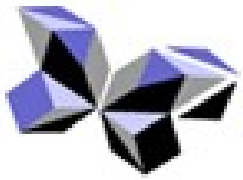

Digital Business Ecosystem

Contract n° 507953

WP16: Service Description Language

D16.2: SDL specification for the DBE Platform

Part 2 – Service Manifest Software Model full definition

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

Contract Number: **507953**

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D 16.2 - part 2

Due Date: July 2006

Delivery Date: August 2006

Short Description: This document provides a full description of the Service Manifest Conceptual Model and the Service Manifest Software Model.

Author: Soluta.net

Partners contributed: Soluta.net

Made available to: Public

Versioning

Version	Date	Author, Organisation
00.01	22/03/2006	Giulio Montanari, Umberto Pernice – Soluta.net
00.02	18/06/2006	Valentino Trentin, Giulio Montanari, Umberto Pernice – Soluta.net
01.00	30/08/2006	Valentino Trentin, Giulio Montanari, Umberto Pernice – Soluta.net

Quality check:

1st Internal Reviewer : Miguel Vidal – SUN Microsystems

2nd Internal Reviewer: Angelo Corallo – ISUFI

Table of Contents

1 .EXECUTIVE SUMMARY....6
2 .INTRODUCTION....7
3 .SERVICE MANIFEST FEATURES....8
3.1 Functional Features....8
3.2 Extra Functional Features....8
4 .CIM AND PIM MODELS OF THE SERVICE....9
5 .SERVICE MANIFEST AND CONCEPTUAL SERVICE (AKA SDNA)....11
5.1 Conceptual Service Model: the Service DNA (SDNA)....11
5.2 Evolutionary aspect of SDNA and "standard de facto"....12
6 .SERVICE MANIFEST....14
6.1 Service Manifest Structure....15
6.1.1 SMID....15
6.1.2 BML & SDL Models....15
6.1.3 Version Number/AncestorSMID/RootSMID....16
6.1.4 Publishing date....17
6.1.5 Last Change Date....18
6.1.6 RegistrarID....18
6.1.7 ServiceType....18
6.1.8 SM Availability/State....19
6.1.9 BML Data....20
6.1.10 Interaction Form....20
6.1.11 BPEL Model....20
6.2 Virtual Data....21
6.2.1 Fitness Data....21
6.2.2 Quality of Service (QoS)....22
6.2.3 Proxy....22
6.3 Service Manifest Definition....22
6.3.1 SM Class Diagram....25
7 .SERVICE MANIFEST LIFE CYCLE....26
7.1 SM States....27
7.2 SM Transitions....28
8 .SM SCOPE/VISIBILITY....29
9 .SM EDITING AND VERSIONING....30
9.1 BML & SDL changes....32
9.2 ServiceType changes....32
9.3 BML Data (M0) changes....33
9.4 Interaction Form changes....33
9.5 Virtual Data changes....33
9.6 BPEL Model changes....33
9.7 Versioning....33
10 .SEMANTIC REGISTRY....35
10.1 Service Manifest Security....35
10.2 Tracing....36
10.3 SM Browse and Discover....36
10.4 CRUDEL Operations....36
10.5 SMID Creations....36
11 .SERVICE MANIFEST, EVE AND FITNESS DATA....37
11.1 SDNA, SM and Fitness Data....37

12 .SERVICE FACTORY – DBE STUDIO....38
12.1 SM Publishing and Editing....38
12.1.1 SM Editor....38
12.1.2 DBE Service Publishing Perspective....38
13 .INTERACTION FORM (IF) AND SELF-CONTAINED SERVICE (SCS)....39
13.1 Self-Contained Service: the simplest way to join the DBE....39
13.2 Interaction Form: “DBE for Dummies”....39
13.3 The Interaction Form creation/build cycle....40
13.4 Service Template or IF Wizard....41
13.4.1 SelfContainedService.User Service Code....43
13.4.2 WizardData.BML Model....43
13.4.3 BML Data Template....43
13.4.4 Service Template Supplier....43
13.5 Service Provider....44
13.6 Customer....44
13.7 Changes required by introducing the Interaction Form/Self-Contained Service....45
13.7.1 Divide SM Creator into two java projects....45
13.7.2 Change the Service Manifest structure....45
13.7.3 Modify the SM Editor in order to support self contained interaction forms....45
13.7.4 Adapt the DBE Portal component....45
13.8 Distributed Interaction Form....46
13.9 Limitations of the Interaction Form and the Self-Contained Service....46
14 .GLOSSARY....47
15 .REFERENCES....52

List of Figures

FIGURE 1 - SERVICE MANIFEST LOCICAL VIEW....14
FIGURE 2 - SERVICE MANIFEST VERSION AND ANCESTOR....17
FIGURE 3 - DBE SERVICE TYPE....19
FIGURE 4 - SERVICE MANIFEST SCHEMA....23
FIGURE 5 - SERVICE MANIFEST XSD....24
FIGURE 6 - SERVICE MANIFEST CLASS DIAGRAM....25
FIGURE 7 - SERVICE MANIFEST LIFE CYCLE....27
FIGURE 8 - SERVICE MANIFEST CHANGES....30
FIGURE 9 - SERVICE MANIFEST VERSIONING....31
FIGURE 10 - SERVICE MANIFEST VERSIONING POLICY....32
FIGURE 11 - SERVICE MANIFEST SECURITY....35
FIGURE 12 - INTERACTION FORM LIFE CYCLE....41
FIGURE 13 - SELF-CONTAINED SERVICE AND IF WIZARD....42

1 .Executive Summary

This document defines the model of the Service Manifest (SM) that is the complete representation of a DBE Service. The SM contains all the information required to describe, retrieve and execute a service in a DBE.

Due to the adoption of the MDA[1] approach, the information contained in the SM is structured using a set of models which define services from two different viewpoints[2]: a business[3] and a technical viewpoint[4]. The business viewpoint is modelled using the BML[5] which describes the business model of the supplier and of the service itself. The technical viewpoint is modelled using the SDL[6] which describes the technical interface of the service at a level of abstraction that is independent of any technical implementation. This latter representation of the service addresses the computational structure of the service: the information described using the SDL will be used to generate the Java code[7].

Services represented by the SM can be a Simple Service or a Service Chain, where the latter is a composition of services that is expressed by a language called BPEL[8].

The Service Manifest can be logically divided into two parts:

- the SDNA which represents a Conceptual Service[9], i.e. the type of the service;
- information about the supplier and data related to the service itself like the price schema, location, discount percentage, i.e. the actual information, the instance of the service.

All the models contained in the SM include links to ontologies because these references allow the discovering, reasoning and the comparison between different services.

Syntactically, the SM is represented by an XML[10] file and the models contained are stored using an MOF[11]-XMI[12] representation.

Finally, two particular services, the Self-Contained Service (SCS) and the Interaction Form (IF), are supported in the SM with the purpose of facilitating the creation and use of DBE services, particularly for those SMEs which do not have a sufficient IT infrastructure.

1 Model Driven Architecture [MDA].
2 In MDA, usually they are three viewpoints; the third is the platform specific model (PSM) [MDA]. In DBE the SM is platform independent, for this reason it is not present.
3 Computation Independent Model (CIM) [MDA].
4 Platform Independent Model (PIM) [MDA].
5 Business Modelling Language
6 Service Description Language
7 Java is the chosen technological platform, at the moment.
8 Business Process Execution Language
9 The SDNA represents the conceptual definition of a service when it is not related to any real service.
10 Extensible Markup Language [XML].
11 Meta-Object Facility [MOF].
12 XML Metadata Interchange [XMI].

2 .Introduction

This document represents the part II of D.16.2 “Specification for the DBE Platform”. It provides a wider coverage of the objectives specified in the task C26 of the DBE project, producing the overall consolidation of the SM related documents as suggested by the EU Commission during the second project review [DBE 2nd Review Report].

This Deliverable defines the model of the Service Manifest, specifically regarding its structure and it describes the SM development from both the CIM and the PIM viewpoint. In addition to the first version of the SM (contained in [D16.1 part II: SM Conceptual and Software Models]), this deliverable now includes and specifies the following topics:

- the use of the SDNA within the SM and not as a separated entity;
- a detailed description of the Self Contained Service;
- a detailed description of the Interaction Form;
- the support for the Semantics of Business Vocabulary and Business Rules models (named SBVR) in addition to the traditional BML approach.

Furthermore, an important modification concerns the overall organization of this deliverable. D.16.2 part II now illustrates the state of the art of the Software Manifest and does not include any more the assumptions, alternative proposals and decisions to be undertaken contained in the previous deliverable [D16.1 part II: SM Conceptual and Software Models], but integrates and closes any issue related to the SM formerly marked as open or “to be discussed” in the previous versions.

3 .Service Manifest Features

The SM is a representation of a service and, as a consequence of that, it has to evolve any time new service features are added in the DBE. This has recently happened, for example, with the introduction of two particular services: the Self-Contained Service (SCS) and the Interaction Form. The XML based structure of the SM, together with versioning support, helps this evolution which is also a key feature of the DBE itself.

The two paragraphs here below indicate its key features, divided into functional and extra-functional.

3.1 Functional Features

Feature name	***Description***
Storing CIM Model	The SM is a composition of CIM Models of the service.
Storing PIM Model	The SM is a composition of PIM models; SDL is used.
Storing service information (the actual service specific data)	The SM also contains service specific data , such as firm location, name, availability, price schema, discount policy and so forth.
Allow service composition	A service can be a Simple Service or a Service Chain. The SM enables the creation of Service Chain or services composition.
Storing data and code for self-contained services.	A Self-Contained Service (AKA Interaction Form and Yellow Pages) is a service which also contains code and data are necessary to the execution of service itself. The SM allows the representation of all this kind of services.
Importing/Exporting models in XMI format	The SM enables the models import/export in XMI format.
Unique identification	Each SM has an unique identification key.

3.2 Extra Functional Features

Feature name	***Description***
XML format	The SM is a pure XML document (not an XMI1.2/MOF1.4.)
Tamper proof	The SM is modifiable only by the creator/owner. Any attempts to change it from unauthorized users is blocked. This feature is related to the DBE Security Architecture and schema, the SME supports it.
Traceable	Any modification to the SM is traced and the author identified. This feature is related to the DBE Security Architecture and schema, the SM supports it.
Identify author of service	It is always possible to identify who has created or published the service. This feature is related to the DBE Security Architecture and schema, the SM supports it.

4 .CIM and PIM Models of the service

The MDA approach states that there are three viewpoints from which any system (or subsystem) should be described in this project, and three related models, as follows:

- the Computation Independent Model (or CIM);
- the Platform Independent Model (or PIM);
- the Platform Specific Model (or PSM).

The CIM describes the business from a computation independent viewpoint. The PIM describes the system in a computational manner, but without any reference to any existing technological platform. The PSM represents the executable form of the system for a specific platform. There might be more PSMs for the same PIM, each one running on a different platform.

The SM is described from these different viewpoints, each of these descriptions being written using specific languages. These languages have to be different because they address different dimensions of the service and different kind of people which approach the system from different perspectives and fields of expertise.

The business representation of the SM, or CIM, is written by using the BML and the SSL[13] or the SBVR[14], while the technical interface of the service, or PIM, is written using the SDL. The PSM representation of the SM is automatically generated from the PIM and is not contained in the SM, this generation in charge of the Service Factory. Other representations or implementations of PSM can be lately provided in the future without affecting the SM structure.

This document does not aim at describing these languages, the reader can refer to [DBECoreArch] for an overview of the CIM and PIM concepts, to [BMLMM] for information about BML, to [KBDI] for information about SSL, to [SBVR Editor & BML framework] for information about SBVR and to [SDL] for information about SDL.

The decision to provide the SM with some degree of autonomy determines the consequence that any SM will contain both its CIM and PIM. Thus that the business model written in BML and SSL and the technical interface of the service written in SDL, are all included in the definition of a service and not simply linked to the SM which therefore becomes a self-contained structure.

All the models representing the Conceptual Model of a service are stored in the Model Repository (that is a part, or a view, of the Knowledge Base) for further reuse. An exception is the PSM which is not stored in the Model Repository, because it will be automatically generated by the Service Factory. When a DBE user intends to deploy a new service, it is strongly recommended to reuse an existing model either for the business models and for the technical one: reuse of models promotes the emergence of de-facto standard services. Refer to chapter 5.2 - Evolutionary aspect of SDNA and “standard de facto” for further discussion about the reuse of standard services.

13 Semantic Service Language. See [DBECoreArch].

14 Semantics of Business Vocabulary and Business Rules

A sequence of operations for creating a service includes, at least, the following steps:

- firstly, it should be created the SDNA of the service (meant as the Conceptual Model) by creating the business model of the service using BML and its technical interface using SDL;
- then the service should be “instantiated” by setting the particular XML file which represents the BML Data that describe the service [BMLMM]. We recall that a separate component, the BML Data Editor, is in charge of supporting the user in creating such information from the BML model.

5 .Service Manifest and Conceptual Service (AKA SDNA)

The most relevant modifications to the SM structure developed during the second phase of the DBE project, and reflected into the transition from D.16.1 to D.16.2, is the elimination of the separation between the SM and the SDNA with the result of having all the information defining a service directly contained in the SM. It can be mentioned that in the first phase of the DBE project the SDNA was described as the conceptual model of the service while the SM as an instance of that service which is directly related to a specific supplier and a specific implementation. Actually in the SM the distinction between SDNA and SM only remains a pure logical distinction. In other words the SDNA just represents a logical concept, a synonym of the Conceptual Service and it is not any more physically stored or managed inside the SM itself (as it was in the previous version of the SM in [D16.1 part II: SM Conceptual and Software Models].Such a decision has been taken to simplify the operational activity of SMEs.

The [DBECoreArch] here can be useful to focus on the difference between the SM/Service Instance and the SDNA/Conceptual Service.

5.1 Conceptual Service Model: the Service DNA (SDNA)

"The SDNA represents the conceptual definition of a service when it is not related to any real service. It is the pair: SDL and BML model" [DBECoreArch].

The SDNA role is well defined in [DBECoreArch] by this sentence: "*The SDNA is an element of reuse across services. It is the conceptual definition of a service when not related to any real service*".Detailed information regarding BML/SSL and SBVR, for the CIM model and SDL for the PIM model can respectively be found in [BMLMM], [SBVR Editor & BML framework], [SDL], however, a brief description of such models is provided here below in order to allow a broader comprehension of the Conceptual Service Model.

The Business Modelling Language allows the representation of business information such as: service offered and requested, resources, processes, business models, policies, agreements, location, event related to business and so on. The main purpose of BML in DBE is to enrich DBE distributed computing architecture creating a new description layer, focused on business knowledge. The adoption of the BML model will make it possible to go further from an architecture centred on service description toward a business model description, focused on business content and business features with the advantage of allowing SMEs to interact as they really do in actual business transactions.

The SBVR model is an alternative notation for representing BML models used for the two main purposes of enabling the construction of business vocabularies and capturing business rules. An SBVR business vocabulary contains both terms defining business elements and expressions describing structural or behavioural aspects. A SBVR-based vocabulary provides in this way the capability to connect concepts that are of interest for the organization and to organize them, creating taxonomies or categorization schemes.

This mechanism allows to create a well-defined structure enabling navigation and search capabilities among different concepts.

The SDL model, representing the technical (but technology independent) interface of the service, enables in DBE the automatic composition of services. For this purpose the SDL has to support semantic descriptions of services.

It is important to evidence that, in the transition from D.16.1 to D.16.2 of the SM, the way to represent the CIM of the service has changed, and now it can be represented both as a BML model and as a SVBR model. The possibility to contemporaneously insert both models into the SM has been provided to enable the use of the SBVR and to give more freedom to the service definition. This could produce some operational difficulties (for example, the automatic creation of the SDL, at the moment, is only defined for BML), but it will facilitate the SM evolution allowing the use of new languages to define services. The modular structure of DBE Studio[15], based on Eclipse plug-ins, will allow to easily add new plug-ins to generate, for example, the SDL starting from the SVBR.

Another important consideration to be highlighted is that, at present, the SBVR physically contains both models and data. This characteristic makes less feasible the idea to distinctly separate models from data and it represents another contribution to the decision of not considering SDNA and SM as two separate entities, although, as previously said, from a logical point of view such a separation still remains.

5.2 Evolutionary aspect of SDNA and “standard de facto”

As experienced by the Fetish Project[16], an European Research project strongly related with the DBE project, it is extremely hard to define a reference model covering all the possible services for any business domain. It might be possible for a single business domain, like the reference model defined by the Open Travel Alliance (OTA)[17] for the Tourism domain, but is not possible for an entire digital ecosystem like the DBE. In this last case, any attempt to define and impose a standard would have very few chances to succeed, because of the lack of the resources needed to maintain it and because it would be extremely hard to have all the community to agree on a single reference model. However this concern does not reduce the importance of standard descriptions for DBE services. The need to define standard services is clearly well known, the example of OTA for travel services is emblematic in this direction; therefore the concept of “standard de facto” has been introduced and considered as a relevant issue in DBE.

The idea of “standard de facto” comes out from the evolutionary aspect of SDNA. If a particular SDNA gets reused by many SM, it will become a “standard de facto” SDNA (a sort of “winning SDNA”); conversely, if no SM will reuse a SDNA, it will be ignored by the recommendation process and it will be forgotten in the KB.

The emergence of standard SDNAs can be crucial for the DBE environment because it eases the service composition, the recommendation process and the interoperability. In

15 The DBE Studio is an Integrated Development Environment (IDE) for the Digital Business Ecosystem (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].

16 See http://www.fetish.t-6.it/

17 Open Travel Alliance. See http://www.opentravel.org/

fact, the peculiarity of the DBE approach is the use of the “standard de-facto” idea: DBE does not aim at defining all the standard services in all the involved business domains, but only to discover and recommend the service descriptions defined by the DBE community under the pressure of the market.

If a SDNA becomes a “standard de-facto” SDNA, its evolution will be faster than the evolution of a less used SDNA because more SM will use the SDNA and more improvements will be done on it. It is probable that from a standard SDNA new SDNA standards will rise.

While it is very important for the success of DBE to allow the emergence of some “standard” SDNAs, it will be somehow dangerous for a SM (a service of a specific company) to become a “standard”, because it means that there is a trust and this situation may slow down the evolution of the service. This leads to an interesting analogy: in nature if an organism becomes a winner, it does not evolve any more because it finds easier to change its ambient rather than change itself. However this may represent another new definition of the “standard de facto”. Another possible dangerous situation is when a SM overgrows and it blocks all its competitors without offering to its users further improvements. However this risk should be overcome with the use of open standards. As the scale-free networks[18] can demonstrate, there will be some SDNAs that will emerge and dominate their environment while other SDNAs will still exist but will have a smaller distribution compared with the “winners”. Such “points of public trust” will self emerge and even disappear according with the needs of the community which shape their environment.

18 Rif. Scale Free Network. See http://en.wikipedia.org/wiki/Scale-free_network

6 .Service Manifest

The SM represents a Real Service[19]. “carrying” both its definition and data needed to fulfil its purpose. It contains all the specifications that describe the real service from the computing and business viewpoints.
The Figure 1 - Service Manifest Locical View here below, shows the SM data structure and a description can be found in chapter 6.1 - Service Manifest Structure. The figure below represents only the principal aspects of the SDNA, but not all its internal details.

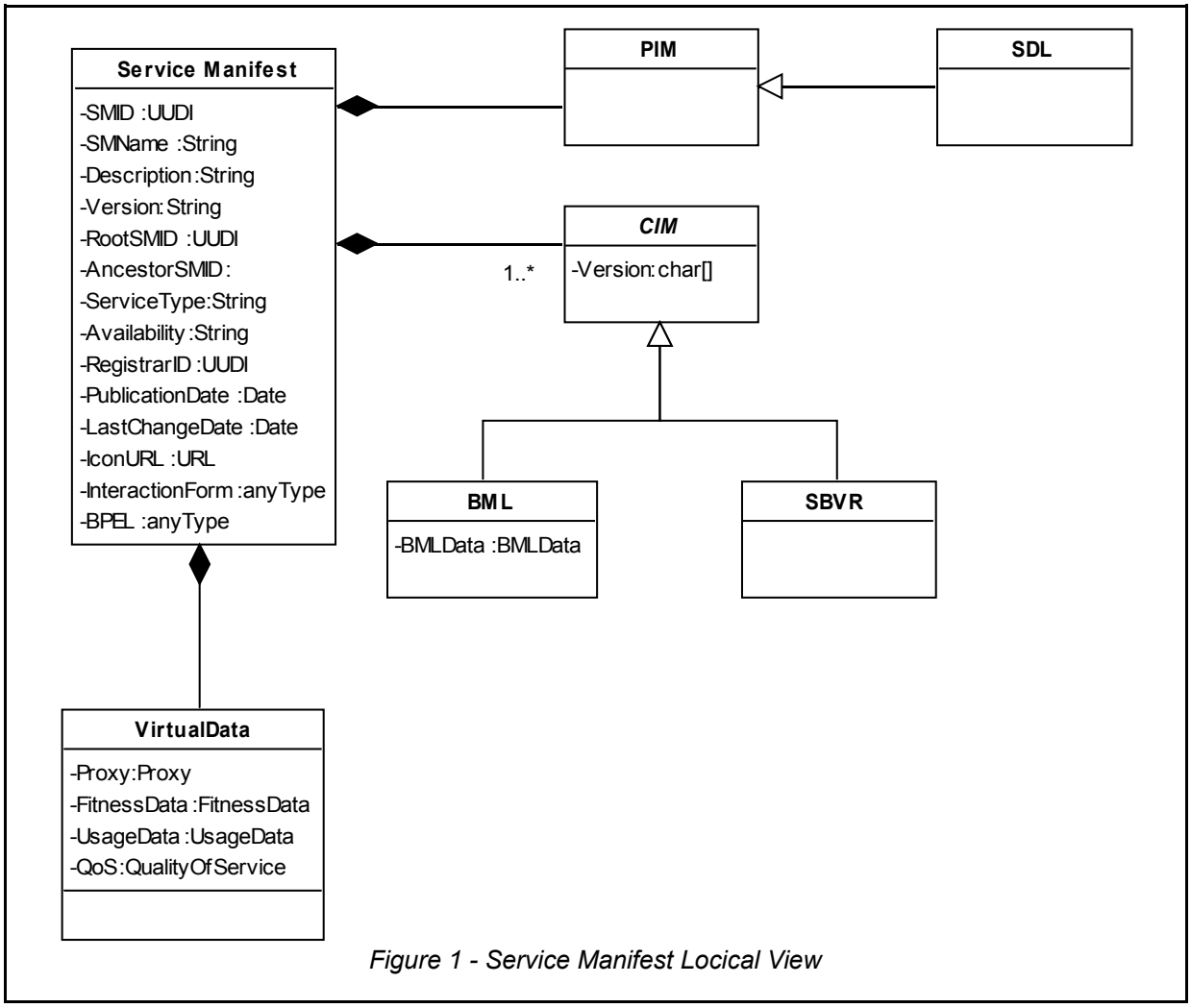

Figure 1 - Service Manifest Locical View

In the SM both the models and the data are stored. However, the structure of the data section will be defined by different DBE tasks[20] and it depends on the CIM used. Further details on the logical composition of the SM are provided in Figure 6 - Service Manifest Class Diagram.

19 A service that really exists in the real world. See [DBECoreArch].
20 For example BML Data is part of the Business Modelling Language.

At the moment the role of the SBVR in the DBE is still an on going process that will be completed far beyond the project life span itself. At any rate, this will not significantly compromise the SM structure, and in all of cases, only the formats are assumed: while models will be encoded in XMI 1.2/MOF 1.4, data, on the other hand, will be in XML. Regarding the SM definition, although it should not be obligatory that all the models are expressed in XMI 1.2/ MOF 1.4, such a possibility is considered as an important decision in order to allow the storage and elaboration of models in a common manner. The models used by a SM are copied from the KB (it is a composition, not a link, i.e. models are actually copied in the SM) and included in the SM. In this way the SM is self-contained and the dependency with the KB is avoided at run-time.

It is defined the possibility to store the BPEL into the SM and this is different from what happens for the Fitness Data[21] (or FD), which are not contained into the SM. The idea of storing the BPEL into the SM reflects the aim of making the SM a “self contained structure”. This means that all the information needed to execute a service (including service description, code for its execution, other necessary data) have to be copied in the SM. However, as lately discussed in paragraph 6.1.11 BPEL Model, the BPEL storage policy makes this possibility as optional, and not as mandatory, leaving to the SMEs the decision to publish or not the BPEL. Concerning FD, they will not be stored in the SM. Where FD will be stored is not an issue of this document, but, from discussions with the EvE group, the FD will be stored in the EvE. Further considerations on FD and the related storage policy can be found in paragraph 6.2.1 Fitness Data.
Virtual Data are data related to services that are logically and not physically contained in the SM; refers to paragraph 6.2 - Virtual Data for further information.

Since the SM is an XML document the natural way to define it is to supply an XML Schema. The Figure 5 - Service Manifest XSD at page 24 represents the SM in the XSD Model.

6.1 Service Manifest Structure

6.1.1 SMID

The SMID identifies the Service Manifest and it allows the identification of the related data (proxies, fitness data etc.). To identify all the proxies which implement it, the SMID is stored as a foreign key in the proxies themselves. It is important to remember that an SM can have more than a single Service Instance; therefore the SMID identifies not just a single proxy but all the proxies implementing the SM. A proxy id may be used for registering/unregistering proxies in FADA. As stated above, it is not possible to keep the proxy id in the SM for two reasons: many proxies may implement a single SM and the ProxyId changes every time the proxy is registered in FADA. The SMID is provided by the SR during the storage of the SM into the SR. It is a task of the SR to provide a unique SMID.

6.1.2 BML & SDL Models

The BML Model and the SDL Model are stored in the SM in XMI 1.2/MOF 1.4 format. For a description of BML, SBVR and SDL refer to chapter 5.1 Conceptual Service Model: the Service DNA (SDNA) .

21 Fitness Data express how fitted is a service for a business. FD are dealt by the EvE.

6.1.3 Version Number/AncestorSMID/RootSMID

One important SM non functional feature is traceability; once the SM is published all the relevant changes are traced, particularly those information of the SM that change at run-time (for instance, the M0 level of the MDA abstraction layers[22]). If the model related information is changed a new version of the SM must be published and traced.
The problem is detailed in chapter 9 - .SM Editing and Versioning.

- **Version Number:** it identifies the version of the SM. It is a free data, and it must be the SM provider who decides its format and meaning. It is suggested a format like xx.yy where xx represents the major release and yy the minor release. The DBE does not impose any specific format on it, but it is suggested to change the major release when SDNA changes. Since the Version Number is unstructured, it can not be used to identify the previous version of the SM, for this purpose the AncestorSMID is used.
- **AncestorSMID:** it represents the previously published SM version. If the SM is the first SM on the top of the chain, the AncestorSMID is omitted.
- **RootSMID:** it represents the first SM in the ancestors chain. It is useful to find all the SMs with a common origin. If the SM is the first SM on the top of the chain, the RootSMID is set to SMID value.

22 Further information on MDA layers for the DBE project can be found on [DBECoreArch]

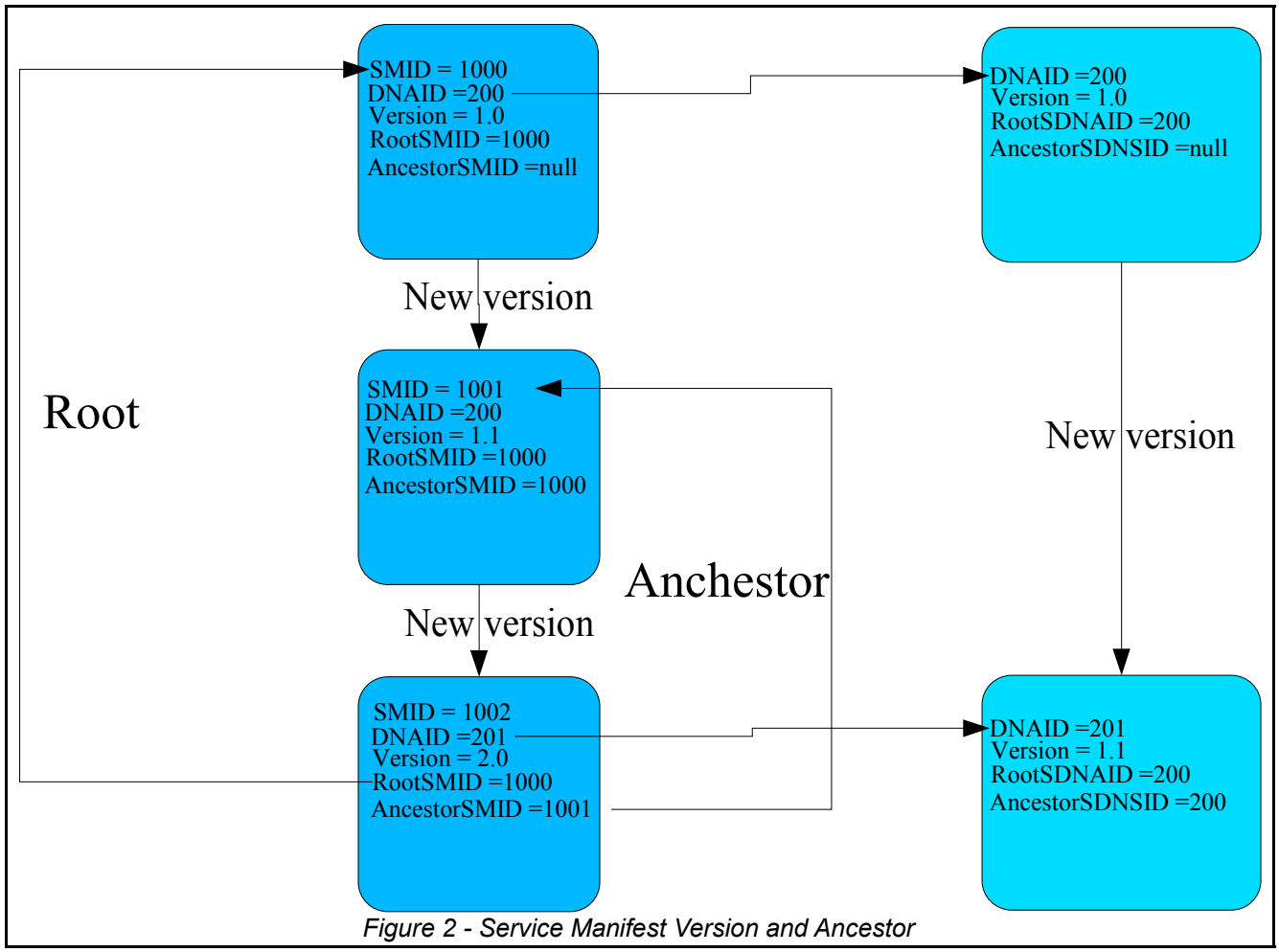

Figure 2 - Service Manifest Version and Ancestor

The Figure 2 - Service Manifest Version and Ancestor here above shows the relationship between Version Number, AncestorSM and RootSM.
Despite the fact that the SDNA is not any more considered as a separated entity from the SM, in this example the concept of DNAID and RootDNAID has been kept with the purpose to highlight that the conceptual model of the service has changed and with this change a new SM must consequently be published. Of course, a new version of the service can be published even if the conceptual service has not been changed, as it happens, for example, when the implementation has changed or, even more commonly, when the SM is copied by a new provider that intends to offer a new service.

6.1.4 Publishing date

It represents the date of publishing of the current version of the SM.

6.1.5 Last Change Date

It represents the date of the last change of the SM. This date may differ from the Publishing Date when changes to the SM do not involve the publication of a new version of the SM.

6.1.6 RegistrarID

The RegistrarID aims at identifying the publisher of the service. Only the registrar can modify the SM, a security procedure to identify the registrar and to stop unauthorized changes has to be supplied by the SR.

6.1.7 ServiceType

The ServiceType defines the type of the service. The identification of the service type is needed at run time to distinguish, for instance, a Business Service from a “Yellow Page”.
The types of services defined are the following (as from [DBEArchReq]) are shown in Figure 3 - DBE Service Type:

6.1.8

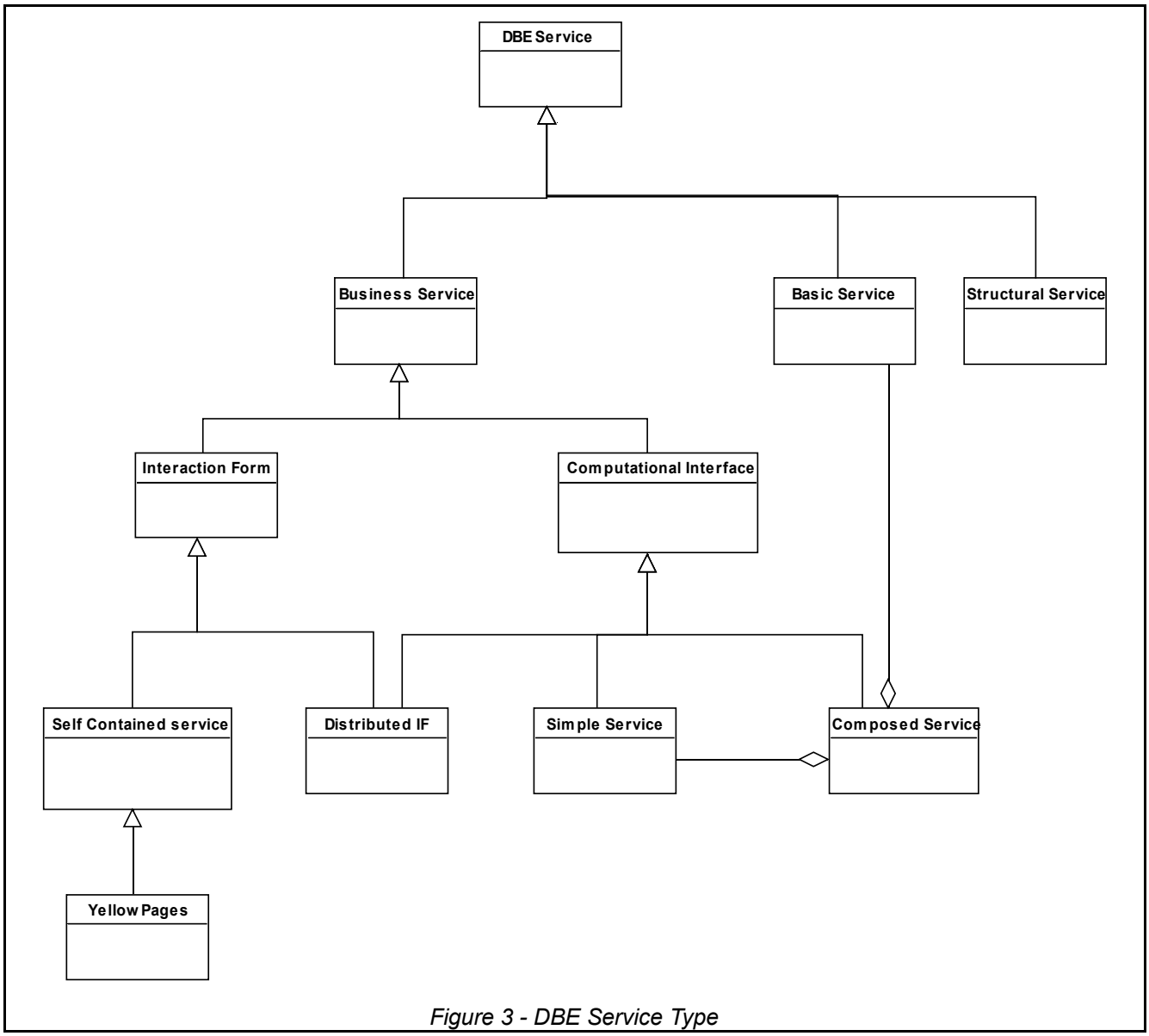

Figure 3 - DBE Service Type

SM Availability/State

As stated in chapter 7 - .Service Manifest Life Cycle, the SM cannot be removed from the SR even if its Service Instance is no longer in use. Then, there is the need to discriminate the Available services from the services which are "dead", or from the services which are still supported but only for the ongoing agreements and they will be no longer supported in the future. For these reasons the Availability/State field has been introduced.

6.1.9 BML Data

The BML data stores the information defined by the BML Model. These information are in XML format[23].

6.1.10 Interaction Form

The Interaction Form is the representation of the User Interface shown when an "Interaction Form" service is executed.
For more information about Interaction Form refers to to chapter 13 - .Interaction Form (IF) and Self-Contained Service (SCS).

6.1.11 BPEL Model

The BPEL (Business Process Execution Language) for Web services is an XML-based language designed to enable task-sharing for a distributed computing, even across multiple organizations, using a combination of Web services.
The BPEL defines a notation to specify business process based on Web Services[24], where business processes can be described in the two following ways:

- Executable business processes model actual behaviour of a participant in a business interaction;
- Business protocols, in contrast, use process descriptions that specify the mutually visible message, exchange behaviour of each of the parties involved in the protocol, without revealing their internal behaviour. The process descriptions for business protocols are called abstract processes.

The BPEL is used to model the behaviour of both executable and abstract processes. The scope includes:

- Sequencing of process activities, especially Web Service interactions
- Correlation of messages and process instances
- Recovery behaviour in case of failures and exceptional conditions
- Bilateral Web Service based relationships between process roles

The adoption of the BML model in DBE takes into account the following considerations:

- The BPEL Model is optional (it is used only for Composed Services)
- The BPEL Model may contain Intellectual Property information and hence has to be considered as non-disclosure
- The BPEL Model may be public (to advertise the service)
- The BPEL Model is needed at runtime to execute the service
- The SM has to be Self Contained[25]

The storage policy for the BPEL in the project has been object of a long evaluation among the DBE partners. At the beginning it was thought to store the BPEL into the KB, both for operational reasons and the absence of a tool like the SMEditor.

23 Refer to [BMLMM] for further information.
24 Definition provided by OASIS. See http://www.oasis-open.org
25 The SM contains all the data needed to define and run a service. For performance reasons it is not suggested to physically distribute the information in different storage supports.

Lately, it was thought to store it into the SM as soon as the SMEditor was made available.
Some DBE partners supported the position of not storing the BPEL into the SM. In particular TCD explains, as a reason for not storing it into the SM, the perception that the BPEL could be something that SMEs do not intend to share, because it is an internal process: apart from defining a business process with trading partners it may contain internal services that are not part of the DBE ecosystem, such as for example, a Customer Relationship Management service or some other internal business-related services. Therefore the BPEL should be considered as something highly guarded from any public eye.
Accordingly to all the aforementioned considerations and also to the fact that Security has not implemented yet, it is left to the SME the decision if storing the BPEL only into the KB or also into the SM, depending on its willingness to respectively make public or not the BPEL.
Then the BPEL is stored in the KB and may be added to the SM just for sharing the BPEL with the DBE Community. All the BPEL stored in the SM are, then, public.

6.2 Virtual Data

Virtual Data are information not contained in the SM, but universally recognized by the DBE community as “SM stuff”. Since the SM is the “representation of a service”, all the data related to a service are “emotionally felt” to be part of the SM. Still, the SM may not contain such data nor a reference to them, while Virtual Data should contain a reference to the SM. For instance in the [SMCM], Proxy and Fitness Data are supposed to be part of the SM: this may result as correct from the business point of view[26], but it is incorrect from the technical one[27]. By introducing the concept of Virtual Data it is assumed that these data are not contained in the SM but only related to it. To implement this relationship the SMID is stored in the Virtual Data and used to find the “owner SM”.

6.2.1 Fitness Data

Fitness Data (FD) are not needed in order to execute the service, for this reason they will not be stored in the SM. The SM is agnostic about the Fitness Data structure. Anyway it is preferred to have a conceptual link[28] to Fitness Data, wherever they will be stored, because these data may frequently change and they are not accessed during the consumptions phase of the service. Furthermore this information might grow considerably over time, although at the moment FD are calculated, used and deleted.
The EvE uses an abstraction of the SM that consists of a pointer to the original SM and other EvE-related data. The same SM can have different "success" in the different DBE EvEs with different FD associated with it; additionally, this representation encapsulates all the data needed for the EvE inside the EvE itself.
Also it should be mentioned that the user profiling, especially the dynamically gathered data during usage of the system, is strongly interlocked with fitness data

26 CIM
27 PIM
28 Not a physical link to the data (i.e. a foreign ID), but the data are reachable/obtainable in someway from the object.

gathering. The EvE and the related FD are not an easy concept, therefore refer to [Report on Fitness Landscape] for further information.

6.2.2 Quality of Service (QoS)

As stated for the Fitness Data, the QoS[29] data are not stored in the SM. The SM has just a conceptual link to the QoS.
Example of QoS of the SM can be represented by the last successful execution, the average time before execution, and so forth.

6.2.3 Proxy

The SM does not contain the proxy and in addition it does not contain a link to the proxy. The link between the SM and the proxies (remember that a SM may have more than one proxy and a proxy might implement more than one SM) is made when every single proxy is registered to FADA. It is the proxy that contains a link to the SM.

6.3 Service Manifest Definition

Figure 4 - Service Manifest Schema here below represents the current version of the SM schema.

29 Data which define the quality of a service.

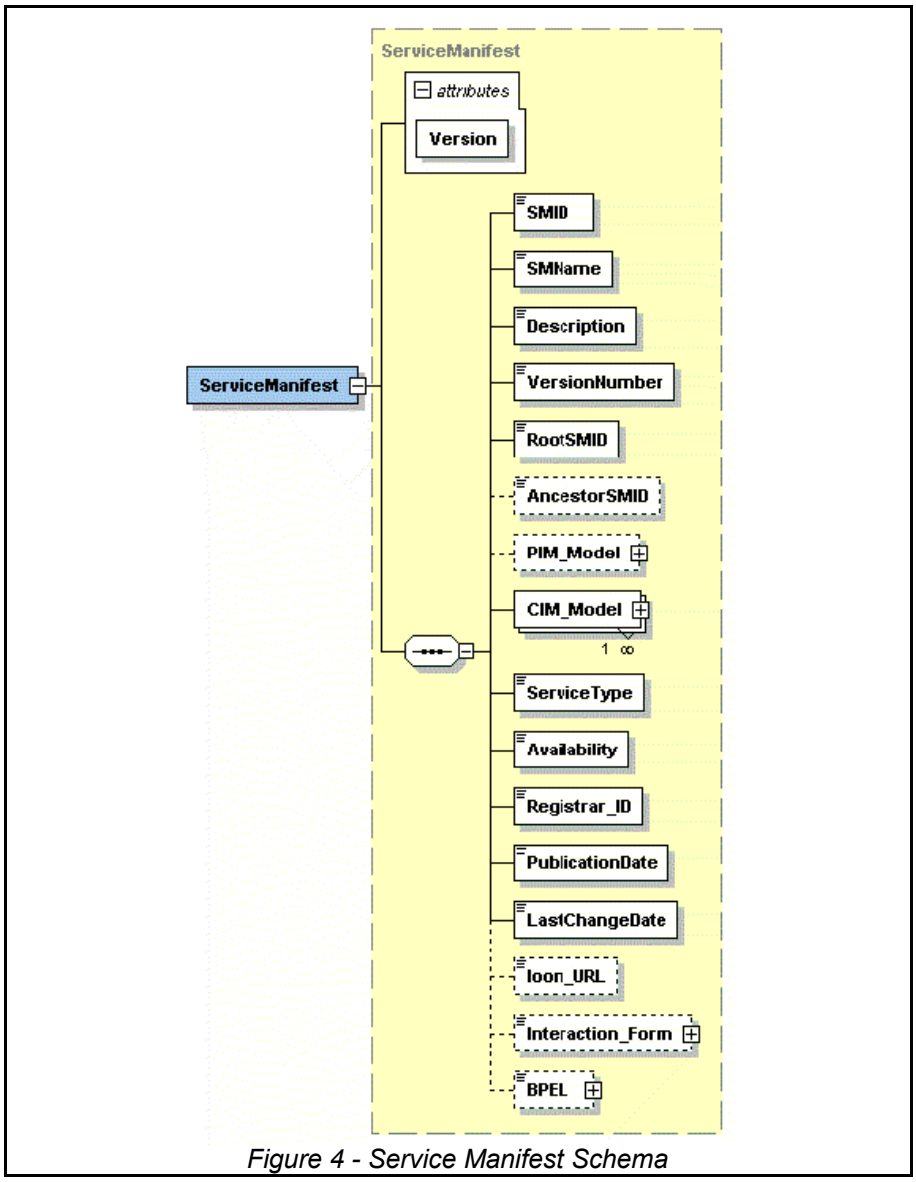

Figure 4 - Service Manifest Schema

In the first version of the SM (00.02), the XSD intended to define also the SM contents (such as SDL, BML etc) in order to provide a "validation" method for the SM and its contents.

Now it is suggested to consider the SM only as a container that has no responsibility on the structure and validation of its contents but just the duty to contain it. For this reasons the XSD represents only the SM structure, but not the structure of the SM contents.

The XSD (which defines the SM model) is used to emphasize that the SM is not a model or a metamodel but only a container of data, some of which might also be models. The Figure 5 - Service Manifest XSD here below contains the SM definition.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xs:element name="ServiceManifest" type="ServiceManifest"/>
	<xs:complexType name="ServiceManifest">
		<xs:sequence>
			<xs:element name="SMID" type="xs:string"/>
			<xs:element name="SMName" type="xs:string"/>
			<xs:element name="Description" type="xs:string"/>
			<xs:element name="VersionNumber" type="xs:string"/>
			<xs:element name="RootSMID" type="xs:string"/>
			<xs:element name="AncestorSMID" type="xs:string" minOccurs="0"/>
			<xs:element name="PIMModel" type="PIM" minOccurs="0"/>
			<xs:element name="CIMModel" type="CIM" maxOccurs="unbounded"/>
			<xs:element name="ServiceType" type="xs:string"/>
			<xs:element name="Availability" type="xs:string"/>
			<xs:element name="RegistrarID" type="xs:string"/>
			<xs:element name="PublicationDate" type="xs:dateTime"/>
			<xs:element name="LastChangeDate" type="xs:dateTime"/>
			<xs:element name="IconURL" type="xs:anyURI" minOccurs="0"/>
			<xs:element name="InteractionForm" type="xs:anyType" minOccurs="0"/>
			<xs:element name="BPEL" type="xs:anyType" minOccurs="0"/>
		</xs:sequence>
		<xs:attribute name="Version" type="xs:string" use="required"/>
	</xs:complexType>
	<xs:complexType name="CIM">
		<xs:sequence>
			<xs:element name="Model" type="xs:anyType"/>
		</xs:sequence>
		<xs:attribute name="Version" type="xs:string" use="required"/>
		<xs:attribute name="Description" type="xs:string"/>
	</xs:complexType>
	<xs:complexType name="BMLModel">
		<xs:complexContent>
			<xs:extension base="CIM">
				<xs:sequence>
					<xs:element name="BMLData" type="xs:anyType"/>
				</xs:sequence>
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
	<xs:complexType name="SBVRModel">
		<xs:complexContent>
			<xs:extension base="CIM"/>
		</xs:complexContent>
	</xs:complexType>
	<xs:complexType name="PIM">
		<xs:sequence>
			<xs:element name="Model" type="xs:anyType"/>
		</xs:sequence>
		<xs:attribute name="Version" type="xs:string" use="required"/>
		<xs:attribute name="Description" type="xs:string"/>
	</xs:complexType>
	<xs:complexType name="SDLModel">
		<xs:complexContent>
			<xs:extension base="PIM"/>
		</xs:complexContent>
	</xs:complexType>
</xs:schema>
```

Figure 5 - Service Manifest XSD

6.3.1 SM Class Diagram

The SM class diagram is useful to understand the logical composition of the SM, and it is depicted in Figure 6 - Service Manifest Class Diagram here below. The SM XML Schema was generated from this class diagram. Note that all the associations are compositions. CIM and PIM are stored into the SM.

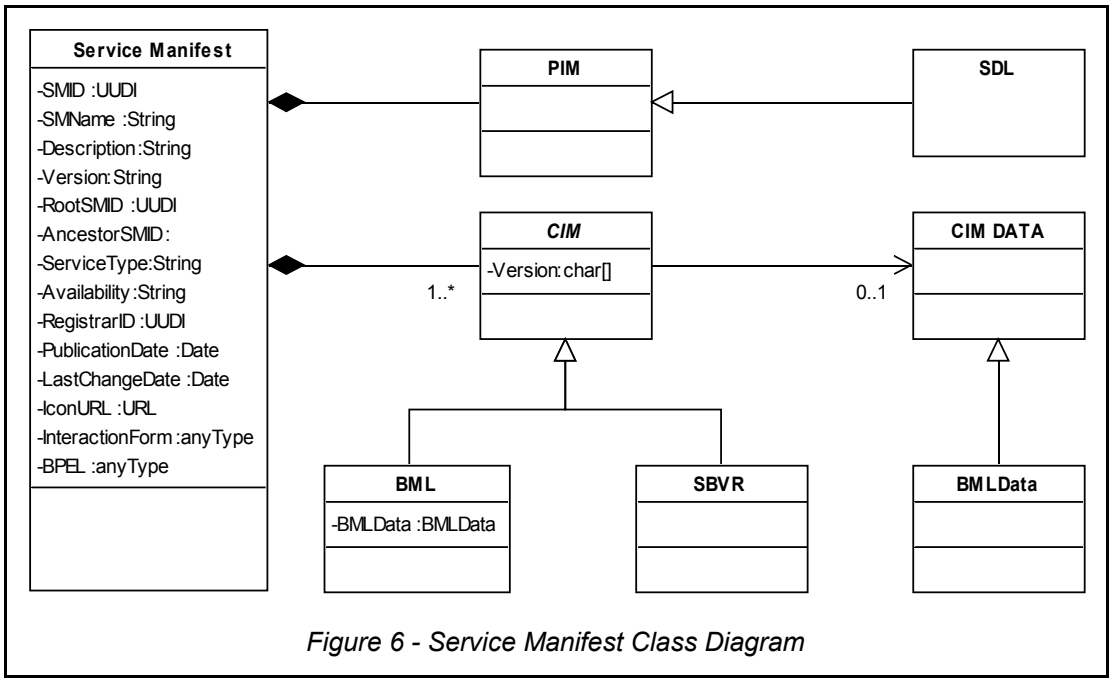

Figure 6 - Service Manifest Class Diagram

7 .Service Manifest Life Cycle

One of the most interesting features of FADA is that if a service is down, the proxy is removed from the network. This allows to search only proxies of active services. As a consequence of that, the life cycle of a proxy is the same of the back-end service.

Now the question is: does the SM and the Proxy have the same life cycle? Moreover: why is this choice important?

We suggest that the availability of the Proxy does not directly impact the life cycle of the SM, i.e. the SM has to be still reachable even if its proxy is not reachable. The different Life Cycle between SM and proxy can also allow the chaining of services with the Composer if, at design time, the proxy is down. For instance, in case a SME has its Information System active only during the office hours, the Service Proxy will be active only 8 hours a day, but the SM can still be reachable.

If the proxy is not continuously present, this will impact the QoS (presumably the uptime) to underline the service availability. As a suggestion, the Servent[30] should be able to understand the different reasons for the proxy to be removed. It can be either for an unexpected network failure or for an explicit command. In this way the QoS (Quality of Service) information can infer the right reasons for the proxy unavailability and evaluate in a different way the two types of values.

Another consideration is that it may be highly time consuming to check through all the network if at least a proxy is still active, this operation can not be frequently executed. Note also that there might be services (i.e. informative or “Yellow Pages” and the Interaction Form belonging to the type Self Contained) for which a Service Instance is not present, in this case it does not make any sense to bind the SM life cycle to the proxies' one.

For all these reasons it is strongly suggested that the SM is removed independently from proxies.

The Figure 7 - Service Manifest Life Cycle here below shows the UML State SM life cycle.

30 The ServENT (SERVer & cliENT) acts as a Server and as a Client inside the DBE. It is the piece of software which transparently creates the DBE P2P network and contains services (infrastructural and custom made)[DBECoreArch].

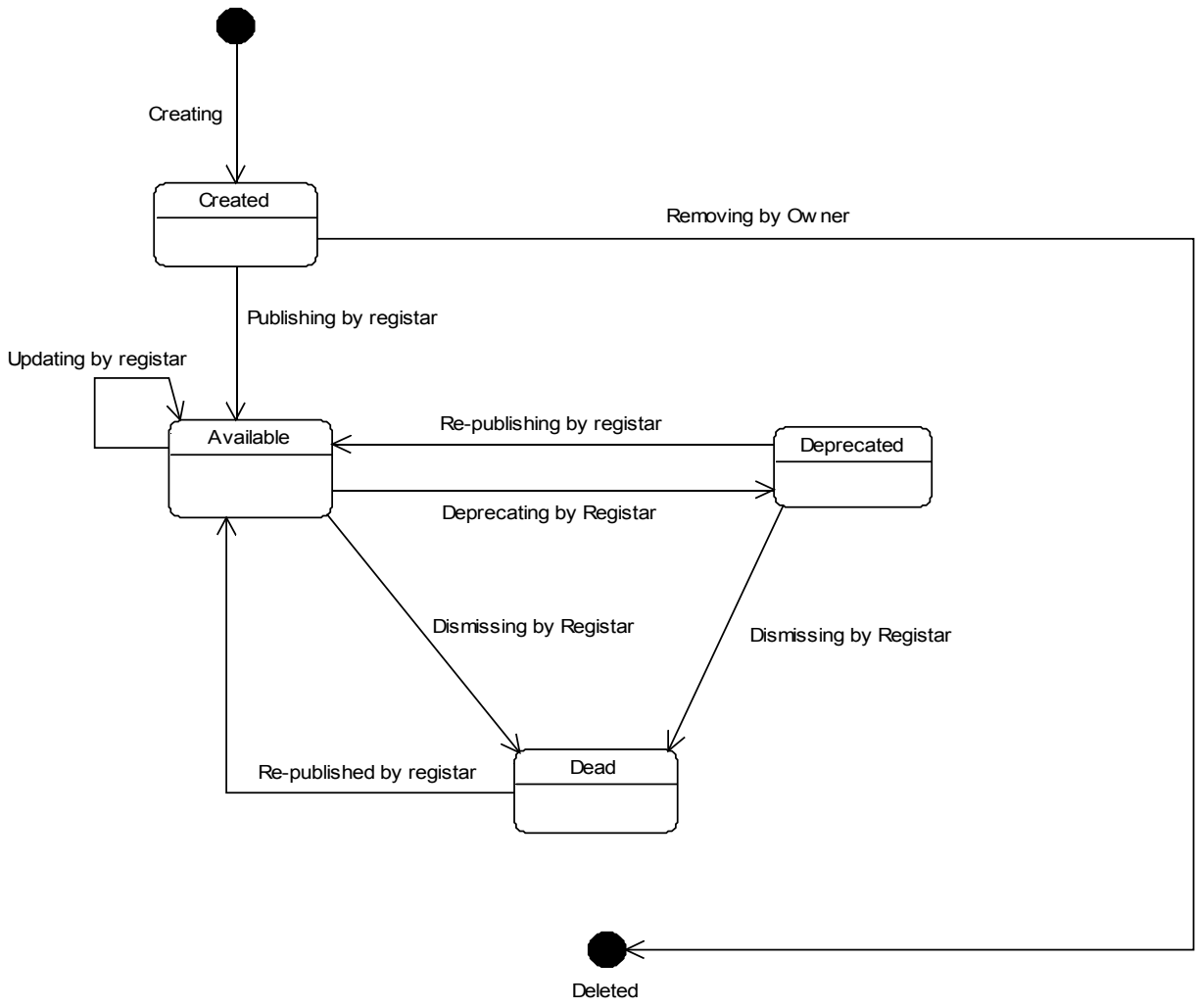

Figure 7 - Service Manifest Life Cycle

7.1 SM States

The states, as reported in the Figure above are the following:

- **Created**: the SM is private to the owner and is not shared with the DBE community. It is stored in its private File System. The owner can modify the SM without restriction.
- **Available**: the SM is shared with the DBE community or groups of users. Changes to the SM are subject to the rules defined in chapter 9 - .SM Editing and Versioning.
- **Deprecated**: the SM still remains in the SR but it is not possible to have new agreements or to compose it in a new service chain. The SM Owner must guarantee the availability of the related proxy until the end of all the ongoing agreements. In any case the SM can be used as an ancestor for the creation of new SM. The registrar may re-publish the service.

- **Dead**: the SM has no proxies available. The SME provider has decided not to support the service any more. There are not ongoing agreements. The SM remains in the SM for historical purpose or it can be used as ancestor for other services.

7.2 SM Transitions

The transitions, as reported in the Figure 7 - Service Manifest Life Cycle above are the following:

- **Creating**: the SM is created by the owner using the Service Factory (SF), it is private to the owner[31] and not shared with the DBE community.
- **Publishing**: the SM is published (registered in the Semantic Register) by the registrar[32]. After publishing, the SM becomes public and only the registrar can change it.
- **Updating by the registrar**: if the SM is updated it still remains Available in the SR. The SM may be updated only by the registrar.
- **Dismissing by the registrar**: the registrar can dismiss the SM. It still remains in the SR but it it is not used. The registrar dismisses the SM when he decides not to maintain the service any more.
- **Re-publishing by registrar**: a dismissed or dead SM might be re-published by the registrar if he wants to rise again the service.
- **Removing by the owner**: the SM can be deleted by the owner before the publication. The DBE community will never know the existence of this service.

It should be stressed that a service published in DBE will never be removed from the SR. There is not a transition from the DEAD state to the final state (Figure 7 - Service Manifest Life Cycle above). The only way to delete a SM is when it is in the “Created” state, before publication.

31 The entity who owns the intellectual propriety of the SM.

32 The person who publishes the SM in the Semantic Register. He/she is the only authorized to edit the SM. He/she may be the SM Owner.

8 .SM scope/visibility

Even if the original intent in the project was to make a SM available to the entire community, it is reasonable to consider the existence of “private” SMs. An SM, in fact, can represent an element of competitive advantage for an SME inasmuch the SM reveals important information about its business and organisational model, policies, processes and so forth. Therefore, an SME user can create a SM for personal use and store it in its private File System and he might wish not to share it with other SMEs competitors.

Nevertheless, from a business perspective, it could be interesting to consider the possible rise of new business models based on the concept of “SM Market” where SMEs exchange their SM (or part of it). However, in such a context new mechanisms for protecting sensitive information of SM (e.g. encryption or identity management techniques) have to be introduced.

To better explore the market visibility of the SM it is important to understand its context within the Evolution Environment (EvE)[33] of DBE and the concept of EvEservice. “*An EvEservice is an individual within the EvE, and its DNA is the Service Manifest it references. An EvEservice is primarily a reference to a Service Manifest, and many EvEservices can point to the same Service Manifest*”, as described in [Control of Self-organization]. The Eve has its own structure to describe services, called EvE SM which contains a link to a SM and Fitness Data (FD). The EvE SM is an entity which is located into the local service pools (called EvEservice-Pool), one Pool per Habitat. Each SME user has a Habitat which contains an EvEservice-Pool with service of potential use to the SME. In other words, the Habitat is the representation of a SME networked with the Servent. Hence each Habitat owns its collection of EvE SMs. These Habitats are linked among each others (by logical links defined through the Servent interface according to specific algorithms). The topography which characterizes the network of these Habitats is a ”Caveman topology” (several clusters interconnected by very weak links) and do not require a complete connected network. In each link there is a value of probability which determines the strengthens of the link. Hence, the EvE includes the concept of closed community (a cluster or component strongly connected which is not linked – or is weakly linked – to the other clusters or components).

33 The EvE represents an alternative way to distribute, locate, and aggregate services. When a service is constructed from the Service Factory, it will be distributed from the SME's own habitat instance automatically across the Habitat Network, becoming available for evolutionary service composition at other SME user habitats. This is possible as the habitats are all interconnected, dynamically clustering around business sectors and supply-demand relationships. So distributing a service from the SME's own habitat will provide the optimal distribution of the service to the users who are most likely to consume the service, based on past experience [DBECoreArch].

9 .SM Editing and Versioning

The SM may be updated in order to reflect some changes to the real service[34] or to the Service Instance service[35]. If changes of the service pertain to the implementation, they do not affect the SM, but if they pertain to the BML, SDL or the service data they will affect the SM. When a SM changes, the event has to be notified to all the DBE parts interested to use that SM.

Changes to the SM can happen before or after the publication. While in the first case it does not represent a problem, in the latter case the re-publication may cause some troubles and must be done with care. To understand the reason why changing a SM might be a problem it must be remembered that the SM can be used in a service chain and that the composition of services is done using the SM.

The Figure 8 - Service Manifest Changes below explains the effect of a change in a SM SDL.

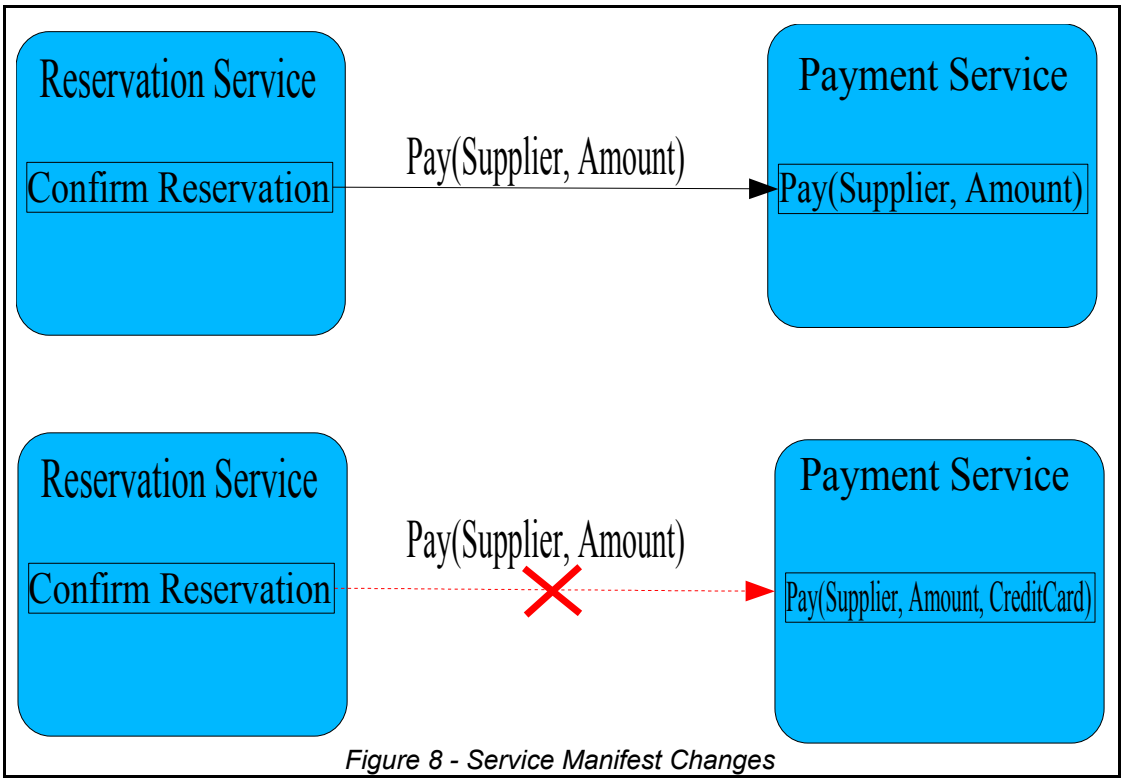

Figure 8 - Service Manifest Changes

As showed in Figure 8 - Service Manifest Changes above, the changes in the SM Payment Service may produce errors in service chaining: the Confirm Reservation method tries to use the Pay(Supplier, Amount) functionality, but the functionality does not exist any more.

To avoid these problems tracing and versioning have been introduced. Once the SM is published all the changes are going to be traced, and if a change is done to a SM a new version of this SM has to be published.

34 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.

35 The IT implementation of a Real Service.

The Figure 9 - Service Manifest Versioning shows how versioning may avoid the problems pointed out by Figure 8 - Service Manifest Changes, above.

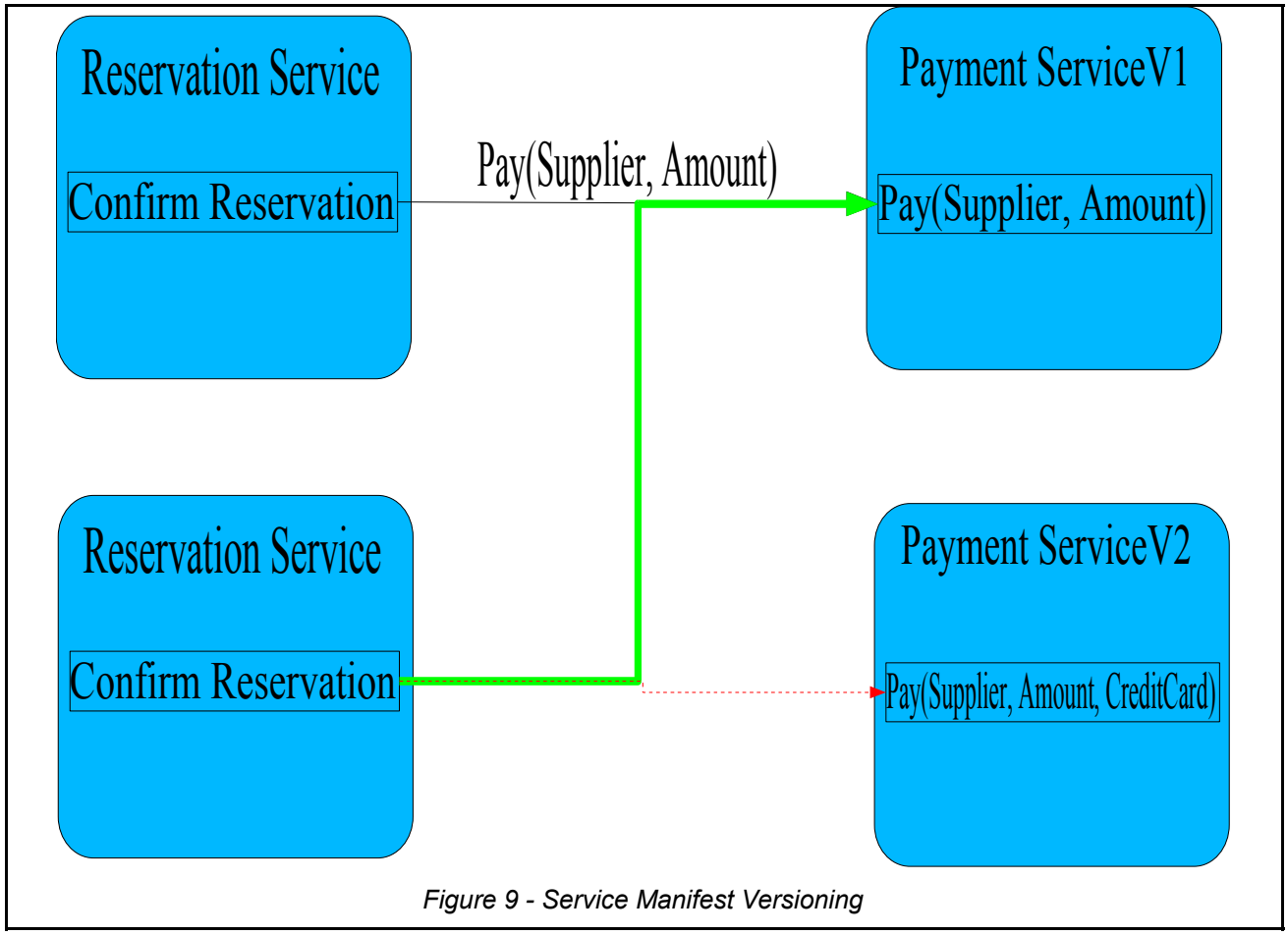

Figure 9 - Service Manifest Versioning

If a change is made on the Payment Service a new SM has to be published, then Reservation Service can use the “old” SM until its “owner” decides to modify it in order to use the new Payment Service.

It has to be noticed that the changes in SM may affect both the data and the model of the service.

The Figure 10 - Service Manifest Versioning Policy summarizes the changes to SM that imply the publication of new SM.

Attribute	*Editable*	*New Version required*
SMID	**N**	**N/A**
SMName	**Y**	**N**
Description	**Y**	**N**
VersionNumber	**Y**	**Y**
RootSMID	**N**	**N/A**
AncestorSMID	**N**	**N/A**
CIM Models	**Y**	**Y**
PIM Model	**Y**	**Y**
BMLData	**Y**	**N**
ServiceType	**Y**	**Y**
Availability	**Y**	**Y**
Registrar_ID	**N**	**N/A**
PublicationDate	**N**	**N/A**
LastChangeDate	**N**	**N/A**
IconURL	**Y**	**N**
Interaction_Form	**Y**	**N**
BPEL	**Y**	**N**

Figure 10 - Service Manifest Versioning Policy

The following sections will investigate the effects of the changes to the SM.

9.1 BML & SDL changes

If there is the need to change the CIM or PIM Model, changes have to be done on the Model Repository and not directly on the SM; then, a new version of SM has to be produced with the new models. However, no changes are possible on the CIM and the PIM Models inside the SM, because models must directly be modified on the KB (or file system) and then imported into the SM. Therefore, models from the SM perspective, are replaced and a new SM version will be published. A particular consideration should be given for the SBVR which contains inside both the business model and the related data. In this case it is hard to understand what has been modified, if the model or just the data. To avoid new useless SM versions when replacing the SBVR, it must always be indicated, together with the new SBVR, if modifications are just related to data or they include the models as well. According to these information it will be decided if a new version of the SM has to be published or not.

9.2 ServiceType changes

It is not possible to change the ServiceType. If the ServiceType is changed a new version of the SM has to be published.

9.3 BML Data (M0) changes

The BML data defines the Service Instance which contains, for example, the specific data of the service and service provider. If the BML Model defines that the service provider must have an address and a telephone number, the BML data should therefore contain the address (e.g. 1505 Piccadilly – London) and the telephone number (e.g. +44 (020) 7493 1234).

The SM Version does not have to be changed. Given the fact that when an agreement is signed the related SMs are copied in another storage area[36] and any change in the published SM will not affect the ongoing agreements unless such changes are agreed between partners.

9.4 Interaction Form changes

Changes to the Interaction Form do not force to publish again a new version of the SM. If the changes only relate to the Interaction Form, it is assumed that the service is not changed but only its User Interface. The end-user is not really affected by these changes.

9.5 Virtual Data changes

All changes to data that are not physically stored into the SM are simply ignored. It should be remembered that Virtual Data are not stored in SM but there is only a conceptual link to them. Any modification to all these data cannot effect the SM.

9.6 BPEL Model changes

Leaving aside the technical aspects, changes on the the BPEL Model affect the implementation of the service and not its functionalities. For this reason it is suggested that changes to the BPEL Model do not impose the publication of a new version of the SM. These changes might be notified to all partners with ongoing agreements because the service could have published the BPEL through the SM and this can be an integral part of the contract. However, such a notification is not directly managed by the DBE infrastructure, but from the provider through its traditional channel.

9.7 Versioning

The SM itself changes to reflect the changes of the service. Generally service evolution is very limited in time and usually it involves slight changes in its functionality. This could also happen when service implementation is changed. Conversely, radical changes often bring to the definition of a new completely different (at least from a business point of view) service. In this latter case, evolution reflects mostly a change in the business model and strategy of the service provider.

36 A service is stored in a local storage area for later retrieval for successive execution

Anyway, when a service changes, in order to keep coherence between the service and its SM, the following rules have to be satisfied:

- The SM has to evolve to follow the evolution of the real service.
- All the alteration of the SM has to be traced.
- When the Registrar (the only that can modify a published SM) tries to change the SM it has to be identified, as suggested later in chapter 10.1 - Service Manifest Security.

When a new SM version is published, it has a different SMID and a different Version Number; the RootSMID does not change and is used to identify all the SM versions “descending” from the same root SM. AncestorSMID is used to identify the “father” of the SM. If a SM has an empty AnchestorSMID and RootSMID = SMID, the SM is the first version of the SM. This means that it has been created from scratch: it is not a modification of an existing SM.

10 .Semantic Registry

This document does not intend to illustrate the Semantic Registry (SR), rather it investigates how the SR stores the SM. In this direction some requirement affecting the implementation of SR implementations and its features will be explicitly indicated (more information about the SR can be found in [SR]).

10.1 Service Manifest Security

The SM has to provide integrity, message authentication and signer authentication. Security is a responsibility of a DBE specific task; nevertheless the specifications contained in this document, by supporting the standard WS Security (i.e. the WS-Security Protocol 1.1 for authentication[37]), enable the implementation of a security strategy in a WS standard fashion.

Security may be implemented by using the XML-Signature Syntax and Processing[38]. Since the SM may reuse the BML and the SDL Model, it is suggested that the signatures are enveloped in each single component. This can guarantee the integrity and the authentication of every single component.

The entire SM may also be signed to guarantee the overall integrity.

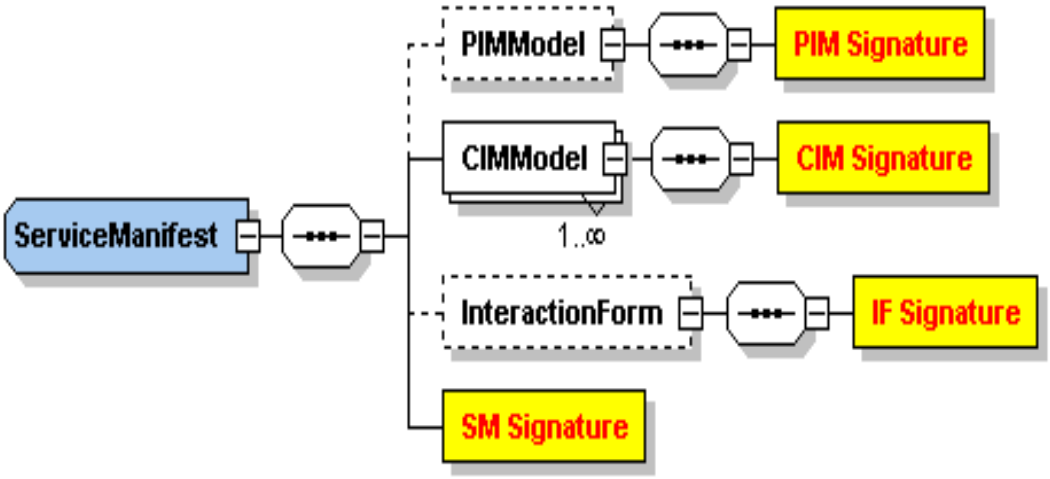

Figure 11 - Service Manifest Security

The Figure 11 - Service Manifest Security, above shows how the signature might guarantee that contents and the SM itself have not been tampered.

The XML Signature recommendation supports the X509 standard certificate. There are a lot of FLOSS[39] Java libraries implementing it: for those reasons it is suggested to use the XML-Signature recommendation of the W3C.

37 http://www.networkworld.com/news/2006/021506-ws-security.html
38 A WC3 recommendation. See [XMLSig].
39 Free/Libre Open Source

Since the SM is divided into two main parts, SDNA (CIM and PIM Models) and Service Instance (CIM Data and SM Data) and a lot of SMs may have the same CIM and PIM Models, each single model will be signed by the author and the whole SM will be signed to obstruct fraudulent data modification or substitution of models.

10.2 Tracing

As stated in chapter 6.1.3 - Version Number/AncestorSMID/RootSMID all the changes in the SM have to be traced. Tracing is useless if there is not any mechanism that allows to navigate through the different available versions; for instance, the SM needs the SR to allow to retrieve all its new versions. Minor changes that do not imply a new version of SM can be stored by the SR using an internal version number.

10.3 SM Browse and Discover

The SR has to supply some API to search the SMs as well as some visual tools to browse and discover them. The SM research have to be based on:

- models semantic
- models features
- SM Data
- SM ancestors/descendant

10.4 CRUDEL[40] Operations

The SR has to supply some APIs to perform the basic CRUDL operations taking into account the security related issues. It is responsibility of the SMEditor and not of the SR to consider the editing rules stated in chapter 9 of this document.

10.5 SMID Creations

The SMID, which identifies the Service Manifest and allows the identification of related data, may be an UUID. As stated in chapter 6.1.1 - SMID, to guarantee across all DBE the uniqueness of the SMID and to have a common SMID creator this task should be performed by the SR during the storage of the SM and not by the single editors.

40 It's an acronym for: Create, Read, Update, Delete, List.

11 .Service Manifest, EvE and Fitness Data

This chapter does not aim at discussing the internal structure of the EvE or Fitness Data but to underline some aspects of EvE and FD that are related to SDNA and SM. In this direction it aims at providing some useful suggestions to the EvE.

11.1 SDNA, SM and Fitness Data

For our goals, Fitness Data might contain information about the usage history of a service. The goal of this paragraph is to highlight the difference between the “service” as represented by the SDNA and the “service” as represented by the SM. Since the SDNA represents the Conceptual Model of a service, the SDNA Fitness Data refer only to the conceptual model usage and might monitor, for instance, the number of SM “implementing” the SDNA and the number of daily access to all the SMs “implementing” the SDNA. On the other hand, the SM Fitness Data refer only to the usage of a single Service Instance represented by an SM and might monitor the customer satisfaction referred to a determinate proxy.

For these reasons it is suggested to consider two kinds of Fitness Data: one for the SDNA and one for the SM. It shall be noticed that this suggestion affects the logical data not the physical data: the SDNA FD might be an aggregation of SM FD. This suggestion aims at underlining the evolutionary aspect of the conceptual service (represented by the SDNA) as well as the Service Instance (represented by the SM).

After a keen analysis, it has been decided that the FD must not be stored in the SM nor in the KB, but in the EvE. The reason is because Fitness Data are not related only to a SM but they are specific for the use of service by an user of a group of users. However this decision does not imply the absence of a relationship between SM and FD. In [SMCM] it is described how FD are considered to be logically part of the SM. This logical link exists even if FDs are physically stored in the EvE and not in the SM. Because the FDs are stored in EvE, any EvE can contain data which refer the same SM and which reflect the use of the SM in relation with that specific EvE. The Usage Data represents general data with global visibility which can be used by anyone who needs data about services. The EvE and the Recommender will have their own private data (perhaps simple aggregations of Usage Data). It is possible that data stored into the DBE Memory can be divided in Usage Data and Service History. And, in turn, the data of Service History can be divided in Migration History, Service Chain History, Service Evolution etc.

At present, the EvE does not define any sort of evolution for Fitness related to the SDNA but only for the single SM. However the evolution of a Conceptual Service (represented by an SDNA) should be considered as a potential valuable development track in the future.

12 .Service Factory – DBE Studio

This document does not aim at discussing the Service Factory[41] (SF) and its internal details as its main goal is just to investigate the interaction between SF and SM. It is expected that the following considerations will influence the implementation and features of the SF. In the following section some SF requirements will be explicitly enlightened from the SM point of view. In particular, the SMEditor will depend on this document as it shall manage the SM according with these specifications.

12.1 SM Publishing and Editing

12.1.1 SM Editor

Such a task has been introduced lately to develop the SMEditor. This software now supplies graphical tools for creating, editing and publishing the SM and also for editing and re-publishing the modified SM. The SMEditor uses the SR API to publish the SM. The first version of the SMEditor allowed only to create the SM, while the latest includes the possibility also to edit the SM.

12.1.2 DBE Service Publishing Perspective

The DBE Service Publishing is one of the Eclipse perspectives in which the DBE Studio is organised. When the BML and SDL for a service have been defined, the DBE Service Publishing perspective can be used to create and publish an SM. These SMs can be thought as representing a public advertisement on the DBE ExE. The BML Data associated with the service can also be edited with this perspective.

Typically the service is deployed after publishing the SM. Accounting handlers[42] can be attached at this stage so that appropriate data can be gathered for billing purposes, and the completed DBE service can be packaged up and deployed onto the DBE network.[An IDE for DBE Studio]

41 Architecturally, the software for the DBE can be considered as consisting of three core components, corresponding to three different environments:

- the Execution Environment (ExE) that hosts the DBE services;
- the Evolutionary Environment (EvE) that continuously analyses and optimizes DBE services;
- the Service Factory (SF) that facilitates the construction of DBE services.

The SF is devoted to service modelling and definition: users of the DBE will utilise this environment to create offers associated to services. [DBECoreArch]

42 In the web services domain, usage metering is often concerned with the extraction of messaging data related to an appropriate charging scheme, which can potentially be specific to an individual service instance. With reference to the billing policies, this is normally achieved through the application of such handlers which “listen” to the messages being passed between the consumer and the service. Further information can be found on [D.36.1 - SOTAComposedServices]

13 .Interaction Form (IF) and Self-Contained Service (SCS)

The following section describes the Interaction Forms and the Self-Contained Service, explaining the reason for their introduction in the DBE. It includes an indication of the advantages offered and changes required to implement the Interaction Form and the Self-Contained Service.

Although DBE has been conceived for SMEs, it might result not very easy for some of them to create, distribute and maintain a service: Self-Contained Service and Interaction Form have been introduced with the purpose of easing the creation of services.

13.1 Self-Contained Service: the simplest way to join the DBE

The Self-Contained Service is a service entirely contained in the SM. Both the code implementing the service and the related data are completely contained in the SM. The service is hence not distributed on a server and there is no proxy, but the code contained in the SM is executed by the Servent. In this way there is no need of hardware apparatus or Servent. Self-Contained Service can even be created, for example, from an Internet Café. The Self-Contained Service, therefore, enables those SMEs which do not own a proper IT infrastructure to access and create DBE services in a simple way, extending in this way the number of potential SMEs involved in DBE.

The backdraw is that this kind of service can not export SME legacy services, it is does not talk with any SME back end services.

13.2 Interaction Form: “DBE for Dummies”

Although the introduction of the Self-Contained Service in DBE has been conceived for providing SMEs with the necessary IT infrastructure to publish their services, it might be still difficult for those SMEs with a very low IT knowledge, or none at all, to create and publish a service.

The Interaction Form has been created with the purpose of enabling SMEs to create and distribute services without requiring any IT expertise, or paying any IT specialist for doing such a task.

The introduction of the Interaction Form in DBE represents an evolution of the “Yellow Page” service[DBEArchReq]. More specifically, the “Yellow page” service of DBE does support a very bare interaction with the customer: only phone or fax number can be read like the regular paper based Yellow Pages. In this context, the Interaction Form adds interaction capabilities allowing users to consume the service by filling in forms, still without the need to write any code.

To summarize, the Self-Contained Service is a technical layer for creating services which does not require an IT infrastructure, while the Interaction Form is a framework that uses the Self-Contained Service to create services using a wizard.

13.3 The Interaction Form creation/build cycle

This section and Figure 12 - Interaction Form Life Cycle here below briefly illustrates the process of creating-publishing DBE services through the use of the Interaction Form. The process involves the following actors:

- the Service Template (AKA IF Wizard) Supplier;
- the Service Provider;
- the Consumer.

The first step is the creation of a Service Template or IF Wizard made by the Service Template Supplier. The IF Wizard is a DBE service itself that has the goal of facilitating the creation and distribution of another service. A Service Template allows to create a single type of service (it can be thought as a unique “Conceptual Service” or service with the same SDNA) and therefore it can solve a specific business need. The IF Wizard can be specific to different business sectors (i.e. Car Selling, Grocery, Book Shop, Hotel Reservation and others).

At a second step, the Service Provider uses the IF Wizard to create and distribute its new service. It then executes the following actions:

- chooses the most suitable Template Service for its business;
- executes the creation process of the Template Service through simple operations which are guided by the IF Wizard itself;
- publishes the Service in DBE through an automatic operation made by the IF Wizard.

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 task like the creation of the BML, SM and the registration and publication are completely transparent.

Finally, the Consumer (which can be both a SME and a single individual) accesses the DBE portal and consumes the service, in the same manner as for traditional services.

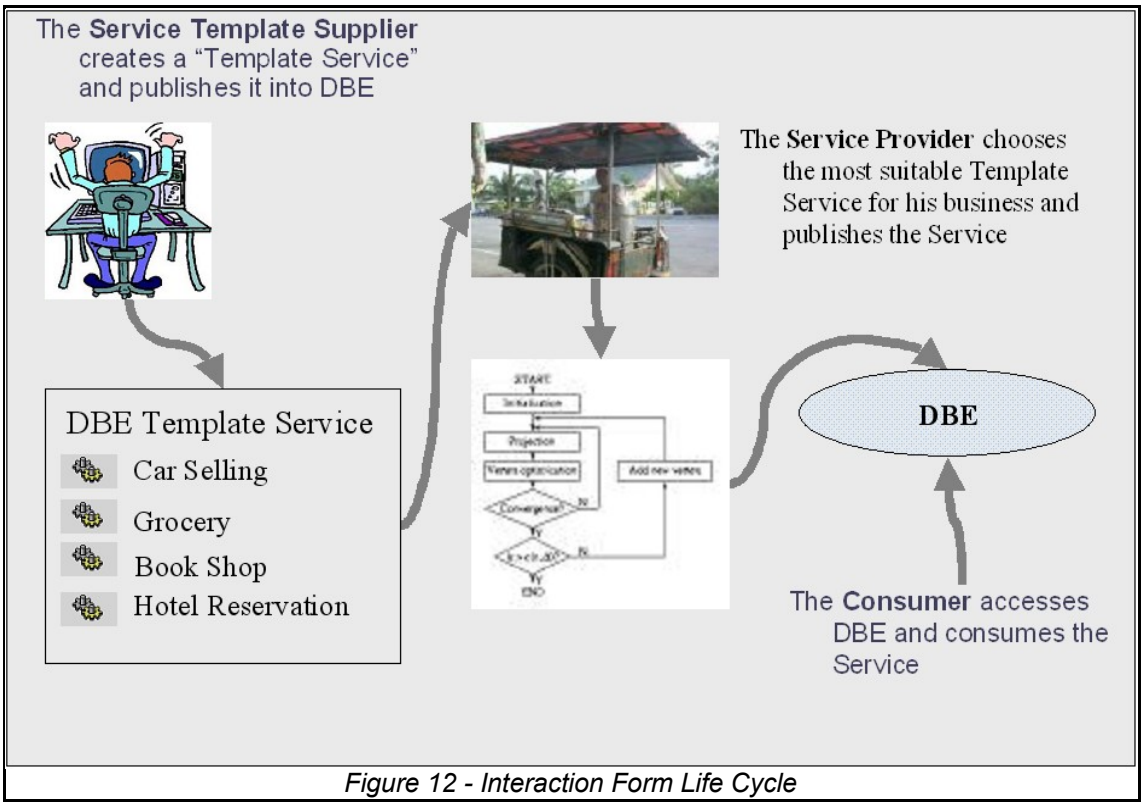

Figure 12 - Interaction Form Life Cycle

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

13.4 Service Template or IF Wizard

The Service Template allows SMEs that are not familiar with tools like BML Editor, BML Data Editor and SM Editor to easily and intuitively create DBE service.

The IF Wizard asks the service provider a set of information and then publishes the service to be used by the consumer. Information are divided into two parts:

- info needed to build-up the BML data (i.e., the location of the book-shop)
- info needed by the customer service (i.e. the list of books, pricing).

The IF Wizard will ask these information to the provider who wants to publish its service. Since the it has specifically been built for a "conceptual service", differently from the other editors the IF Wizard will be able to execute such operations in an easier way for the final user. To give an example, in the case of an IF Wizard related to a library service, it could make questions such as "What is the name of the library?", "In which city is it located?" and so on, till it will have collected all the necessary information. Same logic could be thought for the user data entry, where the IF Wizard asks for information with questions such as: "Do you want to insert a new book? (Y/N)", "What is the book title?", "How much does it cost?", and so forth, making the data entry very simple even for people not confident with computers. Furthermore, possibilities to create and target different wizards for differently skilled persons can be thought, avoiding to

create and target, for example, a question-based interface to an advanced user when such user is not the right target.

In theory, the IF Wizard structure is not pre-determined and can be whatever, even if a series of libraries have been developed to facilitate the IF Wizard creation. Such libraries can only be used if the following structure in Figure 13 - Self-Contained Service and IF Wizard is respected:

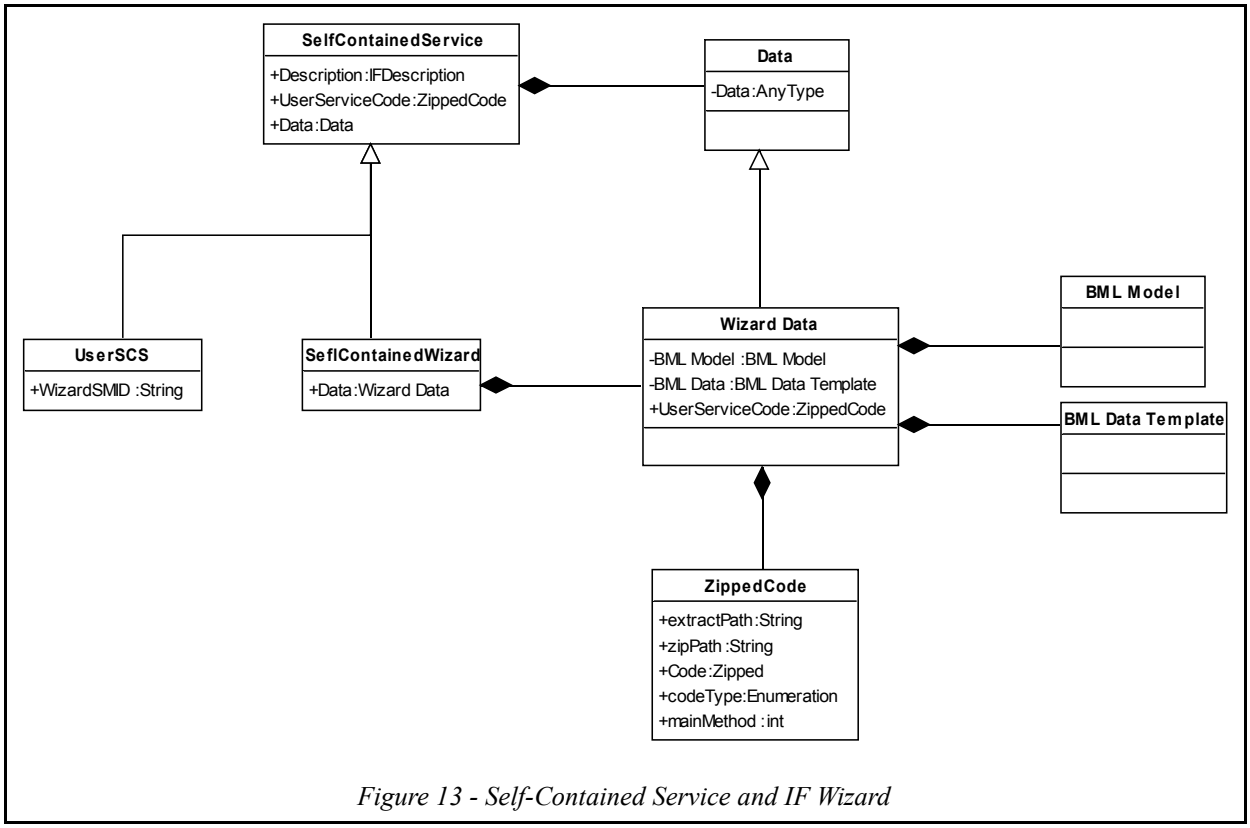

Figure 13 - Self-Contained Service and IF Wizard

An IF wizard is a particular kind of Self-Contained Service composed, as any Self-Contained Service, by the following elements:

- a description
- the service code
- a set of data

The difference between a Self-Contained Service and an IF Wizard is that the IF Wizard contains inside its data a series of information that will be used to create the user service, including;

- BML Model
- BML Data Template
- the code of user service

It must be pointed out that the structure of data is not predetermined and each IF Wizard can create its own structure. Nevertheless the IF Wizard creation will be easier

when respecting the suggested structure of data, as illustrated in the Figure 13 - Self-Contained Service and IF Wizard above.

The following paragraph will analyse in details each single field.

13.4.1 SelfContainedService.User Service Code

The User Service Code is the executable service code that will be consumed by the final user. The reason for having two User Service Codes, one located in the WizardData and the other located in the Self-Contained Service is explained by the fact that the IF Wizard is a particular case of Self-Contained Service. The SCS Code is the code of the Self-Contained Service and it will be executed when the service is invoked. The IF Wizard is a Self-Contained Service and therefore it contains its executable code (that one located in the Self-Contained Service) but it also necessitates the code of the service that it will create (that one located in the WizardData). This code will be used by the IF Wizard to create the User Service. Hence, the UserCode contained in the IF Wizard as a data will become an User Service Code (executable code) of the service created.

13.4.2 WizardData.BML Model

The BML Model describes services from a semantic viewpoint and in a business perspective. This model describes the user service and not the IF Wizard itself that will be described by a BML contained in the SM. The BML Model contained in the WizardData describes the service that will be created and it will be simply copied into the SM of the User Service. All the services created with a the same IF Wizard will therefore have the same BML.

13.4.3 BML Data Template

All the User Services created by the same IF Wizard represent the same Conceptual Service and therefore share the same BML (CIM Model); 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 some libraries have 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 entry. 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.

13.4.4 Service Template Supplier

The Service Template Supplier is a programmer (or an entity that pays the 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 Lazlo for Self-Contained Service) and must be able to set up, configure and use the DBE Studio.

The Service Template Provider builds-up two programs: the IF Wizard and the User service program. This entity must be able to edit (copy/modify) a BML model both for the IF Wizard and the User Service. Also it must use the BML Data Editor to insert the BML Data for the IF Wizard and the BML Data Template for the User Service.
Several entities for several reasons can be interested to make use of the IF Wizard through the programmer, among them being:

- a local government authority (i.e. a region, a local municipality) or a private SME (i.e. a wholesaler) can pay the programmer to make the IF Wizard for launching a service in a particular business sector;
- advertising can be introduced inside the developed service and this can be of interest of the above mentioned entities;
- an SME Software provider can add it among its programs and services to be offered;
- the programmer itself can act as a specialist for creating a specific kind of wizard and selling the use of it to public or private clients. He could also make it just for disseminating the OSS spirit and principle and getting some peer recognitions, or promoting his CV/career;

In the case of Distributed Interaction Form the same programmer can then provide hosting of the distributed service.

13.5 Service Provider

As previously mentioned, the Service Provider can be a private or public SME, a public institution, a local authority, and so forth.

The Service Provider creates the services by simply using the IF Wizard. It will have to execute the following operations:

- Searching the IF Wizard by using the DBE Portal: this operation is very simple and it is the same operation executed by all the user that want to consume a DBE service. Therefore it has been conceived for non expert users;
- Choosing of the most suitable Template Service (IF Wizard) for its business to implement: generally a ranked list of Template Service will be presented, including a description of each single Template Service. The IF Wizard can also present a description of the service (i.e. a short video), including its detailed functioning;
- Executing the IF Wizard: this will require some further actions that are related to the chosen Template Service (for example, in a book shop service, the Provider will insert the list of the book to sell and some information about the shop).

As a final action, the IF Wizard will publish, the service in DBE in a full transparent manner to the Service Provider that will see published its own service.

13.6 Customer

The Customer can be both a SME and a private single individual generating in the first case a B2B scenario (where SMEs will interact among each others) and in the second

case a B2C scenario (where a private individual will search for a particular service to consume).

The Customer uses the Self-Contained Service as a normal DBE service (it is indistinguishable from a regular manually created DBE Service), by searching it through the DBE Portal and then executing it. The only problem is that although an interaction between Customer and Provider exists, such an interaction, for the Self-Contained Service does not occur in real time (like in a deployed service) since the Provider has no Information System to process the Customer's request, but it happens off-line (e.g. through e-mail exchange).

As a consequence of this over semplification in the service creation, the Self-Contained Services are services that do not exploit all the potentialities offered by the system.

13.7 Changes required by introducing the Interaction Form/Self-Contained Service

A list of some major changes need to be done to integrate the new Self-Contained Service into the DBE structure, as follows:

13.7.1 Divide SM Creator into two java projects

Some functionalities offered by the SM Editor, such as, for example, the publication of the SM, are also useful for the IF Wizard that must be able to access them. For such a reason it must be possible to distribute them independently from the SM Editor into the ClientSideServent. The module which contains this functionalities is called SMCore. The implementation of the Self-Contained Service generates some modifications affecting the DBE Portal. As a consequence of that, at present, the orientation is to distribute the SMCore together with the DBE Portal which depends on the SMCore itself.

13.7.2 Change the Service Manifest structure

As explained in this document, the SM has been changed to allow the Self-Contained Service storage. However it has been noticed that modifications represent “additions” allowing to store the necessary information but they do not affect the general structure that reveals to be flexible enough.

13.7.3 Modify the SM Editor in order to support self contained interaction forms

The SM Editor has been modified both in its implementation and functionalities to allow the entering of information (code and data) necessary for the IF Wizard. It must be reminded that, normally, only the IF Wizard can be created by using the SM Editor, although it is possible to directly create an User SCS.

13.7.4 Adapt the DBE Portal component

The DBE Portal has been modified to allow the execution of Self-Contained Service. Differently from traditional services, the User Interface is not required to the proxy of the service but it is directly extracted from the SM. Also in this case, the modifications required have been very limited demonstrating in this way that

that DBE has been conceived to easily support its evolution in the next coming years.

13.8 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 (it can make use, for example, of a Database, some Web Services exposed by the Provider to interact with its IT system) but it must have a Server (and a DBE Servent) where to be executed. 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 furnished, against payment 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 Wizard and a SCS Wizard is that the first must also take care of deploying the service in the DBE Servent.

As the User Service is a traditional DBE service, possible data of a Distributed Interaction Form will not be stored inside the SM but they will be entered into a database. This allows to manage relevant volume of data.

For the Distributed Interaction Form all the functionalities of the accounting service can be used, on the contrary they cannot be for the Self-Contained Service.

13.9 Limitations of the Interaction Form and the Self-Contained Service

Although in theory the Interaction Form and the Self-Contained Service have no particular limitations, some have been defined to facilitate their introduction in DBE and to deal with security.

1) Self-Contained Service cannot be considered as a “complete” DBE service because they are not distributed on a server but they are executed on the browser. It is not possible to use DBE services such as accounting and metering, etc. etc.
2) It has been considered as not secure to execute the code contained into the SM on the CSS (Client Side Server). And, in any case, the presence of a considerable number of Self-Contained Services could have a negative impact on the performance of the server itself.
3) The Distributed Interaction Form are ordinary DBE services without any sort of limitation, but they necessitate a server to run. The Service Template Supplier could also provide the server. These services can access the Database and the IT System of the provider.

14 .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 Link		Abstract Link, Virtual Data	It means that the object has to store some data in order to allow external processes to get the related data. This is done in a CIM perspective. This does not imply that the data are stored in the object nor the object has a link to the data, but only that the data

Term	Acronym	Synonymous	Description
			are reachable/obtainable for the object.
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 manner [ISG].
DBE CA			Certification Authority of DBE
DBE Portal			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.
Enveloped Signature			The signature is over the XML content that contains the signature as an element. The content provides the root XML document element. Obviously, enveloped signatures must take care not to include their own value in the calculation of the SignatureValue [XMLSig].
Enveloping Signature			The signature is over content found within an Object element of the signature itself. The Object (or its content) is identified via a Reference (via a URI fragment identifier or transform) [1]
Fitness Data			In evolutionary computing the fitness is how well it performs relative to the desired goal.

Term	Acronym	Synonymous	Description
			Fitness Data refer to all the data needed to measure the success of a service in an environment.
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.
JAD Session			JAD means Joint Application Development is a methodology that involves the end user in the application's development. A JAD Session is a meeting of a discussion group.
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].
Message Authentication			A signature should identify what is signed, making it infeasible to falsify or alter either the signed matter or the signature without detection [DSG].
Platform Independent Model	PIM		The Platform Independent Model is a viewpoint of a system which focuses on the information about 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

Term	Acronym	Synonymous	Description
			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].
Quality of Service Information	QoS		Information about the technical Quality of the service (i.e. like “average number of active proxy”, “no proxy available time”)
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]
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.

Term	*Acronym*	*Synonymous*	*Description*
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 Usage Data	SUD		When a service is used by a customer the event is logged in the DBE Memory for further usage.
Signer Authentication			A signature should indicate who signed a document, message or record, and it should be difficult for another person to produce the signature without authorization [DSG].
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.
To Be Developed	TBD		Areas in the document which will be developed in the next versions of it
Value Added Service	VAS	Service Chain	A complex service composed by many simple services.
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.

15 .References

An IDE for DBE Studio: J.Kennedy, D.McKitterick, Introducing the DBE Studio: an IDE for the DBE, 2006
BMLMM: ISUFI, BML Metamodel Ver. 2.0.1, December 2004, ISUFI
Control of Self-organization: Briscoe, De Wilde, D.6.2 - Control of Self-organization and a Performance Measure, 30/04/2005, ICL
D.36.1 - SOTAComposedServices: P. Malone, State of the Art in Accounting for composed service, October 2004, WIT
D16.1 part II: SM Conceptual and Software Models: Soluta.net, D16.1 part II: Service Manifest Conceptual and Software Models, October 2005, Soluta.net
DBE 2nd Review Report: DG INFSO - EC, DBE Consolidated Second Review Report, 2006
DBE Studio: David McKitterick, DBE Studio Integration, 2006, INTEL
DBEArchReq: Pierfranco Ferronato, DBE Architecture Requirements Ver 2.2, August 2004, Soluta.net
DBECoreArch: Pierfranco Ferronato, DBE Core Architecture Ver 01.3, 2004/06/04, Soluta.net
DSG: Digital Signature Guidelines, http://www.abanet.org/scitech/ec/isc/dsgfree.html
http://swallow.sourceforge.net: ,
ISG: Internet Security Glossary, http://www.faqs.org/rfcs/rfc2828.html
KBDI: TUC/MUSIC, Knowledge Base Design and Implementation Status Ver 0.1, October 15, 2004, TUC/MUSIC
MDA: David S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, January 10, 2003
MDA Guide: OMG, MDA Guide Version 1.0.1, June 2003, Object Management Group
MOF: Meta-Object Facility,
http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF
Report on Fitness Landscape: STU, D.9.1. DBE REport on Fitness Landscape, 2005,
SBVR Editor & BML framework: , SBVR Editor first release & BML framework, 2005,
SDL: , Service Description Model and Language Definition , April 2005, Soluta
SMCM: Claudius Masuch, Thomas Kurz, Giulio Marcon, Service Manifest Conceptual Model, 15th March, 2005, Salzburg - Austria
SR: TUC, 1st P2P distribuited implementation of the DBE Knowledge Base and Semantic Registry, 2005,
XMI: XML Metadata Interchange (XMI), http://www.omg.org/technology/documents/formal/xmi.htm
XML: Extensible Markup Language (XML), http://www.w3.org/XML/
XMLSig: XML-Signature Syntax and Processing- W3C Recommendation 12 February 2002, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

- end of document -