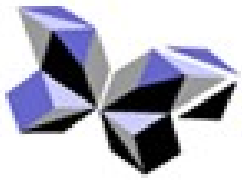

Digital Business Ecosystem

Contract n° 507953

WP21: DBE Architecture

D21.5: Interaction Form Implementation

Project funded by the European Community under the "Information Society Technology" Programme.

Contract Number: **507953**

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D21.5 Interaction Form implementation

Due Date: December 2006

Delivery Date: January 2007

Short Description: This document provides a full description of the Interaction Form implementation

Author: Soluta.net

Partners contributed: Soluta.net

Made available to: Public

Versioning		
Version	**Date**	**Author, Organisation**
00.01	22/03/06	Giulio Montanari, Irina Dumitrascu – Soluta.net
00.02	15/11/06	Giulio Montanari, Irina Dumitrascu – Soluta.net
01.00	11/01/07	Giulio Montanari, Irina Dumitrascu – Soluta.net

Quality check:

1st Internal Reviewer : Javier Val (ITA)

2nd Internal Reviewer: Nagaraj Seetharamon Konda (UCE)

Table of Contents

List of Figures

1 .Executive Summary

DBE allows the execution of a particular kind of service which doesn't need the support of a server to be executed, these kind of services are called Self Contained Service (SCS). They are distributed in a single logical container in which is stored every necessary information related to service execution. An SCS can be searched and located through the DBE Portal and it is automatically downloaded and executed into the client environment. Designing and developing a new DBE service is a task that always requires some significant technical skills, the same applies for SCSs, even if in this case the service structure is simpler and the client-server architecture can be avoided. To help companies which don't want to undertake the investment of developing a service, the IF Wizard has been created. An IF Wizard is a type of an SCS by which, a user who is not skilled in computer programming, is assisted to configure and automatically publish a new DBE service. The simplicity and the effectiveness of this approach would be a driver for fostering the DBE adoption.

Using an IF Wizard requires a minimum knowledge of its structure and functionalities; these are the three main actor involved

- The Service Provider is the user that offers products or services, he or she runs the IF Wizard to create a DBE service; these operations are guided by an intuitive graphical user interface. The creation and publication of a service are managed automatically and transparently by the IF Wizard.
- The Consumer is the user that runs a DBE service, he or she invokes the SCS as any other DBE service, without concerning about the technique by which the service has been created.
- The IF Wizard Supplier, who has the responsibility of designing and develop a new IF Wizard, needs to know its internal structure and its life cycle environment.

Every IF Wizard has been developed to automatically support the creation of a certain kind of service, hence the task of both the Supplier and the Service Provider is simplified. For instance, a wizard can be tailored for Bed & Breakfast or for Book Selling. Currently in DBE it is possible to develop IF Wizards using the Open Laszlo[1] programming language. To construct and deploy a wizard the Supplier must have the knowledge of its technical architecture and must be proficient with the supported programming language.

A tutorial with a complete running example is reported and commented in detail, to help wizard Suppliers in the comprehension of a tangible case and eventually to be the basis on which other IF Wizards would be developed.

1 It produces applications runnable in Internet browsers enhanced with the Macromedia Flash plug-in

2 Forewords

Chapter 3 Introduction is similar to chapter 13 of the deliverable D16.2 [D16.2SMSM]. The chapter was explicitly included into the D16.2 to explain what the IF Wizard was and no external reference to the current document had been made, because this document is newer that D16.2.

After Chapter 3-Introduction which introduces IF Wizard, Service Provider, Consumer and other basic concepts, Chapters 4-IF Wizard Features, 5-IF Wizard Description and 6-User Service Description are useful to understand the functionalities of IF Wizard and User Service, it would be worth for both IF Wizard Suppliers[2] and Service Providers[3] reading this chapters in order to understand the functional aspects related to the creation of an IF Wizard and to the creation/update of a User Service.

Chapters 7-IF Infrastructure and Implementation notes, 8-Tutorial: Creating an IF Wizard service in DBE and 9-Annex A are dedicated specifically to the IF Wizard Supplier who has to understand the whole design and construction process of an IF Wizard; he has to be able to produce wizards tailored to particular business domains.

2 Who develops a DBE IF Wizard
3 Who offers a service in DBE

3 Introduction

The following sections describe, in the context of the DBE environment, the Self Contained Service[4], the IF Wizard and the Interaction Form[5], explaining also the reason for their introduction in the DBE. This deliverable includes comments about the advantages offered by the IF Wizard and the Self Contained Service, and reports in detail the modifications to the DBE project required to implement them.

Although the DBE has been conceived for Small to Medium Enterprises (SMEs), it might result not easy for some of them to create, distribute and maintain a service: Self Contained Services, Interaction Forms and IF Wizards have been introduced with the purpose of easing the creation of services.

3.1 Self Contained Service: the simplest way to join the DBE

The Self Contained Service (SCS) is a service entirely contained in its Service Manifest (SM), that is the object in which all the specifications of a service instance are kept. For SCS both the implementation code and the related data are stored in the SM, as opposed to the regular Business Service[6] case in which only the description of the computing and the business viewpoint are inside the SM, while the actual service code is deployed in one of the DBE's nodes. The SCS is hence not distributed on a server and no service proxy[7] is required, instead the code contained in the SM is executed by the client browser. In this way there is no need for hardware apparatus or Servent. an SCS can even be created, for instance, from an Internet Café using an automatic interactive procedure provided in the DBE environment. The SCS, therefore, enables those SMEs which do not own a proper IT infrastructure to access and create DBE services in a simple way, extending for this reason the number of potential SMEs involved in DBE.

Because SCS was introduced in DBE as an easier way to publish services without managing IT infrastructures, its design doesn't include interfaces to interact with any SME's on-line service. As a consequence of this choice this service type doesn't include a mechanism to exploit SME legacy services, it is not enhanced with a standardised software to interact with any SME's on-line server.

3.2 IF Wizard: “DBE for Dummies”

The IF Wizard is a kind of SCS that has been defined with the goal of enabling SMEs to create and distribute services without the need to write a single line of code, every service produced and published using an IF Wizard is referred as the User Service, which is a synonymous of Interaction Form (IF), a term used only in previous DBE project documents. A significant impact in this approach is that this wizard can be

4 Service entirely contained in the SM.
5 DBE services created and distributed by using an IF Wizard. Interaction Form is usually abbreviated with IF.
6 Business Service is every non structural nor basic service supported by the DBE.
7 Service proxy is the part of a DBE service which references and interacts with the actual service software.

executed from an HTML interface without the need to install the DBE Studio which required over 100Mbyte and a non trivial understanding of software engineering practises.

The introduction of the User Service SCS in DBE represents an evolution of the “Yellow Pages” service [DBEArchReq]. More specifically, the DBE “Yellow pages” service does support a very simple interaction between a service provider and a customer: only phone or fax number can be read, as it happens with the regular paper based Yellow Pages.

The User Service can be easily and automatically generated via an IF Wizard executed by a Service Provider, who is the entity that wants to offer a service. In the DBE context, the User Service adds interaction capabilities allowing users to consume the service by filling in forms, still without forcing the service provider to manually develop a service.

To summarize, the Self Contained Service is a DBE service that does not require an IT infrastructure, while the IF Wizard is a framework based upon the SCS, which enables a Service Provider to offer a service through a simple procedure, that automatically produces an SCS.

3.3 The User Service life cycle

This section briefly illustrates the process of creating and publishing DBE services through the use of the IF Wizard. The process, which is depicted in Figure 3.1.: User Service Life Cycle, involves the following roles:

- the IF Wizard Supplier;
- the Service Provider;
- the Consumer.

The first step in User Service life cycle is the creation of an IF Wizard made by the IF Wizard Supplier. The IF Wizard is a DBE service itself which has the goal of facilitating the creation and distribution of another service. It allows the creation of a single type of service (it can be thought as a unique “Conceptual Service” or a category of services with the same SDNA[8]) and therefore it can satisfy a specific business need. The IF Wizard can be specific to different business areas (i.e. Car Selling, Grocery, Book Shop, Hotel Reservation and others).

To give a clear vision about the IF Wizard it is worth to present an example scenario: Mark Smith is a computer programmer who often develops software for car rental companies, he has the DBE Studio installed in his computer and a good knowledge about the DBE Studio development environment.

- To provide his customers an autonomous way of creating and publishing car rental on-line services, Mark decides to develop a new IF Wizard. Using his own development environment and the available DBE tools he creates the “Rent a Car Wizard by Mark”, an IF Wizard tailored to the automatic creation of car rental related services.

8 Service DNA

- When Mark has tested its new IF Wizard, he publishes it as a new DBE service, giving this way the opportunity to other DBE users to execute it.

In the second step, the Service Provider uses the IF Wizard to create and distribute its new service. In the example scenario we suppose that Pete Swanson, who owns “Pete Rent a Car” business, is willing to create an on-line service to promote his car rental activity. To accomplish this task the following actions are executed:

- Pete Swanson uses his laptop to connect to a DBE Portal via the Internet browser. He searches for “Rent Car Wizard” and he reads “Rent a Car Wizard by Mark” as an available wizard. Pete runs it from the Portal and an application opens up in his browser, ready to allow him to describe his business. He writes “Wimbledon”, “UK” as his office location, then “Sport cars” and “Economy” as car types available, and finally he specifies quantities and rates. Pete is asked by the wizard to provide the mean by which he wishes to be contacted, he specifies “Fax” and adds the number.
- The “Rent a Car Wizard by Mark” at the end summarizes the given information and asks the confirmation “Do you confirm to publish the service?”. Pete presses “yes”.
- As a consequence of Pete's choice the “Pete Rent a Car” service is available as a plain DBE service which is consumable by any User. The “Pete Rent a Car” service is indistinguishable from a regular service developed by hand in the DBE Studio.
-

In other words, the Service Provider executes the IF Wizard which acts as a common “self-installer”, helping the Provider to configure and insert the necessary data and to install (in this case to publish) the service in DBE. All the operations like the creation of the BML, SM, the registration and the publication are completely transparent to the Service Provider.

Finally, the Consumer (which can be either a SME or a single individual) accesses the DBE portal and consumes the service, in the same manner as for Business Services.

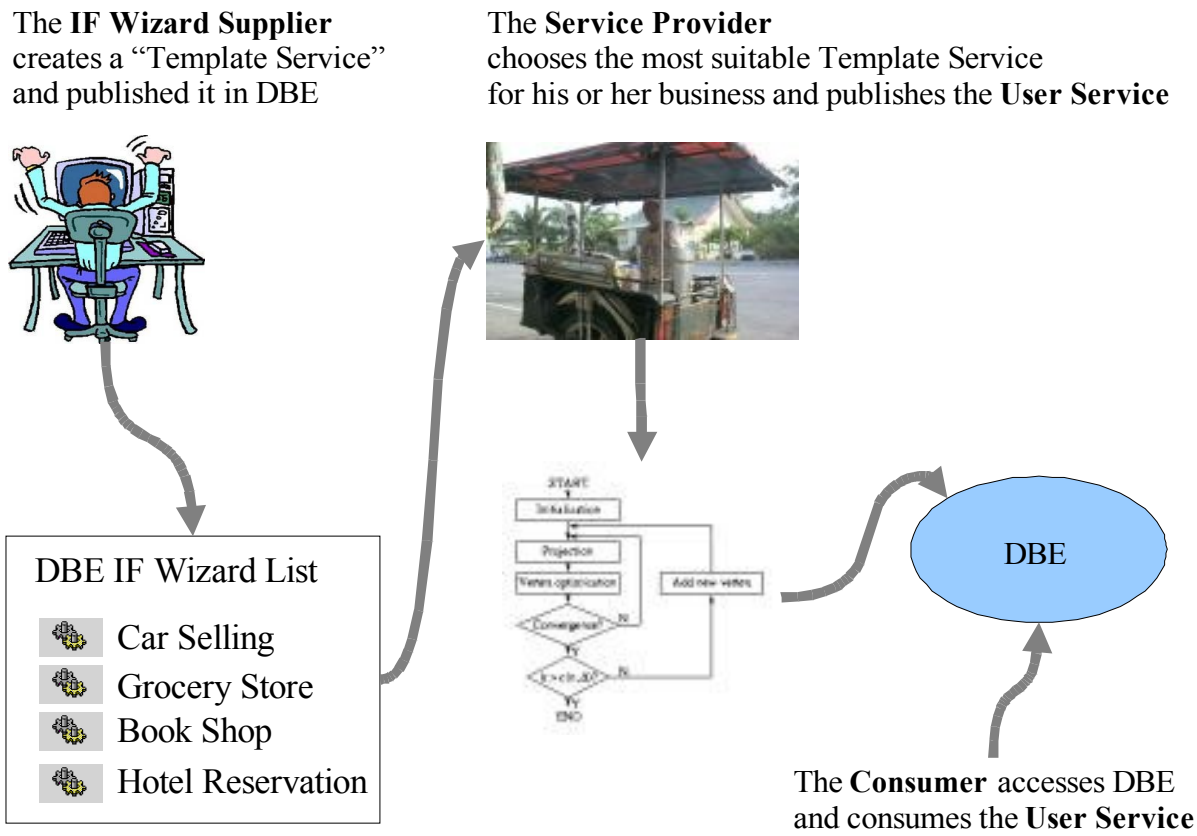

Figure 3.1.: User Service Life Cycle

The following sections provide a more detailed description of the above-mentioned Interaction Form cycle and the related elements introduced.

3.4 Distributed Interaction Form

A Distributed Interaction Form represents an ordinary DBE service but it is directly configured and distributed through the IF Wizard. In this case the service can be more complex than the User Service automatically produced by an IF Wizard (it can make use, for example, of a Database or Web Services exposed by the Provider to interact with its IT system) and it must be executed on a server running a DBE Servent. The server does not necessarily have to belong to the SME, but it can be provided, for example, by an association of category or any other provider, even provided by the supplier who builds the IF Wizard. All the necessary information to distribute the service is contained into the IF Wizard or it is required as an input during the creation of the service.

The main difference between the Distributed Interaction Form and an IF Wizard is that the first must also define a back-end service and take care of deploying the service in the DBE Servent.

Being the User Service a regular DBE Business Service, the data of a Distributed Interaction Form are not stored inside the SM, but they could be for instance saved to a database. This approach allows, as an example, the Distributed Interaction Form to manage relevant volume of data using a back-end database server.

With the Distributed Interaction Form all the functionalities of the accounting service can be used while this is not possible with Self Contained Services.

3.5 Limitations about User Service SCS

Some features about User Service SCS were defined to facilitate their adoption in DBE. SCSs are not fully featured DBE services because they are neither distributed nor executed on a Servent, SCSs have not back-end side services and thus they are not enabled to use DBE services like accounting and metering. The choice of running SCS within the user's computer is due to the consideration that it was not safe to execute untrusted code contained into the SM on the Client Side Servent[9], furthermore the presence of a large number of SCSs could have reduced the performance of the Servent.

Distributed Interaction Forms are instead full featured DBE services; as such they require to be executed on a back-end server. These kind of services can also access on-line applications and IT systems made available by the service provider.

9 Is the Servent when used as a Client. See the Reference section at the end of the current document.

4 IF Wizard Features

The subsequent sections describe the features of the IF Wizard in the context of the DBE, and the complete life cycle of the wizard[10] will be considered: its creation, its execution, and then the User Service creation, publication and execution. To complete the description of the system behaviour, some UML[11] diagram are presented and explained. After discussing the high level features the subsequent chapters explain in detail the characteristics of the environment and of every part of the IF Wizard.

4.1 Actors[12]

The Actors in the following requirement analysis correspond to users in the role of interacting with a system to accomplish tasks. The concept of "system" is not absolute in this type of analysis, it is instead relative to the functionalities provided to an Actor. In the DBE context, the DBE Portal allows a user to search for a service, in this case the DBE portal can be considered a system that offers functionalities to an Actor; even the IF Wizard at run time is a system that provides functionalities to a Service Provider Actor, but the IF Wizard is at the same time a particular kind of Actor – not a human user – that uses the functionalities offered by the Service Manifest Creator. Therefore the IF Wizard at run time plays both the role of a system serving the Service Provider Actor, and the role of an Actor requiring the services offered by the Service Manifest Creator.

4.1.1 IF Wizard Supplier

The IF Wizard Supplier is a computer programmer (or an entity that assigns the development responsibility to a programmer) who owns the required IT skills to create the IF Wizard. The programmer must know at least one programming language among those supported by the DBE platform (for example Open Laszlo for Self Contained Service) and must be able to set up, configure and use the DBE Studio.

The IF Wizard Provider has the responsibility to develop two software applications: the IF Wizard application[13] and the User Service application. The IF Wizard Provider must be able to edit (copy/modify) a BML model both for the IF Wizard and the User Service, furthermore he or she must use the BML Data Editor to insert into the Service Manifest (SM) the BML Data related to the IF Wizard and the BML Data Template related to the User Service.

Several organisations for different reasons can be interested in building and publishing an IF Wizard. For instance a local government authority (i.e. a region, a local municipality) or a private SME (i.e. a wholesaler) could pay a programmer to develop the IF Wizard for launching a service in a particular business area. Even advertising could be introduced in the User Service as a possible value added for the above mentioned organisations. Another case could be that of a Software House, that adds IF Wizard

10 In the context of the current document the term wizard, when not better specified, is always used as a synonym of IF Wizard

11 Unified Modelling Language standardised by Object Management Group (www.omg.org)

12 Actor in this context has the meaning specified by the UML standard by OMG

13 IF Wizard application is the computer program that the Service Provider executes to exploit the wizard

development among its offered programs and services. Even a programmer by itself could act as a specialist in specific kind of wizard creation, producing wizards for his or her clients. He or she could also make wizards just for disseminating the OSS spirit and principle, getting some peer recognitions and promoting their CV/career.

4.1.2 Service Provider

As already mentioned, the Service Provider can be a private or a governmental SME, a public institution or a local authority. To exploit an IF Wizard in the creation of a new User Service, the Service Provider needs to choose, among the published ones, a wizard that fits its business requirements. For instance an IT technical papers seller could search for a wizard that publishes User Service suitable for book or paper document selling; a wine producer and seller could search among wizards that publish Drink & Food on-line selling services. To publish a new Service Instance through an IF Wizard no technical skills are required, the Service Provider interacts with the wizard through a simple graphical user interface. The main interactive task of the wizard program is gathering data to be used by the User Service, so as an example the IT document seller will be asked by the wizard to enter the titles, the descriptions and the prices of the items for sale; the wine producer will be asked to specify the description, the attributes and the prices of the wine bottles for sale. In a similar simple interactive way the wizard will ask the Service Provider to enter the name and the address of the selling company, as well as any other data meaningful for the User Service instance publication.

4.1.3 Consumer

The Customer can be either a SME or a private single individual. The first case is a B2B scenario, in which SMEs interact among each others, while in the second case is a B2C scenario, where a private individual searches for a particular service to consume. The Consumer uses the User Service (that is an SCS) as any other DBE service, by searching it through the DBE Portal and then executing it. From the point of view of the Consumer the User Service instance generated by an IF Wizard is identical to any other hand coded DBE Service, what happens in the system is transparent to the Consumer. SCS actually executes in the Consumer's client and no interaction mechanism with a Service Provider's running service is provided, for this reason interactions between the Consumer and the Service Provider take place out of this usage context, interactions are performed for instance through e-mail messages or other kind of applications. Typically, at the end of its execution, a User Service sends data entered by the Consumer to the Service Provider. For instance an hypothetical service for selling goods on line, would send a new order made by the Consumer to the Service Provider, through a formatted e-mail message.

4.2 Overall Process Model

The overall use case scenario, already presented in the introduction and depicted in figure Figure 4.1.: IF Wizard High Level Overall process, is here presented as an Activity Diagram, that enumerates the main activities of the life cycle of the IF Wizard, indicating the user role responsible for each step of the sequence. The diagram in figure 4.1 has the semantic of a

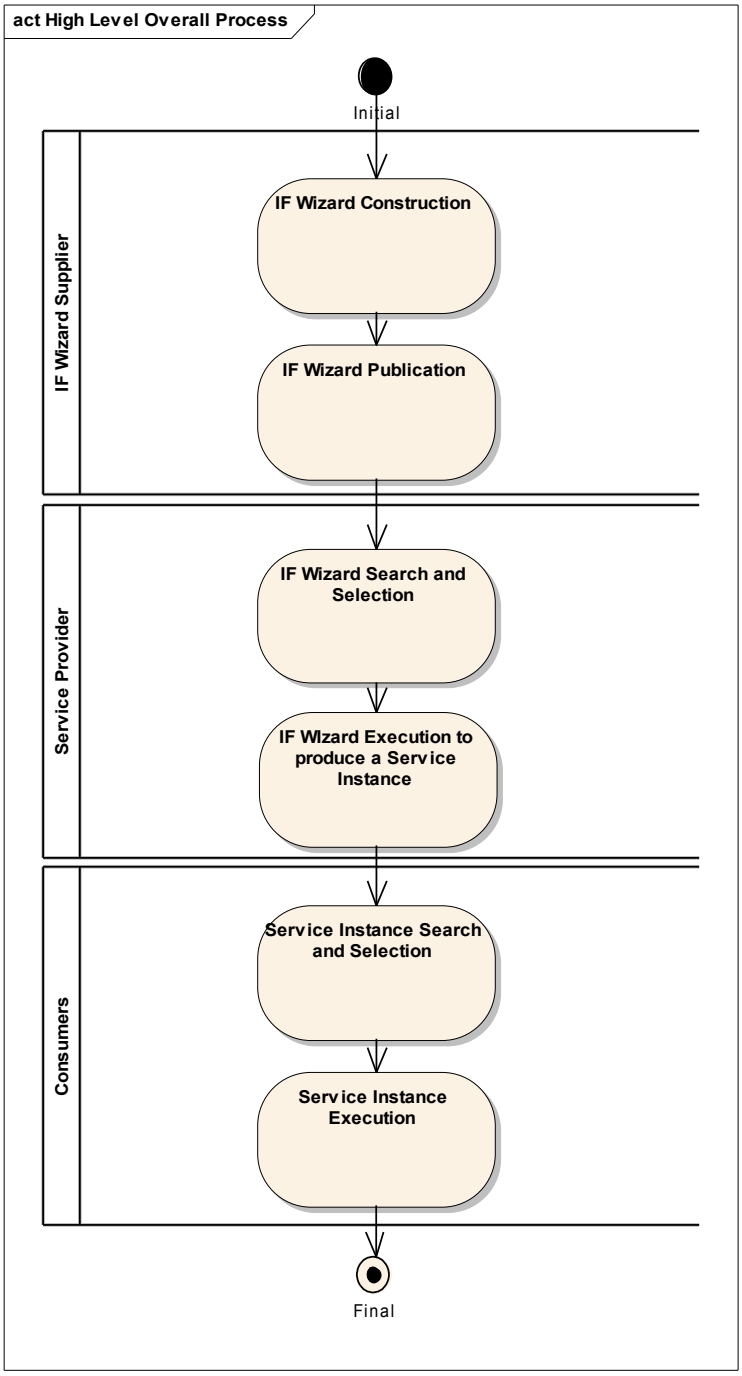

Figure 4.1.: IF Wizard High Level Overall process

Process Model, i.e. a model that depicts the starting and the ending point of the activities, and the necessary transitions from any activity stage to another.

4.3 The IF Wizard Supplier Viewpoint

The IF Wizard usage scenario involves the three main user roles – or Actors – already presented in the previous sections: the IF Wizard Supplier, the Service Provider and the

Consumer. The responsibility of the IF Wizard Supplier is constructing and deploying a wizard related to a specific conceptual service, for instance a wizard that could be used by an on-line book shop to create a book selling and delivering service. The Service Provider has the opportunity to use a specific wizard to create an instance of a service related to its own activity, for instance a book shop could start a new on-line selling service. Finally the Consumer has the possibility to execute an on-line application to obtain a specific service, for instance to buy books from an on-line seller specialised in technical publications. Figure 4.2 will be commented in the remaining of this section.

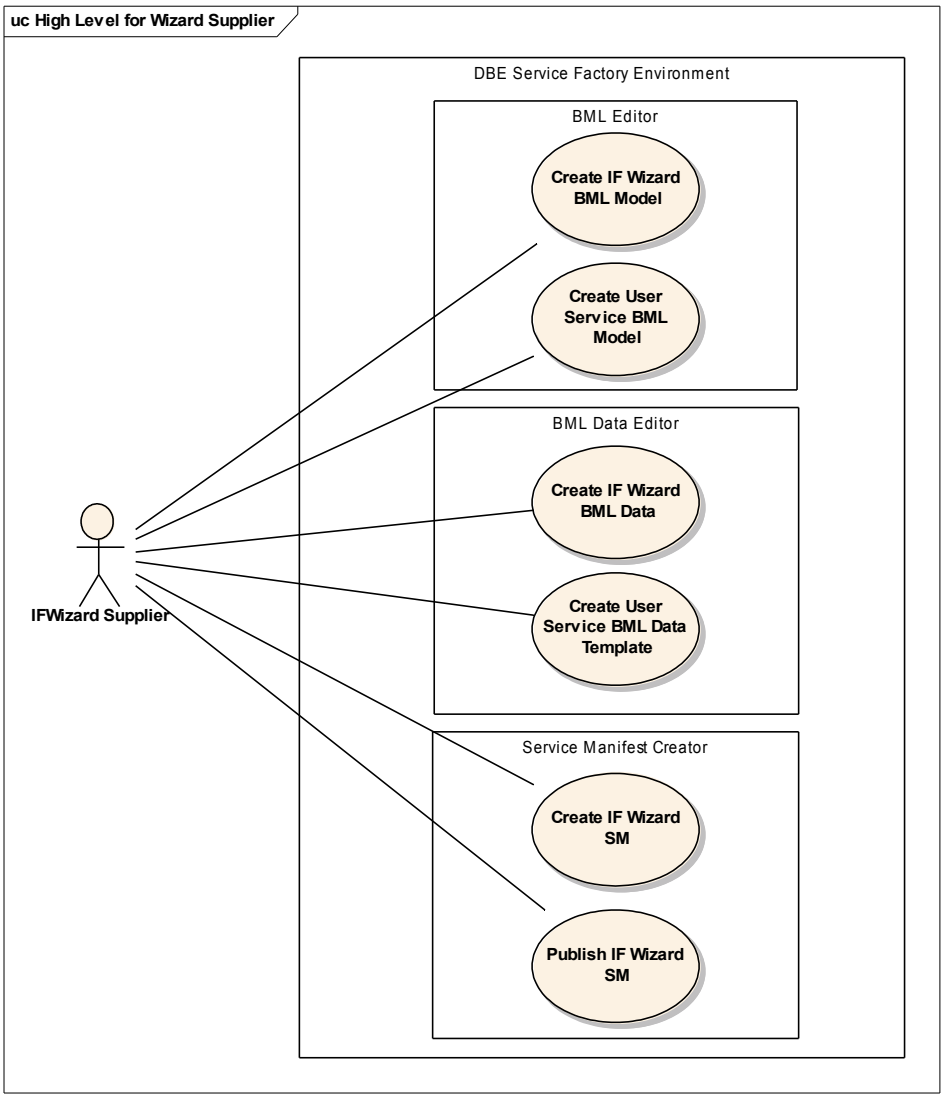

Figure 4.2.: IF Wizard Supplier Use Case Diagram

The IF Wizard Supplier role is played by an actual entity (a company, an agency, a software house, a freelance consultant) that has the necessary technical skills to design and construct a wizard in the DBE environment. To build and publish a new wizard the supplier has the objective to develop a system which automatically produces services, which are instances of the same conceptual “template” service. For example an IF Wizard Supplier could produce a wizard bounded to the Bed & Breakfast reservation

business domain, and in that case the development effort would be aimed at constructing all the necessary software to enable a Bed & Breakfast owner to define and publish its own reservation service in the DBE environment. The IF Wizard is a DBE service by itself, so the first task of the IF Wizard Supplier is to produce a valid Service Manifest (SM). As well as the wizard, the User Service is a DBE service, therefore a SM is needed even for the User Service. The IF Wizard Supplier uses the DBE BML Editor tool to define both the wizard Model and the User Service Model. In the same way the DBE BML Data Editor is used to make up both the wizard BML Data and the User Service BML Data Template.

The BML Data Template, structure of which will be delved in the following chapters (please refer to 5.2 IF Wizard Class Diagram Model, 8.5 User Service BML Data definition, 8.8 User Service BML Data Template definition, and to Figure 8.7.: User Service BML Data Template Example), can be defined at design time, because the target User Service is a well known service, with the particularity that it must be filled by data specific to an instance. As an example, the IF Wizard Supplier in charge of the development of a wizard for the on line book selling domain, knows the structure of the BML Data of the User Service, that probably would contain information like the seller's company name, its e-mail address, the categories of the books in the catalogue, and so forth. Anyway the BML Data of the User Service instance should be filled with actual data provided by the Service Provider, data like “Company Name = Best Paperback Shop”, “Contact = info@bestpaperback.com”, “Category = Classic”, an so forth. For this reason it is necessary for the IF Wizard Supplier to define a template, starting from the User Service BML Data, to drive the wizard at run time during the data gathering phase. While the definition of the User Service BML Data is supported by the DBE BML Data Editor tool, the definition of the template is not, so the IF Wizard Supplier must complete this particular task with his or her own tools, out of the DBE studio environment.

After these preliminary activities with the DBE environment tools and after other specific activities that will be described in a following section related to the wizard construction, the IF Wizard Supplier interacts with the system using the DBE Service Manifest Creator to make up the SM of the wizard, and finally to publish the newly created wizard service. As this last step is completed, the IF Wizard will be available in DBE to be searched and used by Service Providers.

4.4 The Service Provider Viewpoint

The Service Provider role is played by an actual entity (a company, a book seller, a Bed & Breakfast owner, a wine producer) that wants to publish an on line service to sell its products or services. In the IF Wizard scenario the Service Provider doesn't have the necessary skills, the resources or the will to develop a complete DBE service from scratch. In this context the IF Wizard is a suitable means to publish a new service with the minimum effort. Figure 4.3 will be commented in the remaining of this section.

4.4.1 Creation of a new User Service

The Service Provider creates a new User Service instance using the IF Wizard, executing the following operations:

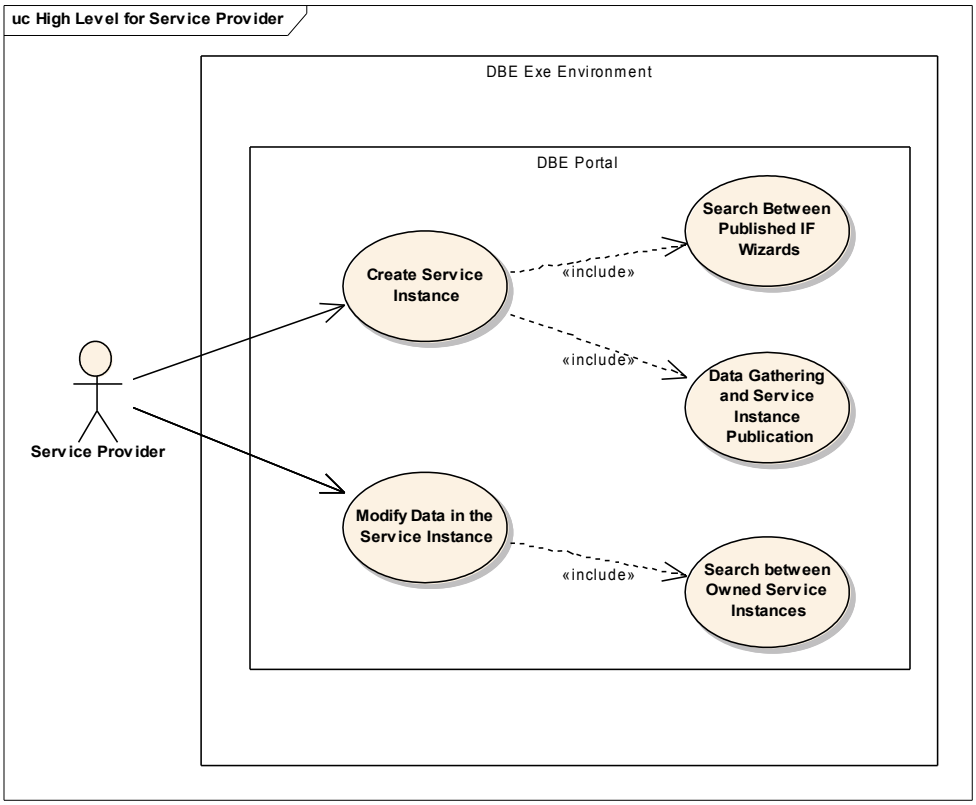

Figure 4.3: Service Provider Use Case Diagram

- Research of the IF Wizards through the DBE Portal: this operation is very simple and it is the same operation executed by all the users which want to consume a DBE service. Therefore it has been conceived for non expert users;
- Choice of the most suitable IF Wizard for its business: generally a ranked list of IF Wizard is presented as result of a research, including a description of each single IF Wizard. The IF Wizard can also be attached to a description of the service (i.e. a short video), including its detailed functioning;
- Execution of the IF Wizard: this requires some further actions that are specific to the chosen IF Wizard (for example, in a book shop service, the Provider will insert the list of the books to sell and some information about the shop).

As a final action, the IF Wizard publishes the service in DBE in a full transparent manner to the Service Provider. Refer to chapter 8 Tutorial: Creating an IF Wizard service in DBE for further details.

4.4.2 Update of an existing User Service

The other main interactions between the Service Provider and the DBE system in the IF Wizard scenario is the update of a published service. After the publication of a new service, it would raise for the Service Provider the need to update the service instance data. For instance, a book seller would like to add new titles to its published catalogue. To update one of its services the Service Provider searches for its own published services through the DBE Portal tool, and then it launches again the wizard to modify the data. When the operation is completed the updated User Service instance is

automatically re-published in DBE by the wizard. Refer to chapter 8 Tutorial: Creating an IF Wizard service in DBE for further details.

4.5 The Consumer Viewpoint

The Consumer role is played by a user that operates either by itself or on behalf of an actual entity (a company, an agency). The goal of the Consumer is to access a service or buy goods that are available through a DBE public service. From the point of view of the Consumer there is no difference between a DBE Business Service and a service published by an IF Wizard. Figure 4.4 will be commented in the remaining of this section.

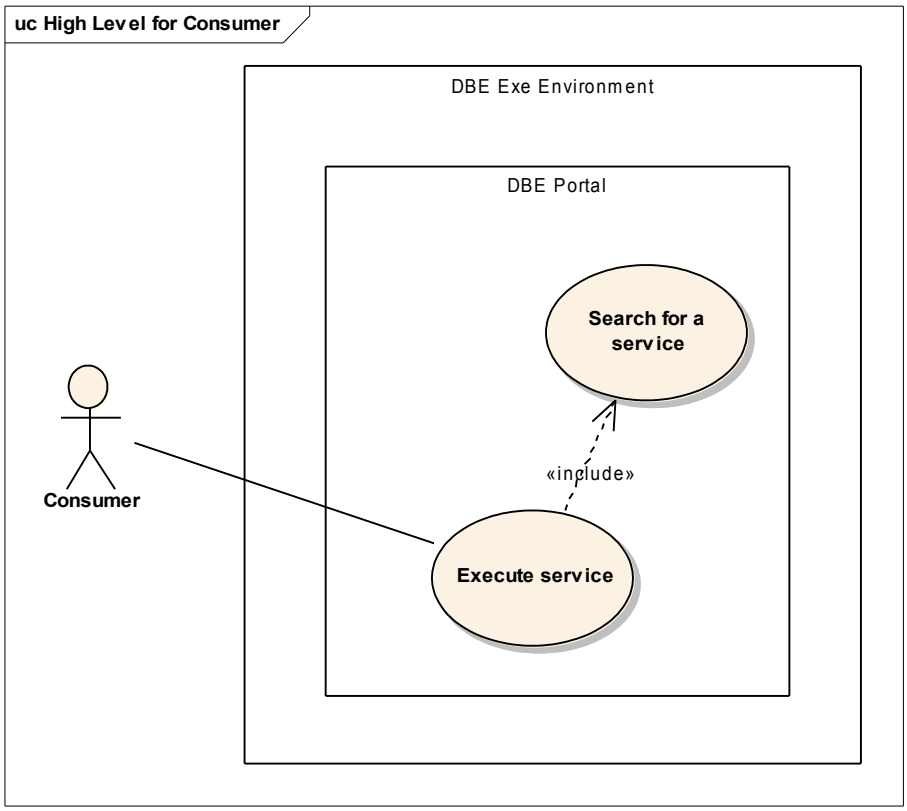

Figure 4.4.: Consumer Use Case Diagram

To execute a published service the Consumer interacts with the DBE Portal to search for a suitable offer, and then he or she executes the service. As already mentioned the functional difference between a DBE Business Service and a User Service created by a wizard, is due to the fact that in the latter case the communication between the service run time and a back end server[14] is not provided. The User Service application executes only in the Consumer's client environment, where no standard functionalities are available to communicate synchronously with the Service Provider. Typically the User Service at the end of its execution, sends data to the Service Provider with the actions requested by the user. For instance using an SCS to buy books, a Consumer couldn't

14 A server used by a Service Provider to provide interactive synchronous services to the Consumer

be able to check the actual stock availability for a certain book, because the User Service created by the wizard is not synchronously communicating with the provider. At the end of the session the service will send a formatted e-mail message to the Service Provider with the requested books. After this request the communication between the Consumer and the Service Provider is out of the scope of the on line User Service instance.

5 IF Wizard Description

The IF Wizard allows SMEs which are not accustomed to the use of tools like BML Editor, BML Data Editor and SM Editor to easily and intuitively create DBE services.

In order to enable a Service Provider to publish a new service - the User Service - the IF Wizard Application at run time gathers information and then automatically publishes the service on behalf of the Service Provider. Once published, the new service is ready to be used by a Consumer. Information gathered by the IF Wizard are grouped in two classes:

- info needed to compose the BML data (e.g. the location of the book-shop, the e-mail address for info requests, the book categories available)
- info needed and used by the User Service (e.g. the book catalogues, the title of the books, the authors, the prices).

Since the IF Wizard has been built bounded to a “specific conceptual service”, differently from other editors the IF Wizard will define and publish a new service offering the final user a simple and straightforward interface.

As an example, in case of publishing of a Book Shop related service, IF Wizard might ask questions like

- “Which is the name of the Book Shop?”;
- ”In which city is it located?”;
- and so on.

Aiming at collecting all the necessary information. A similar scheme would be applied for service related data, where IF Wizard might ask questions like:

- “Do you want to insert a new book? (Y/N)”;
- “Which is the book title?”;
- “How much does it cost?”;
- and so forth,

making data entry very simple even for people not confident with computer programs. Furthermore, it would be possible to create different wizards versions targeted to differently skilled persons, avoiding for instance offering a question-based interface to an advanced computer user.

IF Wizard structure is not fixed or constrained and can be freely conceived, anyway to support and to facilitate this activity a series of libraries have been developed. Such libraries can only be used if the structure described in section 5.2-IF Wizard Class Diagram Model is respected.

An IF Wizard is a particular kind of SCS composed, as any SCS, by the following elements:

- a description;
- the service code;

- a set of data.

The major difference between an SCS and an IF Wizard lays in the fact that IF Wizard contains inside its data a series of information that will be used to create the User Service, including:

- User Service's BML Model;
- User Service's BML Data Template;
- User Service's application code.

The following sections in this chapter will analyse in detail the structure of the IF Wizard service.

5.1 IF Wizard Application Use Case Model

As already mentioned in section 4.1-Actors, from the point of view of Use Case analysis an Actor is not always a human user of a system, it is worth to consider the Use Case model also from the perspective of IF Wizard Application while interacting with the DBE system. The IF Wizard Application is the executable element of the IF Wizard, the computer program that at run time helps the Service Provider producing and publishing a new instance of a User Service.

In figure 5.1 Figure 5.1.: IF Wizard Application Use Case Model Diagram are depicted the functionalities of the DBE system that the IF Wizard Application uses at run time. When the wizard application[15] has completed the interaction with the Service Provider user, and when it has collected all the data necessary to publish the new User Service instance, it uses the operations provided by the Service Manifest Creator to publish the Service Manifest of the new User Service instance. When the new service is published it is available to all the potential Consumers.

15 Wizard application in this document, when not better specified, is used as a synonym of IF Wizard Application

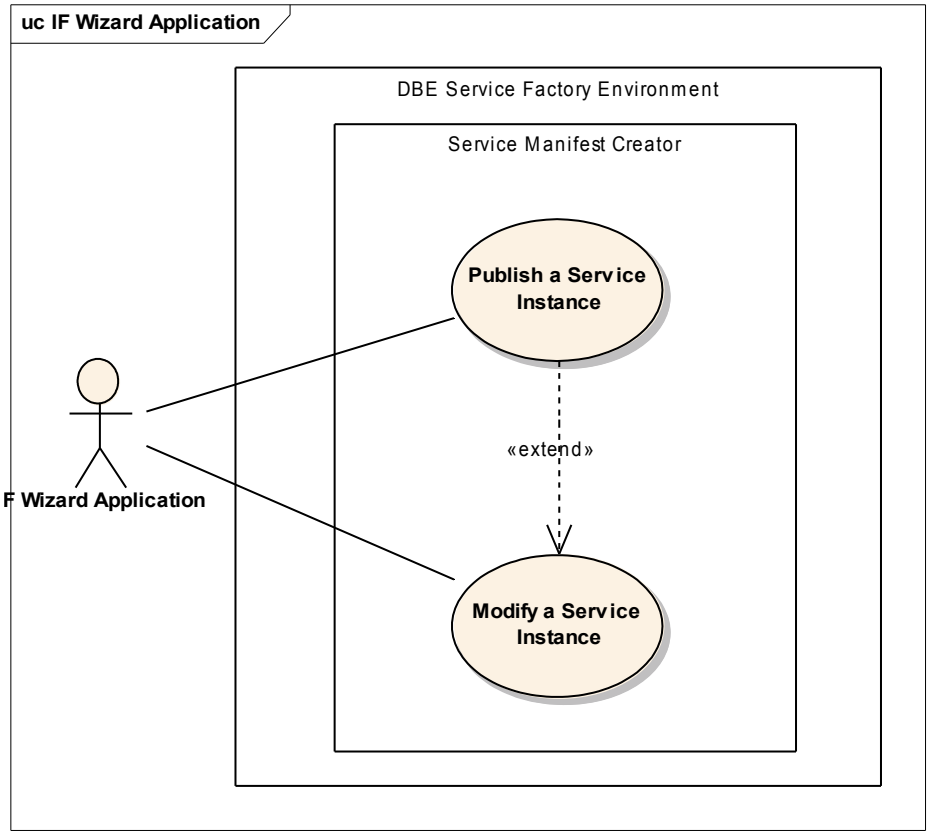

Figure 5.1.: IF Wizard Application Use Case Model Diagram

IF Wizard Application is also used when an update on a Service Instance is required. When a Service Provider wants to modify the data associated with a service already published through a wizard, he or she searches the service by the DBE Portal tool and then he or she executes the selected User Service instance, as already said in section 4.4-The Service Provider . Though the service executed by the Service Provider was the User Service, the actual SM update and its publication would be performed by the wizard application. When the user confirms the updates on a User Service instance, the IF Wizard Application uses the operation provided by the Service Manifest Creator to compose a new version of the SM and to publish the updated version of the service.

5.2 IF Wizard Class Diagram Model

Basically an SCS in DBE context is a service which is deployed as a proper application instead of a reference to an application. The Service Manifest (SM) of an SCS is a particular type of SM, beyond SMID[16] and other common SM data it contains also the actual application code of the service. When a DBE user executes a service that is an SCS, the application code is downloaded and executed in the user's client environment [D16.2SMSM]. Figure Figure 5.2.: Self Contained Service Class Diagram depicts the Class Diagram model of a generic DBE SCS.

16 Service Manifest Identification code

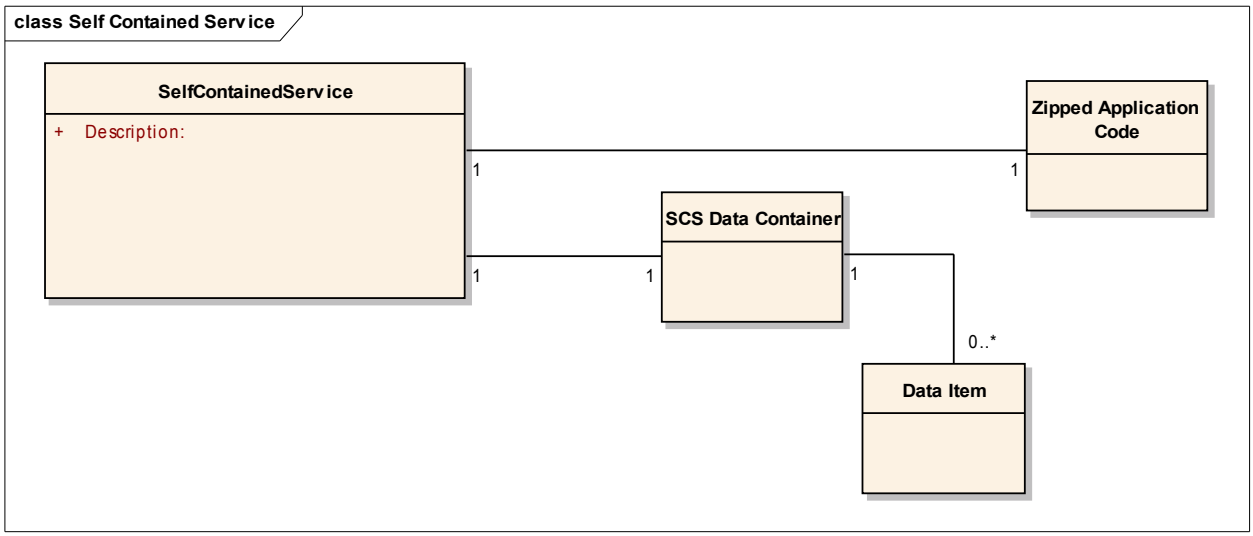

Figure 5.2.: Self Contained Service Class Diagram

It is clear from the model of figure 5.2 that every SCS has its own application code, that is stored in a compressed (zipped) file format to save disk space allocation. The “Zipped Application Code” class corresponds to the application that is executed when a DBE user executes an SCS instance. Beyond the application code, the SCS could also contain some data to be used by the service application. For instance, if an SCS was a service used to subscribe to an e-mail delivered newspaper, the “SCS Data Container” would be associated with the name of the newspaper, the frequency of the delivering, the cost of the subscription; therefore at execution time the service would be able to show all this data to the user.

An IF Wizard is used by a Service Provider to easily publish a User Service, which is a kind of SCS. The Class Diagram of the User Service is depicted in figure Figure 5.3.: User Service Class Diagram

A User Service is a particular type of “Self Contained Service”, for this reason diagram depicts “User Service” as a specialised “Self Contained Service” class. This means that every User Service instance has its own application code and in case also some extra data items associated. Beyond these entities the User Service stores the ID of the IF Wizard from which it has been generated: a User Service can't be constructed by a programmer, it is assumed to be automatically published by a wizard. The reason for associating the wizard ID to the SCS will be discussed in section 6.1-Service Provider Process Model and Use Case Model.Even an IF Wizard is a particular type of SCS and a comprehensive Class Diagram is depicted in figure Figure 5.4.: Self Contained Service and IF Wizard Class Diagram.

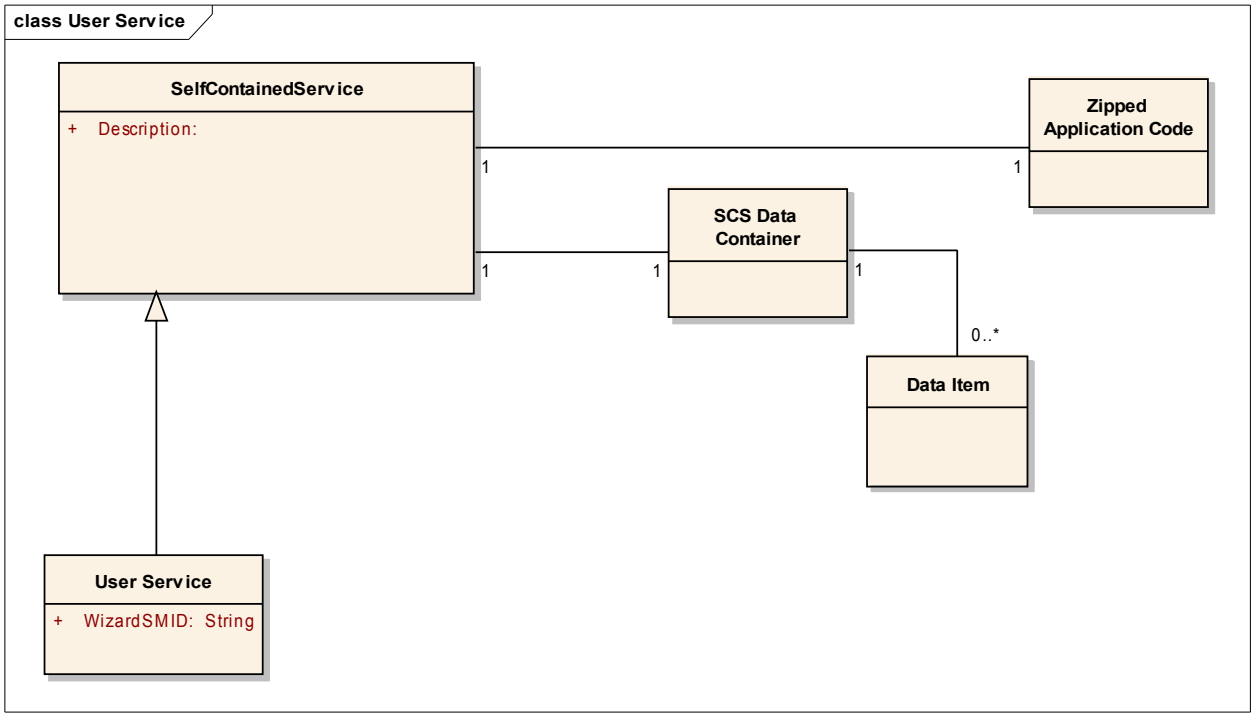

Figure 5.3.: User Service Class Diagram

An IF Wizard SCS aims at supporting a Service Provider in publishing an instance of a service, for this reason a wizard carries within it the information needed to build and publish a User Service. The IF Wizard is an SCS which contains an IF Wizard application, which is the computer program that interacts with the Service Provider user to let him or her specify the characteristics of the User Service. In the “SCS Data Container” of an IF Wizard there are the “User Service BML Model” and the “User Service BML Data Template”. The “User Service BML Data Template” is used by the wizard in order to make up the User Service's BML Data. Furthermore, in a particular IF Wizard there could be even one or more “Data Items”, representing data used at run time by the wizard application to accomplish some specific task (configuration data, internationalisation data, copyright information, etcetera). Beyond these data, the “SCS Data Container” must be associated also to the User Service application, which is the computer program that is executed when a Consumer decides to run a User Service. This requirement is described by the association between the “Wizard Data” class and the “Zipped Application Code” class in diagram Figure 5.4.: Self Contained Service and IF Wizard Class Diagram

There are no structural models to be applied to the “Data Item” and to the “Zipped Application Code” parts of the IF Wizard, it is up to the IF Wizard Supplier to decide how to structure the data items and the applications. Anyway, the IF Wizard Supplier has to bear in mind that, when the wizard application builds the User Service SM, all the expected rules and models must be complied by the SM components [D16.2SMSM] [D15.5BMLED] [D20.8BMLDATA].

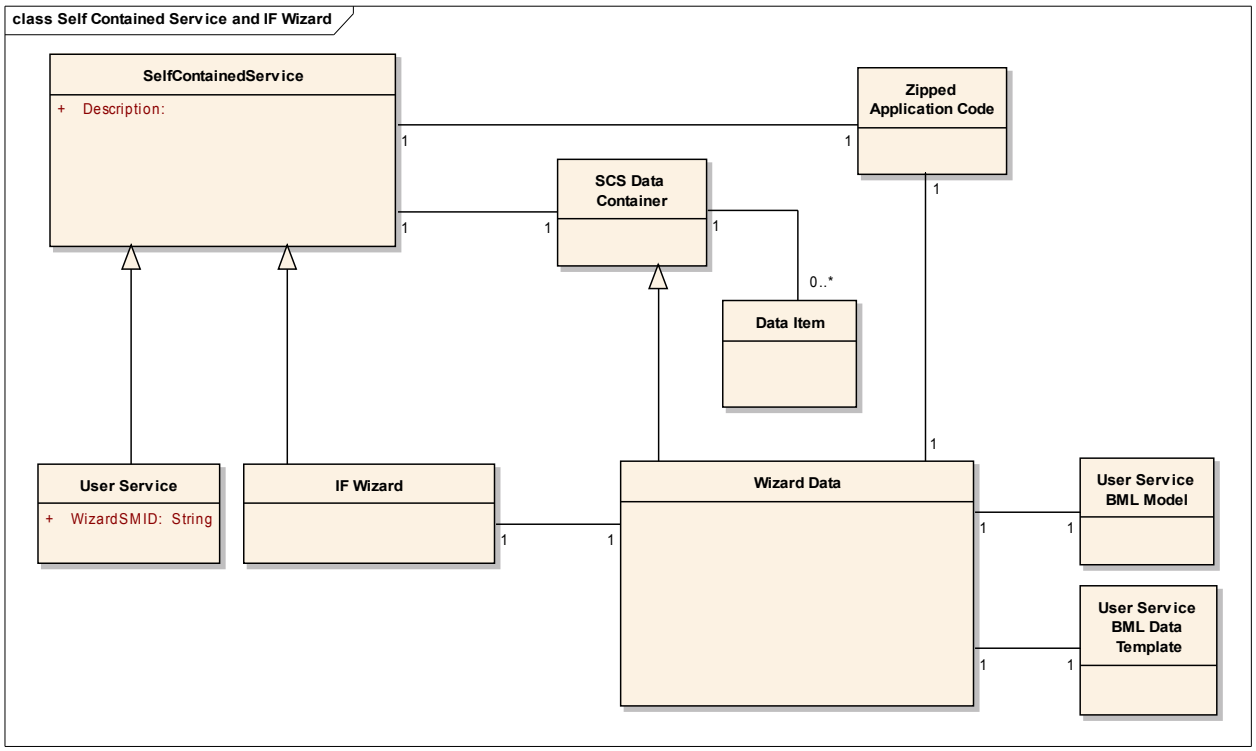

Figure 5.4.: Self Contained Service and IF Wizard Class Diagram

5.3 IF Wizard Supplier Process Model

The Use Case scenario for the IF Wizard Supplier in section 4.3-The IF Wizard Supplier Viewpoint depicted and explained the interactions between the IF Wizard Supplier and the DBE system. Anyway not all the work performed by the wizard programmer can be done interacting with DBE system, many activities are to be completed using other support tools. In Figure 5.5.: IF Wizard Supplier Process Model Diagram is represented the Process Model that comprehends all the main activities necessary to produce a wizard.

The activity flow proposed in model 5.5 is not mandatory, the Activity Diagram is so structured only to separate the environments in which every activity can be completed, for instance it is possible for the wizard programmer to start with BML Data Editor instead of BML Editor, or it is admissible to write the applications before the definition of the BML Model and Data. Before the SM composition anyway, every part of the IF Wizard must be completed, to be provided to the Service Manifest Creator.

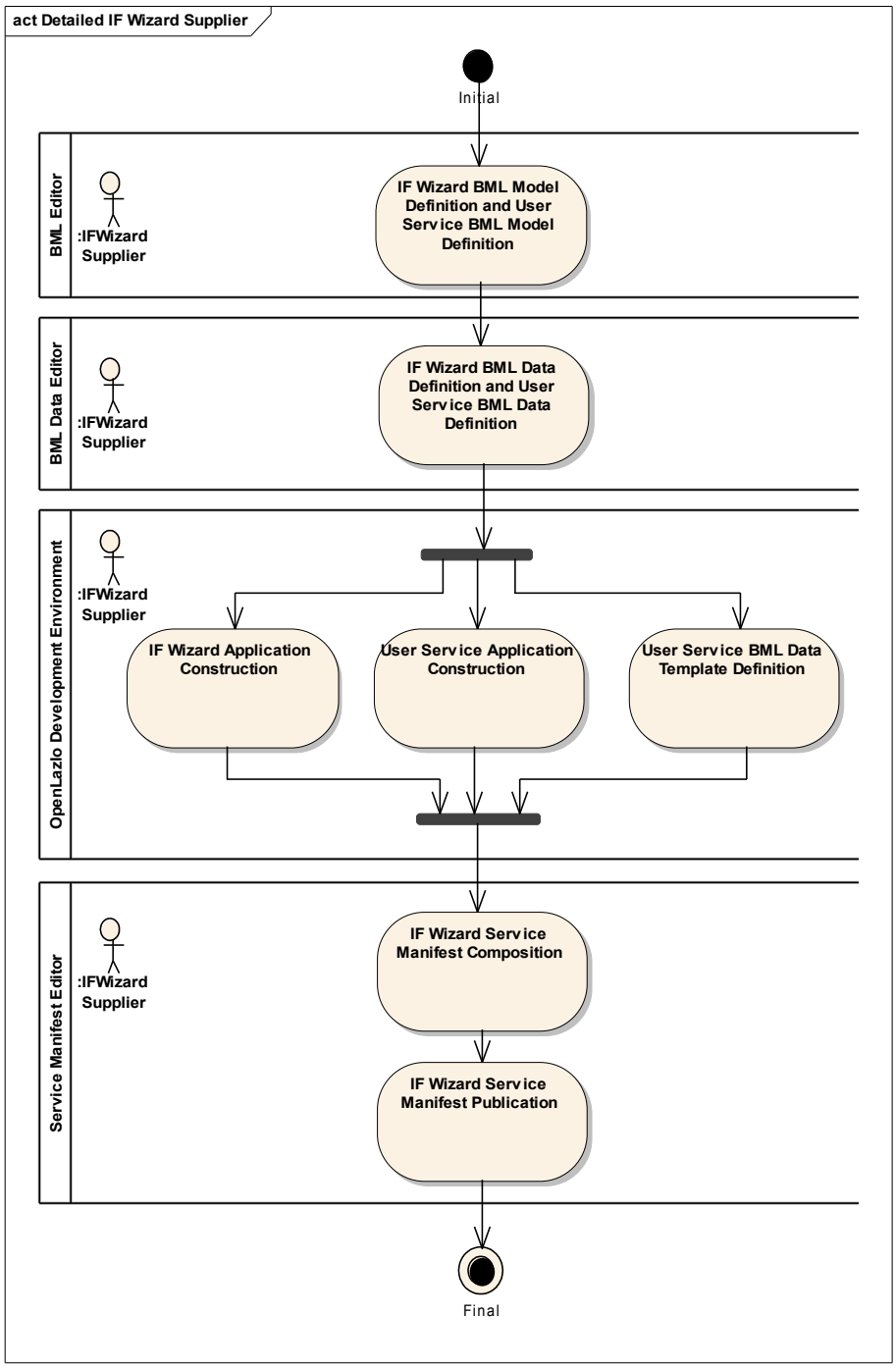

Figure 5.5.: IF Wizard Supplier Process Model Diagram

Using the BML Editor the wizard developer can define two BML models: the first is related to the wizard itself, and the second is related to the User Service.

In a similar manner the wizard programmer can use the BML Data Editor to construct the BML Data for the wizard and for the User Service. While the BML Data of the wizard is definitive after the completion of this activity, the User Service BML data is not. When the User Service is being published its BML Data must include the actual data specified

by the Service Provider during the gathering phase driven by the wizard, as explained in the previous chapters. Therefore, the User Service BML Data produced by the BML Data Editor can only be used as a base on which building the User Service BML Data Template. This latest activity must be completed then by the wizard programmer using tools from its own development environment, because a specific tool is not provided by DBE project.

Currently DBE provides support for IF Wizard Applications written in Open Laszlo language, for this reason the wizard programmer can develop the wizard application and the User Service application with the support of one of the available Open Laszlo programming environments.

At the end of the development phase, which in the model of Figure 5.5.: IF Wizard Supplier Process Model Diagram corresponds to the programming activities, the IF Wizard Supplier can proceed to the SM composition. This particular task can be completed using the DBE Service Manifest Editor tool: the SM Editor asks the wizard programmer to upload the service components – BML model, BML Data, SCS, etc – and finally it publishes the newly composed wizard in DBE.

As soon as the IF Wizard service has been published, a Service Provider could search for it through the DBE portal, to decide whether it is suitable or not for his or her purposes.

6 User Service Description

The User Service is the DBE Service Instance produced by the IF Wizard. Every wizard is constructed to be bounded to a particular class of services, for instance it might exists a wizard for the book selling business, another one for the Bed & Breakfast reservation services. The specific characteristics of a User Service depends upon which information the Service Provider supplies during the interactive definition process. The Consumer is who uses the User Service to obtain a service, for instance a Consumer could execute a specific service to buy a book from an on line selling company. In this chapter it is described in detail the Use Case scenarios and the processes related to the Service Provider and to the Consumer.

6.1 Service Provider Process Model and Use Case Model

The two main Use Cases related to a Service Provider Actor are the creation of a new User Service and the update of an existing and already published User Service. Figure 6.1: Service Provider Use Case Model Diagram depicts the Use Case model for the Service Provider.

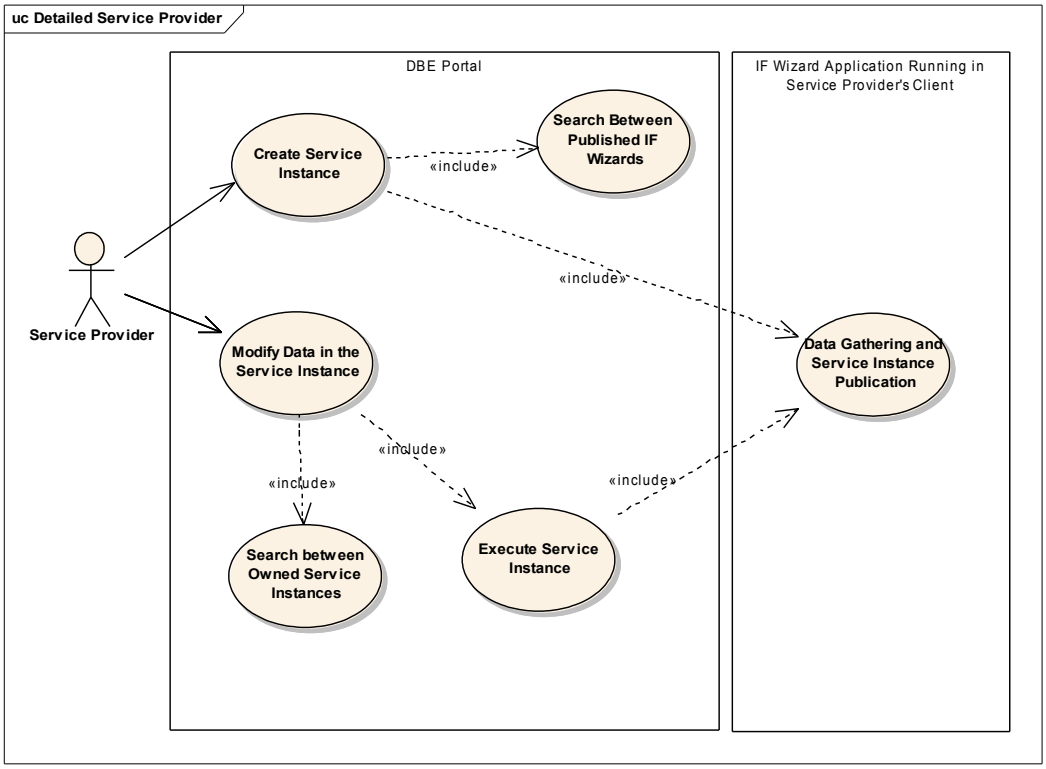

Figure 6.1: Service Provider Use Case Model Diagram

The diagram Figure 6.1: Service Provider Use Case Model Diagram highlights that the execution of the IF Wizard Application is supported by Service Provider's Client system, while the other interactions take place between the Service Provider and the DBE Portal

tools. There are two main searches offered by the DBE Portal to the Service Provider: the search among the available IF Wizards and the search among the owned User Service instances. The first type of search is useful to the Service Provider when he or she is going to decide which is the best wizard to be used to produce a User Service that fits his or her business; the second type of search helps the Service Provider finding a specific User Service instance that has to be updated, for example with up-to-date characteristics related to new products for sale. In the next two sections the main Service Provider Use Cases will be discussed.

6.1.1 User service Creation

In Figure 6.2: User Service Creation Activity Diagram is depicted the Activity Diagram that represents the creation model of a new User Service.

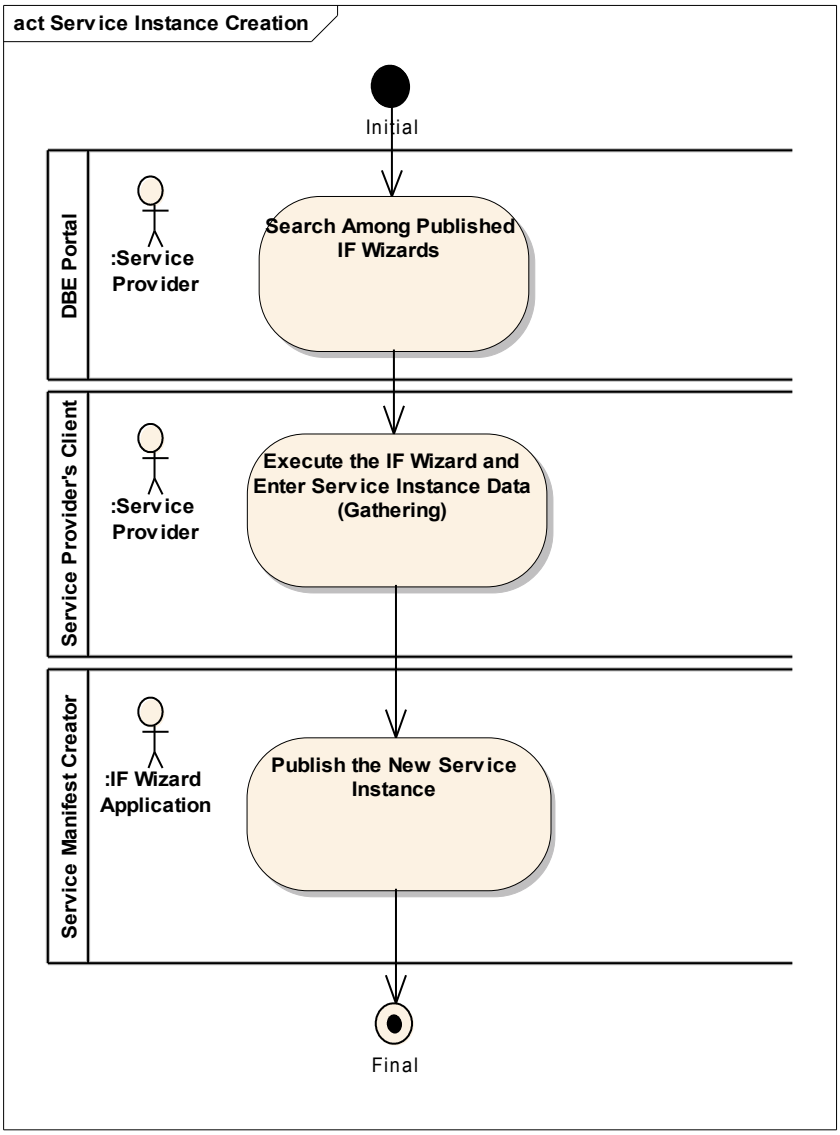

Figure 6.2: User Service Creation Activity Diagram

In the creation process of a new User Service the first activity is performed by the Service Provider, who searches among the published wizards to find a suitable one.
As an example scenario Carl Brown, which is a Book Shop owner, aims at publishing an on-line book selling service for his own business, to reach his goal he has to go through the following procedure:

- during the first step Carl needs to find an IF Wizard tailored for Book Selling. In order to find the right wizard Carl uses the DBE Portal to specify his search criteria and to browse among the matching results.
- Once chosen, the wizard is downloaded and executed within the Internet browser installed in Carl's personal computer. The Internet browser needs Macromedia plug-in to run the Open Laszlo application. Carl interacts with the wizard application to supply the information needed to define the User Service. The wizard application during the data gathering phase would ask Carl questions like

  "Which is the name of the Book Shop?",
  "Which categories of books are for sale?".

  These types of data are necessary for the wizard application to define the User Service BML Data [D20.8BMLDATA]. After the BML Data, the wizard application would gather information about the items that are purchasable through the User Service, then it would ask questions like

  "Which is the title of the book?",
  "Who is the author?",
  "How much does it cost?",
  "Do you want to insert another item?".

  Carl might answer with data like

  "The Adventures of Tom Sawyer",
  "Mark Twain",
  "8 €",
  "Yes" (to insert another book),
  "Wuthering Heights",
  "Emily Bronte",
  "10 €",
  "No" (to stop inserting books).

  These data shall be part of a data set used by the User Service to let a Consumer to place an order using the on line book shop.
- At the end of the data gathering activity, the process ends with the publication of the new User Service performed by the IF Wizard application. In this step the wizard application interacts with the DBE Service Manifest Creator to make up a Service Manifest, and finally to publish the new User Service instance.

6.1.2 User Service Update

In Figure 6.3.: User Service Update Activity Diagram is depicted the Activity Diagram model that represents the update process of an already published User Service.

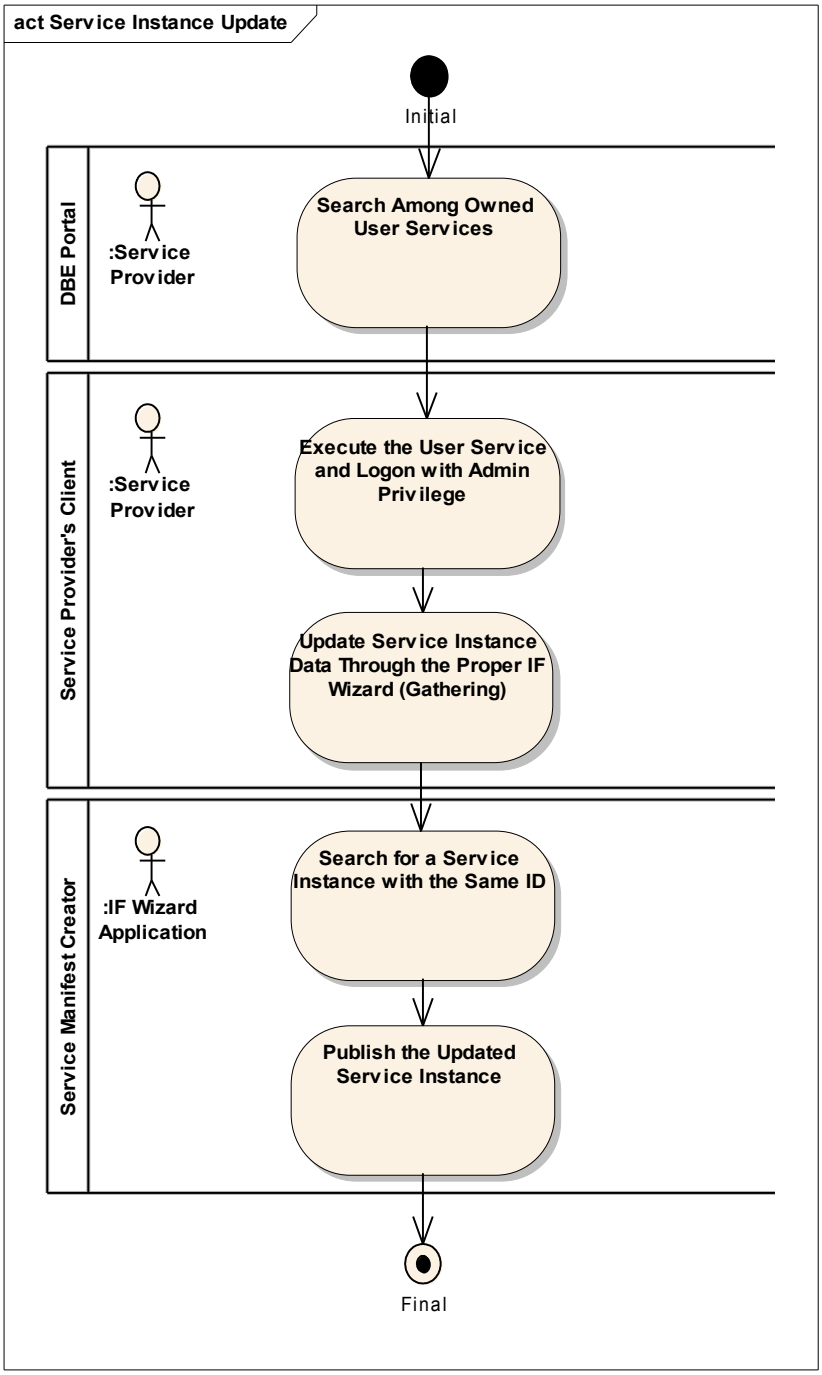

Figure 6.3.: User Service Update Activity Diagram

After a User Service publication, it might arise the need to update its data. For instance, in the case of a User Service created to sell book on-line, it would be necessary to add new items to the book catalogue, or to update the price of some items. In cases like this one the Service Provider needs a mechanism to access the already published service, to update it and to publish it again.
In the example scenario, Carl Brown, the Book Shop owner, has to go through the following procedure to update its on-line service:

- The first activity performed by the Carl is the search for the service instance to be updated. Carl uses the DBE Portal to search among the owned User Service instances, when the instance is located Carl executes it.
- The User Service application, as well as the wizard application, executes into the Service Provider Client's environment, that is in this case Carl's Internet browser, and allows the user to log on with administrator privilege. The log on as User Service administrator is performed by entering user name and password in the User Service interface.
- After Carl has logged in with administrator privileges, the User Service searches the IF Wizard from which it has been generated, this operation is possible because the unique identifier of the IF Wizard is stored into the User Service's data structure. Once located, the IF Wizard is automatically executed in a totally transparent way for the Carl.
- The wizard application, running within the Carl's Internet browser, let Carl to update the instance data. This interaction doesn't have a mandatory scheme to adhere to, it is up to the wizard programmer establishing the right user interface to accomplish the update task. The IF Wizard for the book selling business, would ask Carl to update the books catalogue, it would let him to insert a new item, to modify an existing one or to update the price of a book.
- At the end of data update activity, the wizard application would interact with the DBE Service Manifest Creator to locate the previous version of the wizard and finally to publish again the updated User Service instance.

6.2 Consumer Process Model

In figure Figure 6.4: Consumer Activity Diagram is depicted the Consumer process model.

The Consumer uses User Services like any other DBE service, on the first step of the process the Consumer searches and selects the desired service through DBE Portal, and then he or she executes it. As well as for the wizard application, currently the User Service can be only an Open Laszlo program and it runs into the Consumer's Client environment. The working mechanism is similar to that of the wizard application: once selected, the User Service is downloaded and executed into the Customer's Internet browser. While running, the User Service can interact with the user, without any constraint imposed to the type of interface or to the sequence of the operations. For instance, a User Service published to sell books on line would help the Consumer to place an order, selecting certain books, the quantities and the delivering address. At the end of the interaction with the user the User Service transmits the data to the Service Provider.

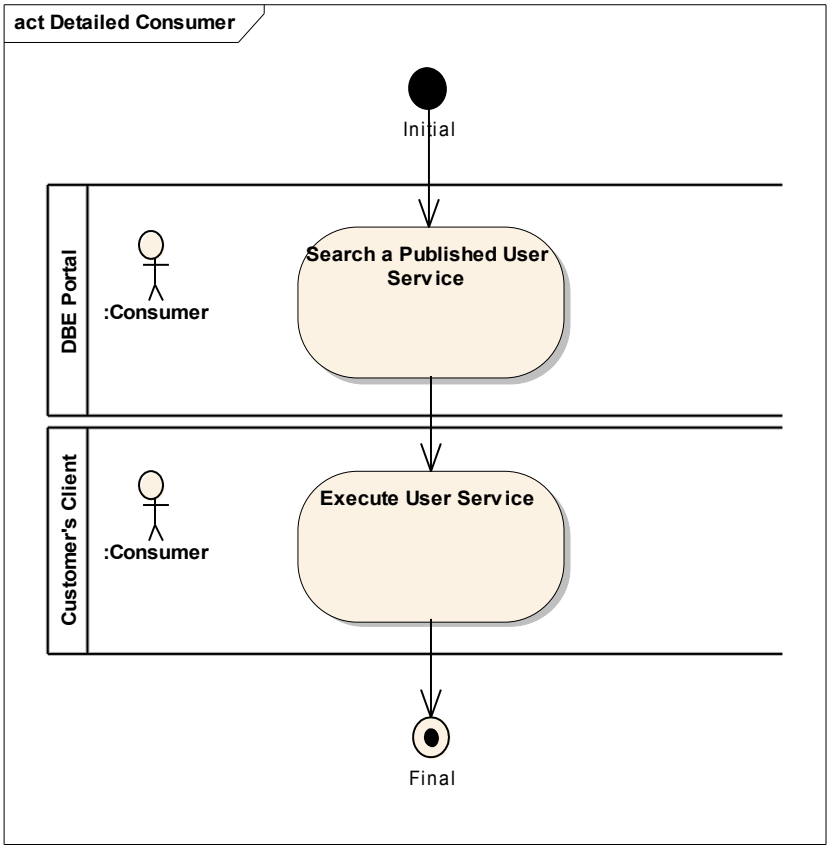

Figure 6.4: Consumer Activity Diagram

The limit with the current approach is that although an interaction between the Consumer and the Service Provider exists, such interaction does not occur synchronously (as well as for a deployed DBE Business Service) since the Provider has no Information Systems available to process the Consumer request, but it happens through asynchronous messages (e.g. through e-mail exchange).

As a consequence of their simple structure and behaviour, the SCSs are services that do not exploit all the potentialities offered by DBE system.

7 IF Infrastructure and Implementation notes

In order to integrate the Self Contained Service and to support the IF Wizard in the DBE structure, some of the existing components had been changed. In this chapter these modifications will be discussed, together with other implementation notes useful to understand DBE SCS management and IF Wizard development.

7.1 SM Creator division in two java projects

Some functionalities offered by the SM Editor, such as the publication of the SM, are also useful to the IF Wizard that must be able to access them. For such a reason it is now possible to distribute these core functionalities independently from the SM Editor and not only in bundle with the Client Side Servent[17]. The module which contains this Service Manifest related functionalities is called SM Core. Moreover, the implementation of the SCS required some modifications to the DBE Portal, for instance to enable the portal to execute also the SCS, or to expose the API offered by SM Core to publish Service Manifests. As a consequence of that, it has been decided to distribute the SM Core in bundle with DBE Portal, which now depends on the SM Core itself.

The code of the previous SM Editor has been split in two separate projects:

- SM Core
  contains all the classes corresponding to the core functionalities: service manifest xml parser/writer; constants needed to identify service manifest type; create, add, remove, update, a service manifest.

- SM Creator User Interface
  a separate project that contains the actual Eclipse plug-in and uses the functionalities of the “core” project. It just implement the UI of SM according to the MVC[18] paradigm.

7.1.1 Service Manifest Core API: SMHelper

SMCore is a Java project containing libraries to manage DBE Service Manifests. SMHelper is an SMCore Java class which supply an high level interface for SM operations: for instance it provides methods to get and set BML in a SM, to get and set SDL, to create an XMI file from a SM and vice versa.
In Figure 7.1.: SMHelper Class Diagram is reported the SMHelper class diagram.

17 Is the Servent when used as a Client
18 Mode View Control

Figure 7.1.: SMHelper Class Diagram

In Table 1: SMHelper Public Methods public methods of SMHelper Class are reported, for each method the return type and a note are specified, every method of the class acts on a SM instance.

Method	Type	Notes
SMHelper ()	public:	Costructor
setStringSM (*String*)	public: *void*	param: s [String - in]
handleError (*Throwable*)	public const: *void*	param: e [Throwable - in]
getStringSM ()	public: *String*	
getSm ()	public: *SM*	
setSm (*SM*)	public: *void*	param: sm [SM - in]
refreshSM ()	public: *void*	public void createSMFile();
getAncestorId ()	public const: *String*	@return Returns the ancestorId.
setAncestorId (*String*)	public const: *void*	param: ancestorId [String - in] The ancestorId to set.
getAvailability ()	public const: *String*	@return Returns the availability.
setAvailability (*String*)	public const: *void*	param: availability [String - in] The availability to set.
getBmlModel ()	public const: *String*	@return Returns the bml.
setBmlModel (*String*)	public const: *void*	param: bml [String - in] The bml to set.
getBmlData ()	public const: *String*	@return Returns the bml.
setBmlData (*String*)	public const: *void*	param: bml [String - in] The bml to set.
getBpel ()	public const: *String*	@return Returns the bpel.
setBpel (*String*)	public const: *void*	param: bpel [String - in] The bpel to set.
getDescription ()	public const: *String*	@return Returns the description.
setDescription (*String*)	public const: *void*	param: description [String - in] The description to set.
getIconUrl ()	public const: *String*	@return Returns the iconUrl.
setIconUrl (*String*)	public const: *void*	param: iconUrl [String - in] The iconUrl to set.
getId ()	public const: *String*	@return Returns the id.
setId (*String*)	public const: *void*	param: id [String - in] The id to set.
getLastChangeDate ()	public const: *java.util.Date*	@return Returns the lastChangeDate.
setLastChangeDate (*java.util.Date*)	public const: *void*	param: lastChangeDate [java.util.Date - in] The lastChangeDate to set.
getName ()	public const: *String*	@return Returns the name.
setName (*String*)	public const: *void*	param: name [String - in] The name to set.

getNoName ()	public const: *String*	@return Returns the noName.
setNoName (*String*)	public const: *void*	param: noName [String - in] The noName to set.
getPublicationDate ()	public const: *java.util.Date*	@return Returns the publicationDate.
setPublicationDate (*java.util.Date*)	public const: *void*	param: publicationDate [java.util.Date - in] The publicationDate to set.
getRegistrarId ()	public const: *String*	@return Returns the registrarId.
setRegistrarId (*String*)	public const: *void*	param: registrarId [String - in] The registrarId to set.
getRootId ()	public const: *String*	@return Returns the rootId.
setRootId (*String*)	public const: *void*	param: rootId [String - in] The rootId to set.
getSbvr ()	public const: *String*	@return Returns the sbvr.
setSbvr (*String*)	public const: *void*	param: sbvr [String - in] The sbvr to set.
getSdl ()	public const: *String*	@return Returns the sdl.
setSdl (*String*)	public const: *void*	param: sdl [String - in] The sdl to set.
getServiceType ()	public const: *String*	@return Returns the serviceType.
setServiceType (*String*)	public const: *void*	param: serviceType [String - in] The serviceType to set.
getSmVersion ()	public const: *String*	@return Returns the smVersion.
setSmVersion (*String*)	public const: *void*	param: smVersion [String - in] The smVersion to set.
getVersion ()	public const: *String*	@return Returns the version.
setVersion (*String*)	public const: *void*	param: version [String - in] The version to set.
getXmlns ()	public const: *String*	@return Returns the xmlns.
setXmlns (*String*)	public const: *void*	param: xmlns [String - in] The xmlns to set.
getSCSServiceData ()	public const: *String*	@return Returns the scs.
getServiceSCSMetaDescription ()	public const: *String*	@return Returns the scs.
getServiceSCSMetaName ()	public const: *String*	@return Returns the scs.
getServiceSCSMetaVersion ()	public const: *String*	@return Returns the scs.
getServiceSCSServiceZippedCodeMain ()	public const: *String*	@return Returns the scs.
getSCSServiceSCSServiceZippedCodeType ()	public const: *String*	@return Returns the scs.
getSCSServiceZippedCodeUiZipContents ()	public const: *String*	@return Returns the scs.
getSCSWizardData ()	public const: *String*	@return Returns the scs.
getSCSWizardSCSMetaDescription ()	public const: *String*	@return Returns the scs.
getSCSWizardSCSMetaName ()	public const: *String*	@return Returns the scs.
getSCSWizardSCSMetaVersion ()	public const: *String*	@return Returns the scs.
getSCSWizardSCSServiceBMLData ()	public const: *String*	@return Returns the scs.
getSCSWizardSCSServiceBMLModel ()	public const: *String*	@return Returns the scs.

getSCSWizardZippedCodeMain ()	public const: *String*	@return Returns the scs.
getSCSWizardZippedCodeType ()	public const: *String*	@return Returns the scs.
getSCSWizardZippedCodeUiZipContents ()	public const: *String*	@return Returns the scs.
setSCSServiceData (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setServiceSCSMetaDescription (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setServiceSCSMetaName (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setServiceSCSMetaVersion (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setServiceSCSServiceZippedCodeMain (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSServiceSCSServiceZippedCodeType (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSServiceZippedCodeUiZipContents (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardData (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardSCSMetaDescription (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardSCSMetaName (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardSCSMetaVersion (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardSCSServiceBMLData (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardSCSServiceBMLModel (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardZippedCodeMain (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardZippedCodeType (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.
setSCSWizardZippedCodeUiZipContents (*String*)	public const: *void*	param: s [String - in] @return Returns the scs.

Table 1: SMHelper Public Methods

For example, the following code:

```
SMHelper smH = new SMHelper();
smH.setStringSM(smString);//set the string (SM in format XML)
```

creates an instance smH of SMHelper, and then the call to setString() transforms an XMI smString to a SM object.

```
smH.getBmlModel();//get the BML Model as XML string
smH.getBmlData();//get the BML Data as XML string
```

In the example above, these two methods get BML Model and Data from SM currentrly handled by smH.

```
smH.setBmlModel(xmlBmlModel);
smH.setBmlData(xmlBmlData);
```

In the example above the two methods set BML Model and Data in the SM currently handled by smH.

```
String result = smH.getStringSM();
```

In the example above, the result string is an XMI string which represents the updated version of the SM.

7.2 Service Manifest structure modification

The SM has been changed to allow the SCS specific storage model, the necessary modifications to support SCSs have been implemented as “additions” to the previous SM structure.. Refers to [D16.2SMSM] for further information about the SM Structure.

7.3 SM Editor modification to support SCS IF Wizard

Even the SM Editor has been adapted to take into account the SM structure for the SCS. The SM Editor has been modified to manage the data required by the IF Wizard. It is worth remembering that only the IF Wizard, among the SCSs, can be created using the SM Editor. Although it is possible to develop an SCS by hand-coding, this task is usually accomplished using the IF Wizard that is specifically built for this purpose.

Refer to [D16.2SMSM] and [D18.5SMED] for further information.

7.4 DBE Portal adaptation

The DBE Portal has been modified to allow the execution of Self Contained Services. Differently from Business Services, the SCS User Interface is not provided by the

service proxy, but it is directly extracted from the SM. Even in this case, the modifications required have been very limited, demonstrating that DBE has been conceived to easily support its evolution in the next coming years.

For the regular non SCS services, the portal stores the user interfaces within its context and then serves the files when requested by the client browser. Refers to [D26.6PORTAL] for more information about this mechanism and the DBE Portal.

Currently only *Open Laszlo SCSs*[19] are supported by the Servent, but other SCS languages may be supported in the future, the DBE Portal will be modified consequently to be able to run the new SCS types.

The Open Laszlo (OL) language was the natural choice for the SCS because the Servent was already supporting the execution of OL programs. Every time an SCS is executed by a user, the contained OL code is extracted from the SM by the Servent, and it is "deployed" in a specific location (folder) in the DBE Portal context.

In the DBE context every SCS, when invoked, is deployed to a file system folder whose name follows the pattern:

`<SERVENT_DEPLOY_DIR>/<PORTAL_UI_CACHE_DIR>/scs/<SMID>`[20]

where <SERVENT_DEPLOY_DIR> is the folder in which the Servent is deployed, <PORTAL_UI_CACHE_DIR> is the folder that contains the current portal's cache, and <SMID> is the unique identifier of the SCS.

For every deployed SCS the corresponding invoking URL adheres to the pattern:

`http://<hostname><:port>/portal/scs/<SMID>/<OL_SERVICE_NAME>.lzx?smid="<WIZARD_SMID>"&smiduser="<USER_SERVICE_SMID>"`

where <hostname> and <port> correspond to the Servent, <SMID> is the unique identifier of the SCS, <OL_SERVICE_NAME> is the name of the Open Laszlo program that implements the service. This pattern is applied to invoke both an IF Wizard and a User Service, in every case two parameters are passed to the server: the <WIZARD_SMID> that is the unique ID of an IF Wizard, and the <USER_SERVICE_SMID> that is the unique ID of a User Service. To understand how these parameters are interpreted, it necessary to consider three possible scenarios of invocation of the SCS:

19 SCS written in Open Laszlo language

20 The file separator character (/) depends on the operative system on which the portal is deployed. All the examples in the current document use the slash (/) symbol as the separator character.

Scenario	***<SMID> (invoked service)***	***<WIZARD_SMID> (IF Wizard SIMD)***	***<USER_SERVICE_SMID> (User Service SMID)***
User Service Creation (an IF Wizard is invoked)	IF Wizard SMID	IF Wizard SMID	Empty
User Server Execution	User Service SMID	SMID of the IF Wizard by which the User Service has been generated	User Service SMID
User Service Update (the IF Wizard which has generated the User Service is invoked)	IF Wizard SMID	IF Wizard SMID	SMID of the User Service to be update

Table 2: SCS invocation scenarios

In figure Figure 7.2.: DBE Service Execution Process is depicted, as an Activity Diagram, the process of execution of a generic Open Laszlo based DBE service.

When a service is selected by a user through the DBE Portal the getUIURL() method is invoked, and the service's SMID is used to identify the kind of Service being executing. When executing a service the DBE Portal loads its entire Service Manifest using the Servent SR service, which locates the specific service manifest by its SMID.

If the service is an SCS then its SM is retrieved and the Application Code files (see Figure 5.4.: Self Contained Service and IF Wizard Class Diagram at page 28) are extracted from the SM and deployed to a separate folder, these files are the actual Open Laszlo application in the DBE Portal context. If a previous version of the Open Laszlo files are already stored in the portal context, they are overridden by those just extracted from the SM. Otherwise, if the requested service is not an SCS, then its proxy is retrieved and its User Interface (UI) is obtained from the proxy. Before deploying the service UI the portal checks to verify if the same version is already in the portal context, thus only if UI files are newer they are actually deployed in the portal context folder.

At the end of these activities a URL pointing to the main .lzx file of the Open Laszlo application is returned to the user. The patterns of the application file location and of the URL follow the rules already explained at the beginning of this section. If any .lzx file was already present in the DBE Portal application folder it would be replaced by the files extracted from the Service Manifest. This procedure is performed because it is an SM responsibility to contain the up to date version of the SCS. If the SCS was replaced by an updated version since the last service execution, then the new version extracted from the SM would replace the old one in the DBE Portal file system.

When the user gets the URL of the application from the DBE Portal, the Open Laszlo program is downloaded to the User's Client Environment and executed, as represented by the activity named “Open Laszlo Service Execution” in figure 7.2. The DBE Servent

includes an Open Laszlo Compiler[21] that compiles the .lzx files, therefore what is actually sent to the client Internet browser is a file that can be interpreted by the Macromedia Flash Player[22] plug-in. As a consequence of this serving mechanism, it is a requirement to use an Internet browser with Flash Player plug-in installed as the client platform from which DBE SCS OL services can be invoked.

If the OL service executed by the user client's browser is an SCS User Service (i.e. an instance of service created by an IF Wizard) then the process in figure 7.2 terminates, possibly with a data packet sent to a certain Internet service (for instance an e-mail message sent to the Service Provider, or a call to another on line service), otherwise, as the last operation, the OL program invokes a Remote Call to the DBE Portal, passing the <WIZARD_SMID> and <USER_SERVICE_SMID> parameters following the convention detailed at the beginning of current section.

21 See www.openlaszlo.org/architecture for further details about Open Laszlo technical architecture and behaviour.
22 See http://www.macromedia.com/software/flash/about/ for copyright and technical details about the plug in.

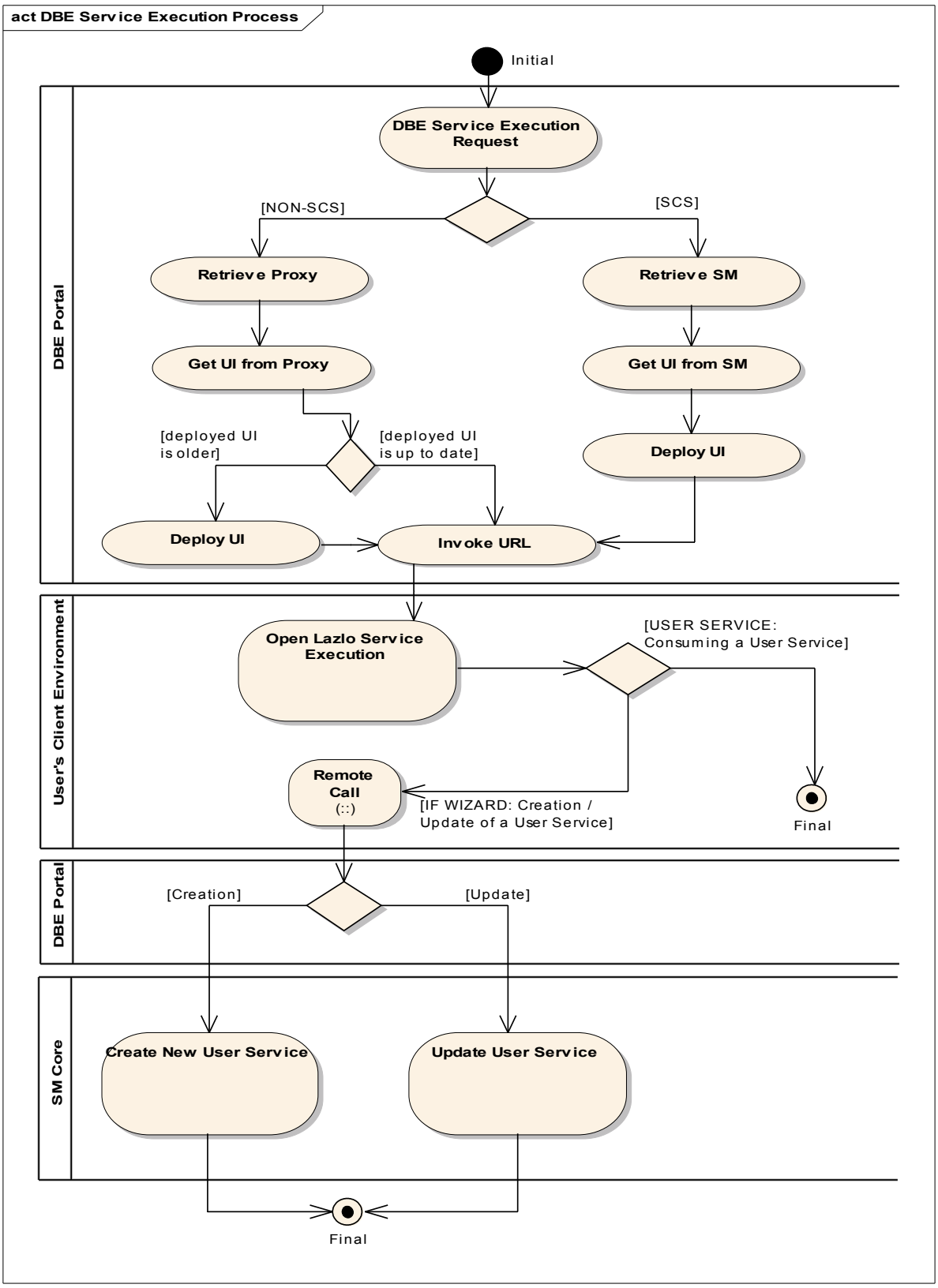

Figure 7.2.: DBE Service Execution Process

When the control is back to the DBE Portal, the ExecuteIF() method is invoked to parse the request: according to Table 2: SCS invocation scenarios and a creation or an update of a User Service is executed. After the User Service publication, either as a new or as an updated service, the process in Figure 7.2.: DBE Service Execution Process terminates.

The DBE Portal exposes a set of API corresponding to the SM Core functionalities, that the IF Wizard can call to perform SM related tasks. For instance the IF Wizard might call SM Core methods to pack and store a service manifest to the SR (see 7.1.1-Service Manifest Core API).

The DBE Portal takes care of the deployed service UIs and implements a clean up policy, i.e. if a certain amount of time has passed since the last usage of an application folder, then the corresponding application is considered expired and its files are deleted from the file system.

7.5 Security Issues

The SCS execution model doesn't includes the Security/Identity (refer to [M24.9IDT]) and Metering (refer to [D36.2ACC] and [D36.3ACC] for further information) mechanism used by distributed DBE Services, instead a simple Security mechanism (based upon user and password) has been implemented to support SCS services. This technical choice is not to be considered a problem because SCSs are suggested only for simple interactions between Consumer and Service Provider. It would have been possible to integrate the DBE Security and Metering services in the SCS execution mechanism, but the Identity was delivered quite late in the DBE Project release schedule, therefore the effort necessary to integrate the Identity and the Metering with SCSs was considered not worthy of the resulting security improvement.

Currently, in DBE project, a better security mechanism is not really mandatory for the execution of the SCS.

7.6 Open Laszlo Library

To ease the creation of IF Wizards and User Services, more than SM Core, an Open Laszlo Library has been developed. Refer to 8.7.1-ifLib Open Laszlo Library for a complete description of the library.

8 Tutorial: Creating an IF Wizard service in DBE

This chapter explains in detail, with code examples, which steps are to be completed to develop a DBE SCS service with the Open Laszlo language. The tutorial is intended for the IF Wizard Suppliers who are already familiar with the DBE environment and tools (BML Editor, BML Data Editor, Service Manifest Creator) and with the Open Laszlo language.

As already mentioned in chapter 5-IF Wizard Description to create a wizard an IF Wizard Supplier has to build both the wizard application and the User Service application. Either the wizard application and the User Service application must be currently implemented as Open Laszlo programs.

An IF Wizard aims at supporting a user – the Service Provider – in building his or her specific service and in publishing it in the DBE service catalogue. The end user of the service published by the wizard is the Consumer, that chooses and executes the User Service application to obtain a specific service. Definition, construction and deploy of an IF Wizard is responsibility of the IF Wizard Supplier.

To successfully build and deploy a new IF Wizard, the IF Wizard Supplier has to design and to code two applications:

- the wizard application, that enables the Service Provider to define and publish a new instance of the User Service;
- the User Service application, that executes when a Consumer wants to obtain a specific service.

In building these applications the IF Wizard Supplier has to bear in mind the structure of the conceptual service which is the target of the wizard, its BML definition and particularly its BML Data section, which has to be populated with actual data related to the service instance.

The wizard application itself has to perform some major tasks at run time:

- Gathering information related to User Service's BML Data;
- Inserting that information into User Service's BML Data Template;
- Building a User Service's Service Manifest;
- Publishing the User Service's Service Manifest.

The complete life cycle of the IF Wizard is depicted in Figure 8.1.: IF Wizard construction life cycle, after an introductory section about functionalities offered by the example wizard used in the tutorial, each one of the activity, task and step necessary to build a wizard will be discussed in the following sections.

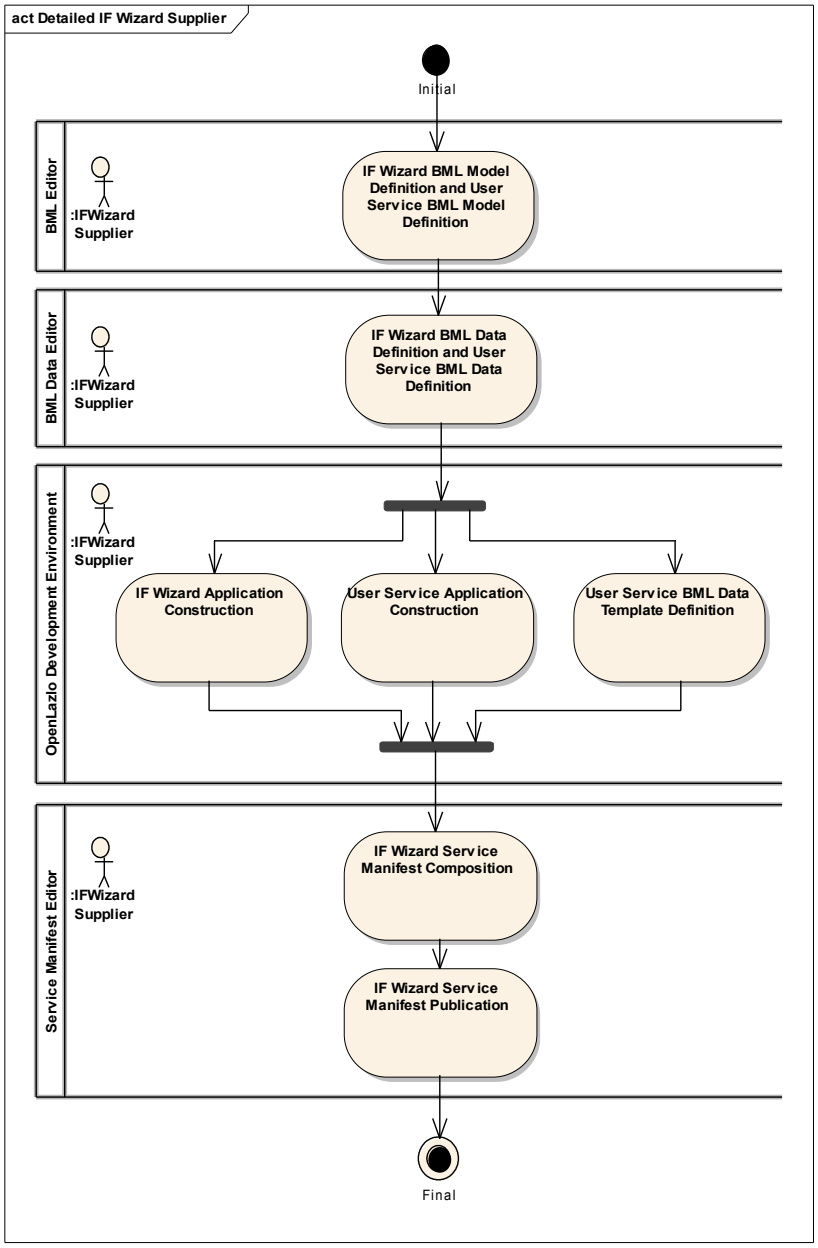

Figure 8.1.: IF Wizard construction life cycle

8.1 Bed & Breakfast Reservation Service Functionalities

To illustrate and to explain all the parts that compose the wizard application and the User Service, this tutorial presents a simple actual service: the Bed & Breakfast reservation service (B&B Service). The goal of the B&B Service wizard application is to permit a B&B owner, i.e. a Service Provider, to publish a User Service that contains a simple list of available rooms, along with their description and number of available beds. The User Service automatically produced by the wizard, once executed by the Consumer, it will look like Figure 8.2: B&B User Service GUI.

Figure 8.2: B&B User Service GUI

The B&B Service example is not intended to be a production level IF Wizard, it has been created to be the subject of this tutorial, hence the layout of the User Service GUI[23] is extremely simple, to let the reader concentrate on the internal structure instead on the presentation level. Even the Service Provider data (for instance, the name and the location of the B&B) useful to classify the service when published in DBE, are reduced to the minimum for the same reason. Only the B&B mailbox address is managed, because it has a specific meaning in this tutorial. A more rich example about using Open Laszlo for the presentation layer is given in the IF Wizard application, where more sophisticated UI components are used, like dialog boxes, tabbed forms, combo boxes, radio buttons and others.

> A production level IF Wizard is available as one of the examples which are automatically copied to the file system by the DBE Studio set up. The complete source code of the wizard is contained in the "ECLIPSE_HOME\plugins\org.dbe.studio.core.examples_0.3.x\LibraryIFExample" folder. The wizard is currently used by a pool of public libraries to publish services about cultural events organisation and scheduling. The complete source of the wizard is also directly available from the DBE project CVS repository at the URL: http://dbestudio.cvs.sourceforge.net/dbestudio/dbestudio/studio-core/studio-examples/src/conf/eclipse/LibraryIFExample

The Room List in the User Service GUI will present to the user the descriptions of all the rooms in the B&B structure, selecting one of the room in the list the user will read the room details. In Figure 8.2: B&B User Service GUI the user has selected the last room in the list, which is room number 2 on first floor, which has 2 double-beds inside and doesn't have enough space to add an extra single bed. Pressing the "Reserve" button, an automatic email message will be formatted on behalf of the Consumer, addressed to the B&B public mailbox and containing details about the preferred room. The Consumer will complete the email message adding the reservation dates and any other question or useful comment to help booking the room.

To publish the service, the Service Provider will execute the wizard application that will help in the definition of the service. The wizard graphical user interface looks like in Figure 8.3: B&B wizard application GUI.

23 Graphical User Interface

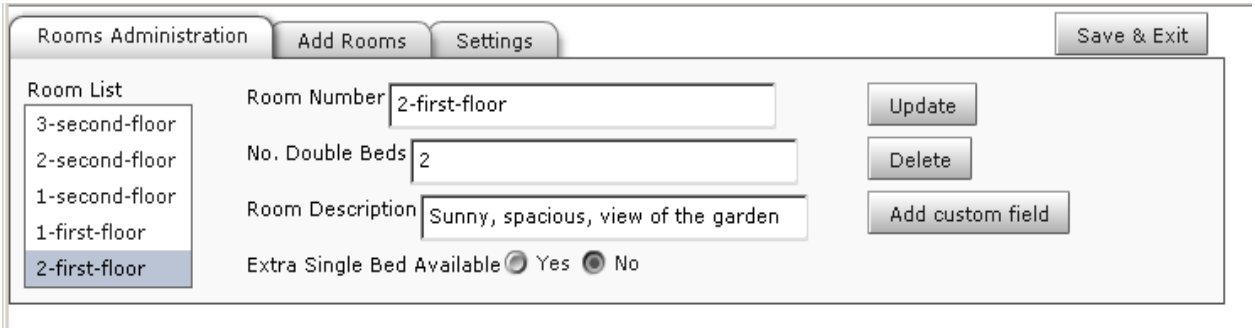

Figure 8.3: B&B wizard application GUI

The wizard GUI allows the Service Provider to edit the room list, i.e. to add, modify or delete rooms displayable in the User Service, and finally to modify the basic background colour of the User Service GUI. In the example of Figure 8.3: B&B wizard application GUI the Service Provider user has selected the last room in the room list, room details are then available to be modified or deleted. A special feature allows the user to add an extra attribute field to better describe the room characteristics (pressing button “Add custom field”), for instance the user could add a field named “air conditioning” to specify when an air conditioning system is available in one the room. Once terminated the data gathering the Service Provider user will end its work and will publish the User Service by pressing the “Save & Exit” button.

The complete source of the B&B Service example is available in 9-Annex A, there are four file related to this example: ifLib.lzx, wizardService.lzx, userService.lzx and data.lzx, which will be extensively described in the following sections. It is important to notice that the file names “wizardService.lzx” and “userService.lzx” are not mandatory, but they must be defined to be distinct in the actual services implementation.

8.2 IF Wizard BML Model

Concepts and DBE functionalities about BML Models are documented in [D15.5BMLED], for the aims of this tutorial a simple example of IF Wizard BML Model is defined. With BML Editor tool three attributes are used to describe the semantic of the wizard service: serviceName, wizardSupplier and isWizard flag, as shown in Figure 8.4.: IF Wizard BML Model Example. serviceName should be instantiated for example with a string like “Wizard for B&B Reservation Service”, wizardSupplier with the name of the IF Wizard Supplier and isWizard with the boolean value “true”, aiming at identifying the published service as a wizard.

Furthermore, selecting the “BedAndBreakfast_Wizard” box in the Semantic Description it could be specified “Accomodation” as the main service property.

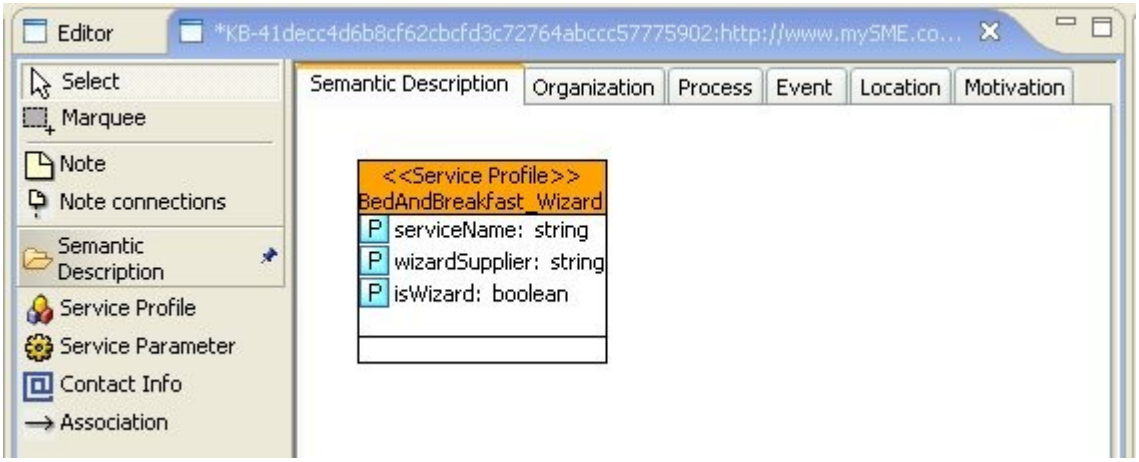

Figure 8.4.: IF Wizard BML Model Example

8.3 User Service BML Model

A BML Model describes a service from a semantic viewpoint and in a business perspective. In the IF Wizard scenario the User Service BML Model describes the User Service and not the IF Wizard itself, which is described instead by a BML contained in the IF Wizard SM. The User Service BML Model contained in the WizardData describes the service that will be created and it will be simply copied to the SM of the User Service. All the services created with the same IF Wizard will therefore have the same BML: indeed this BML describes the Conceptual Model of the category of User Services which are created and published by a specific wizard.

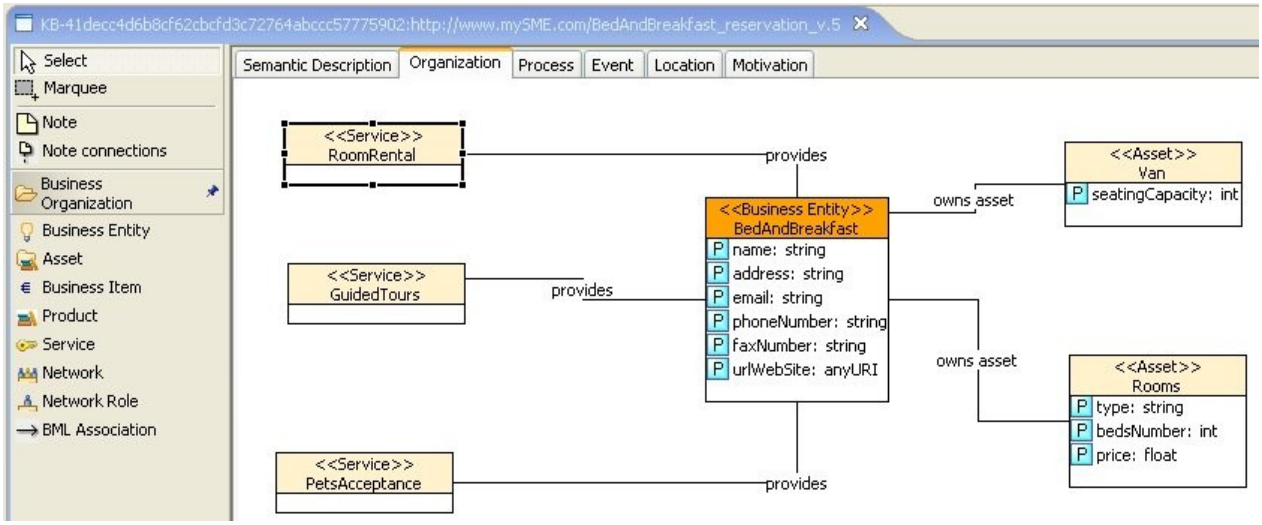

Figure 8.5.: User Service BML Model Organization Example

In Figure 8.5.: User Service BML Model Organization Example an example of User Service BML Model is shown. In the example the B&B would provide room rental service, pet acceptance and guided tours service. From the model it results clear that B&B has rooms available to let and a van to transport guests. Of course, in actual service instances, not all characteristics could be applicable.

8.4 IF Wizard BML Data definition

A complete description of concepts and DBE functionalities about BML Data definition could be found in [D20.8BMLDATA]. In the tutorial example the BML Model is defined as shown in Figure 8.6.: IF Wizard BML Data Editor Example. serviceName has been instantiated with the string “Bed&Breakfast wizard” as a description of the wizard characteristics, wizardSupplier has been instantiated with “Soluta.net” as the name of the IF Wizard Supplier and finally isWizard as been instantiated with the boolean value “true” to indicate that this service “is a” wizard.

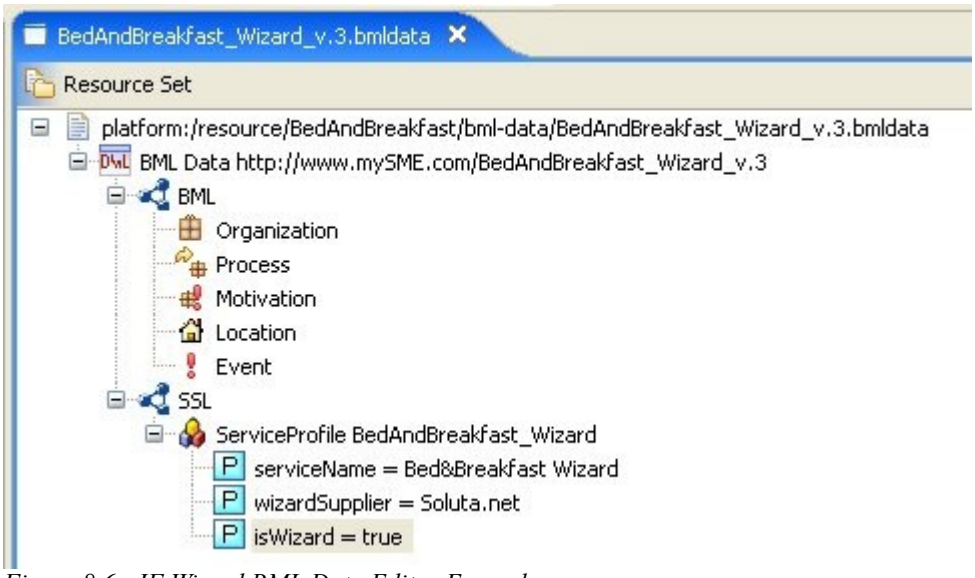

Figure 8.6.: IF Wizard BML Data Editor Example

8.5 User Service BML Data definition

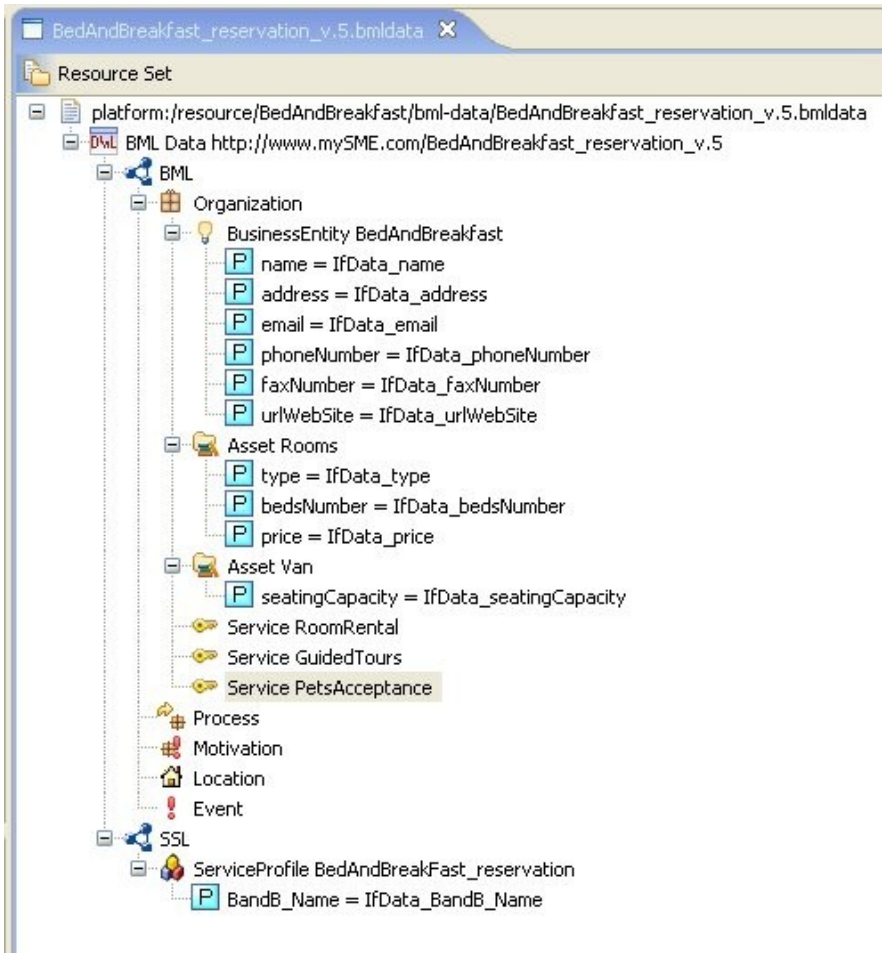

Figure 8.7.: User Service BML Data Template Example

As already mentioned the User Service BML Data definition requires a parametric XML to be produced. To obtain a BML Data template BML Editor can be used as depicted in Figure 8.7.: User Service BML Data Template Example. In this example all the attributes have been instantiated with tag which follow the pattern IFData_<name>. In the tutorial example "IfData_name", "IfData_address" and "IfData_email" will be saved in the bmlData section of “data” *Dataset*. During the IF Wizard publishing phase these place holders ("IfData_name", "IfData_address" and "IfData_email") will be substituted by the SMCore with the actual values gathered from the Service Provider by the wizard application.

8.6 IF Wizard Application Construction

8.6.1 IF Wizard data structure

In the B&B Service example the data gathering process is completely wired in the wizard application code, there is no other model to describe data structure or input sequence, and the service BML model is neither read nor interpreted. This

approach was followed to demonstrate the simplest way to build an IF Wizard, thus the overhead of a model definition was avoided. The main concern of the wizard application is to collect data to be used in the User Service, in this specific example collectable data are referred to the rooms available in the B&B structure. In Figure 8.8: XML snippet from “data.lxz” file is shown a snippet of XML taken from “data.lzx” file, which is the file contained in the “SCS Data Container” of the B&B IF Wizard. It is important to notice that “data.lzx” is the mandatory file name, both in the wizard and the User Service case.
The file named “data.lzx” actually is the “database” of the User Service at run time, it contains rooms details collected during the execution of the wizard application, and the data section reported in figure 8.8 is specific to a particular instance of

```
<rooms>
        <room>
                <id>1</id>
                <field label="Room Number" type="Text">1-first-floor</field>
                <field label="No. Double Beds" type="Text">1</field>
                <field label="Room Description" type="Text">Quiet, king size bed,
                lovely view of the hills</field>
                <field label="Extra Single Bed Available" type="Yes/No">Yes</field>
        </room>
        <room>
                <id>2</id>
                <field label="Room Number" type="Text">2-first-floor</field>
                <field label="No. Double Beds" type="Text">2</field>
                <field label="Room Description" type="Text">Sunny, spacious, view of
                the garden</field>
                <field label="Extra Single Bed Available" type="Yes/No">No</field>
        </room>
...
```

Figure 8.8: XML snippet from “data.lxz” file

User Service. When B&B IF Wizard is published, the file “data.lzx” contains only the basic set of data not referred to a particular instance of the service. In Figure 8.9: “data.lxz” file as packaged to the IF Wizard data structure is reported the file “data.lzx” as packaged within the IF Wizard data structure when the wizard is published in DBE.
The structure of the XML file follows the rules of the OL “dataset” definition, all XML elements are contained in the external tag “dataset” which is named “data”. This format ease the manipulation of XML data by the OL program, as it will be seen in the source code of the wizard application where objects like “datapointer” and primitives like setXPath(), addNode() and setNodeText() are used to read and write XML data structures (these data types and primitives are specific to the OL programming language). In the B&B Service example data gathered during execution of the wizard application are not stored as IF Wizard BML Data, all the data necessary to the wizard application and to the User Service are stored in the file “data.lzx”. Even this design approach has been followed to simplify the construction of both IF Wizard and User Service.

```
<dataset name="data">
        <admin>
                <user></user>
                <pwd></pwd>
        </admin>
        <infoMail></infoMail>
        <rooms>
        </rooms>
        <roomType>
                <id />
                <field label="Room Number" type="Text" />
                <field label="No. Double Beds" type="Text" />
                <field label="Room Description" type="Text" />
                <field label="Extra Single Bed Available" type="Yes/No" />
        </roomType>
        <settings>
                <layout>Green</layout>
        </settings>
        <bmlData>
                <di></di>
        </bmlData>
</dataset>
```

Figure 8.9: "data.lxz" file as packaged to the IF Wizard data structure

As shown in Figure 8.9: "data.lxz" file as packaged to the IF Wizard data structure the default structure of data attributes related to a certain room is wired in the <roomtype>[24] XML element: "Room Number" defines the text string which identifies the room, "No. Double Beds" contains the number of double beds available in each room, "Room Description" is a free text that reports main characteristics of a room, while "Extra Single Bed Available" is a flag based value (Yes/No) that indicates the possibility to add an extra single bed for every room.
Another XML element of file "data.lzx" is <infoMail>, which is there to store the Service Provider email address to which reservation request messages are sent. The element named <settings> traces which background colour is to be used in the User Service GUI. The <admin> XML element is used to store username and password used by the Service Provider to execute the wizard to update a User Service instance data.
As well as for the rooms instance data, these last mentioned XML elements are completely specified by the Service Provider during the wizard data gathering phase.
The XML element named <bmlData> is contained in the file "data.lzx", it is used in the B&B Service wizard and User Service example to contain name, address and email address of the B&B.

24 In the tutorial the XML elements will be represented between angle brackets, like in the case of <roomtype>.

The example data stored within “data.lxz” file are used in this context only for the tutorial purpose, in case of an actual service there could be a far greater number of entries coded in the XML dataset to be used by the User Service.

8.6.2 Wizard Application Structure

The wizard application is the OL program, contained in the IF Wizard SCS, which enables the Service Provider to input necessary data to publish a User Service instance. Every wizard application is designed to produce a specific class of User Services, the current tutorial is about a wizard application designed to publish User Services which are instances of a B&B Reservation Service.
This tutorial presents, discusses and comments a specific example of IF Wizard application and User Service application, aiming at giving a practical help to the IF Wizard Supplier who would produce other kinds of IF Wizards. For tutorial purposes Bed & Breakfast business domain has been chosen because of the simplicity of the context, in which a Service Provider builds a list of available rooms and publishes a User Service to make reservation requests. The example is not intended to be a “production level” service, it has been developed to be tutorial matter, and it has been kept very simple accordingly. Nevertheless these applications (wizard application and User Service application) could be used as a base on which a complete production service could be developed with a limited effort. All the functionalities of the applications are introduced and discussed in this tutorial.

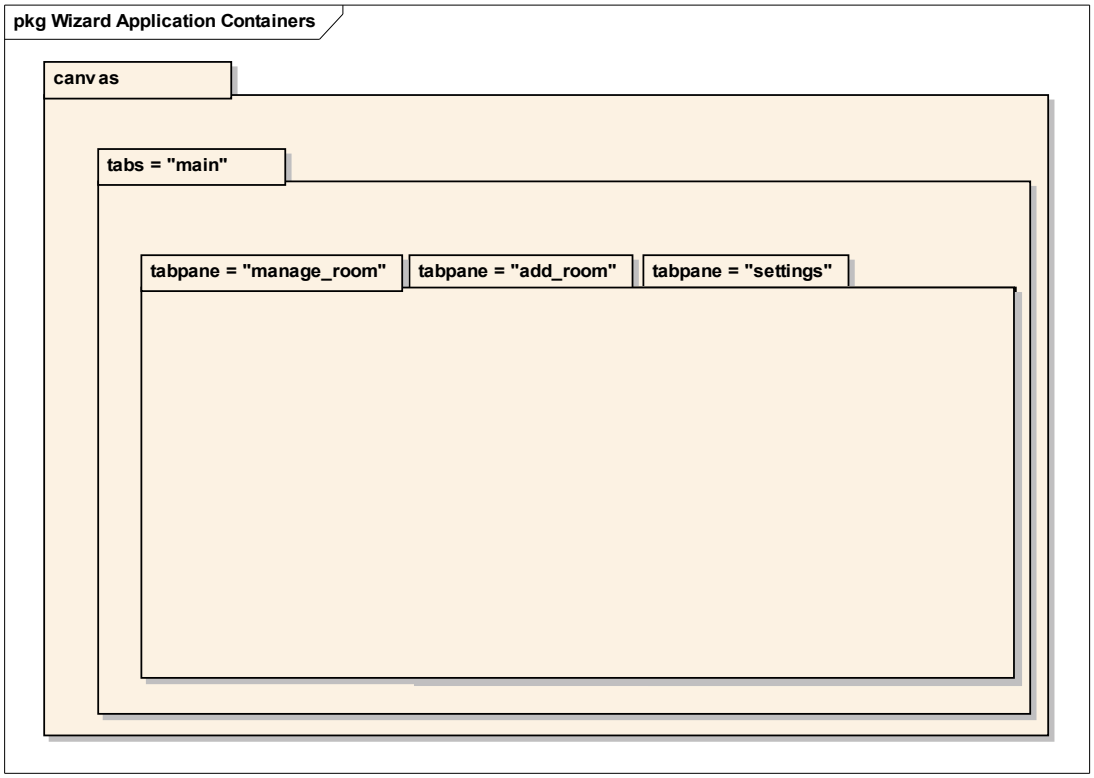

Figure 8.10.: Wizard Application Containers

At run time the GUI of the B&B wizard application looks like Figure 8.3: B&B wizard application GUI.
In Figure 8.10.: Wizard Application Containers is draft the hierarchy of the main containers defined and used by the OL program “wizardService.lzx”.
The most external element is <canvas>, which is the container for all views within an OL application. In the B&B example <canvas> element defines a visible area for the wizard application and by default enables the use of the interactive debugging window. In a production version of the same application the debug flag should be assigned to “false”. The canvas is the direct container of the “Save & Exit” button (see Figure 8.3: B&B wizard application GUI) and of the <tabs> visual element named “main”. Within “main” tabs container are defined three single <tabpane>: “manage_room”, “add_room” and “settings”. Each of these <tabpane> is a container for other functional-related UI elements. All code and XML examples in the current tutorial are referred to a specific User Service instance, the component files of which are reported in 9-Annex A.
The “manage_room” tabpane is depicted in Figure 8.3: B&B wizard application GUI and it is the first view shown to the user when the wizard application starts; it contains a list of the B&B available rooms, selecting one of the elements in the room list, its details are presented in the middle form where is also possible to modify values. When a specific room is selected within the list, “Update” and “Delete” buttons enables, giving the user the possibility to modify or delete the item. “Add custom field” button is always enabled to permit the definition of a new room attribute, for example a new field named “air conditioning”, which specifies which rooms have an air conditioning system installed. The always available “Save

& Exit" button lets the user to end his session and to save every modification made to the instance data.

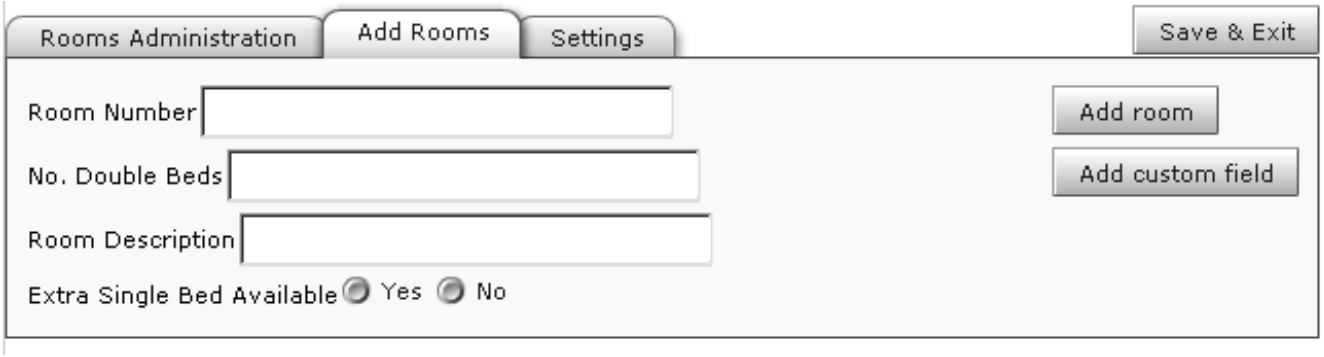

Figure 8.11.: Add rooms tabpane

The "add_room" tabpane is depicted in Figure 8.11.: Add rooms tabpane.
Using the "add_room" tabpane the Service Provider user has the opportunity to add new rooms items to the User Service instance data. For instance the user could specifies a new room numbered "4-second-floor" described as "Green room, view of the woods", furnished with 1 double bed, with the possibility to add an extra single bed. To complete its task the user could press "Add room" button, and then

Figure 8.12.: Settings tabpane

he or she could go back to room administration tabpane before confirming the modifications.
The "setting" tabpane is very simple and is depicted in Figure 8.12.: Settings tabpane. Using the options proposed by the combo box in the "settings" tabpane the Service Provider user could change the main background colour of the User Service instance. For instance the user could choose "Green" and confirm his or her choice pressing "Save & Exit" button. All available colour options are coded in "data.lzx" XML file (see annex 9-Annex A).

8.6.3 Definition of Wizard Application Behaviour

The complete source of the wizard application written in OL language is reported in annex 9-Annex A. In the following sections the source files will implicitly be

referenced when an OL feature, an OL statement, an application variable or an application object is mentioned.

8.6.4 Application Startup

The complete source of the application wizard is saved in the file “wizardService.lzx”.
The wizard application is invoked, as any other Internet resource, via a specific URL. The wizard can be executed to complete two tasks: to create a new User Service instance or to modify an already published User Service instance (see 5.1-IF Wizard Application Use Case Model).
Figure 8.13.: Wizard Application Startup depicts the activity diagram of the wizard application startup. When the wizard is executed to create a new service instance, no parameters are passed in the invoking URL, so the canvas-scope variable named "smIdUserService" has no value assigned. In this case the wizard requires the user to define a Username and a Password, used to grant future accesses to the User Service instance produced. To accomplish this task the wizard application set the “adminDefinitionForm” “visible” property to “true”.

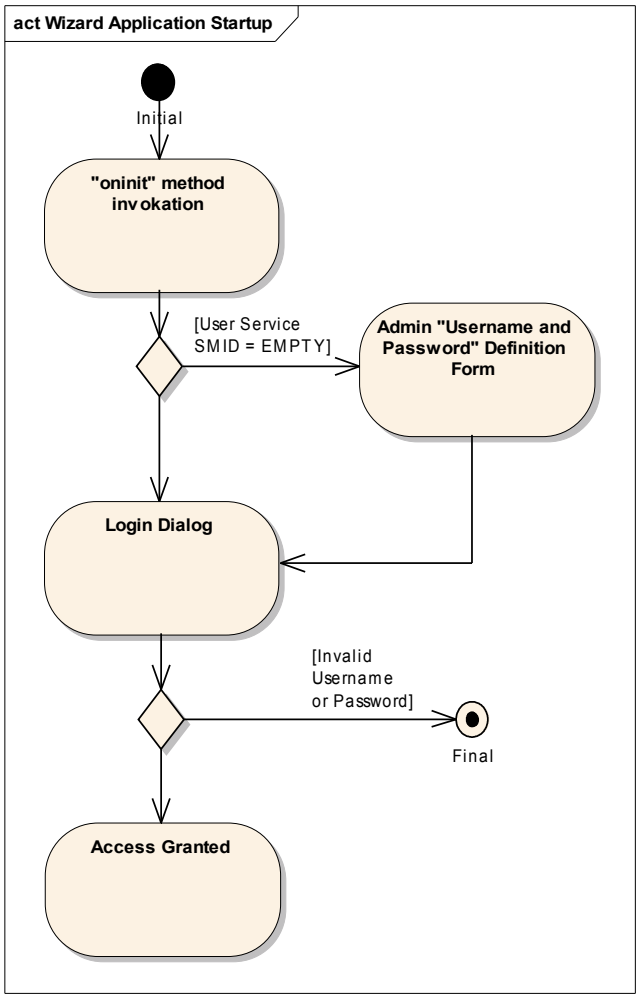

Figure 8.13.: Wizard Application Startup

When the “adminUserDefinitionForm” is displayed, the user must specify a Username, a Password, the B&B name, the B&B address and an email address to which messages automatically generated by the User Service execution are sent. If the user omits one of the required values an error message box is displayed, i.e. an OL alert object named “alertError” is opened by the call `alertError.open()`. After the successful execution of the administrator user definition form, the login dialog box is opened through the call `dlgLogin.animOpen.doStart()`. The user has then the opportunity to enter username and password to gain access to the application.
When the wizard is executed to update an already published User Service, then the “userServiceSMID” URL parameter is assigned to an existing SMID, and the flow of the application startup goes directly to the login form.

At wizard application startup, the GUI focus is given to the “manage_room” tabpane, and the method which corresponds to its “onselected” event is fired.
The execution of the “onselected” event method completes the following tasks: the room list of the “manage_room” tabpane is populated, this first task is performed

by `doCreateRoomList()` call. In its initial condition the tabpane doesn't have a selected room, thus the “room” form in the middle of the “manage_room” tabpane is empty and the “Update” and “Delete” buttons are disabled.

The wizard application uses two local OL datasets, named “list” and “data”. When the application starts the “data” dataset is initialised with all elements contained in the “data.lzx” XML file. Instead “list” dataset is a in-memory-only dataset, used to contain a simple list of data displayed by “manage_room” tabpane. When the Service Provider user adds, modifies or deletes items during the gathering phase, the “data” dataset is updated to reflect the modifications made by the user. Every time a refresh of the room list in “manage_room” tabpane is required, the `doCreateRoomList()` method rebuilds the “list” dataset from scratch, reading actual value from “data” dataset. Data contained into the “list” dataset is displayed by the tabpane because an Open Laszlo datapath has been defined to show data items within a list box UI view (datapath between OL textlistitem named "datasetBoundedRoomList" and “list” dataset).

8.6.5 OL Canvas

The wizard application, as any other OL application, defines the *Canvas* element as a container for all the UI elements. In Illustration 8.14.: Wizard OL Application canvas component is depicted the component diagram related to wizard application *Canvas*.

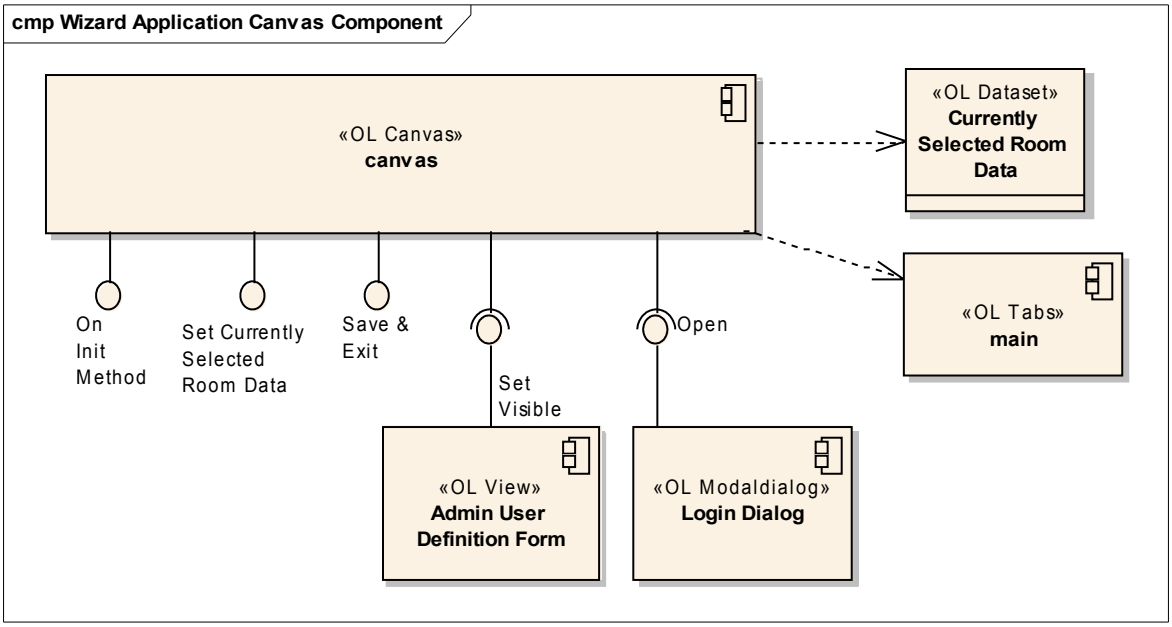

Illustration 8.14.: Wizard OL Application canvas component

The “On Init Method” provided interface actually exposes an OL method invoked when the application starts, to locate the body definition of this methods look the wizard application source for its definition: `<method event="oninit"> ... </method>`. This interface is required by the OL runtime, that automatically consumes it during application startup. As already mentioned in the previous

section, this method checks if the User Service SMID has been passed as a parameter to the application - `if (smIdUserService) ...` - and consequently passes control either to “Login Dialog” or to “Admin User Definition Form”.
These latter UI components offers their interfaces to *Canvas*, which uses them through

```
dlgLogin.open();
```

or

```
adminUserDefinitionForm.setAttribute("visible",true);
```

“Set Currently Selected Room Data” is an interface offered to the UI components contained within *Canvas* and it is used to display the details of a room. When an inner component need to show every attribute value associated with a specific room, it loads the room attributes to the “Currently Selected Room Data (OL Dataset)” and then uses the “Set Currently Selected Room Data” provided interface, that displays room details in a form placed inside an OL view. Room details are displayed because a datapath has been defined between “Currently Selected Room Data (OL Dataset)”, implemented as a OL *Dataset* named "currentlySelectedRoomData", and a UI view:

```
<datapath xpath="currentlySelectedRoomData:/roomType/*" />
```

Application *Canvas* depends also from the inner defined OL views, that have the responsibility to manage the interactions with the user. As shown in Figure 8.10.: Wizard Application Containers, there are three main OL *Tabpanes*, compounded into a single OL *Tab* object. The major functionalities offered by the OL *Tabpanes* are room management (modify, add, delete rooms) and application settings (User Service background colour).
When “Admin User Definition Form” is made visible during execution of “On Init Method”, the UI in Figure 8.15.: Admin User Definition UI is shown to the user. The form defines six editable fields: username, password, B&B name, B&B address and B&B E-Mail info. The user should enter all the values and then should press the “Create User” UI button, which fires the appropriate event:

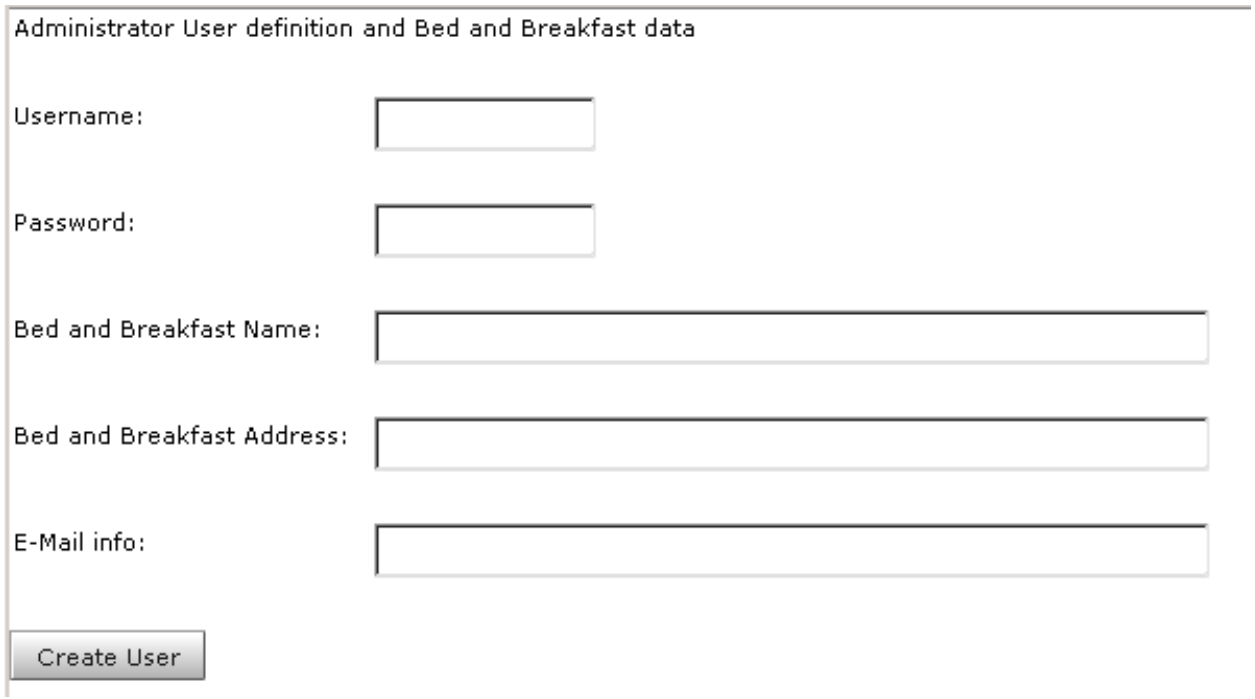

Figure 8.15.: Admin User Definition UI

```
<button text="Create User" onclick="parent.addUser()" />
```

The “addUser” OL method ensures that all the six values are entered by the user and then saves these values to the “data” OL *Dataset:* username, password and email are saved within elements named “admin” and “infoMail”, while B&B name, B&B address and email are saved within bmlData element, using functions provided by ifLib library (see 8.7.1-ifLib Open Laszlo Library on page 75); email address is saved twice to give an example of wizard application data manipulation flexibility.

```
tmp.setXPath("data:/admin");
tmp.setXPath("data:/infoMail");
```

Data saving in bmlData element is performed by the following code:

```
var utils = new IfUtility;
utils.setBmlDataAttribute("IfData_name", bbname_view.bbname.getText());
utils.setBmlDataAttribute("IfData_address",
      bbaddress_view.bbaddress.getText());
utils.setBmlDataAttribute("IfData_email",
      bbmail_view.bbmail.getText());
```

the names "IfData_name", "IfData_address" and "IfData_email" are chosen in accordance with the User Service BML Data template definition (see 8.8-User Service BML Data Template definition).

As shown in Figure 8.13.: Wizard Application Startup before accessing the wizard application the admin user (i.e. the Service Provider user) must enter valid Username and Password, using the dialog of Figure 8.16.: Login Dialog.

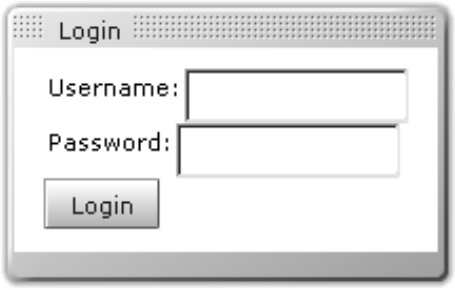

Figure 8.16.: Login Dialog

When “Login” UI Button is pressed, “addUser” method is invoked:

```
<button text="Create User" onclick="parent.addUser()" />
```

If Username and Password match with those stored in “data” OL *Dataset* in element “data:/admin”, then the user is granted to continue with the application.

“Save & Exit” interface is offered by canvas to be used by the method associated with the OL *Button*. When, by the end of its activity with the wizard application, the user presses the “Save & Exit” UI button, the corresponding method is invoked:

```
<button name="btnExit"...> <method event="onclick">...
```

The detailed behaviour of this method is discussed in 8.6.10-Save and Exit section.

8.6.6 Updating or Deleting a Room

The “manage_room” OL *Tabpane* is the application element which allows the user to do update or delete a room from the list of the rooms availables at the Bed & Breakfast. The user can also add a new attribute field to better describe the characteristics of every room, for instance the user could add the field “air conditioning”, with Yes and No as possible values, to describe which of the rooms have an air conditioning system installed. Figure 8.17.: manage_room” OL component represents “manage_room” in a component diagram. In the OL application this *Tabpane* is defined as:

```
<tabpane name="manage_room" text="Rooms Administration">
```

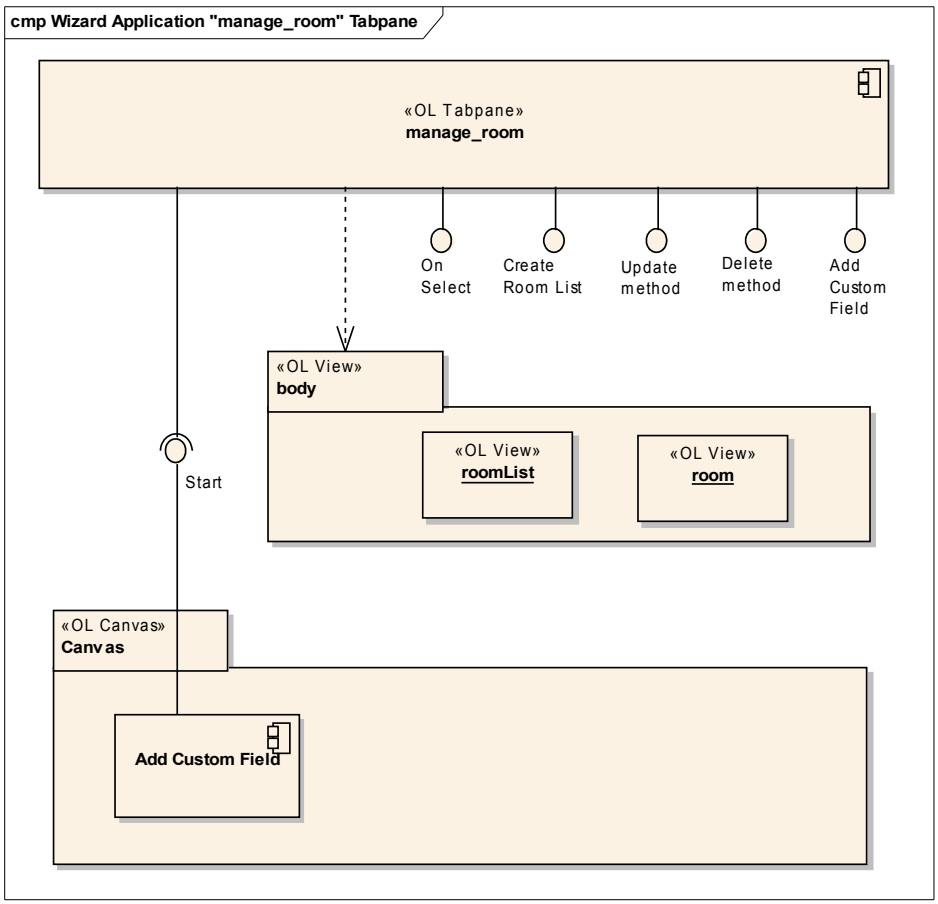

Figure 8.17.: manage_room" OL component

Inside "manage_room" another OL View named "body" is defined, it has the main purpose of grouping together other two OL *Views*: "roomList" and "room". "roomList" contains an OL *List* object, which owns a UI object ("datasetBoundedRoomList") bounded to the list of all the room available at the Bed & Breakfast (the "list" OL *Dataset*). The other OL *View* named "room" contains a UI form with the details of the currently selected room.

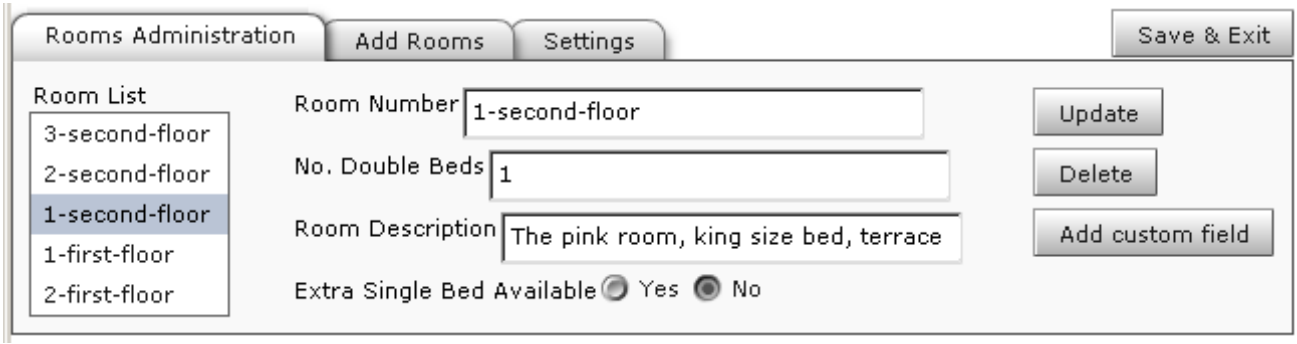

Figure 8.18.: "manage_room" UI

As an example, in Figure 8.18.: "manage_room" UI the room list is filled with the 5 available rooms (on the left part of the *Tabpane*, within the not-visible "roomList" *View*), while in the middle of the "Rooms Administration" *Tabpane* the UI form is filled with details about room number "1-second-floor" (within the not-visible "room" *View*).

At application startup the "manage_room" OL *View* got the UI focus, and the "On Select" provided interface is invoked by OL runtime. The method associated with this interface uses the "Create Room List" to populate the listbox.

When the "Room List" is just loaded there is no room selected in the listbox, thus the "Update" and "Delete" buttons are disabled. If the user selects one of the item in the "Room List", the "currentlySelectedRoomData" OL *Dataset* (owned by the *Canvas*) is loaded with the room values, and then the correspondent UI form displays such values, because a proper OL *Datapath* definition exists inside the "room" OL *View*:

```
<view name="roomData">
<datapath xpath="currentlySelectedRoomData:/roomType/*" />
```

The selection of a particular room in the "Room List" enables the "Update" and "Delete" buttons:

```
// Enables "Update" and "Delete" buttons
parent.parent.parent.btnUpdate.setAttribute("enabled", true);
parent.parent.parent.btnDelete.setAttribute("enabled", true);
```

The "Update" button is associated with the "Update method" provided interface, this method updates values in the "data" OL *Dataset* using values contained into "currentlySelectedRoomData" OL *Dataset.* Finally "Create Room List" interface is used to update the content of the listbox.

The "Delete" button is associated with the "Delete method" provided interface, this method deletes values from the "data" OL *Dataset* using key value (room ID) contained into "currentlySelectedRoomData" OL *Dataset.* Finally "Create Room List" interface is used to update the content of the listbox.

8.6.7 Adding Custom Field

The “Add custom field” button is associated with the “Add Custom Field” interface provided by “manage_room” OL *Tabpane*. Figure 8.17.: manage_room” OL component shows that OL *Canvas'* “Add Custom Field” component provides an interface to execute a UI modal dialog box. The UI of this dialog box is shown in Figure 8.19.: “Add custom field” dialog UI. The “Add Custom Field” dialog is activated within the “manage_room” OL *Tabpane* by the code:

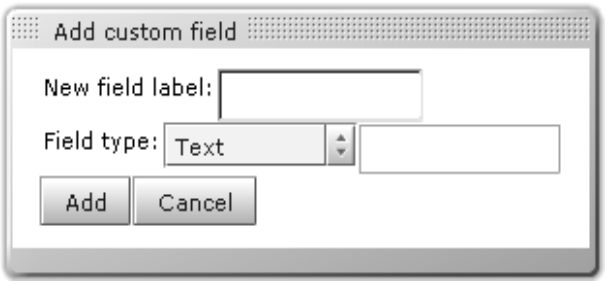

Figure 8.19.: “Add custom field” dialog UI

```
<button text="Add custom field" x="500" y="65"
 onclick="canvas.dlgAddField.animOpen.doStart()" />
```

The user can specify a label for the new room attribute, for instance “air conditioning”, and then chooses from the two field types available: Text or Yes/No. If the user, for instance, add the field “air conditioning”, then he or she should select Yes/No as the associated data type. If the “Add” button is pressed, then the new attribute is added to all room instances stored in “data” OL *Dataset*, even the “currentlySelectedRoomData” OL *Dataset* is updated. To complete the task a refresh operation is invoked to display room details with the new attribute:

```
// Refresh the currently selected room form
// and closes the modal dialog box
//
canvas.setCurrentlySelectedRoomData();
parent.parent.close();
```

8.6.8 Adding New Rooms

The Service Provider uses the wizard application to publish a User Service instance tailored for his or her own business. During the data gathering phase in the example of this tutorial, the user adds new rooms to the room list. The Consumer will use the room list as a base for his or her reservation request. To add a new room the Service Provider user select the tab named “Add Rooms” in the wizard application UI. Figure 8.20.: “Add Rooms” UI shows the “Add Rooms” wizard UI.

In Figure 8.21.: “add_room” OL Tabset component diagram is shown the “add_room” OL *Tabset* component model diagram.

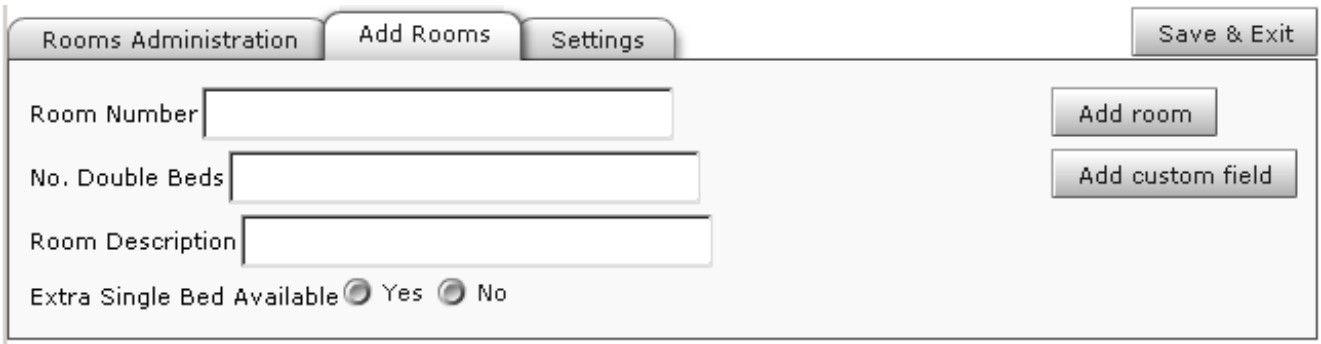

Figure 8.20.: "Add Rooms" UI

When the user selects the "Add Rooms" tab, the "On Select" provided interface is used to prepare the input form:

```
<method event="onselected" reference="this.tab">
      if (this.tab.selected)
      {
            canvas.setCurrentlySelectedRoomData();
      }
</method>
```

The wizard application uses the "roomData" OL *View* to manage the room data input form. As the data entry is completed the user can actually add the new room pressing button "Add room", which is associated with the "Insert method" interface. "Insert method" adds the the new room to the "data" OL *Dataset*, and then prepares the input form for the data entry of the next room data:

```
// Resets the input form reading the values
// from "data:/roomType",
// all form fields are empty.
//
canvas.setCurrentlySelectedRoomData();
```

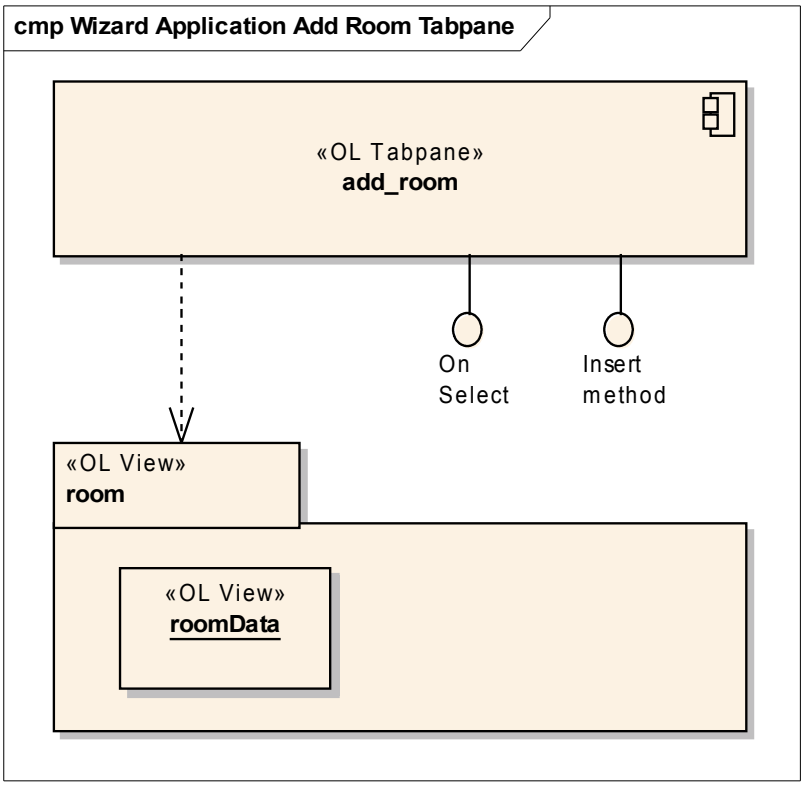

Figure 8.21.: "add_room" OL Tabset component diagram

8.6.9 Settings

The UI of the "Settings" OL *Tabset* has already been shown in Figure 8.12.: Settings tabpane. The "settings" OL Tabset is a very simple OL element, which contains only one major OL View. No interfaces are provided by this OL component. The only "settings" internal view, named "layoutSettings", defines a OL *Combobox* which contains all the possible values for the User Service background colour (Default, Green, Blue, Purple). When the "Save & Exit" button is pressed, the value of the OL *Combobox* is read and saved to "data" OL *Dataset* structure:

```
// Saves the background colour setting
tmp.setXPath("data:/settings/layout");
tmp.setNodeText(main.settings.layoutSettings.layout.getText());
```

8.6.10 Save and Exit

At the completion of data gathering phase, the Service Provider user can save and publish the User Service by pressing the "Save & Exit" UI button. This action corresponds to the execution of two main steps: the User Service data and the User Service BML Data (contained in memory by the "data" OL *Dataset*) are packed together into a single XML structured variable, and then a remote method call to DBE portal publishes the User Service instance. This behaviour is followed in both the case of a new User Service instance and the case of the updating of an

already published User Service, is up to the DBE portal method to perform the correct operation using the passed actual parameters.
When the “Save & Exit” button is pressed by the user, the “confirm” OL *Alert* method is invoked:

```
// Builds (User Service Data + BML data) into
// "ifDsString" local variable
// It is used the "getIfDs()" library function.
var utils = new IfUtility;
var ifDsString = utils.getIfDs();
```

this code loads the User Service data and the User Service BML Data to the in memory variable “ifDsString”, reading from the “data” OL *Dataset*.
Then the remaining of the method is executed:

```
// The "executeIF" method of class
"org.dbe.toolkit.portal.client.DBEPortalClient"
// is invoked.
// The meanings of the parameters are:
// smIdProviderService = wizard service's SMID
// smIdUserService     = User Service's SMID
// ifDsString          = User Service Data + BML data
//
this.executeIF.invoke([canvas.smIdProviderService,
canvas.smIdUserService, ifDsString]);
```

this code calls the remote “executeIF” method provided by DBE Portal, which makes up the SM and publish the service. If the “smIdUserService” parameter has an assigned value then the operation is an update of an existing User Service, otherwise the operation is the publishing of a new instance of User Service.

The remote call is possible because of an OL definition:

```
<javarpc name="DBEPortalClient"
   scope="session" autoload="true"
   classname="org.dbe.toolkit.portal.client.DBEPortalClient">
</javarpc>
```

8.6.11 Other implementation details

The details about other minor functionalities granted by the wizard application are in the source code of the Open Laszlo program, which is extensively commented to help the reader understanding the design and implementation issues about the wizard. The complete source of the wizard application is in the file “wizardService.lzx” reported in annex 9-Annex A.

8.7 User Service Application Construction

The User Service application is the executable service that will be consumed by the final user: the Consumer. The User Service is an SCS, hence the code of its OL application is contained into its Service Manifest (SM). When the Consumer executes an SCS User Service, the OL application is downloaded and executed into its Internet browser. The complete source of the tutorial Bed & Breakfast example is reported in 9-Annex A in the file “userService.lzx”. The SM of the published User Service contains also the XML structure with the specific service instance data, as it has been created by the wizard application at the end of the data gathering phase. The complete XML data for this tutorial example is reported in file “data.lzx” in 9-Annex A.

In Figure 8.22.: Bed & Breakfast User Service GUI is shown the GUI of the tutorial example as it is executed into the Consumer Internet browser.

Figure 8.22.: Bed & Breakfast User Service GUI

In Figure 8.23.: User Service application component model is depicted the OL component model of the User Service application.

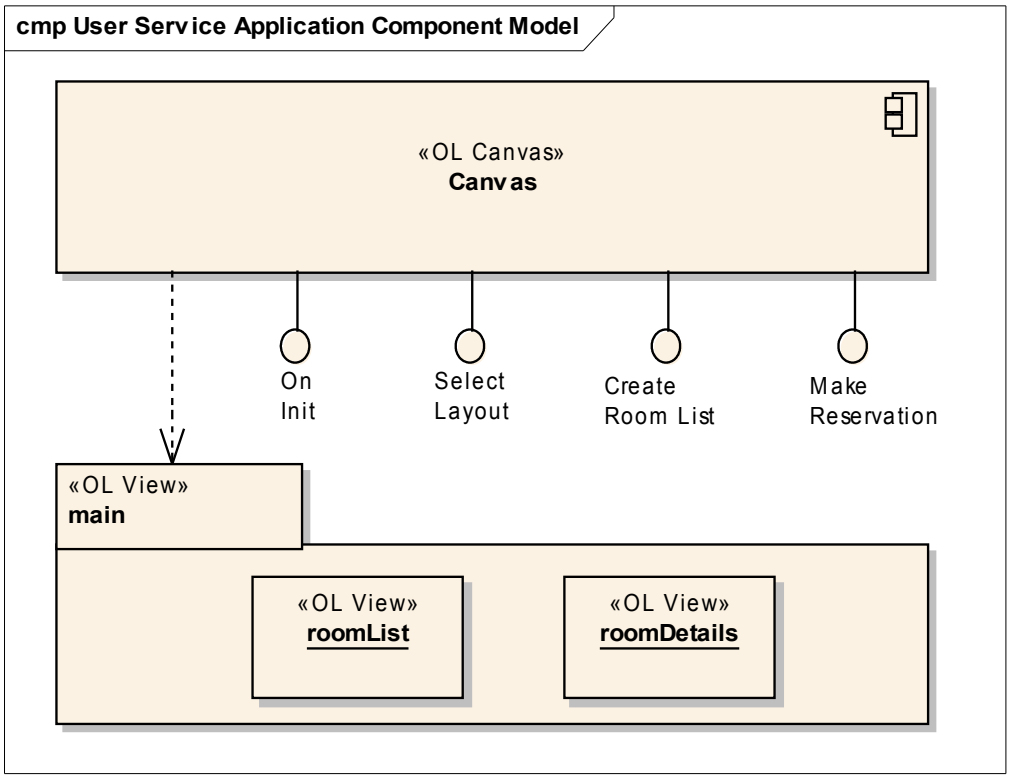

Figure 8.23.: User Service application component model

When the application[25] is executed, XML data contained in "data.lzx" are loaded in memory in the "data" OL *Dataset*, the "data.lzx" file is located through its URL by the subsequent OL statement:

```
<!-- Includes external dataset  -->
<include href="./data.lzx" />
```

Soon after the XML data loading the OL runtime invokes the "On Init" provided interface, which uses two other application provided interface: "Select Layout" and "Create Room List". "Select Layout" is implemented into the method:

```
<method name="doSelectLayout"> ... </method>
```

it reads the setting section from "data" OL *Dataset*, positioning a data pointer to:

```
tmp.setXPath("data:/settings/layout");
var layout = tmp.getNodeText();
```

and then applies the selected background colour (one of the three available):

```
if (layout == "Green")
      greenStyle.setAttribute("isdefault", true);
else if (layout == "Blue")
      blueStyle.setAttribute("isdefault", true);
else if (layout == "Purple")
```

25 Application is used as a synonym of User Server application in this context.

```
purpleStyle.setAttribute("isdefault", true);
```

"Create Room List" is implemented into the method:

```
<method name="doCreateRoomList"> ... </method>
```

and its purpose is that of displaying the list of the Bed & Breakfast available rooms.

To display the room list, the "doCreateRoomList" method uses a OL *Datapath* definition between an OL Textlistitem named "roomTextListItem" (contained into "roomList" OL *View*) and the OL *Dataset* named "list":

```
<textlistitem name="roomTextListItem" datapath="list:/room"
text="$path{'Room Description/text()'}"
value="$path{'id/text()'}" />
```

The "list" OL *Dataset* is a temporary XML *Canvas* scope variable which contains a simplified list of rooms read from the "data" OL *Dataset*.

When the execution of the "doCreateRoomList" is completed, OL runtime displays the application UI, giving the focus to "main" OL View, which contains "roomTextListItem" bounded to the actual list of available rooms. Hence at the end of the application initialisation the list of available rooms is displayed to the Consumer user.

If the user selects one of the room in the list, the "roomDetails" OL *View* is populated by room's attribute values, as shown in Figure 8.22.: Bed & Breakfast User Service GUI. The details display method is coded into the "roomList" OL *View*:

```
<!--
This method is invoked when the user selects one of the items
in the "room list". The detail about the selected room are
displayed in the right half of the "main" UI view.
-->
<method event="onselect"> ... </method>
```

The OL application element which actually shows room details is "roomDetail" OL *View*.

When a room is selected within the room list and its details are displayed to the user, even the "Reserve" button is enabled:

```
parent.parent.btnReservation.setAttribute("visible", true);
```

If the Consumer wants to send a reservation request to the Bed & Breakfast owner, he or she selects the preferred room in the list and then presses the "Reserve" button. The method associated with the "Reserve" button-clicked event prepares an email message addressed to the Bed & Breakfast mailbox, as read from the "data" OL *Dataset*. "data:/infoMail" section. The email message has an object that specifies the description and the number of the requested room, and a body in which the user could add its name, address and any other useful information. As the message has been completed the user can send it to the Bed & Breakfast mailbox, as well as any other regular email message sent from its client. From this point on, the task of the User Service is completed and the interaction between the Consumer and the Service Provider could go

on through alternative channels, for instance through email, phone calls, or whatever is appropriate.

8.7.1 ifLib Open Laszlo Library

As already mentioned in previous chapter, an Open Laszlo Library was developed to support IF Wizard Suppliers in the creation of new wizards and User Service applications. The Library offers operations to manipulate “data” OL *Dataset* and the contained “bmlData” XML element which stores the User Service's BML Data definition. The whole “data” OL *Dataset* is contained in “data.lzx” file. It is important to notice that “data.lzx” is the mandatory file name for both the wizard and the User Service case.

The XML structure of “data” OL *Dataset* must comply with the following pattern:

```
<dataset name=”data”>
      [other XML elements here]
      <bmlData>
            [User Service's BML data here]
      </bmlData>
      [other XML elements here]
</dataset>
```

The dataset name is specified by the “name” attribute, and conventionally its value must be set equal to “data”. Within the dataset an element surrounded by

```
<bmlData> ... </bmlData>
```

must be included, this corresponds to the section in which User Service's BML Data are stored.

To use the library in an Open Laszlo program, the library itself must be included:

```
<!-- Includes library file -->
<include href="./ifLib.lzx" />
```

then before using one the provided functions it is necessary to instantiate an handle to the library:

```
var utils = new IfUtility;
```

in the following sections every operation of the library is discussed indetail.

8.7.1.1 getIfDs()

getIfDs requires no parameters. This function reads the “data” OL *Dataset* and returns an XML text value formatted as follow:

```
<IfDs>
      <bmlData>
            [User Service's BML data here]
      </bmlData>
      <serviceData>
```

```
            [other service's data here]
        </serviceData>
</IfDs>
```

in which <bmlData> element contains the User Service's BML Data definition and <serviceData> element contains every other piece of information from “data” OL *Dataset*.
When the remote “executeIF” is invoked the “ifDsString” parameter must be set to the value returned by “getIfDs” call, as in the following example taken from the tutorial's IF Wizard application example:

```
// Builds (User Service Data + BML data) into
// "ifDsString" local variable
// It is used the "getIfDs()" library function.
var utils = new IfUtility;
var ifDsString = utils.getIfDs();


// The "executeIF" method of class
// "org.dbe.toolkit.portal.client.DBEPortalClient"
// is invoked.
// The meanings of the parameters are:
// smIdProviderService = wizard service's SMID
// smIdUserService = User Service's SMID
// ifDsString = User Service Data + BML data
//
this.executeIF.invoke([canvas.smIdProviderService, canvas.smIdUserService,
ifDsString]);
```

8.7.1.2 setBmlDataAttribute

“setBmlDataAttribute” requires two parameters:”name” and “val”. This function creates a node within the <bmlData> XML element of “data” OL *Dataset*, the following bmlData section is taken from the tutorial example (file data.lxz)*:*

```
<bmlData>
      <IfData_name>
      </IfData_name>
      <IfData_address>
      </IfData_address>
      <IfData_email>
      </IfData_email>
</bmlData>
```

If the node named “name” already exists in the XML “bmlData” element, then the call to this function updates the value of the corresponding node.
This function should be used by the wizard application to add name/value pair to the User Service's BML Data definition.

8.7.1.3 GetBmlDataAttribute

"GetBmlDataAttribute" requires one parameter: "name". This function returns the value of the node identified by "name", searching within <bmlData> element in "data" OL *Dataset*. If such a node doesn't exists the function returns an empty string.
This function should be used by the wizard application to obtain the value of a node from User Service's BML Data definition.

8.7.1.4 deleteBmlDataAttribute

"deleteBmlDataAttribute" requires one parameter: "name". This function deletes the node identifyed by "name" from <bmlData> element in "data" OL *Dataset*. If such a node doesn't exists the function returns without signalling an error.
This function should be used by the wizard application to delete name/value pair from the User Service's BML Data definition.

8.8 User Service BML Data Template definition

All the User Services created by the same IF Wizard represent the same Conceptual Service and therefore share the same BML; however they are different Service Instance and therefore have different BMLData. These data will be required by the IF Wizard to the Provider that could easily enter them without using the BML Data Editor. Each IF Wizard can use, of course, its own technique to do that, but a library has been elaborated to allow who is writing the IF Wizard to make that simpler by creating in advance a BML Data Template that will consequently be populated by the IF Wizard. This Template is a BML Data in which values are replaced by some "marks" in the form **IfData_UniqueID** where "IfData" is a constant to identify that it is a user data and the "UniqueID" is the ID that identifies the single data element. It will then be easier to replace real data with marks from the Template by using the appropriate tool provided by the library created ad hoc. The complete source of the IF Wizard utility library is reported in file "ifLib.lxz" in 9-Annex A.
What follows is the tutorial example BML Data Template definition in XML format:

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<XMI xmi.version = '1.2' timestamp = 'Mon Nov 27 17:36:40 CET 2006'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
  </XMI.header>
  <XMI.content>
    <IMM.Object xmi.id = 'a1' type = 'BedAndBreakfast' namespace =
'http://www.mySME.com/BedAndBreakfast_reservation_v.5'>
      <IMM.Object.slot>
        <IMM.Slot xmi.id = 'a2' type = 'name'>
```

```
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a3' type = 'http://www.w3c.org/2001/XMLSchema#string'
                literal = 'IfData_name'/>
            </IMM.Slot.value>
          </IMM.Slot>
          <IMM.Slot xmi.id = 'a4' type = 'address'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a5' type = 'http://www.w3c.org/2001/XMLSchema#string'
                literal = 'IfData_address'/>
            </IMM.Slot.value>
          </IMM.Slot>
          <IMM.Slot xmi.id = 'a6' type = 'email'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a7' type = 'http://www.w3c.org/2001/XMLSchema#string'
                literal = 'IfData_email'/>
            </IMM.Slot.value>
          </IMM.Slot>
          <IMM.Slot xmi.id = 'a8' type = 'phoneNumber'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a9' type = 'http://www.w3c.org/2001/XMLSchema#string'
                literal = 'IfData_phoneNumber'/>
            </IMM.Slot.value>
          </IMM.Slot>
          <IMM.Slot xmi.id = 'a10' type = 'faxNumber'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a11' type = 'http://www.w3c.org/2001/XMLSchema#string'
                literal = 'IfData_faxNumber'/>
            </IMM.Slot.value>
          </IMM.Slot>
          <IMM.Slot xmi.id = 'a12' type = 'urlWebSite'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a13' type = 'http://www.w3c.org/2001/XMLSchema#anyURI'
                literal = 'IfData_urlWebSite'/>
            </IMM.Slot.value>
          </IMM.Slot>
        </IMM.Object.slot>
      </IMM.Object>
      <IMM.Object xmi.id = 'a14' type = 'Rooms' namespace =
'http://www.mySME.com/BedAndBreakfast_reservation_v.5'>
        <IMM.Object.slot>
          <IMM.Slot xmi.id = 'a15' type = 'type'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a16' type = 'http://www.w3c.org/2001/XMLSchema#string'
                literal = 'IfData_type'/>
            </IMM.Slot.value>
          </IMM.Slot>
```

```
          <IMM.Slot xmi.id = 'a17' type = 'bedsNumber'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a18' type = 'http://www.w3c.org/2001/XMLSchema#int'
                literal = 'IfData_bedsNumber'/>
            </IMM.Slot.value>
          </IMM.Slot>
          <IMM.Slot xmi.id = 'a19' type = 'price'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a20' type = 'http://www.w3c.org/2001/XMLSchema#float'
                literal = 'IfData_price'/>
            </IMM.Slot.value>
          </IMM.Slot>
        </IMM.Object.slot>
      </IMM.Object>
      <IMM.Object xmi.id = 'a21' type = 'Van' namespace =
'http://www.mySME.com/BedAndBreakfast_reservation_v.5'>
        <IMM.Object.slot>
          <IMM.Slot xmi.id = 'a22' type = 'seatingCapacity'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a23' type = 'http://www.w3c.org/2001/XMLSchema#int'
                literal = 'IfData_seatingCapacity'/>
            </IMM.Slot.value>
          </IMM.Slot>
        </IMM.Object.slot>
      </IMM.Object>
      <IMM.Object xmi.id = 'a24' type = 'RoomRental' namespace =
'http://www.mySME.com/BedAndBreakfast_reservation_v.5'/>
      <IMM.Object xmi.id = 'a25' type = 'GuidedTours' namespace =
'http://www.mySME.com/BedAndBreakfast_reservation_v.5'/>
      <IMM.Object xmi.id = 'a26' type = 'PetsAcceptance' namespace =
'http://www.mySME.com/BedAndBreakfast_reservation_v.5'/>
      <IMM.Object xmi.id = 'a27' type = 'BedAndBreakFast_reservation' namespace =
'http://www.mySME.com/BedAndBreakfast_reservation_v.5'>
        <IMM.Object.slot>
          <IMM.Slot xmi.id = 'a28' type = 'BandB_Name'>
            <IMM.Slot.value>
              <IMM.DataValue xmi.id = 'a29' type = 'http://www.w3c.org/2001/XMLSchema#string'
                literal = 'IfData_BandB_Name'/>
            </IMM.Slot.value>
          </IMM.Slot>
        </IMM.Object.slot>
      </IMM.Object>
    </XMI.content>
</XMI>
```

From the complete BML Data Template, for the sake of simplicity in this tutorial example only three values have been saved in the bmlData section of “data” *Dataset*:

```
<bmlData>
```

```
        <IfData_name>
        </IfData_name>
        <IfData_address>
        </IfData_address>
        <IfData_email>
        </IfData_email>
</bmlData>
```

The values in the bmlData section correspond to the B&B name, B&B address and B&B information email.

8.9 IF Wizard Service Manifest composition

A complete guide to Service Manifest concept and SM Composition can be found in [D18.5SMED].Anyway to compose the SM of an IF Wizard some extra steps are necessary, as documented in the current section.

Before invoking SM composition interface two zipped files must be created with an external archive utility, refer to Figure 8.24.: SM Composition Wizard zip file. The first file to be included in this archive is the file which should contain the code of the wizard application (wizardService.lzx, the name given here is only an example, the actual file name is up to the IF Wizard Supplier), then the IF Wizard library file is needed (ifLib.lzx, as provided by DBE) and at last the “data” *Dataset* file must be added (data.lzx, as well as the name of the application file, this file name is up to the IF Wizard Supplier). The file name of the resulting zipped archive is a temporary file name, it will be used during the SM composition phase, thus no specific pattern should be applied to this file name.

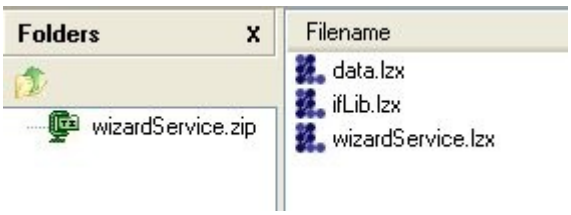

Figure 8.24.: SM Composition Wizard zip file

The other zipped archive is that which contains the user service, as depicted in Figure 8.25.: SM Composition User Service zip file. As for the previous archive this one should contain IF Wizard library file and “data” *Dataset* file, plus the User Service application file (userService.lzx in the current example, but this name is not mandatory and can be freely chose by the IF Wizard Supplier). Even in this case the name of the zipped archive file is temporary, to be used during SM composition phase. Any other support file can be included in this archive, without limitations, for example extra library files and picture files can be added to be used by the User Service application.

When the SM composer is invoked, the User interface form of Figure 8.28.: SM Composition is displayed, in this form it is necessary to provide attributes of the IF Wizard service: SMName, Description and VersionNumber have to be specified as for any other DBE Business Service, IF Wizard BML Model and BML Data must be specified while SDL Model is not necessary in case of ServiceType equals to

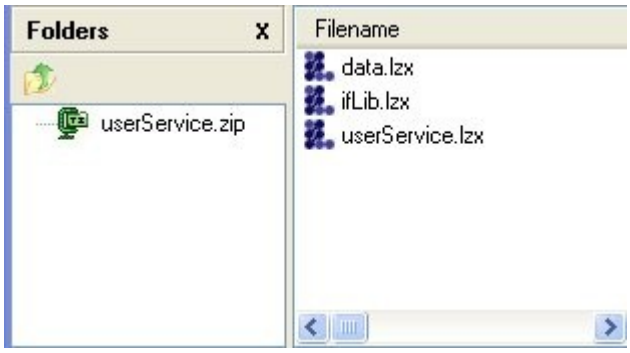

Figure 8.25.: SM Composition User Service zip file

"InteractionForm". Given this set of attributes, pressing the "IF" button correspondent to the "InteractionForm" row the Wizard definition UI form of Figure 8.26.: SM Composition, Wizard definition is activated: Name, Description and Version are descriptive fields (the version of the wizard application can be changed without changing the IF Wizard version, for example after some bug fixing activity), then the complete path of the wizard zipped archive file must be provided in the "ZipPath" field, "openlaszlo" must be specified as "Code Type", and "UI Main" field must be set equal to the main wizard application file name chosen by the IF Wizard Supplier (this value is case sensitive). The form is confirmed and closed pressing "next" button.

The next step is the Service definition of Figure 8.27.: SM Composition, Service definition: Name, Description and Version are values used to create User Service instances in DBE, "ZipPath" must be set to the service zipped archive file name, "Code Type" must be set to "openlaszlo" and "UI Main" must be set to the file name of the main User Service application (this value is case sensitive). The Service definition phase ends with pressing the "next" button.

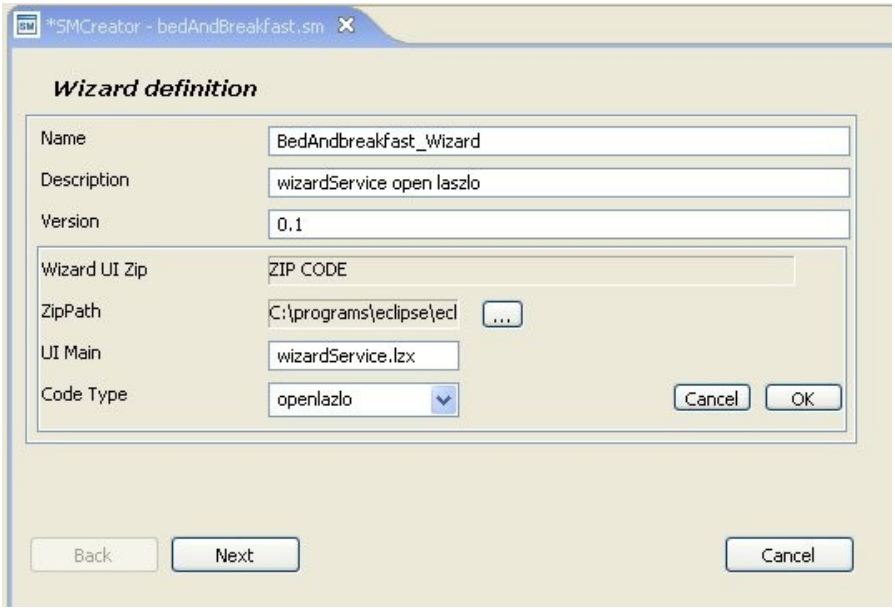

Figure 8.26.: SM Composition, Wizard definition

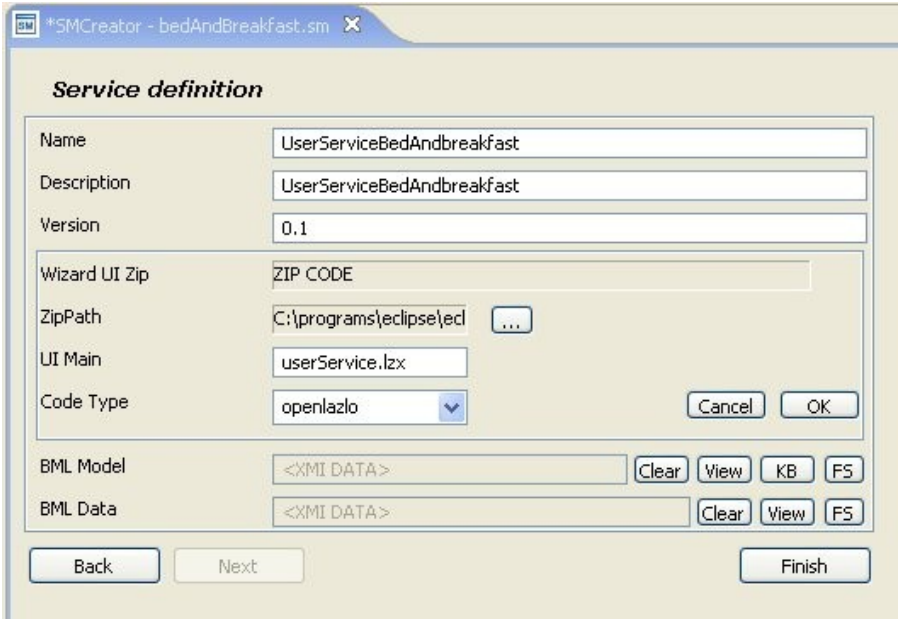

Figure 8.27.: SM Composition, Service definition

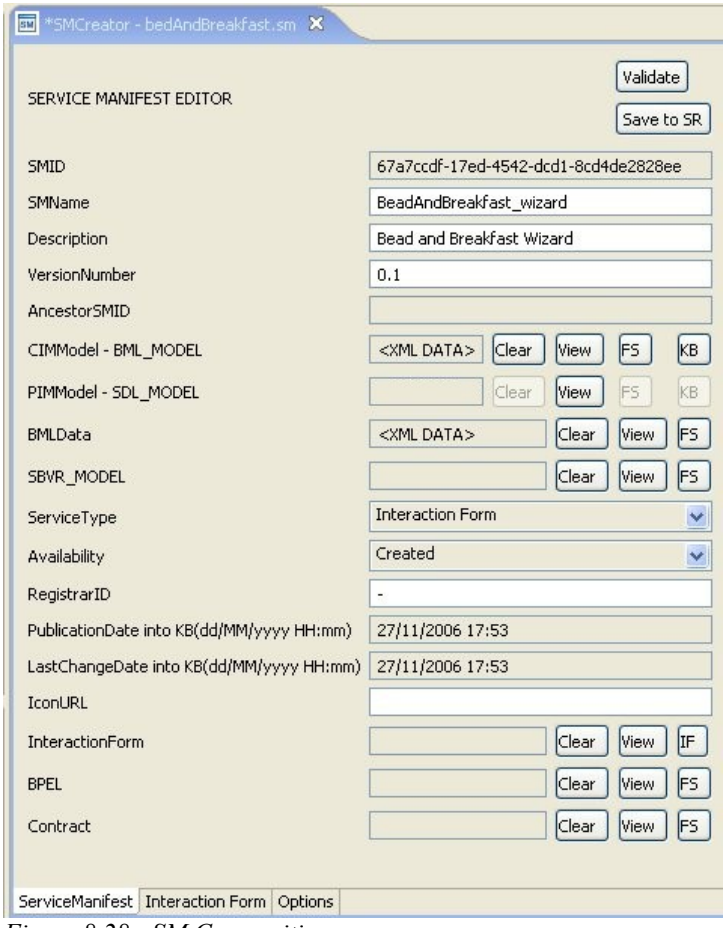

Figure 8.28.: SM Composition

8.10 IF Wizard Service Manifest publication

At the end of the steps described in the previous section the IF Wizard service is ready to be published, to complete the operation “Save to SR” button must be pressed. When the publication completes the SM Editor confirm the success operation showing the SMID identification number assigned to the just published service.

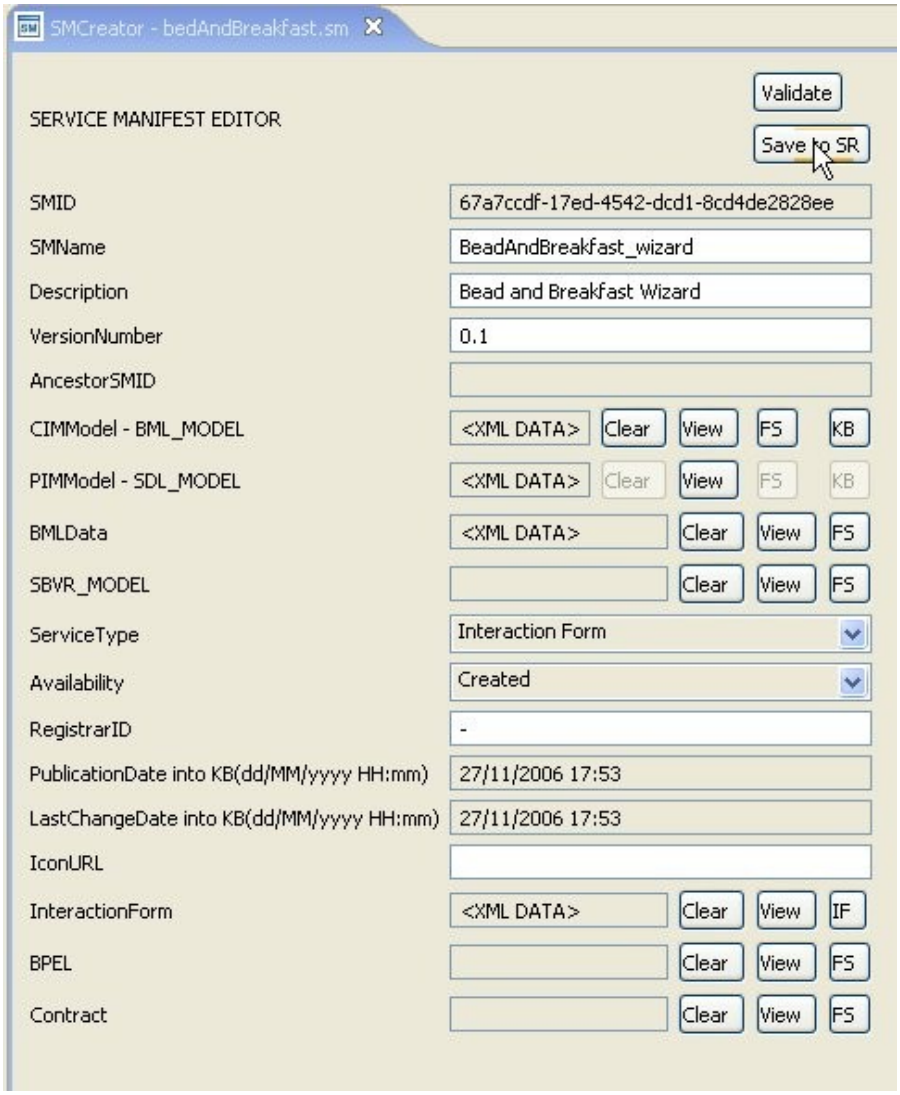

Figure 8.29.: SM Composition, IF Wizard publication

8.11 DBE Portal Screen Shots

As a usage example of the DBE Portal, in this chapter are reported some screen shots. In Figure 8.30.: DBE Portal Home Page a link to “Search & browse” is available, in Figure 8.31.: Service search form the search interface is shown, and finally Figure 8.32.: Search results is the screen shot reporting the matching services. To activate a service, either an IF Wizard or a User Service, the User has to follow the “Execute service” link.

Figure 8.30.: DBE Portal Home Page

Figure 8.31.: Service search form

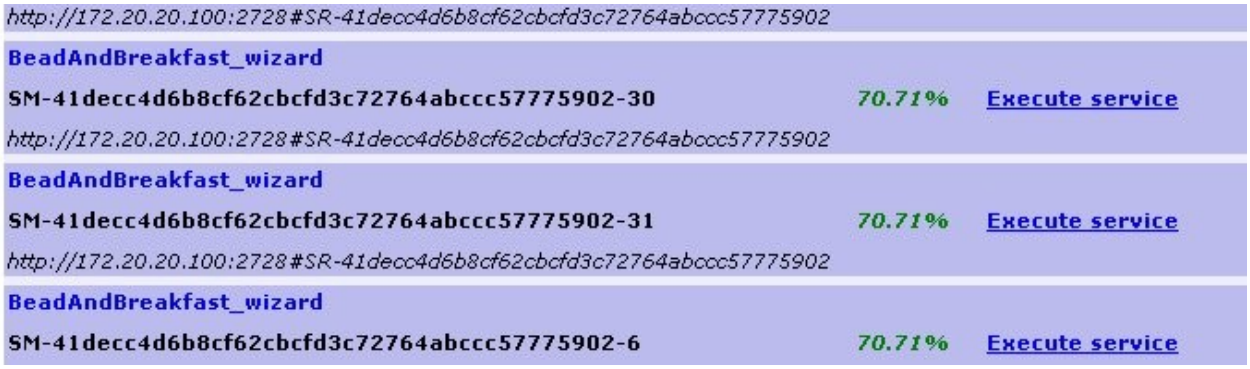

Figure 8.32.: Search results

9 Annex A: IF Wizard Bed & Breakfast Example Sources

9.1 data.lzx[26]

```
<dataset name="data">

	<admin>
		<user></user>
		<pwd></pwd>
	</admin>

	<infoMail>info@bandbitaly.it</infoMail>

	<rooms>
		<room>
			<id>1</id>
			<field label="Room Number" type="Text">1-first-floor</field>
			<field label="No. Double Beds" type="Text">1</field>
			<field label="Room Description" type="Text">Quiet room, king size bed,
lovely view of the hills</field>
			<field label="Extra Single Bed Available" type="Yes/No">Yes</field>
		</room>
		<room>
			<id>2</id>
			<field label="Room Number" type="Text">2-first-floor</field>
			<field label="No. Double Beds" type="Text">2</field>
			<field label="Room Description" type="Text">Sunny room, spacious, view of
the garden</field>
			<field label="Extra Single Bed Available" type="Yes/No">No</field>
		</room>
		<room>
			<id>3</id>
			<field label="Room Number" type="Text">1-second-floor</field>
			<field label="No. Double Beds" type="Text">1</field>
			<field label="Room Description" type="Text">The pink room, king size bed,
terrace</field>
			<field label="Extra Single Bed Available" type="Yes/No">No</field>
		</room>
		<room>
			<id>4</id>
			<field label="Room Number" type="Text">2-second-floor</field>
			<field label="No. Double Beds" type="Text">2</field>
			<field label="Room Description" type="Text">Family room, spacious,
terrace</field>
			<field label="Extra Single Bed Available" type="Yes/No">Yes</field>
		</room>
		<room>
			<id>5</id>
			<field label="Room Number" type="Text">3-second-floor</field>
			<field label="No. Double Beds" type="Text">1</field>
			<field label="Room Description" type="Text">Honeymoon room, quiet
atmosphere, view of the hills</field>
			<field label="Extra Single Bed Available" type="Yes/No">Yes</field>
```

26 “data.lxz” is the mandatory file name both in the wizard and the User Service case.

```
            </room>
        </rooms>

        <roomType>
            <id />
            <field label="Room Number" type="Text" />
            <field label="No. Double Beds" type="Text" />
            <field label="Room Description" type="Text" />
            <field label="Extra Single Bed Available" type="Yes/No" />
        </roomType>

        <settings>
            <layout>Green</layout>
        </settings>

        <bmlData>
            <IfData_name>
            </IfData_name>
            <IfData_address>
            </IfData_address>
            <IfData_email>
            </IfData_email>
        </bmlData>
        <!-- (MB) Placeholder field to manage the wizard SMID -->
        <WSMID>x</WSMID>
</dataset>
```

9.2 WizardService.lzx[27]

```
<?xml version="1.0" encoding="UTF-8" ?>
<canvas width="1000" height="600" debug="true">

        <!--
        Canvas-scope variable used to read parameter from the URL line.
        -->
        <attribute name="smIdUserService"     value="111111111111" type="string" />
        <attribute name="smIdProviderService" value="111111111111" type="string" />

        <!--
        Methods automatically invoked at application startup.
        It reads "Service Manifest ID" and "Service Manifest ID User" from application URL.
        "Service Manifest ID" is not used in this wizard example.
        -->
        <method event="oninit">
            smIdUserService = LzBrowser.getInitArg("smIdUser");
            smIdProviderService = LzBrowser.getInitArg("smid");
            // If "Service Manifest ID User" (read from URL) is not empty,
            // then an account already exists, stored in file "data.lzx".
            // Otherwise, the wizard is executed by the first time,
            // then opens the interactive form to collect
            // "username" and "password" values.
            //
            <!--The smIDUserService it is always present:
```

27 “wizardService.lzx” is not the mandatory file name, but must differs from User Service's file name.

```
                  if it is equal to smIdProviderService it is the first time otherwise it is an
update functionality
                -->
                if (smIdUserService != smIdProviderService)
                        dlgLogin.open();
                else
                        adminUserDefinitionForm.setAttribute("visible",true);
        </method>

        <!--
         Sets "currentlySelectedRoomData" dataset equals to
         "roomType" element of "data" dataset.
        -->
        <method name="setCurrentlySelectedRoomData">
                tmp.setXPath("currentlySelectedRoomData:/roomType");
                tmp.deleteNode();
                tmp.setXPath("currentlySelectedRoomData:/");
                tmp1.setXPath("data:/roomType");
                tmp.addNodeFromPointer(tmp1);
        </method>

        <!-- Includes library file -->
        <include href="./ifLib.lzx" />

        <!-- Includes "data" dataset file definition  -->
        <include href="./data.lzx" />

        <!-- Dataset temporanei -->
        <dataset name="currentlySelectedRoomData">
                <newField>
                        <label />
                        <type />
                </newField>
        </dataset>
        <dataset name="list" />

        <!-- Dataset Pointers -->
        <datapointer name="tmp" />
        <datapointer name="tmp1" />

         <!--
         Button "Save & Exit", asks confirmation, and then saves and exits the wizard application.
         -->
        <button name="btnExit" text="Save &amp;amp; Exit" x="548" y="3" visible="false"
onclick="confirmExit.open()" />

        <javarpc name="DBEPortalClient" scope="session" autoload="true"
                 classname="org.dbe.toolkit.portal.client.DBEPortalClient">
         </javarpc>

        <alert name="confirmExit" button1="Yes" button2="No">
                Confirm Save and Exit action?
                 <method event="onresult" args="yes_selected">
                          if (yes_selected)
                        {
                                // Builds (User Service Data + BML data) into "ifDsString" local
variable
```

```
                                // It is used the "getIfDs()" library function.
                                var utils = new IfUtility;
                                var ifDsString = utils.getIfDs();
                                // Debug.write(ifDsString);

                                // The "executeIF" method of class
"org.dbe.toolkit.portal.client.DBEPortalClient"
                                // is invoked.
                                // The meanings of the parameters are:
                                   // smIdProviderService = wizard service's SMID
                                // smIdUserService     = User Service's SMID
                                   // ifDsString          = User Service Data + BML data
                                   //
                                   this.executeIF.invoke([canvas.smIdProviderService,
canvas.smIdUserService, ifDsString]);

                                   // The Internet browser could now be closed
                        }
                </method>

                <!--
                Creates an Open Laszlo hook to a remote mothod.
                The class is "org.dbe.toolkit.portal.client.DBEPortalClient" and the invoked
method
                is equal to funcname, i.e. "executeIF".
                In this definition the attribute "name" is not defined, by default name=funcname,
                i.e."executeIF".
                -->
                <remotecall funcname="executeIF" remotecontext="$once{canvas.DBEPortalClient}">
                        <method event="onerror" args="error">
                                Debug.write('Error report:', error);
                                Debug.inspect(error);
                        </method>
                </remotecall>
        </alert>


        <edittext name="temp" x="10" y="200" multiline="true" width="300" height="400"
visible="false" />

        <!--
         The "adminUserDefinitionForm" form is used to collect "username" and "password" values
from the user.
         It is executed when the element "data:/admin" is empty, and the wizard application is
         invoked by the first time (URL argument named "smiduser" missed).
        -->
         <view name="adminUserDefinitionForm" visible="false">
                <!--
                 This method is invoked when the user presses "Create User" button
                 in the user definition form.
                -->
                 <method name="addUser">
                        // Username, Password and email address are mandatory values,
                        // if not specified shows an error box to the user.
                        //
                        if ((user_view.usr.getText() == "") ||
                              (pwd_view.pwd.getText() == "") ||
                              (bbname_view.bbname.getText() == "") ||
```

```
                        (bbaddress_view.bbaddress.getText() == "") ||
                        (bbmail_view.bbmail.getText() == ""))
                {
                        alertError.open();
                        return;
                }

                // Updates "data" dataset with Username, Password and email address.
                  tmp.setXPath("data:/admin/user");
                tmp.setNodeText(user_view.usr.getText());
                tmp.setXPath("data:/admin/pwd");
                tmp.setNodeText(pwd_view.pwd.getText());
                tmp.setXPath("data:/infoMail");
                tmp.setNodeText(bbmail_view.bbmail.getText());

                // Sets values in bmlData element
                // The names "IfData_name", "IfData_address" and "IfData_email"
                // are chosen in accordance with the User Service BML Data template
definition
                //
                var utils = new IfUtility;
                utils.setBmlDataAttribute("IfData_name", bbname_view.bbname.getText());
                utils.setBmlDataAttribute("IfData_address",
bbaddress_view.bbaddress.getText());
                utils.setBmlDataAttribute("IfData_email", bbmail_view.bbmail.getText());


                // Go to login dialog box
                dlgLogin.animOpen.doStart();
        </method>

        <alert name="alertError">
                <text>All fields are mandatory.</text>
        </alert>

        <simplelayout axis="y" spacing="25" />
        <text>Administrator User definition and Bed and Breakfast data</text>
        <view name="user_view">
                <simplelayout axis="x" />
                <text width="175">Username:</text>
                <edittext name="usr" />
        </view>
        <view name="pwd_view">
                <simplelayout axis="x" />
                <text width="175">Password:</text>
                <edittext name="pwd" />
        </view>
        <view name="bbname_view">
                <simplelayout axis="x" />
                <text width="175">Bed and Breakfast Name:</text>
                <edittext width="400" name="bbname" />
        </view>
        <view name="bbaddress_view">
                <simplelayout axis="x" />
                <text width="175">Bed and Breakfast Address:</text>
                <edittext width="400" name="bbaddress" />
        </view>
```

```
                <view name="bbmail_view">
                        <simplelayout axis="x" />
                        <text width="175">E-Mail info:</text>
                        <edittext width="400" name="bbmail" />
                </view>
                <button text="Create User" onclick="parent.addUser()" />
        </view>

        <!--
        Modal dialog box named "dlgLogin": allows the user to enter "username" and "password"
        to access the wizard application. Valid username/password pair is stored in "data" dataset
        in the element "data:/admin/"; "data" dataset is automatically loaded from file "data.lzx"
        during the startup of the wizard application.
        -->
         <modaldialog name="dlgLogin" title="Login">
                <animatorgroup name="animOpen" start="false" duration="1000"
process="simultaneous">
                        <animator attribute="opacity" to="0" target="adminUserDefinitionForm"
onstop="adminUserDefinitionForm.setAttribute('visible',false)" />
                        <animator attribute="opacity" to="1" onstart="dlgLogin.open()" />
                </animatorgroup>

                <!--
                 Checks username and password, if they correspond to those
                 in "data:/admin/", then the access is granted, otherwise an
                 error box is shown to the user.
                -->
                 <method name="doLogin">
                        tmp.setXPath("data:/admin/user");
                        tmp1.setXPath("data:/admin/pwd");
                        if ((user_view.usr.getText() == tmp.getNodeText()) &amp;&amp;
(pwd_view.pwd.getText() == tmp1.getNodeText()))
                        {
                                mainAnimOpen.doStart();
                                main.manage_room.doCreateRoomList();
                        } else {
                                alertError.open();
                                return;
                        }
                </method>
                <alert name="alertError">
                        <text>Wrong username or password!</text>
                </alert>

                <simplelayout axis="y" />
                <view name="user_view">
                        <simplelayout axis="x" />
                        <text>Username:</text>
                        <edittext name="usr" />
                </view>
                <view name="pwd_view">
                        <simplelayout axis="x" />
                        <text>Password:</text>
                        <edittext name="pwd" password="true" />
                </view>
                <button text="Login" onclick="dlgLogin.doLogin()" />
        </modaldialog>
```

```
        <animatorgroup name="mainAnimOpen" start="false" duration="1000" process="simultaneous">
                <animator attribute="opacity" to="0" target="dlgLogin" onstop="dlgLogin.close()" />
                <animator attribute="opacity" to="1" target="main"
onstart="main.setAttribute('visible',true)" />
                <animator attribute="opacity" to="1" target="btnExit"
onstart="btnExit.setAttribute('visible',true)" />
        </animatorgroup>

        <tabs name="main" visible="false">
                <tabpane name="manage_room" text="Rooms Administration">

                        <!--
                        When current tab ("manage_room") is selected,
                        buttons are disabled and "room list" is reloaded.
                        -->
                      <method event="onselected" reference="this.tab">
                              if (this.tab.selected)
                              {
                                      this.doCreateRoomList();
                                      btnUpdate.setAttribute("enabled", false);
                                      btnDelete.setAttribute("enabled", false);
                              }
                      </method>

                        <!--
                        Creates the room "list" dataset.
                        This method is invoked when it is necessary to refresh the "room list"
                        shown in the left half of the "manage_room" UI view.
                        It deletes everything from "list" dataset, and then refresh "list" dataset
                        reading "room" elements from "data" dataset.
                        "list" dataset is bounded to "roomElement" UI textlistitem through
                        a datapath definition.
                        -->
                      <method name="doCreateRoomList">

                              // Deletes every data contained in "list" dataset
                                tmp.setXPath("list:/");
                              while (tmp.selectChild())
                              {
                                      tmp.deleteNode();
                                      tmp.setXPath("list:/");
                              }

                              // for each "room" in "data" dataset (initially loaded from fine
"data.lzx")
                              //     copy values from "data" to "list" dataset
                              //
                              // ("list" dataset is a minimalistic list of rooms, suitable to be
displayed)
                              //
                              // "tmp" is a datapointer to "data" dataset, it scrolls through all
existing rooms.
                              // "tmp1" is a datapointer to the "list" dataset under contruction.
                              //
                                tmp.setXPath("data:/rooms");
                              var tot = tmp.getNodeCount();
                              for (var i = 1; i &lt;=tot; i++)
```

```
                                {
                                        // Adds an empty "room" node to the "list" dataset
                                        tmp1.setXPath("list:/");
                                        tmp1.addNode("room", "", "");

                                        // Adds room ID
                                        tmp1.setXPath("list:/room["+i+"]");
                                        tmp.setXPath("data:/rooms/room["+i+"]/id");
                                        tmp1.addNode("id", tmp.getNodeText(), "");

                                           // Adds pairs "field-name/value"
                                           tmp.setXPath("data:/rooms/room["+i+"]");
                                        var tot2 = tmp.getNodeCount()-1;
                                        for (var j = 1; j &lt;= tot2; j++)
                                        {
                                                tmp.setXPath("data:/rooms/room["+i+"]/field["+j+"]");
                                                tmp1.setXPath("list:/room["+i+"]");
                                                tmp1.addNode(tmp.getNodeAttribute("label"),
tmp.getNodeText(), "");
                                        }
                                }
                        </method>

                        <!--
                          The user modifies values in the middle form of "manage_room" tabpane,
                          then presses "Update" button and this method is fired.
                          The "Update" button is enabled only when the user selects one of
                          the items in the "room list" as the currentrly selected room.
                        -->
                          <button name="btnUpdate" text="Update" x="500" y="5" enabled="false">
                                <method event="onclick">

                                        // Updates the dataset, i.e. transfers data from UI back to
dataset)
                                        parent.body.room.roomData.datapath.updateData();

                                        // Reads ID of the currentrly selected room
                                        tmp.setXPath("currentlySelectedRoomData:/roomType/id");
                                        var id = tmp.getNodeText();

                                        // looks up for the specific room in "data" dataset
                                        tmp.setXPath("data:/rooms");
                                        var tot = tmp.getNodeCount();
                                        var i = 1;
                                        tmp.setXPath("data:/rooms/room["+i+"]/id");
                                        while ((i &lt;= tot) &amp;&amp; (id != tmp.getNodeText()))
                                        {
                                                i++;
                                                tmp.setXPath("data:/rooms/room["+i+"]/id");
                                        }

                                        // Deletes previous version of the room from "data" dataset
                                        tmp.setXPath("data:/rooms/room["+i+"]");
                                        tmp.deleteNode();

                                        // Adds updated room data to "data" dataset
                                        tmp.setXPath("currentlySelectedRoomData:/roomType");
```

```
                                        tmp1.setXPath("data:/rooms");
                                        tmp1.addNodeFromPointer(tmp);
                                        tmp1.setXPath("data:/rooms/roomType");
                                        tmp1.setNodeName("room");

                                        // Rebuilds room list and sets the currently selected room
                                        canvas.setCurrentlySelectedRoomData();
                                        parent.doCreateRoomList();
                                </method>
                        </button>

                        <!--
                          The user presses the "Update" button and this method is fired,
                          the currently selected room, shown in the middle form of
                          "manage_room" tabpane, is removed from room list.
                          The "Update" button is enabled only when the user selects one of
                          the items in the "room list" as the currentrly selected room.
                        -->
                          <button name="btnDelete" text="Delete" x="500" y="35" enabled="false">
                                <method event="onclick">

                                        // Reads the ID of the currently selected room
                                        tmp.setXPath("currentlySelectedRoomData:/roomType/id");
                                        var id = tmp.getNodeText();

                                        // Looks up in "data" dataset the room with that ID
                                        tmp.setXPath("data:/rooms");
                                        var tot = tmp.getNodeCount();
                                        var i = 1;
                                        tmp.setXPath("data:/rooms/room["+i+"]/id");
                                        while ((i &amp;lt;= tot) &amp;amp;&amp;amp; (id != tmp.getNodeText()))
                                        {
                                                i++;
                                                tmp.setXPath("data:/rooms/room["+i+"]/id");
                                        }

                                        // Remove the room from "data" dataset
                                        tmp.setXPath("data:/rooms/room["+i+"]");
                                        tmp.deleteNode();

                                        // Rebuilds room list and sets the currently selected room
                                        canvas.setCurrentlySelectedRoomData();
                                        parent.doCreateRoomList();
                                </method>
                        </button>

                        <button text="Add custom field" x="500" y="65"
    onclick="canvas.dlgAddField.animOpen.doStart()" />

                        <view name="body">
                                <simplelayout axis="x" spacing="30" />

                                <view name="roomList">
                                        <simplelayout axis="y" />
                                        <text name="labelRoom">Room List</text>
                                        <list name="roomList">
```

```
                                                    <!--
                                                    This method is invoked when the user selects one
of the
                                                    items in the "room list" listbox on the left of
the
                                                    "manage_room" tabpane.
                                                    Once selected, the form in the middle of the
"manage_room" tabpane
                                                    is filled with values from the currently selected
room.
                                                    At the end of this method, "Update" and "Delete"
buttons are enabled.
                                                    -->
                                                <method event="onselect">
                                                        // Looks up for the room using the selected ID
                                                        tmp.setXPath("data:/rooms");
                                                        var tot = tmp.getNodeCount();
                                                        var i = 1;
                                                        tmp.setXPath("data:/rooms/room["+i+"]/id");
                                                        while ((i &lt;= tot) &amp;&amp;
(tmp.getNodeText() != this.getValue()))
                                                        {
                                                               i++;
        tmp.setXPath("data:/rooms/room["+i+"]/id");
                                                        }

                                                        if (i &lt;= tot)
                                                        {
                                                                    // Copies data from selected room
TO "currentlySelectedRoomData" dataset
                                                                    //
                                                               canvas.setCurrentlySelectedRoomData();
        tmp.setXPath("currentlySelectedRoomData:/roomType");
        tmp1.setXPath("data:/rooms/room["+i+"]");
                                                               var tot_old = tmp1.getNodeCount();
                                                               var tot_new = tmp.getNodeCount();
                                                               tmp.deleteNode();
        tmp.setXPath("currentlySelectedRoomData:/");
                                                               tmp.addNodeFromPointer(tmp1);
        tmp.setXPath("currentlySelectedRoomData:/room");
                                                               tmp.setNodeName("roomType");

                                                               // Adds field values to
"currentlySelectedRoomData" dataset
                                                               //
                                                               for (var j = tot_old; j &lt; tot_new;
j++)
                                                               {
        tmp.setXPath("currentlySelectedRoomData:/roomType");
        tmp1.setXPath("data:/roomType/field["+j+"]");
                                                                   tmp.addNodeFromPointer(tmp1);
                                                               }
                                                        }
```

```
                                                    // Enables "Update" and "Delete" buttons
                                                        parent.parent.parent.btnUpdate.setAttribut
e("enabled", true);

        parent.parent.parent.btnDelete.setAttribute("enabled", true);

                                                </method>

                                                <textlistitem name="datasetBoundedRoomList"
datapath="list:/room" text="$path{'Room Number/text()'}" value="$path{'id/text()'}" />
                                        </list>
                                </view>

                                <view name="room">
                                        <simplelayout axis="y" spacing="5" />

                                        <view name="roomData">
                                                <datapath
xpath="currentlySelectedRoomData:/roomType/*" />
                                                <simplelayout axis="x" />
                                                <view>
                                                        <text datapath="@label" resize="true"
valign="middle">
                                                                <method event="oninit">
                                                                        // Displays the
appropriate UI widget,
                                                                         // "Text" and "Yes/No" are
currentrly supported
                                                                         //
                                                                        if
(parent.parent.datapath.getNodeName() == "id")
                                                                        {

        this.setAttribute("visible", false);
                                                                        }
                                                                        if
(parent.parent.datapath.getNodeAttribute('type') == 'Text')
                                                                        {

        parent.parent.roomField.string.setAttribute('visible', true);

        parent.parent.roomField.string.setAttribute('width', 230);
                                                                        }
                                                                        else if
(parent.parent.datapath.getNodeAttribute('type') == 'Yes/No')
                                                                        {

        parent.parent.roomField.radio.setAttribute('visible', true);
                                                                        }
                                                                </method>
                                                        </text>
                                                </view>

                                                <view name="roomField">
                                                        <edittext name="string" datapath="text()"
visible="false">
                                                                <method event="ondata">
                                                                        // If a "Text" value is
available, then display it
```

```
                                                                        if
(parent.parent.datapath.getNodeAttribute('type') == 'Text')

        this.setAttribute('datapath', 'text()');
                                                                        else

        this.setAttribute('datapath', '');
                                                                </method>
                                                        </edittext>
                                                        <radiogroup name="radio" datapath="text()"
layout="axis:x" visible="false">
                                                                <method event="ondata">
                                                                        // If a "Yes/No" value is
available, then display it
                                                                        if
(parent.parent.datapath.getNodeAttribute('type') == 'Yes/No')

        this.setAttribute('datapath', 'text()');
                                                                        else

        this.setAttribute('datapath', '');
                                                                </method>
                                                                <radiobutton name="Yes" text="Yes" />
                                                                <radiobutton name="No" text="No" />
                                                        </radiogroup>
                                                </view>
                                        </view>

                                </view>

                        </view>

                </tabpane>

                  <!--
                  The "add_room" tabpane allows the user to add a new room to the "room list".
                  The user selects this tabpane, then fills the form on the left of the tabpane
                  (room number, number of double beds, room description, extra single bed
available),
                  and finally presses the "Add room" button to confirm the action.
                  -->
                <tabpane name="add_room" text="Add Rooms">

                        <!--
                        This method is invoked when the user selects the "add_room" tabpane.
                        On tabpane selection, the input form on the left of the tabpane
                        is prepared with fieldnames read from "data:/roomType",
                        all the values are empty.
                        -->
                       <method event="onselected" reference="this.tab">
                              if (this.tab.selected)
                              {
                                      canvas.setCurrentlySelectedRoomData();
                              }
                       </method>

                        <!--
                        This method is invoked when the user presses the "Add room" button
                        in the "add_room" tabpane. Room values, specified in the input form,
```

```
                        are added to the "room list".
                        -->
                    <button name="btnInsert" text="Add room" x="500" y="5">
                        <method event="onclick">

                            // Updates the dataset connected with the input form
                            parent.room.roomData.datapath.updateData();

                            // Calculates the new ID for the room
                            tmp.setXPath("currentlySelectedRoomData:/roomType/id");
                            tmp1.setXPath("data:/rooms");
                            var tot = tmp1.getNodeCount()+1;
                            tmp.setNodeText(tot);

                            // New room values are added to "data" dataset
                            tmp.setXPath("currentlySelectedRoomData:/roomType");
                            tmp1.setXPath("data:/rooms");
                            tmp1.addNodeFromPointer(tmp);
                            tmp1.setXPath("data:/rooms/roomType");
                            tmp1.setNodeName("room");

                            // Resets the input form reading the values from
"data:/roomType",
                            // all form fields are empty.
                            //
                            canvas.setCurrentlySelectedRoomData();
                        </method>
                    </button>

                    <button text="Add custom field" x="500" y="35"
onclick="canvas.dlgAddField.animOpen.doStart()" />

                    <view name="room">
                        <simplelayout axis="y" spacing="5" />

                        <view name="roomData">
                            <datapath xpath="currentlySelectedRoomData:/roomType/*" />
                            <simplelayout axis="x" />
                            <text datapath="@label" resize="true" valign="middle">
                                <method event="oninit">
                                    // Displays the appropriate UI widget
                                    // "Text" and "Yes/No" are supported
                                    //
                                        if (parent.datapath.getNodeName() == "id")
                                    {
                                        this.setAttribute("visible", false);
                                    }
                                    if (parent.datapath.getNodeAttribute('type')
== 'Text')
                                    {
                                        parent.string.setAttribute('visible',
true);
                                        parent.string.setAttribute('width',
230);
                                    }
                                    else if
(parent.datapath.getNodeAttribute('type') == 'Yes/No')
                                    {
```

```
                                                        parent.radio.setAttribute('visible',
true);
                                                }
                                        </method>
                                </text>
                                <edittext name="string" datapath="text()" visible="false">
                                        <method event="ondata">
                                                // If a "Text" value is available, then
disaply it.
                                                if (parent.datapath.getNodeAttribute('type')
== 'Text')
                                                        this.setAttribute('datapath',
'text()');
                                                else
                                                        this.setAttribute('datapath', '');
                                        </method>
                                </edittext>
                                <radiogroup name="radio" datapath="text()" layout="axis:x"
visible="false">
                                        <method event="ondata">
                                                // If a "Yes/No" value is available, then
disaply it.
                                                if (parent.datapath.getNodeAttribute('type')
== 'Yes/No')
                                                        this.setAttribute('datapath',
'text()');
                                                else
                                                        this.setAttribute('datapath', '');
                                        </method>
                                        <radiobutton text="Yes" selected="true" />
                                        <radiobutton text="No" selected="false" />
                                </radiogroup>
                        </view>
                </view>
        </tabpane>

        <!--
        The "settings" tabpane allows the user to change the background colour
        of the User Service's UI. The tabpane contains only one "listbox" widget,
        with the current list of available colours: Green, Blue and Purple.
        If no specific colour is chosen, then the default Open Laszlo colour
         would be used for the User Service's UI.
        -->
        <tabpane name="settings" text="Settings">
                <simplelayout axis="y" />
                <view name="layoutSettings">
                        <simplelayout axis="x" />
                        <text>Layout userService:</text>
                        <combobox name="layout" editable="false">
                                <textlistitem text="Default" selected="true" />
                                <textlistitem text="Green" />
                                <textlistitem text="Blue" />
                                <textlistitem text="Purple" />
                        </combobox>
                </view>
        </tabpane>
</tabs>
```

```
        <!--
        The modal dialog box named "dlgAddFields", allows the user to add extra pair field/value
        to the room attributes. As an example, the user could be able to add the field "air
conditioning"
        with "Yes/No" as admitted values. Once created, the new field is added to each item in
"data" dataset.
        The new field is added even to "data:/roomType" XML element in "data" dataset.
         -->
         <modaldialog name="dlgAddField" title="Add custom field">
                <animatorgroup name="animOpen" start="false" duration="1000"
process="simultaneous">
                        <animator attribute="opacity" to="1" onstart="canvas.dlgAddField.open()" />
                </animatorgroup>
                <datapath xpath="currentlySelectedRoomData:/newField" />
                <simplelayout axis="y" />

                <view name="viewLabel">
                        <simplelayout axis="x" />
                        <text>New field label:</text>
                        <edittext name="fieldLabel" datapath="label/text()" />
                </view>

                <view>
                        <simplelayout axis="x" />
                        <text>Field type:</text>
                        <combobox name="type" datapath="type/text()" editable="false">

                                <!--
                                "Text" and "Yes/No" data type are available for added fields.
                                -->
                                <textlistitem text="Text" />
                                <textlistitem text="Yes/No" />

                                <!--
                                Display only the UI widget compatilbe with the user choice.
                                If "Text" data type has been chosen, then "string" UI widget
                                is enabled (visible=true) and "radio" widget is disabled
                                (visible=false). Else, vice versa.
                                -->
                                <method event="onselect">
                                     if (this.getValue() == "Text")
                                     {
                                             parent.string.setAttribute("visible", true);
                                             parent.radio.setAttribute("visible", false);
                                     }
                                     else if (this.getValue() == "Yes/No")
                                     {
                                             parent.string.setAttribute("visible", false);
                                             parent.radio.setAttribute("visible", true);
                                     }
                                </method>
                        </combobox>

                        <edittext name="string" enabled="false" visible="false" />

                        <radiogroup name="radio" layout="axis:x" enabled="false" visible="false">
                                <radiobutton text="Yes" selected="true" />
```

```
                        <radiobutton text="No" selected="false" />
                    </radiogroup>
                </view>

                <view>
                    <!--
                    Two buttons available: "Add" and "Cancel",
                    to confirm or to abort the add-new-field operation.
                    -->
                    <simplelayout axis="x" />
                    <button name="btnAdd" text="Add">
                        <method event="onclick">
                            // Checks if the "field label" has been entered by the user
                            if ((parent.parent.viewLabel.fieldLabel.getText() == ""))
                            {
                                parent.parent.alertMiss.open();
                                return;
                            }

                            // Updates the datapath between the dialog and
"currentlySelectedRoomData:/newField"
                            dlgAddField.datapath.updateData();

                            // Add the new field/value pair among the room attributes in
"data" dataset
                            tmp1.setXPath("currentlySelectedRoomData:/newField/label");
                            var label = tmp1.getNodeText();
                            tmp1.setXPath("currentlySelectedRoomData:/newField/type");
                            var type = tmp1.getNodeText();
                            tmp.setXPath("data:/roomType");
                            tmp.addNode("field", "", {label : label, type : type});

                            // Refresh the currently selected room form
                            // and closes the modal dialog box
                            //
                            canvas.setCurrentlySelectedRoomData();
                            parent.parent.close();
                        </method>
                    </button>

                    <!--
                    Nothing happens if the user presses the "Cancel" button.
                    The modal dialog is closed.
                    -->
                    <button text="Cancel" onclick="parent.parent.close()" />
                </view>

                <alert name="alertMiss"><text>Please specify the "field label": is a mandatory
value.</text></alert>

        </modaldialog>

</canvas>
```

9.3 userService.lzx[28]

```
<?xml version="1.0" encoding="UTF-8" ?>
<canvas oninit="doCreateRoomList(); doSelectLayout()" debug="true">
<!-- (MB)
        Security: enable OpenLaszlo to use a specific java classes
-->
        <security>
               <allow>
                      <pattern>org.dbe.kb.qi.utils.ServiceDiscoverer</pattern>
                      <pattern>org.dbe.toolkit.portal.client.DBEPortalClient</pattern>
               </allow>
       </security>
       <include href="./ifLib.lzx" />
<!-- (MB)
     javaRpc: Add the remote call to enable the update feature
-->
<javarpc name="executeService" scope="session" autoload="true"
classname="org.dbe.toolkit.portal.client.DBEPortalClient">

               <method event="onload">
                      Debug.write("Execute service loaded");
        </method>

               <method event="ondata" args="res">
                   Debug.write("Requested SMID:" + canvas.smId);
                      Debug.write("Trying to execute service from url : " + res + " ...");
                      LzBrowser.loadURL(res, '_blank');
               </method>

               <method event="onerror" args="res">
                      Debug.write("Error: Execute service :  #", res);
               </method>

               <remotecall name="getURL" funcname="getUI">
                      <param value="${canvas.smId}"/>
               </remotecall>

       </javarpc>
<!-- javaRpc -->

        <!--
        Canvas-scope variables used to pass the selected room description and number
        to the email message method.
        -->
       <attribute name="selectedRoomNumber"      value="nil" type="string" />
       <attribute name="selectedRoomDescription" value="nil" type="string" />

        <!--
        The "doSelectLayout" is invoked during the initialisation of the User Service application.
        If a colour has been chosen and saved in "data.lzx" settings element, then the
        background colour of the UI is changed accordingly. Otherwise, the Open Laszlo default
        colour is used.
        "data.lzx" is automatically loaded into "data" dataset during application startup.
        -->
```

28 “userService.lzx” is not the mandatory file name, but must differs from Wizard Service's file name.

```
        <method name="doSelectLayout">
               // Sets layout as specified in configuration data in "data:/settings/layout".
                //
               tmp.setXPath("data:/settings/layout");
               var layout = tmp.getNodeText();
                if (layout == "Green")
                       greenStyle.setAttribute("isdefault", true);
               else if (layout == "Blue")
                       blueStyle.setAttribute("isdefault", true);
               else if (layout == "Purple")
                       purpleStyle.setAttribute("isdefault", true);
        </method>

         <!--
         The "doCreateRoomList" is invoked during the initialisation of the User Service
application.
         The method creates a new dataset named "list", containing a list of all the rooms stored
         in the <rooms> element of "data" dataset.
         "data.lzx" is automatically loaded into "data" dataset during application startup.
         -->
        <method name="doCreateRoomList">
               tmp.setXPath("data:/rooms");
               var tot = tmp.getNodeCount();
               for (var i = 1; i &lt;=tot; i++)
               {
                       // adds a room
                       tmp1.setXPath("list:/");
                       tmp1.addNode("room", "", "");

                       // adds room ID
                       tmp1.setXPath("list:/room["+i+"]");
                       tmp.setXPath("data:/rooms/room["+i+"]/id");
                       tmp1.addNode("id", tmp.getNodeText(), "");

                         // Adds pairs "field-name/value"
                         tmp.setXPath("data:/rooms/room["+i+"]");
                       var tot2 = tmp.getNodeCount()-1;
                       for (var j = 1; j &lt;= tot2; j++)
                       {
                              tmp.setXPath("data:/rooms/room["+i+"]/field["+j+"]");
                              tmp1.setXPath("list:/room["+i+"]");
                              tmp1.addNode(tmp.getNodeAttribute("label"), tmp.getNodeText(), "");
                       }
               }
               tmp.setXPath("list:/");
        </method>

        <!-- Includes external dataset  -->
        <include href="./data.lzx" />

        <!-- Temporary Datasets -->
        <dataset name="list" />

        <!-- Dataset pointers -->
        <datapointer name="tmp" />
        <datapointer name="tmp1" />
```

```
        <!-- Layouts -->
        <greenstyle name="greenStyle" />
        <purplestyle name="purpleStyle" />
        <bluestyle name="blueStyle" />

        <!--
        This is the main view of the User Service Open Laszlo application.
        On the left it presents the "room list", when an item is selected by the user
        in the "room list", then room details are shown on the right half of the view.
        The view owns a "Reserve" button which allows the user to send an emain message
        addressed to the Service Provider, the message contains a reservation request.
        -->
         <view name="main">
               <simplelayout axis="y" spacing="50" />

                <!-- (MB)
                     Link to update the userService
                -->
                <text width="110" fontstyle="bold" fgcolor="blue"
                      onmouseover="parent.parent.setBGColor(0xDDDDFF);setAttribute('fontsize',12);
"
                     onclick="canvas.setAttribute('smId',canvas.smId);
                     executeService.getURL.invoke();">
                     <u>Admin: Update Service</u>
               </text>
                <!--
                This is the UI view which diplays the room list.
                Room list is initially read from "data.lxz" file,
                and are kept in memory in a simple structured "list" dataset.
                -->
                <view name="roomList">
                      <simplelayout axis="y" />
                      <text name="labelRoom" fontstyle="bold">Room List</text>
                      <list name="roomList" width="350">

                                <!--
                                This method is invoked when the user selects one of the items
                                in the "room list". The detail about the selected room are
                                displayed in the right half of the "main" UI view.
                                -->
                                <method event="onselect">
                                     // Looks up for the room in the "list" dataset.
                                     // this.getValue() returns the ID of the selected item in the
list,
                                     // it scans all "room" elements in "list" dataset,
                                     // when the IDs match, the element is found.
                                     //
                                     tmp.setXPath("list:/");
                                     var tot = tmp.getNodeCount();
                                     var i = 1;
                                     tmp.setXPath("list:/room["+i+"]/id");
                                     while ((i &lt;= tot) &amp;&amp; (tmp.getNodeText() !=
this.getValue()))
                                     {
                                            i++;
                                            tmp.setXPath("list:/room["+i+"]/id");
                                     }
```

```
                                        // If element is found, then refreshes the selected room
details.
                                        // Finally, enable the "Reserve" button.
                                        //
                                    if (i &lt;= tot)
                                    {
                                            main.roomDetails.fieldData1.setAttribute("datapath",
"list:/room["+i+"]");
                                            main.roomDetails.fieldData2.setAttribute("datapath",
"list:/room["+i+"]");
                                            parent.parent.btnReservation.setAttribute("visible",
true);
                                    }

                                        // Sets value of canvas-scope variables
"selectedRoomNumber" and
                                        // "selectedRoomDescription". Ther are used in building
the
                                        // automatically generated email message.
                                        //
                                        tmp.setXPath("list:/room["+i+"]/Room Number");
                                    selectedRoomNumber = tmp.getNodeText();
                                        tmp.setXPath("list:/room["+i+"]/Room Description");
                                    selectedRoomDescription = tmp.getNodeText();

                            </method>

                            <textlistitem name="roomTextListItem" datapath="list:/room"
text="$path{'Room Description/text()'}" value="$path{'id/text()'}" />
                        </list>
                </view>

                <!--
                 This is the UI view which displays the details about the room selected in "room
list".
                -->
                 <view name="roomDetails">
                        <simplelayout axis="x" />
                        <view name="fieldData1" datapath="">
                                <simplelayout axis="y" />
                                <text name="empty1" fontstyle="bold"></text>
                                   <text datapath="*/name()" resize="true" fontstyle="bold">
                                        <method event="ondata">
                                                // ID is not displayed
                                                if ((this.datapath.getNodeName() == "id") ||
(this.datapath.getNodeText() == "") || (this.datapath.getNodeText() == "null"))
                                                        this.setAttribute("visible", false);
                                        </method>
                                </text>
                        </view>
                        <view name="fieldData2" datapath="">
                                <simplelayout axis="y" />
                                <text name="empty2" fontstyle="bold"></text>
                                <text datapath="*/text()" resize="true">
                                        <method event="ondata">
                                                // ID is not displayed
                                                if ((this.datapath.getNodeName() == "id") ||
(this.datapath.getNodeText() == "") || (this.datapath.getNodeText() == "null"))
```

```
                                                        this.setAttribute("visible", false);
                                        </method>
                                </text>
                        </view>
                </view>

                <!--
                Button "btnReservation", when presses invokes a method that automatically sends an
                email message addressed to the Service Provider (for instance the Bed and Breakfast
owner).
                -->
                 <button name="btnReservation" text="Reserve" x="500" y="10" visible="false">
                        <method event="onclick">
                                var msgBody = "Automatically generated Room Reservation Message.%0A"
+
                                                "Please, submit your data and any other question about
the reservation request.%0A%0A" +
                                                   "Best Regards.%0A" +
                                                   "The Italy Bed and Breakfast Staff.";
                                tmp.setXPath("data:/infoMail");
                                var mail = tmp.getNodeText();
                                var subject = "Reservation request for room: " +
selectedRoomDescription + " (" + selectedRoomNumber + ")";
                                LzBrowser.loadURL("mailto:"+mail+"?subject=" + subject +
"&amp;body="+msgBody);
                        </method>
                </button>
        </view>

</canvas>
```

9.4 IfLib.lzx

```
<?xml version="1.0" encoding="UTF-8" ?>
<library>
        <class name="IfUtility">
                <datapointer name="tmp" />

                <method name="getIfDs">
                <![CDATA[
                        var dataString = data.serialize();

                        var i = dataString.indexOf("<bmlData>");
                        var j = dataString.indexOf("</bmlData>");

                        var bmlString = dataString.substr(i,j-i+10);

                        var fullString =
"<IfDs>"+bmlString+"<serviceData>"+dataString+"</serviceData></IfDs>";

                        return fullString;
                ]]>
                </method>

                <method name="setBmlDataAttribute" args="name, val">
                <![CDATA[
                        tmp.setXPath("data:/bmlData");
```

```
                        var i = 1;
                        var tot = tmp.getNodeCount();
                        tmp.selectChild();
                        while (( i<= tot) && (tmp.getNodeName() != name))
                        {
                                tmp.selectNext();
                        }
                        if (i <= tot)
                        {
                                tmp.setNodeText(val);
                        } else {
                                tmp.addNode(name, val, "");
                        }
                ]]>
                </method>

                <method name="getBmlDataAttribute" args="name">
                <![CDATA[
                        tmp.setXPath("data:/bmlData/"+name);
                        return tmp.getNodeText();
                ]]>
                </method>

                <method name="deleteBmlDataAttribute" args="name">
                <![CDATA[
                        tmp.setXPath("data:/bmlData/"+name);
                        tmp.deleteNode();
                ]]>
                </method>
                 <!-- (MB) Retrieves the wizard SmID from the data repository -->
                 <method name="getwSmId" >
                <![CDATA[
                        tmp.setXPath("data:/WSMID");
                        return tmp.getNodeText();
                ]]>
                </method>
        </class>
</library>
```

10 .Glossary

Term	*Acronym*	*Synonymous*	*Description*
Business Modelling Language	BML		BML is a language through which SMEs can describe their business model in order to enhance mechanism of discovery and selection of e-business partner within the DBE.
Basic Service			Basic Services are DBE services that pertain to the following categories: payments, information carriers. The list could increase further on [DBECoreArch].
Business Process Execution Language	BPEL		BPEL is a language for the formal specification of business processes.
Business Service			The actual service supported by the DBE that is not either structural or basic
Client Side Servent	CSS		"The Servent is a piece of code that can be used as client and server at the same time. Clients will find services and invoke them through the Servent".[http://swallow.sourceforge.net] The CSS is the Servent when used as a Client. The DBE Portal run in the CSS.
Computational Independent Model	CIM		The Computational Independent Model is a viewpoint of a system which focuses on the environment and the requirements for the system, but does not show details of the structure of the system [MDA Guide].
Conceptual Service			A conceptual service is the description of a service that is not related to a service instance.
CRUDEL			Create Read Update Delete List
Data Integrity			The property that data has not been changed, destroyed, or lost in an unauthorized or accidental

Term	Acronym	Synonymous	Description
			manner [ISG].
DBE Portal		DBPortal	The DBE Studio is an Integrated Development Environment (IDE) for the DBE. It is represented by a collection of open-source Eclipse plug-ins that allow business analysts and software developers to model, design, implement, publish, deploy, compose, search and execute DBE services. [DBE Studio]
Distributed Interaction Form			Interaction Form type services created through an IF Wizard. They are not Self Contained Service but they can be distributed over a servent provided by a DBE actor.
FLOSS			Free/Libre and Open Source Software.
Interaction Form			DBE services created and distributed by using an IF Wizard. Usually are simple services targeted to SMEs unequipped of IT staff or financial resources to invest for realizing a service. They include Self Contained Service and Distributed Services.
IF Wizard		IF Wizard Application	Refers to IF Wizard Application
IF Wizard Application			Application, written for instance in OpenLaszlo language, contained in the IF Wizard SM. It is executed by the Service Provider that interactively creates a new Service Instance.
Knowledge Base	KB		Is the part of the DBE system where the DBE knowledge is stored and managed. Such knowledge refers to ontologies, business and service description [KBDI].
Platform Independent Model	PIM		The Platform Independent Model is a viewpoint of a system which focuses on the information about

Term	Acronym	Synonymous	Description
			the specific technology that is used in the realization of a particular platform. A PIM combines the specifications in the PIM with details that specify how that system uses a particular type of platform [MDA Guide].
Platform Specific Model	PSM		The Platform Specific Model is a viewpoint of a system which focuses on the operation a system while hiding the details necessary for a particular platform. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platform of similar type [MDA Guide].
Real Service			"By Real Service we mean a service that exists in the real world: for example the supplier of a real service has a VAT number, a snail mail address or an IP address." [DBECoreArch]
Self Contained Service	SCS		A Self Contained Service is a service containing in the SM its description, code and other possible necessary data for its execution
Semantic Registry	SR		It is the component of the DBE Knowledge Base that hosts the service description published in the DBE environment and available for discovery and consumption.
ServENT			"A peer2peer application container that isolates the programmer from the peer2peer coding complexity. Applications are coded as a POJO (Plain Old Java Object) object without any single acknowledge of distributed programming."[http://swallow.sourceforge.net]

Term	*Acronym*	*Synonymous*	*Description*
Service DNA	SDNA		The SDNA is an element of reuse across services. It is the conceptual definition of a service when not related to any real service; it represents an element of reuse (aka *Service Definition*).[DBEArchReq]
Service Instance			It is the IT implementation of a Real Service.
Service Manifest	SM		All the specifications that describe an instance of a real service from the computing and business viewpoint.
Service Manifest Identifier	SMID		A code which univocally identifies the SM
Service Manifest Owner			The entity who owns the intellectual propriety of the SM.
Service Manifest Registrar			The person who publishes the SM in the Semantic Register. He/she is the only authorized to edit the SM. He/she might correspond to the SM Owner.
Service Template			An alias for IF Wizard
Standard “de facto”			An extensive consensus on a particular choice which has not been ratified by any official standards body, but which has a large market share.
User Service			This concept is used in the context of SCS and indicate the Istance Service generated by the IF Wizard.
Yellow Page Service	YPS		A very simple service only showing some data that publish the real service. It does not define any interaction with the user.

11 .References

D16.2SMSM: Valentino Trentin, Giulio Montanari, Umberto Pernice, D16.2: SDL specification for the DBE Platform - Part 2 Service Manifest Software Model full definition, 2006
DBEArchReq: Pierfranco Ferronato, DBE Architecture Requirements Ver 2.2, August 2004, Soluta.net
D15.5BMLED: TUC, BML Data Editor Final Release, 2006,
D20.8BMLDATA: Victor Bayon (UCE), Workpackage 20: DBE User Interfaces Deliverables D20.8: BML Data Editor/BMLWizard, 2006,
D18.5SMED: Giulio Montanari, Umberto Pernice, WP 16: Service Description Language D18.5: Implementation of the SM Editor, ,
D26.6PORTAL: Intel Ireland Ltd., Workpackage WP26: DBE Portal Deliverable D26.6: DBE Portal Specification, 2006
M24.9IDT: Dominik Dahlem - TCD, Final release of Distributed Identity Framework and Service,
D36.2ACC: FRANK WALSH - WIT, D36.2: A Set of Accounting SoftwareBuilding Blocks, 2005,
D36.3ACC: Paul Malone, Frank Walsh, Brendan Jennings - WIT, State of the Art in Accounting for Composed Services, 2005,
DBECoreArch: Pierfranco Ferronato, DBE Core Architecture Ver 01.3, 2004/06/04, Soluta.net
http://swallow.sourceforge.net: ,
MDA Guide: OMG, MDA Guide Version 1.0.1, June 2003, Object Management Group
ISG: Internet Security Glossary, http://www.faqs.org/rfcs/rfc2828.html
DBE Studio: David McKitterick, DBE Studio Integration, 2006, INTEL
KBDI: TUC/MUSIC, Knowledge Base Design and Implementation Status Ver 0.1, October 15, 2004, TUC/MUSIC

- end of document -