been proposed for this kind of composite services (packages). The composition and invocation of composite services has been also detailed, making use of the orchestration and choreography proposed in WSMO.

Our next steps for the application is the improvement of the description by means of mediators, taking advantage of the different kinds of mediators which are supported by WSMO (mediators for not only data mismatching in terms of ontologies but also for the interconnection of goals and web services and among them).

The implementation of the system is scheduled for the near future and we plan to improve its features including security aspects and the inclusion of tools for the graphical creation and management of Goals templates based on the SWS Designer of the INFRAWEBBS Project.

Acknowledgments

We would like to thank the aid of the Infrawebs team for their fruitful discussions concerning this use case. This work is supported by the EC funded IST project Infrawebs (FP6-511723).

References

1. Frequent Flyer.com. Frequent Flyer Marketing. <http://www.frequentflyer.com/ffp-007.htm>
2. Priest, C.: "A conceptual architecture for Semantic Web Services". In proceedings of the Third International Semantic Web Conference (ISWC2004), November 9-11, 2004, Hiroshima, Japan.
3. López-Cobo, J.M., Pezuela, C.: "Deliverable D10.1-2-3-4.1 Requirements Profile and Knowledge Objects". Infrawebs Project (FP6 – 511723).
4. Infrawebs Brochure. http://www.fh-bochum.de/infrawebs/get_file.php?id=73
5. WSMO. D. Roman, U. Keller, H. Lausen (eds.): *Web Service Modeling Ontology (WSMO)*, WSMO Working Draft D2, final version 1.2, 13 April 2005, available at <http://www.wsmo.org/TR/d2/v1.2/20050413/>
6. WSMO Choreography. J. Scicluna, A. Polleres, D. Roman, D. Fensel (eds.): *Ontology-based Choreography and orchestration of WSMO Web Services*. WSMO Working Draft D14, final version 0.2, 9 December 2005, available at: <http://www.wsmo.org/TR/d14/v0.2>
7. WSML. Jos de Bruijn (eds): *Web Service Modeling Language (WSML)*, WSML Working Draft D16.1 final version 0.2, 20 March 2005, available at <http://www.wsmo.org/TR/d16/d16.1/v0.2/20050320/>
8. Mizoguchi R, Vanwelkenhuysen J, Ikeda M (1995). "Task Ontology for reuse of problem solving knowledge". In: Mars N (ed) *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KBKS'95)*. University of Twente, Schede, The Netherlands. IOS Press, Amsterdam, The Netherlands.
9. van Heijst, G., Schreiber, A. Th., Wielinga, B. J. : "Using explicit ontologies in KBS development". *International Journal of Human-Computer Studies* 45:183-192, 1997.