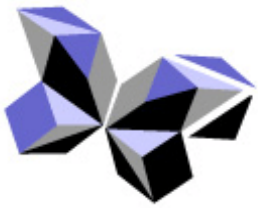

Digital Business Ecosystem

Contract n° 507953

# WP14: DBE Knowledge Base

# D14.1: DBE Knowledge Representation Models

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

**Contract Number**: 507953

**Project Acronym**: DBE

**Title**: Digital Business Ecosystem

**Deliverable N°**: D14.1

**Due date**: 30/4/2005

**Delivery Date**: 27/04/2005

**Short Description**:

This document describes the Knowledge Representation Framework and the Knowledge Representation Models that have been developed in the DBE project. The DBE knowledge is kept in the Knowledge Base (KB) of the system. The overall approach to the knowledge modeling and representation in the developed framework has been chosen in order to meet the functional and technical requirements of the DBE knowledge management and exploitation. To better illustrate the Knowledge Base contents, the overall Business Knowledge Spectrum of DBE and the inter-dependencies existing among the various segments of knowledge are presented.

The role of ontologies in the developed framework is crucial in order to support semantic interoperability among the SMEs that form a Business Ecosystem. Thus, representation of ontologies in a way that will enable their streamlined integration and exploitation in the entire DBE Knowledge Representation Framework was a fundamental objective of the DBE Knowledge Base. To this end, an Ontology Definition Metamodel has been developed in DBE as a generic Knowledge Representation Model and is presented in this document. Moreover, mechanisms for the specification of Business Model and Service Ontologies have been developed in order to capture the semantics of Businesses and Services in a way that allows at the same time interoperability and differentiation of business services in a digital business ecosystem. These ontologies are knowledge representation models that are integral parts of the DBE Knowledge Representation Framework.

**Partners owning**: TUC

**Partners contributed**: TUC, ISUFI

**Made available to**: All project partners and the EC

**Versioning**

| Version | Date | Author, Organization | Description |
|---|---|---|---|
| **0.1** | **10/04/2005** | **NEKTARIOS GIOLDASIS - TUC**<br>**FOTIS G. KAZASIS – TUC**<br>**YIANNIS MARAGOUDAKIS – TUC** | Initial Document Creation |
| **0.2** | **22/04/2005** | **PROF. STAVROS CHRISTODOULAKIS - TUC** | Revisions/Additions |
| **0.3** | **27/04/2005** | **ANGELO CORALLO - ISUFI**<br>**MAURIZIO DE TOMASSI - ISUFI** | Addition of BML short Description |
| **0.4** | **19/05/2005** | **NEKTARIOS GIOLDASIS – TUC**<br>**MAURIZIO DE TOMASSI – ISUFI** | Revisions according to the internal review process |

**Quality check**

**1st Internal Reviewer**: THOMAS KURZ – STU
**2nd Internal Reviewer:** PAUL MALONE - WIT

# Knowledge Representation Models

## Table of Contents:

## List of Figures:

## List of Tables:

# Executive Summary

This document describes the Knowledge Representation Framework of DBE Knowledge Base and its constituent Knowledge Representation Models. The role of the DBE Knowledge Base is to maintain the shared knowledge of the Digital Business Ecosystem and enable its exploitation by the ecosystem habitats through other system components like the Semantic Discovery Tool, the Recommender, the Fitness Landscape in the Evolutionary Environment (EvE), etc. The exploitation of this shared knowledge will enable semantic interoperability among SMEs that participate in the Digital Business Ecosystem.

The Knowledge Representation Framework of the DBE Knowledge Base (DBE KB) follows the OMG's MOF metadata architecture [5] where four different abstraction layers are defined for modelling, representing and organizing both data and metadata. According to this architecture, each piece of information stored in the KB has been produced and placed as an instance of a model at a higher layer (which actually describes how the information in the lower layer will be organized).

The core of the DBE Knowledge Representation Framework (DBE KRF) consists of domain specific ontologies. These ontologies capture community accepted semantics (concepts and relationships between them) of a particular business domain and they are the reference point of every specific model or semantic description allowing real knowledge sharing and semantic interoperability among SMEs. That is, SMEs model and semantically describe their business and their offerings (services) by re-using community-wide accepted and understood concepts.

The representation of domain specific ontologies in the adopted framework, which is based on the MOF metadata architecture, means that a new metamodel was needed. Such a metamodel should be able to describe new ontologies (that will be developed by regional catalysts or business experts) and at the same time it should be able to represent existing ontologies from the semantic web in order to make the Knowledge Representation Framework of the DBE KB, and thus the DBE itself, an open environment.

The need for an ontology definition metamodel in the MOF framework was also identified by the OMG itself which issued a specific Request for Proposals (RFP) [4] for this purpose. Leveraging the existence of this RFP, DBE has developed the needed Ontology Definition Metamodel consistently with this RFP. The DBE Ontology Definition Metamodel (ODM) is presented in this document.

The common conceptualisation of a business domain is captured in domain specific ontologies defined with the Ontology Definition Metamodel. For the modelling and description of specific businesses and services specific business model and service ontologies were needed. However, such ontologies must be specific to business and service modelling. That is, the mechanisms that will be used for their definition needs to enforce the user to define concepts that have specific semantics under the prism of their purpose. For example, a business model ontology that defines a business model needs to make use of primitives that have a clear meaning and interpretation in business modelling and not in generic knowledge representation. The modelling and representation of such ontologies is made by using specific modelling languages which provide the required modelling primitives to the business experts. Such languages are the Business Modelling Language (BML) and the Semantic Service Language (SSL). The abstract syntaxes of these languages (their meta-models) have been defined using MOF and are constituent parts of the DBE Knowledge Representation Framework.

These languages are used for modelling businesses and services. That is, they are defining the meta-data (M1 Layer) of the business descriptions according to the MOF meta-data architecture. The actual information (data) that describes a business or a service is an instance of either a BML model or an SSL model. This information is particularly important in DBE and thus the DBE Knowledge Base should be able to host and manipulate it in an effective way. That is, the DBE Knowledge Representation Framework should take care to provide the appropriate mechanisms that will allow the representation of this information (knowledge) in a consistent way and its smooth integration into the entire framework. The MOF meta-data architecture does not specify how raw information (M0 data) is encoded and represented. The DBE Knowledge Representation Framework has developed a specific mechanism for this purpose. This mechanism is the DBE Instantiation Metamodel that is used to represent the instances of BML and SSL models (i.e. the BML and SSL descriptions) and it is presented in this document.

In the second phase of the project the DBE Knowledge Representation Framework will be extended to allow representation of the biosphere of an ecosystem. With the term biosphere we mean those external factors that influence the life of an organization in the business ecosystem. In this project the biosphere will be consisting of the Regulatory Framework that holds for an ecosystem since it is an external factor that has an effect on how companies operate (behave) in the ecosystem. User Profiles (created with the User Profiling Mechanism) are also important knowledge that needs to be captured in the DBE Knowledge Base. Thus, the User Profile Representation Model that will be developed in the second phase of the project needs to be integrated into the entire Knowledge Representation Framework.

# 1. Introduction

The DBE Knowledge Base (KB) provides a consistent description of the DBE world and its dynamics, as well as the external factors of the biosphere affecting it. Its contents according to the DBE KRF include:

- **Representations of domain specific ontologies** (shared conceptualisation in a particular domain);
- **Descriptions of the SMEs themselves** in terms of business vocabularies, business models, policies, strategies, views etc.;
- **Description of the SME value offerings (services)** and the achieved solutions (service chains/compositions) for particular SME needs.
- **Description of the Regulatory Framework** that will define rules of operation for the SMEs participating in DBE.

The DBE Knowledge Base is mainly used to provide a consistent knowledge representation framework and knowledge management infrastructure that will be the main input in the service creation and composition process and in the service discovery and the recommendation process. Moreover, the DBE Knowledge Base will represent entities, relationships, and rules of the DBE world, and of its surrounding biosphere in a way that will be easily accessed and exploited by the other components of the DBE infrastructure (e.g. composer, fitness landscape, etc.).

Several knowledge representation models are needed in order to effectively describe the various aspects of DBE world. However, all these models need to be integral parts of a coherent Knowledge Representation Framework which will take into account all the functional and technical requirements of the DBE. This way, all Knowledge Representation Models will have a common approach and a certain role under the umbrella of the entire DBE Knowledge Representation Framework. It is thus crucial (as in every project) to start by investigating the functional and technical requirements that should be met by the DBE Knowledge Representation Framework (DBE KRF) and then to design and develop it in a way that will satisfy these requirements.

In the following sections we discuss the requirements that affect the design and development of the DBE Knowledge Representation Framework and its models. We discuss separately the functional from the technical requirements in order to distinguish requirements coming from the exploitation plans of the DBE Knowledge and the sustainability aspects of the project (i.e. functional requirements) with requirements coming from the system level, architectural considerations.

## 1.1 Functional Requirements

Functional Requirements for the DBE Knowledge Representation Framework and its models come mainly from the role that the DBE Knowledge Base is intended to play in the entire DBE system. The DBE Knowledge is the memory of the ecosystem; the place where each inhabitant stores its own knowledge, both private and public. SMEs (the inhabitants) are storing in the DBE Knowledge Base their own description, the description of their offerings, their preferences, etc. Moreover, knowledge that is not generated by the ecosystem itself, but that affects its operation will be also placed there. Such knowledge is the Regulatory Framework that affects not only the behaviour of the ecosystem's inhabitants, but also their existence. Our purpose is to exploit this knowledge within DBE in two ways: a) to enable semantic interoperability among SMEs allowing at the same time for business differentiation,

and b) to allow software SMEs to (semi-) automatically generate software from the description (knowledge) of the real-world behaviour (activities) of the SMEs.

### 1.1.1 Semantic Interoperability and Business Differentiation

In the past years, many mechanisms and standards have been developed that allow syntactic interoperability among systems developed in different platforms, from different software companies, and are owned by different organizations. The emergence of the eXtensible Markup Language (XML) [10] has fired a new revolution in software integration and interoperability. The ability of XML to encode and describe any kind of data that are to be transmitted between systems and the adoption of XML-Schema for defining custom schemas for XML documents has led to an increased syntactic interoperability among software systems. A set of XML derivative technologies has led to a new generation of distributing computing applications and in the industry has enabled many use cases for business collaborations. Such technologies refer to encoding and transmitting of XML messages over the internet (i.e. SOAP) [14], to description of software program interfaces in a way that other programs can understand (i.e. WSDL) [13], and to transformations of data from one structure to another (i.e. XSL-T) [15].

This syntactic interoperability is a strong requirement of system integrators and it is satisfied to a large degree. However, to enable true collaboration of industries we need semantic, not just syntactic, interoperability of business systems. Semantic interoperability means that there is a set of common semantics (e.g. a common terminology) used in business collaborations over the web. This need has been identified in the recent years, mainly by the academic community and there are many ongoing efforts for bringing semantics to the web. The main result of these efforts is the use of Ontologies as the main mechanism for describing a community-wide accepted conceptualization (i.e. model) of a specific domain of discourse. Many languages have been developed for describing ontologies and after the required “evolution”, one of them has been standardized (Ontology Web Language / OWL) [1]. Through the use of ontologies, semantic interoperability among parties is achieved by having those parties to adopt the same model of knowledge representation.

In DBE, the use of ontologies as an important mechanism for knowledge sharing and semantic interoperability was identified from the very beginning of the project. Business Model Ontologies and Service Ontologies are recognised as the mechanisms that will capture the semantics of the businesses and their services accordingly.

While an ontology based approach is reasonable for semantic interoperability purposes, one has to be aware of an additional important functional requirement for the business environments that has to be taken into account. This has to do with the need of businesses to differentiate from each other in order to avoid market commoditization. If every company was competing in a business domain offering exactly whatever the other businesses were offering, the competition among businesses would be based on price. This is an undesirable business environment for many businesses and especially SMEs.

For the DBE Knowledge Representation Framework (DBE KRF) this functional requirement of DBE means that SMEs should be able to describe themselves in their own way (i.e. using their own model which may be adopting certain domain ontologies if they wish, or parts of more than one ontology, and/or define their own model completely). That is, the DBE Knowledge Representation Framework should include mechanisms that give the ability to define both semantic models and instances. Thus, this framework should provide a set of meta-modeling mechanisms that will allow the definition of semantic business and service models. Moreover, it needs to provide appropriate mechanisms that will allow SMEs not only to define models, but also to create and represent instances of these models.

### 1.1.2 Knowledge Exploitation in Software Generation

One of the strategic objectives of DBE is to help software provider SMEs to generate software structures that meet the requirements of SMEs (their clients) and to evolve these structures as the requirements are changing. This process of software generation and evolution should be automated as much as possible. That is, some kind of software should be (semi-) automatically generated (or evolved) according to the semantic descriptions of the SME Business Model or Services that it wants to offer.

For the Knowledge Representation Framework this functional requirement means that the semantics of business models and services should be captured and represented in a formal and unquestionable way that will facilitate their "translation" into models of software systems from which software structures could be generated in a framework of layered model specification (from more abstract models to more specific and software oriented ones).

## 1.2 Technical Requirements

Beyond the functional requirements that are driving the design and development of the DBE KRF and its models, there are also important technical requirements that influence this process. These requirements are related to the detailed technical issues that need to be addressed by the developed system in order to meet the functional requirements, but also for ensuring the effectiveness and the efficiency of the entire system. Two important requirements that affect the entire Knowledge Representation Framework refer to Syntactic Interoperability and Knowledge Integrity Enforcement.

### 1.2.1 Syntactic Interoperability

Syntactic interoperability is not only a strong requirement in system integration but also provides the basic "infrastructure" for building semantic interoperability and knowledge sharing in business communities. The use of XML and its derivatives is today universally accepted as the most appropriate mechanism for information representation and encoding in business application environments in the web.

In DBE, independently of the internal knowledge representation structures that each SME is using, the shared Knowledge should be encoded and communicated in XML format. There is however an important peculiarity in DBE with respect to typical distributed systems. This peculiarity has to do with the nature of knowledge that needs to be represented and communicated among SMEs. While in typical distributed systems the communication among interacting parties might involve XML documents that conform to a pre-defined and pre-agreed schema, in DBE there is no pre-agreed schema at all. The DBE Knowledge Representation Framework should provide meta-modelling mechanisms that will be used by the DBE users to define new models (schemas).

### 1.2.2 Knowledge Integrity Enforcement

In DBE, where there is no pre-defined (global) schema defining the way that the semantics of businesses and their services are represented, we need to transmit both models and instances (data) in the same document. Thus, a clear technical requirement for the DBE Knowledge Representation Framework is to provide a mechanism that is able to describe in a uniform way the arbitrary structures (models/schemas) of the information (semantics) to

be exchanged, as well as the information (semantics) itself. Such a mechanism will not only allow the expression of both models and documents in the same way, but also it will facilitate the enforcement of data integrity in a framework that (conceptually) there is no global schema.

# 2. The DBE Knowledge Representation Framework

The DBE Knowledge Representation Framework needs to provide a set of integrated modelling mechanisms (modelling languages) that will allow domain ontologies to be described and also to allow SMEs to describe their semantics in their own way. Moreover, it needs to represent the instances of these models in a uniform way.

The development of such a framework is a meta-modelling process (i.e. development of metamodels or abstract syntaxes of modelling languages) which has to satisfy another important objective of the DBE project: to participate/follow (and contribute) to the state-of-the art standardisation activities regarding ontology, business, and service modelling as well as software design and implementation methodologies.

Before presenting the DBE Knowledge Representation Framework, we present the best known meta-data architectures in the literature. These are the metadata architecture of the semantic web, and the MOF meta-data architecture.

## 2.1 The Metadata Architecture of the Semantic Web

In [6] Berners-Lee has outlined the architecture of the Semantic Web. In this architecture the expressive primitives are incrementally introduced from languages in the lower lever (i.e. metadata layer) to those in the higher layer (e.g. logical layer), so that the languages of each layer can satisfy the requirements of different kinds of applications. The different layers that are defined by this architecture are as follows:

a) **The metadata layer:** This layer introduces a simple and general model for semantic assertions. This model contains just the concepts of resource and property which are used to express the meta-information and will be needed by languages in the lower layers. The Resource Description Framework (RDF) [7] is considered to be the general model in the metadata layer. The RDF data model provides an abstract, conceptual framework for defining and using metadata [8]. The basic data model consists of three object types:

   a. **Resources:** All entities being described by RDF expressions are called *resources*.

   b. **Properties:** A property is a specific aspect, characteristic, attribute, or relation used to describe a resource.

   c. **Statements:** A specific resource together with a named property and the value of that property for that resource is an RDF statement.

   Generally speaking, the RDF data model is “property-centric” (resource, property, value) mechanism.

b) **The schema layer:** This layer introduces simple Web ontology languages. These languages will define a hierarchical description of concepts (is-a hierarchy) and properties. When referring to classes (concepts) an *“is-a”* relation means that the instances of one class (sub-class) are also instances of the other (the super-class), while when referring to properties an *“is-a”* relation means that two classes that are related to one property (sub-property) are also related with the other (super-property). They use the general model of the metadata layer to define the basic metamodeling architecture of web ontology languages. RDF Schema (RDFS) [9] is the most widely accepted ontology schema layer language.

Although powerful and general, RDF primitives are very basic. One may define "Class" and "subClassOf" as resources in RDF. However RDF provides no standard mechanisms for declaring classes and (global) properties, nor does it provide any mechanisms for defining the relationships between properties or between classes. That is the role of RDFS–a Semantic Web language in this layer. RDFS is expressed by using the RDF data model. It extends RDF by giving an externally specified semantics to specific resources.

In RDFS, rdfs:Class is used to define concepts, i.e. every class must be an instance of rdfs:Class. Resources are described by RDF expressions and they are considered as instances of the class rdfs:Resource. The class rdf:Property is the class of all properties used to characterise rdfs:Resource instances. The rdfs:ConstraintResource defines the class of all constraints. The rdfs:ConstraintProperty is a subset of rdfs:ConstraintResource and rdf:Property. All of its instances are properties used to specify constraints, e.g. rdfs:domain and rdfs:range.

In RDFS, all properties are instances of rdf:Property. The rdf:type property models instance-of relationships between resources and classes. The rdfs:subClassOf property models the subsumption hierarchy between classes, and is transitive. The rdfs:subPropertyOf property models the subsumption hierarchy between properties, and is also transitive. The rdfs:domain and rdfs:range properties are used to restrict domain and range properties. In general, properties are regarded as sets of binary relationships between instances of classes. RDFS uses some primitives to define other modeling primitives. RDFS can be used to define ontologies as well, which makes it rather unique when compared to conventional modeling and metamodeling approaches, and makes the RDFS specification very difficult to read and to formalize [11].

c) **The logical layer:** In the logical layer, more powerful Web ontology languages are introduced. These languages are based on the basic metamodeling architecture defined in the schema layer, and define a much richer set of modelling primitives that may be mapped to very expressive Description Logics. Web Ontology Language (OWL) [1] is considered to be the accepted standard in this layer.

OWL builds on languages from the lower layers (RDF and RDFS), and extends these languages with much richer modelling primitives. OWL provides modelling primitives commonly found in frame-based languages. It has a clean and well defined semantics based on Description Logics. A complete description of the data model of OWL is beyond the scope of this document. However, we will illustrate how OWL extends RDFS by introducing some new subclasses of rdfs:Class and rdf:Property. One of the most important classes that OWL introduces is owl:Datatype. OWL divides the universe into two disjoint parts, the object domain and the datatype domain. The object domain consists of objects that are members of classes described in OWL. The datatype domain consists of the types defined in the XML Schema datatypes part. Both owl:Class (object class) and owl:Datatype are rdfs:subClassOf rdfs:Class. Accordingly, properties in OWL should be either object properties, which relate objects to objects (and are instances of owl:ObjectProperty), or datatype properties, which relate objects to datatype values and are instances of owl:DatatypeProperty.

## 2.2 The MOF Metadata Architecture

According to the Object Management Group (OMG) [5], *"the central theme of the MOF approach to metadata management is extensibility. The aim is to provide a framework that*

*supports any kind of metadata, and that allows new kinds to be added as required. In order to achieve this, the MOF has a layered metadata architecture that is based on the classical four layer metamodeling architecture popular within standardization bodies such as ISO and CDIF. The key feature of both the classical and MOF metadata architectures is the meta-metamodeling layer that ties together the metamodel and model"*. The traditional four layer metadata architecture introduces the following layers:

1. **The meta-metamodel layer.** It consists of the description of the structure and semantics of meta-metadata. In other words, it is the "abstract language" for defining different kinds of metadata.
2. **The metamodel layer.** It consists of the descriptions (i.e., meta-metadata) that define the structure and semantics of metadata. Meta-metadata constructs are informally aggregated into metamodels. A metamodel is essentially an "abstract language" for describing different kinds of data.
3. **The model layer.** It comprises the metadata that describe data in the information layer. Metadata is informally aggregated into models.
4. **The information layer.** It consists of the data that we wish to describe.

The example in Figure 1 is provided by OMG and exemplifies the classical four layer meta-modelling framework showing the organisation of metadata for some simple records along with the "RecordTypes" metamodel used to define the structure and the semantics of the metadata.

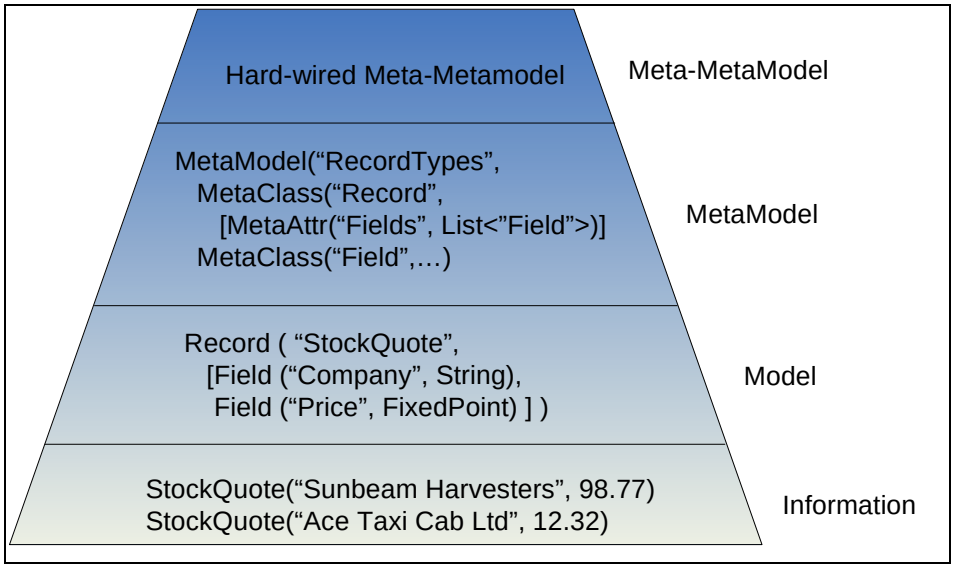


**Figure 1: The four-layer Metadata Architecture**

While the example shows only one model and one metamodel, the main aim of having four meta- layers is to support multiple models and metamodels.

According to the OMG [5], *"…The classical four layer metadata architecture has a number of advantages over simple modeling approaches. If the framework is designed appropriately:*

- *it can support any kind of model and modeling paradigm that is practically conceivable,*
- *it can allow different kinds of metadata to be related,*
- *it can allow metamodels and new kinds of metadata to be added incrementally, and*
- *it can support interchange of arbitrary metadata (models) and meta-metadata (metamodels) between parties that use the same meta-metamodel."*

The MOF metadata architecture, illustrated in Figure 2 is based on the traditional four-layer metadata architecture described below. This example shows a typical instantiation of the MOF metadata architecture with metamodels for representing UML diagrams and OMG IDL.

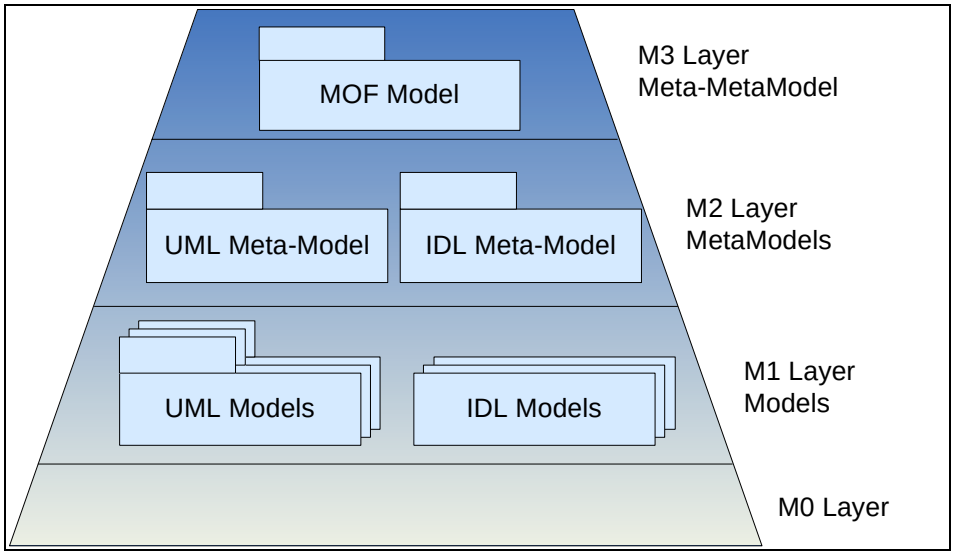


**Figure 2 : The MOF metadata architecture**

## 2.3 Evaluation of Metadata Architectures

Studying carefully the previously described metadata architectures one come to the conclusion that there are significant differences that make difficult the mapping from one approach to another. Pan and Horrocks in [12] have distinguished between fixed and non-fixed metamodelling architectures. In Non-Fixed metadata architectures (like that of the Semantic Web) it is possible to have any number of layers of classes. This happens due to the fact that RDFS does not distinguish between modelling primitives used for the ontology modelling and those used in the language level. In a fixed metadata architecture there is a predefined number of classes (in particular 2) and the instantiation between the different layers has clear semantics.

The two approaches have other differences too. For example, the rdfs:class is set-based. That is, a class is defined by its instances, while in the MOF approach the Class defines its instances. Also, in the RDFS an individual may be a member (instance) of more than one class, while in MOF each instance must have exactly one class. Moreover, in RDFS a class can be instance of itself, while in MOF this is not acceptable.

Although the metadata architecture of the semantic web is intensively used in the literature for the definition of ontology languages and ontologies in DBE we opted to follow the OMG's MOF metadata architecture for one simple reason; MOF, in addition to its metadata modelling provision, has a more software engineering orientation; it is used as the core component in the Model Driven Architecture (MDA) [27] of the OMG which is the state-of-the-art technology for (semi-) automatic software generation. This feature of MOF is particularly important to DBE given its strategic objective for exploiting business knowledge for (semi-) automatic software generation and evolution.

The MDA "*...defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. The MDA approach and the standards that support it allow*

*the same model specifying system functionality to be realized on multiple platforms through auxiliary mapping standards, or through point mappings to specific platforms, and allows different applications to be integrated by explicitly relating their models, enabling integration and interoperability and supporting system evolution as platform technologies come and go*" [27]. The five important concepts related to this technology are:

1. **Model** - A model is a representation of a part of the function, structure and/or behaviour of an application or system. A representation is said to be formal when it is based on a language that has a well–defined form ("syntax"), meaning ("semantics"), and possibly rules of analysis, inference, or proof for its constructs. The syntax may be graphical or textual. The semantics might be defined, more or less formally, in terms of things observed in the world being described (e.g. message sends and replies, object states and state changes, etc.), or by translating higher-level language constructs into other constructs that have a well-defined meaning. The optional rules of inference define what unstated properties can be deduced from the explicit statements in the model. In MDA, a representation that is not formal in this sense is not a model. Thus, a diagram with boxes and lines and arrows that is not supported by a definition of the meaning of a box, and the meaning of a line and of an arrow is not a model—it is just an informal diagram.
2. **Platform** – A set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.
3. **Platform Independent Model (PIM)** – A model of a subsystem that contains no information specific to the platform, or the technology that is used to realize it.
4. **Platform Specific Model (PSM)** – A model of a subsystem that includes information about the specific technology that is used in the realization of that subsystem on a specific platform, and hence possibly contains elements that are specific to the platform.
5. **Mapping** – Specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel. A mapping may be expressed as associations, constraints, rules, templates with parameters etc.

The DBE Knowledge Representation Framework has used MOF as a meta-metamodelling mechanism to develop Knowledge Representation Models that stand at both CIM and PIM layers of the MDA architecture.

Following the MOF metadata architecture and using MOF as a generic mechanism for the development of the Knowledge Representation Models DBE achieves the following two objectives:

a) It allows the definition, using the same mechanism (MOF), of modelling mechanisms that capture the semantics of the structural and behavioural aspects of the SMEs and their services in both business and software platform layers.

b) It enables the adoption of the Model Driven Architecture and thus the ability to leverage existing technologies for automatic or semi-automatic software generation.

## 2.4 The MOF-Based Knowledge Representation Framework of DBE

According to the followed approach (adoption of the MOF metadata architecture and use of MOF as a meta-metamodelling mechanism) each segment of information that will be stored in the KB should be placed as instance of a model at a higher layer of the OMG's MOF meta-data architecture. That is, MOF-based modelling languages (metamodels) will be defined for defining each segment of information.

Currently, the DBE Knowledge Representation Framework comprises the following Knowledge Representation Models (metamodels):

a) The Ontology Definition Metamodel (ODM) for modelling and representing ontologies

b) The Business Modelling Language (BML) for modelling and representing business model semantics,

c) The Semantic Service Language (SSL) for modelling and representing the semantics of the business offerings in DBE.

d) The Service Composition Metamodel (SCM) for modelling and representing the internal logic of the composite services

e) The Service Description Language (SDL) for modelling and representing the programmatic interface of a service.

An overview of the DBE Knowledge Representation Framework is illustrated in Figure 3, where each Knowledge Representation Model (both CIM and PIM level) is positioned in the corresponding layer of the MOF metadata architecture.

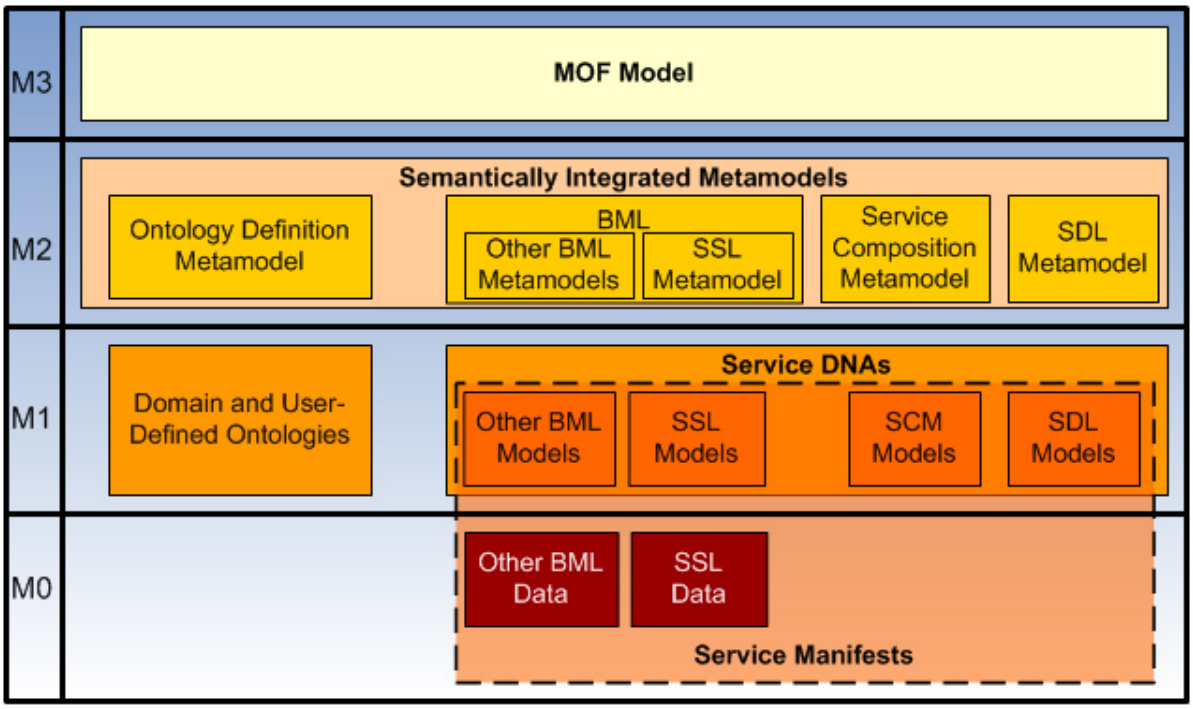


**Figure 3 : The MOF-Based Knowledge Representation Framework of DBE**

For ontology representation under the MOF perspective, the DBE project has decided to follow the OMG's Analysis and Design Platform Task Force (ADTF) and the Ontology Platform Special Interest Group approach of Ontology modelling [28]. The Group's main

objective is to coordinate all the semantic foundations of the platform independent models within the OMG domain activities and to encourage the various domain-oriented groups to adopt appropriate ontology technologies. To this end, the group has issued a Request for Proposals (RFP) [4] in order to define an Ontology Definition Metamodel capable of representing ontologies in the MOF metadata architecture. In the scope of the DBE Knowledge Representation Framework such an Ontology Definition Metamodel (ODM has been developed using MOF and is presented in chapter 4.

For the Business Knowledge Representation part of the Knowledge Representation Framework the DBE project has decided to join OMG's Business Enterprise Integration Domain Task Force (BEIDTF) vision for business modelling (i.e. Business Rules, Business Processes, and Business Organization) [29]. For the implementation of this vision, the group has identified a set of modelling components that are required for providing a business modelling framework and has issued a set of RFPs for some of these components. DBE has opted to join this vision and to respond to these RFPs by developing its own mechanisms for business modelling adhering as much as possible to this vision.

The OMG's framework for business modelling defines several metamodels (modelling mechanisms) that will allow the specification of several business aspects. In particular it has identified metamodels for modelling the Business Organisation, Business Motivation, Business Processes, and Business Locations. In addition to these business aspects, in DBE it has been identified that there is a need to model products and agreements. Thus, this framework will be enhanced with appropriate mechanisms for modelling also these business aspects.

Moreover, one of DBE's objectives is to capture the semantics of the SME business offerings (services) in the ecosystem. For this reason, we opted to extend the OMG's framework for business modelling by incorporating into it a metamodel that provides for modelling the semantics of services named Semantic Service Language (SSL).

All these metamodels (which are business and service knowledge representation mechanisms) constitute the abstract syntax of the DBE Business Modelling Language (BML) and they are enriched with appropriate graphical notations in order to be used for modelling purposes (non-graphical notations are also possible).

Typically, the BML models (M1 models in terms of the MOF metadata architecture) that a user will make using BML do not describe a particular business or service. Rather they define a blueprint for describing services of a particular type. The real description of a particular service is captured by instantiating an M1 BML model (i.e. creating an M0 layer of the MOF metadata architecture). As mentioned, MOF does not define how data at M0 is represented.

For the uniform representation of both models and data that is needed according to the technical requirements mentioned in section 1.2, a similar mechanism was needed in order to be able to represent (using MOF) the instances (M0 data) of BML models. For this reason, a new Knowledge Representation Metamodel has been developed, named Instantiation Metamodel (see chapter 7), which gives the ability to describe M0 data using MOF. This metamodel is not illustrated in Figure 3 since it does not allow the representation of a new kind of knowledge but only the representation of model instances.

For the technical description of the services (business offerings) there have been developed appropriate PIM level (in terms of MDA) metamodels that capture this kind of knowledge. There are two kinds of technical information that are needed for a given service: a) the internal workflow specification of composite services and b) the exposed service interface.

The knowledge regarding internal work flow and communication logic of composite services in DBE is captured through a Service Composition Metamodel (SCM) that has been developed to be compatible with the corresponding widely accepted specification for web services (i.e. BPEL4WS [19])

The same holds for the description of the service interface. A PIM level of a Service Description Language (SDL) [30] has been developed in compliance with the widely accepted Web Service Description Language (WSDL) [13].

The set of models for describing the aspects of an SME related to a specific real world service (i.e. the aspects that a candidate customer would wish to know), the model (or models) for semantically describing this service, and the model (or models) for describing technically the functional interface of the service implementation constitute a logical and physical unit named **Service DNA**. A Service DNA provides a complete blueprint for describing several services with similar semantic and technical characteristics. It is important to mention that the Service DNA does not describe a specific service (like Amazon Bookstore Service). Rather, it provides a model (more precisely a set of models) for describing services of this specific type (e.g. hotel reservation services, book store services, etc.). In MOF terms, a Service DNA is a set of M1 models for describing services (and -part of- their provider).

The real description of specific services is captured at a lower level (i.e. M0 in MOF terms) where each SME provides its own specific data (for itself, and its service). The logical and physical unit that contains the Service DNA and the data that describe a specific service offered by a specific SME constitute a **Service Manifest (SM)**. Service Manifests are the units that are published by service provider SMEs and searched by Service Consumer SMEs.

Future extensions of the current DBE KRF may include representation of user profiles (defined in WP7-User Profiling Mechanism) and representation of the regulatory framework (defined in WP32-DBE Regulatory Framework).

# 3. The MDA Approach to Knowledge Exploitation

An important reason for the selection of the MOF metadata architecture was due to its software orientation. MOF (along with UML) is the core modelling mechanism in the Model Driven Architecture (MDA) technology [27]. MDA is concerned with organising models used in the s/w development process so that developers can move from abstract models to more concrete models. This focus emphasizes the use of Computation Independent Models (CIM models), Platform Independent Models (PIM models), Platform Specific Models (PSM models) and mappings that allow the transformation of one model into another.

Figure 4 provides an overview of how the DBE Knowledge can be exploited in the (semi-) automatic software generation objective of the DBE. The creation of models in the lower layers of the MDA stack can leverage the corresponding models in the upper layer of the stack.

At the business level (mainly used by business analysts and experts) one can see Computation Independent Models (CIM) including the models used to describe from a business point of view the SMEs and their offerings. The M1 models created with the ODM (for describing domain specific ontologies), the Business Modelling Language (BML) and the Semantic Service Language (SSL) are all examples of the DBE CIM models.

At the information system level (mainly used by software architects and designers) one can see Platform Independent Models (PIM) describing a software design in a way that does not assume any specific implementation. Instances of the Service Composition Metamodel (SCM) and the Service Description Language (SDL) are typical examples of the DBE PIM Models. Moreover, Ontologies defined with ODM could also be referenced during the creation of the SCM and/or SDL models. That is, ontologies are used in both Business Level (CIM) and Information System Level (PIM) models.

Last, at the Information System Implementation Level (mainly used by the s/w developers and programmers) one can see PSM models that assure a target platform. Java, BPEL4WS and WSDL are possible examples of this layer.

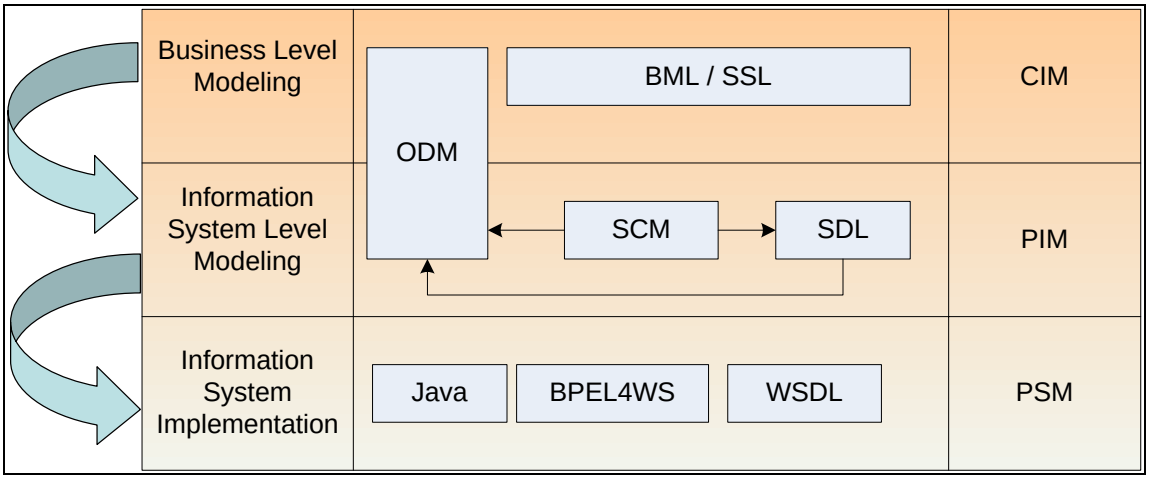


**Figure 4: The MDA Approach of DBE**

In the DBE Studio (an integrated toolkit that includes all DBE tools and allows the users to model businesses and to model, compose, or develop services), appropriate perspectives are provided for CIM and PIM modelling. The CIM perspective includes the BML/SSL editor

and allows business modelling of Businesses and Services. The PIM perspective integrates the Composer and the SDL Editor and allows the technical specification of the software components that will implement real world services. This description is done independently of the details of the underlying implementation platform.

Appropriate transformation tools can be developed to (semi-) automatically generate PIM and PSM models from the CIM and the PIM ones correspondingly. One such a tool is, for example, the SSL compiler (WP 16) defined for the second phase of the project, which will be able to generate SDL descriptions from SSL. Moreover, tools like the SDL-to-WSDL and SCM-to-BPEL would be further steps in this software generation process.

It should also be mentioned that for the accessing of the DBE knowledge a similar approach has been followed. A MOF-Based Knowledge Access Language (QML – Query Metamodel Language) has been defined that allows a uniform and platform independent definition of the knowledge access specifications for all the kinds of knowledge stored in the DBE Knowledge Base. The implementation of this language into the DBE Knowledge Base follows the MDA approach by allowing automatic generation (though appropriate mappings) of executable code from PIM level QML models (queries). More details regarding this aspect of the project can be found in the deliverable D17.1 – Recommender [32].

# 4. Representing Ontologies

This chapter describes the Ontology Definition Metamodel (hereafter ODM) that has been developed in DBE and is used for the representation of domain specific ontologies. Throughout this section examples will be used to illustrate how the various concepts of the metamodel are used in practice. The major design goal is the use of MOF for the definition of a metamodel compatible with the OWL language. Thus, OWL, RDFS, MOF and UML related terms will be frequently mentioned and a familiarity of the reader with these standards is assumed.

## 4.1 Metamodeling Approach to ODM

As mentioned in section 2.3, MOF uses a fixed, four-layer metadata architecture for metadata management while the metadata architecture of the semantic web is non-fixed, and defines its own layers with different semantics than those of MOF. These differences mean that in order to be able in the DBE to manage ontologies (as the term has been defined in the Semantic Web Community) we need to define our own Ontology Definition Metamodel using MOF. They also show that since MOF does not provide the expressive power that RDFS provides, special care should be taken to achieve the desired expressiveness in ODM. It should be noted at this point that OMG is currently working in the same direction. It is not ensured that the Ontology Definition Metamodel that is presented in this document will be compliant with the standard that OMG will finally produce. However, since the ODM developed in DBE meets the objectives set by OMG, we believe that the two metamodels will be highly compatible.

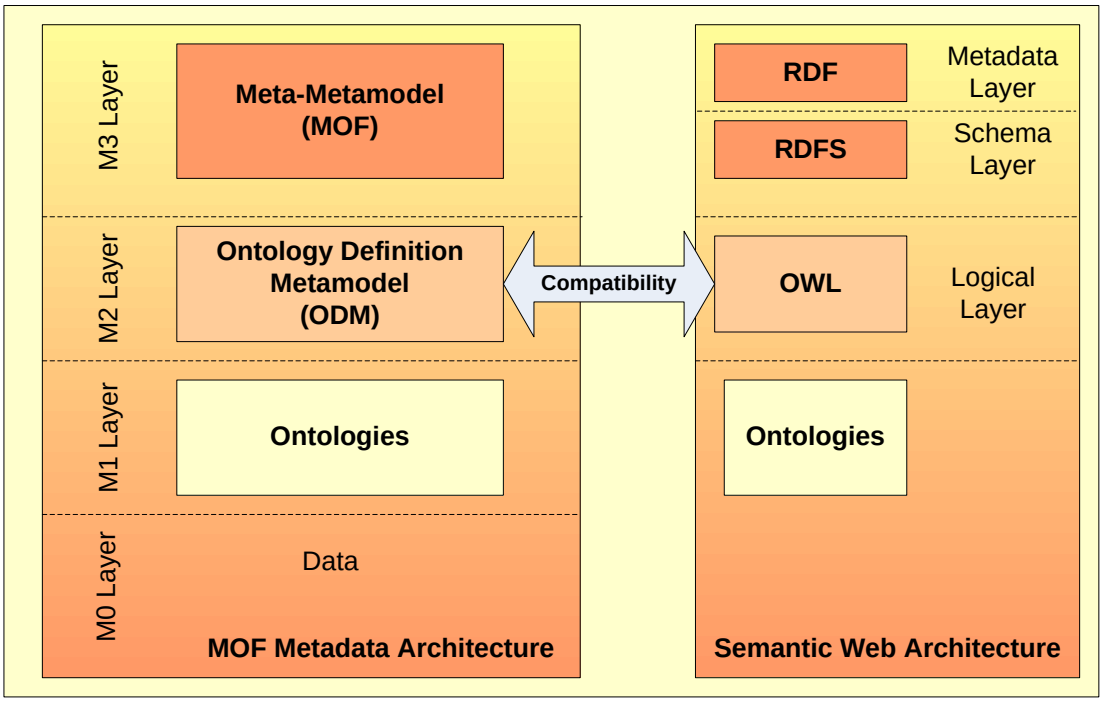


**Figure 5: Metamodeling Approach to ODM**

Figure 5 shows the ODM in the context of MOF metadata architecture and where it stands with respect to the Semantic Web's metamodeling architecture.

The Ontology Definition Metamodel presented here is an M2 layer Model in the MOF metadata architecture. Instances of this metamodel (M1 Layer Models) are the ontologies that the experts or the users define. It should be noted that according to ODM defined in this document, it is possible to describe both classes and their instances (individuals) in M1 layer. This has been done for two reasons:

a) To have the same mechanism used to represent both classes and individuals that are integral parts of ontologies.

b) To allow DBE to be able to handle existing ontologies described with OWL (since OWL describes in the same ontology both class definitions and individuals)

For the specification of the ODM metamodel (as well as for the other metamodels described in this document) the following MOF (meta-metamodel) primitives have been used:

1. **MOF Class** which models MOF meta-objects
2. **MOF Attribute** which models notional slot of value holder of a meta-object
3. **MOF Association** which models binary associations between meta-objects
4. **MOF Datatype** which models other data (e.g. primitive types, external types, etc.)
5. **MOF Package,** which modularizes models

## 4.2 Model Organization of the ODM

The Ontology Definition Metamodel consists of a number of packages each of which defines a specific aspect of the metamodel. Figure 6 shows the package view of the metamodel.

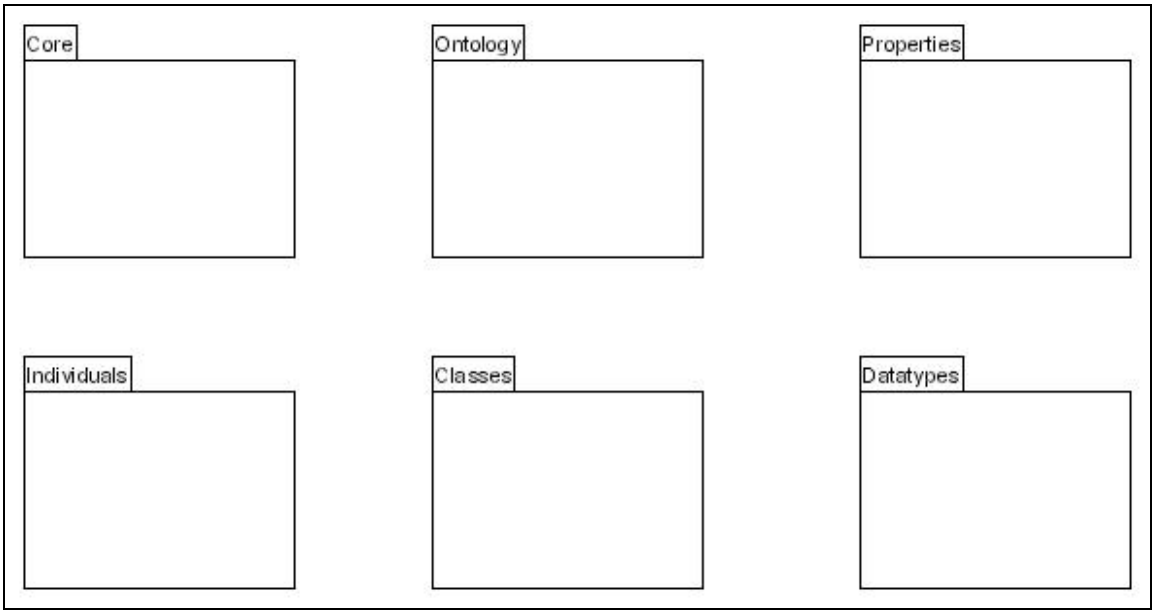


**Figure 6: Package View of the ODM**

- The package *"Core"* is one of the most important packages of the metamodel and defines the abstract concepts that are used as super-classes of any other concept in the metamodel.
- The package *"DataTypes"* defines also a fundamental part of the metamodel as it is introducing the concept of data type and its related concepts.
- The package *"Ontology"* defines what an Ontology is and how it is related with other concepts.
- The concept of *class* and its related concepts (e.g. class axioms, class descriptions, annotations, etc.) are defined in the package *"Classes"*.
- The package *"Properties"* defines various concepts related to properties of classes.
- The package *"Individuals"* defines how the instances of the ontology concepts are described and related to each other.

## 4.3 ODM Abstract Concepts (Core Package)

As mentioned above, the *Core* package defines the abstract meta-classes (hereafter classes) that are used as super-classes of all the other meta-classes in the ODM.

The root class in ODM is the class ***Element***. Element is an abstract class that is common super-class for all classes in the ontology definition metamodel. Every element that may be defined in an ontology and should be described by a name, is modelled with the abstract class ***NamedElement*** which is derived from the class *Element*. Named Elements are identified by an ***id*** which is unique in a given *namespace*. The abstract class ***NameSpace*** is used for scoping modelling elements. A namespace is a special kind of model element (a container) that contains other model elements. Thus, every element in an ontology belongs to one (at most) namespace. In the current version of the metamodel only an ontology can be used as a namespace for other elements (its enclosed concepts). Figure 7 illustrates what has been described in this paragraph.

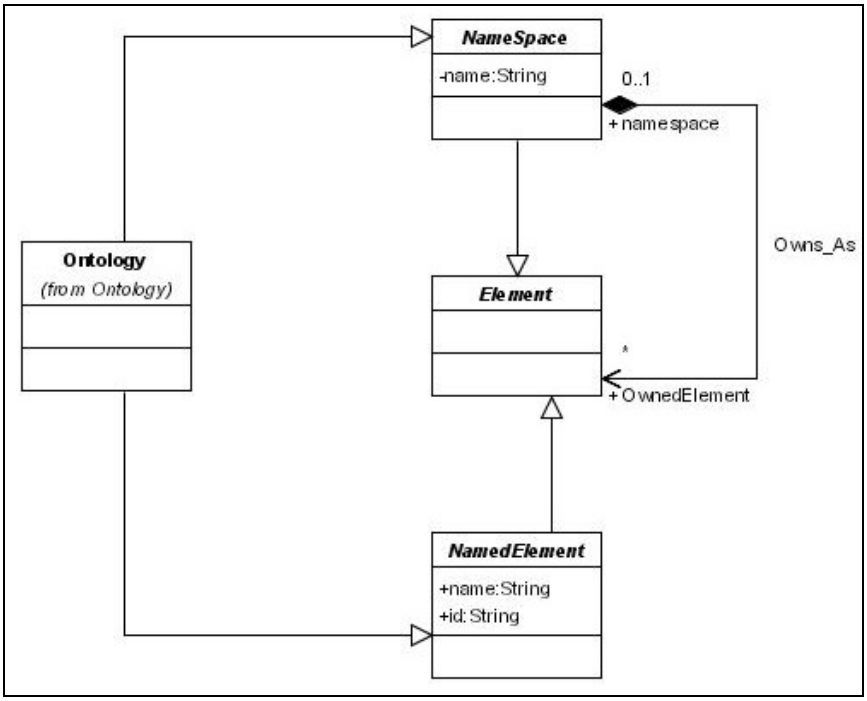


**Figure 7: ODM Abstract Concepts**

From the class diagram in Figure 7 one can come to conclusion that:

a) an ontology may contain other ontologies[1], and

b) an element may, or may not have a namespace[2]

Neither of these statements should be true. Thus, two well-formedness rules (i.e. constraints) must be enforced by the modelling tools in order to ensure correct modelling.

## 4.4 Ontologies

An ***Ontology*** defines the various elements that can be used to describe and represent a domain of knowledge. It includes definitions of basic concepts within a domain or across domains and the relationships among them, using elements, such as classes, individuals, properties etc. defined elsewhere in ODM. As illustrated in Figure 7, an ontology defines a namespace for the elements that it encloses. Broadly speaking, an ontology contains concept definitions/classifications (i.e. classes) like the definition of the concept "person", properties of concepts (e.g. the person's *name* or *age*), relationships between concepts (e.g. a *person has_friend* another *person*), and things or individuals (e.g. "John" who has_friend "Nick") which are actually instances of concepts (classes). Although these are the main primitives used to define an ontology, other constructs (such as restrictions, lists, etc.) can be defined based on them.

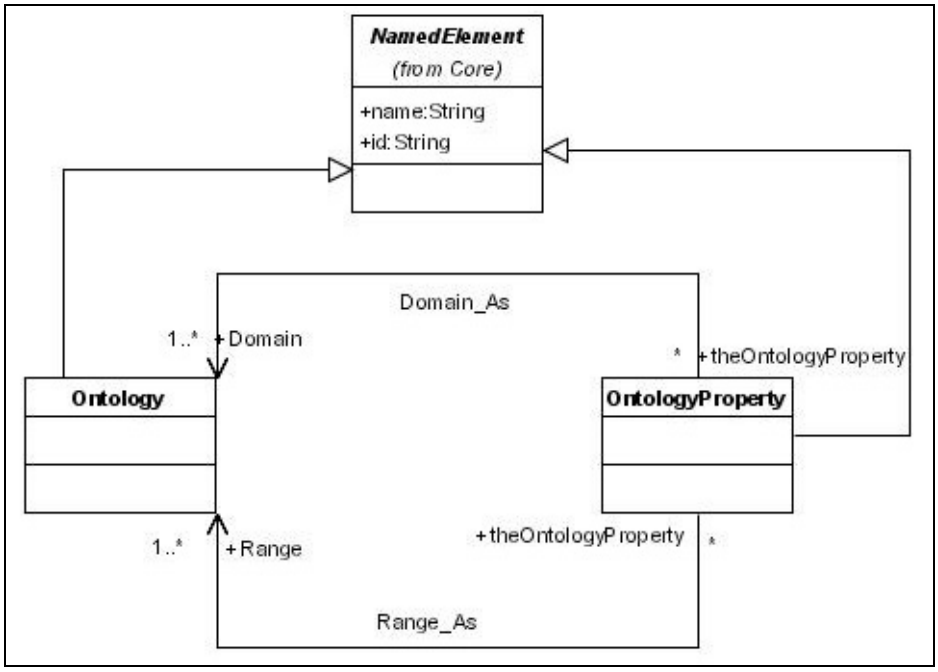


**Figure 8: ODM Ontology and Ontology Properties**

[1] Although an ontology cannot contain other Ontologies, it may import other ontologies by making use of the "imports" OntologyProperty described below.

[2] ODM does not allow ontology concepts (classes, properties, things, etc.) to be defined without a specified namespace. Only an *Ontology* can be specified without a namespace associated.

In the semantic web community (e.g. OWL DL), ontologies have specific properties that are used for semantic reasons. Such properties are modelled in ODM with the class ***OntologyProperty***. More precisely, an *OntologyProperty* relates ontologies to other ontologies. A number of constructs, such as versioning constructs, can be defined as instances of the *OntologyProperty* class. Instances of this class must be associated with specific ontologies (instances of the class *ontology*). As illustrated in Figure 8 an *OntologyProperty* instance must have an associated ontology as *domain* (source) and another associated ontology as *range* (target). Examples of such *OntologyProperty* instances are the properties *"imports*", *"priorVersionOf*", *"backwardCompatibleWith*" and *"incompatibleWith*" that define specific relationships between ontologies.

## 4.5 Classes

A fundamental concept in the Ontology Definition Metamodel is that of a ***class***. A *class* represents the grouping of similar elements (named things or individuals) inside an ontology. Thus, every class is associated with a set of individuals which is called the ***class extension***. The individuals that belong in a class extension are called the instances of the class associated with this class extension. A class has an intentional meaning (the underlying concept) which is related but not identical to its class extension. Thus, two classes may have the same class extension, but still be different classes.

It should be stressed that there is a fundamental difference between the concept class defined in RDFS (hereafter RDFS Class) and the one defined in MOF (hereafter MOF Class). A MOF class is the specification of its instances. That is, you should firstly define a class and then instantiate it when needed. All instances of that class have the same structure, relationships, constraints, etc. Instances of a MOF class cannot be at the same time instances of another class.

On the other hand, an RDFS class represents the grouping (i.e. the classification) of things or individuals (which are actually the instances of this class). For example, the class "DBEPartner" may be defined in the "DBE Ontology" which contains info in order to classify all the partners (institutes and companies) of the DBE project. In this example (the "DBE Ontology Example"), each partner is an instance of the class "DBEPartner". At the same time, in the same ontology, "TUC" may appear as an instance of the class "University", "IBM" as an instance of the class "Company", and so forth.

This is an important (conceptual) difference between MOF and RDFS standards and it is critical for the reader to understand what its effects are.

## 4.6 Class Definitions

In ODM a class may be defined either by a class name or by specifying the class extension of an unnamed class. As OWL does, ODM distinguishes six types of class definitions:

a) **As a new class with an identifier** (i.e. a class name and an id). This kind of class definition is similar to that followed by MOF. That is, a class that defines all its instances.

b) **As an exhaustive enumeration of individuals that together form the instances of a class.** As stated above, a *class* may be defined as a set of individuals (i.e. instances). Such a definition is achieved through the *oneOf*

association in Figure 9. The class extension is then all the things that are associated with the class through the (instances of) the *oneOf* association. For example the class "DBEPartner" in the DBE Ontology example could be defined as one of "TUC", "ISUFI", "TCD", "SUN", "SOLUTA", etc.

c) **As a property restriction.** For a detailed description of this kind of definition look at section 4.9.

d) **As the intersection of two or more class definitions.** This can be thought as the AND operator applied to the related class definitions and is done with the *intersectionOf* association. For example, the class "AcademicPartner" in the DBE example can be defined as the intersection of the classes "University" and "DBEPartner".

e) **As the union of two or more class definitions.** This can be thought as the OR operator applied to class definition and is done through the *unionOf* association. For example, the class "DBEStakeholder" in the DBE example can be defined as the union of the instances of the classes "DBEPartner", and "EC_Officer".

f) **As the complement of a class definition.** This can be thought as the NOT operator applied to class definition and is done through the *complementOf* association. For example, the class "FullyFundedPartner" could be defined as complement of the class "PartiallyFundedPartner" in the DBE example.

The first definition type is special in the sense that it defines a class as a «standalone» concept. The other five types of class descriptions describe an anonymous class by placing constraints on the class extension of other classes.

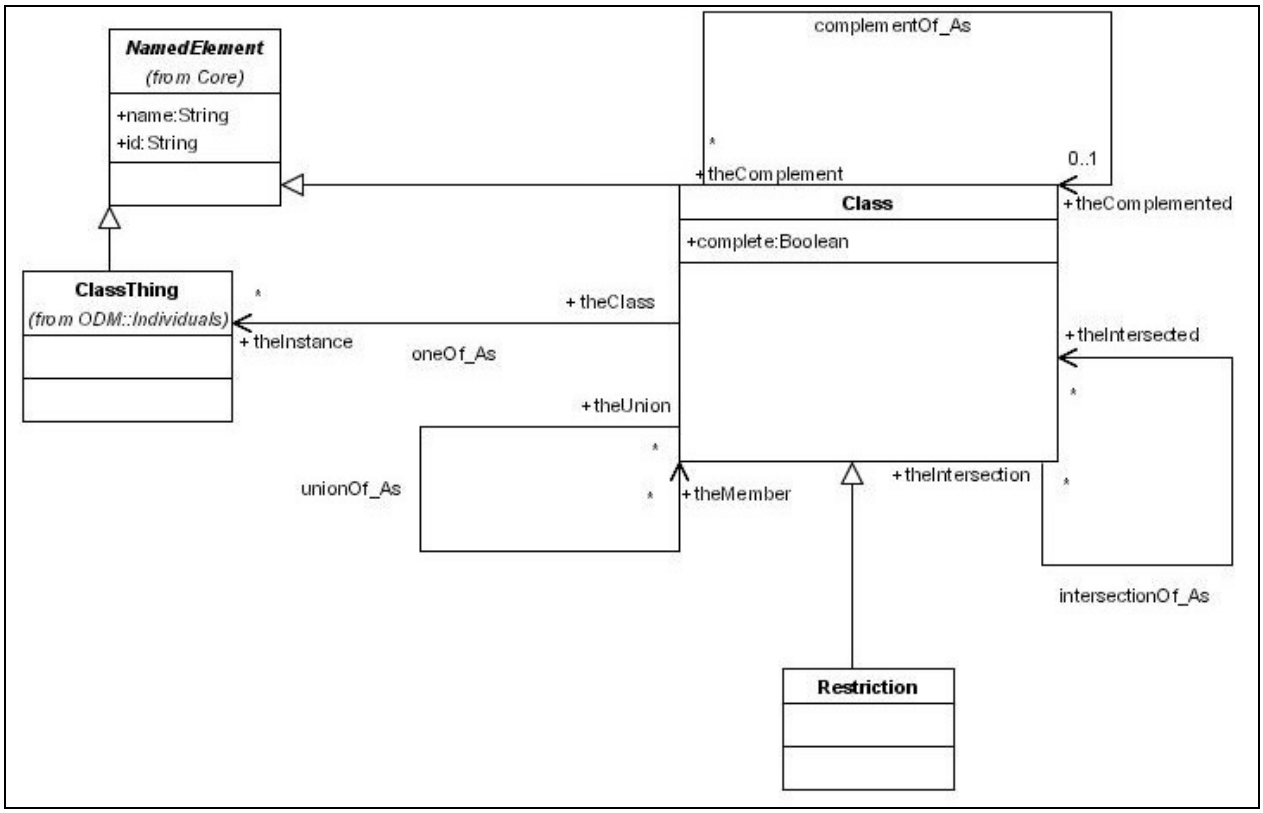


**Figure 9: ODM Class Definitions**

## 4.7 Class Axioms

The definition of a class may be separated or in connection with the definition of other classes. Consider for example the class “Scientific_Coordinator” defined in the DBE ontology separately from the other classes in that ontology. Although it is syntactically correct according to ODM, it does not tell us much about the concept “Scientific Coordinator”. If for example we knew that scientific coordinator is a special kind of “DBEPartner” it would be most useful and accurate in the specific domain of knowledge. To represent such a fact, special ***axioms*** (i.e. constructs that formally relate concepts) are needed. As OWL does, ODM defines specific constructs for combining class definitions into class axioms in order to provide a mechanism for relating classes inside or cross ontologies. In particular, it defines three language constructs for combining class descriptions into class axioms:

a) **subClassOf:** allows one to express that the class extension of a class definition is a subset of the class extension of another class definition. For example, a “DBEAcademicPartner” is a subClassOf “DBEPartner”. In principle, this definition implies that a class is a subclass of itself. However, since such a function has nothing to provide from a semantic point of view, ODM does not allow class definitions to be defined as sub-classes of themselves.

b) **equivalentClass:** allows one to say that a class definition has exactly the same class extension as another class definition. For example, we could define that the class “DBEAcademicPartner” is equivalentClass with the class “FullyFundedPartner” (since they have the same class extension).

c) **disjointWith:** allows one to say that the class extension of a class definition has no members in common with the class extension of another class definition. For example the class “DBEAcademicPartner” is disjointWith “DBEIndustrialPartner”.

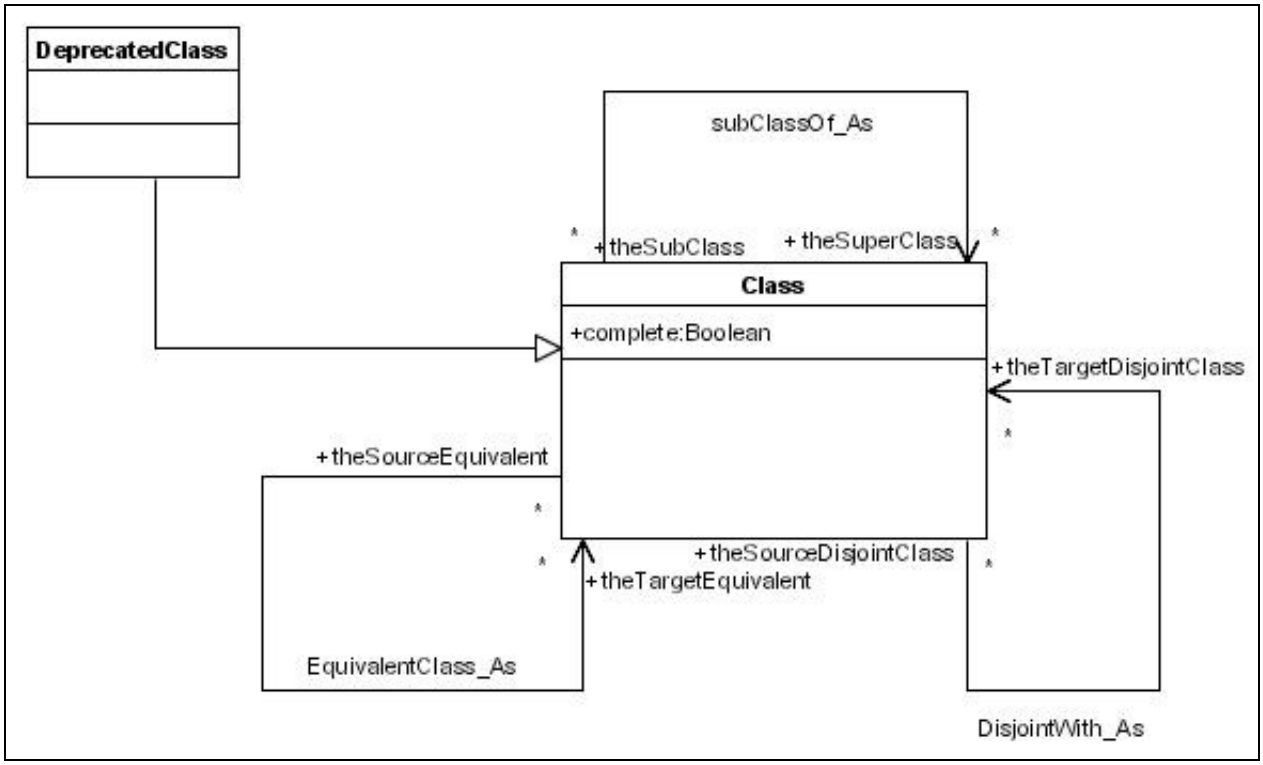


**Figure 10: ODM Class Axioms**

## 4.8 Class Properties

A class is used to describe a concept such as the concept “person”, the concept “car”, etc. in a specific domain of knowledge. In the real world, concepts have specific characteristics and/or may be related with other concepts. To address this need, the RDFS uses the *property* primitive. A property may be also considered as an advanced mechanism for relating specific individuals to other individuals or to specific data values. In the DBE ontology example, the individual “BML Editor”, that is an instance of the class “DBE_Task”, *isAssignedTo* the individual *“TUC”* which is an instance of the class “DBEPartner”. In this example, the predicate *isAssignedTo* has been defined as a property (of type “DBEPartner”) on the class “DBE_Task”. An instance of this property relates two individuals. Consider now that we want to describe that the task (individual) “BML Editor” *lastsFor* 12 months. Actually we now want to relate the individual “BML Editor” with the literal “12” with specific semantics.

The discussion in the previous paragraph clearly illustrates that a class may have two kinds of properties: properties that link instances of a class with the instances of another class, and on the other hand, properties that relate the instances of a class with specific data values (literals). If we consider properties as predicates that relate subjects with objects, then what is different in these two kinds of properties is the object. However, the subject (i.e. the domain) of both properties is the same (a class).

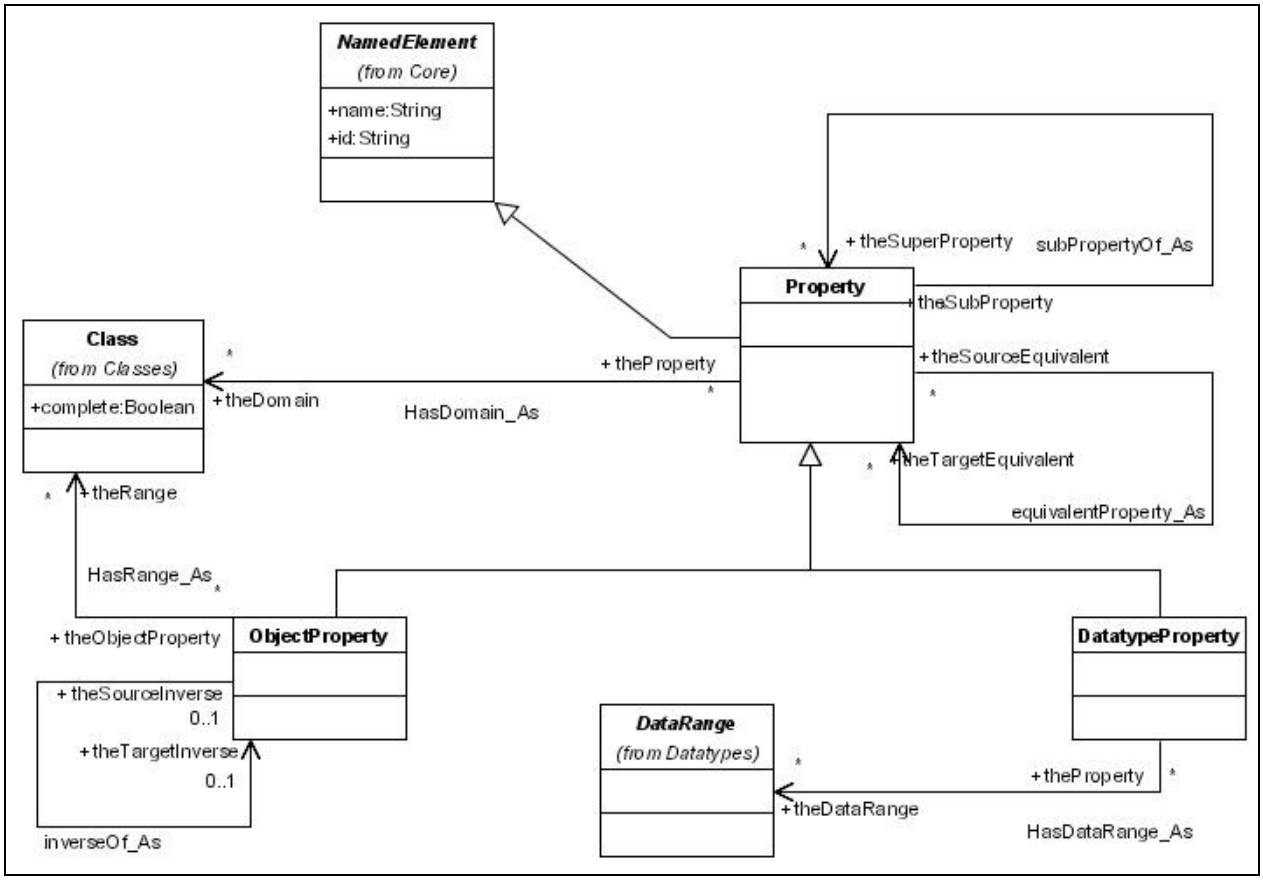


**Figure 11: Properties in ODM**

As far as the object (i.e. range) of a property is considered, ODM defines two kinds of properties as illustrated in Figure 11:

**a)** ***Object Property:*** a property that has as object (range) another class. As already mentioned, this kind of property relates instances of one class with the instances of another class. Since object properties relate classes to other classes, it may happen that an object property is the *inverseOf* another object property. For example, the object property "isAssignedTo" of the class "DBE_Task" in the DBE example, is *inverseOf* the object property "isInChargeOf" of the class "DBEPartner". The *inverseOf* association between properties is symmetric. That is, if A is inverseOf B, then B is inverseOf A too.

**b)** ***Datatype Property*****:** a property that has specific data values as object (range). For this purpose, the concept of *DataRange* has been defined. As it is described in section 4.13, the DataRange is an abstract construct to define that the range of a datatype property can be either a specific datatype (e.g. integer), or a set of data values. For example, one can define that the datatype property "lastsFor" of the class "DBE_Task" is an integer number (e.g. 12) of months, while the datatype property "Status" is set either to "Completed" or to "inProgress".

Properties in ODM, may be related to each other. In particular, a property may be defined to be *equivalentProperty* with another property. For example, the property *"lastsFor"* may be equivalentProperty with the property *"hasDuration"* of the classes "DBE Task" and "DBE WP" correspondingly in the DBE example.

Furthermore, in ODM a property (either ObjectProperty, or DatatypeProperty) may be sub-property of another property, thus allowing the definition of property hierarchies. Consider for example an object property *"hasRelative"* which has as domain and range the class *"Person"* in some ontology. The object property *"hasFather"* with the same domain and range (i.e. the class "Person") could be defined as *"subPropertyOf"* the property *"hasRelative*". This sub-classing of properties is another conceptual difference between RDFS and MOF. In MOF, class attributes and associations between classes are not generalisable elements. ODM offers this ability and special treatment should be foreseen in the notation that will be provided for modelling with the ODM.

## 4.9 Property Restrictions

ODM defines ***restriction*** as a special kind of *class* associated (through the *"onProperty"* association) with some already defined property. It is an anonymous class, namely a class of all individuals that satisfy certain property restriction. There are two kinds of property restrictions: ***cardinality constraints*** and ***value constraints***.

Any instance of a class may have an arbitrary number of values for a particular property. A cardinality constraint puts constraints on the number of values a property can take, in the context of this particular class description. Consider for example that the class "DBE Partner" in the DBE ontology example has an object property "assignedTask" that relates a "DBEPartner" instance with a "DBE_Task" instance. In this specific example, a cardinality constraint should define that there is no partner with no tasks assigned to it (i.e. the "AssignedTask" object property of the "DBEPartner" class should have "minCardinality=1" constraint defined). Formally speaking, the cardinality constraints are used to link a *restriction class* to a data value belonging to the value space of the XML Schema datatype nonNegativeInteger.

ODM defines three types of cardinality constraints:

a) **minCardinality:** it describes a class of all individuals that have *at least* N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint (a non-negative integer number).

b) **maxCardinality:** it describes a class of all individuals that have *at most* N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint (a non-negative integer number).

c) **Cardinality:** it describes a class of all individuals that have *exactly* N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint (a non-negative integer number).

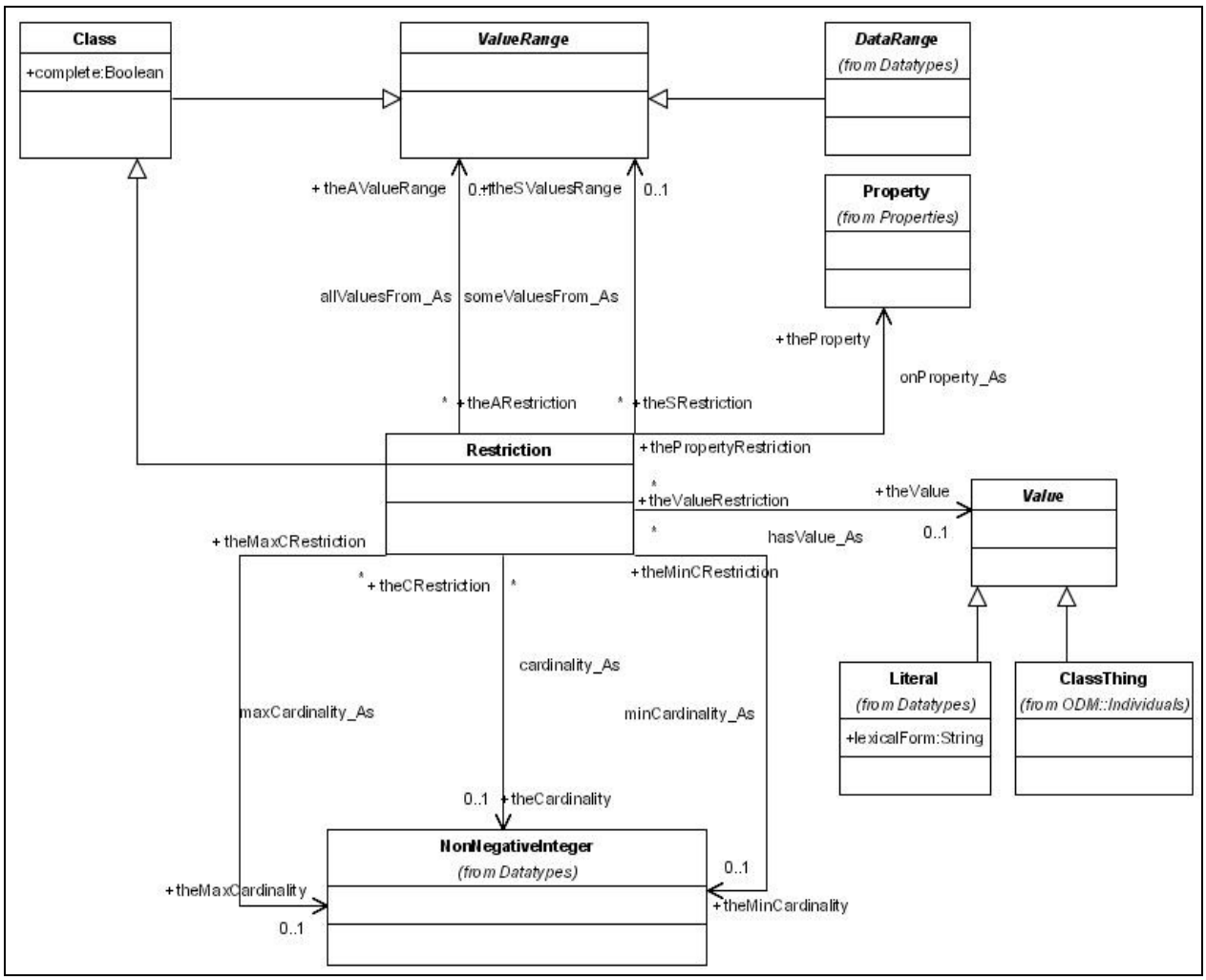


**Figure 12: ODM Property Restrictions**

A value constraint puts constraints on the *range* of the property when applied to this particular class definition. For example, suppose that in the DBE Ontology example there is a sub-class of the "DBE_Task" named "WPAdministration". Consider now that we want to create a special type of "DBEPartner" which is the "WPLeader" in some work package. This could be achieved by defining the "WPLeader" class as a value restriction on the "isInChargeOf" property of the class "DBEPartner". The restriction should define that this property must have *someValuesFrom* "WPAdmininstration". This means that a "WPLeader" "isInChargeOf" at least one "WPAdministration".

ODM defines three types of value constraints:

a) **allValuesFrom**: it links a restriction class to either a *class* definition, or a *DataRange* (through the abstract class *ValueRange*). It is used to describe a class comprised of all individuals for which all values of the property under consideration are either members of the class extension of the associated class definition or are data values within the specified data range.

b) **someValuesFrom**: it links a restriction class to either a *class* definition, or a *DataRange* (through the abstract class *ValueRange*). It is used to describe a class of all individuals for which at least one value of the property under consideration is an instance of the class definition or a data value in the data range.

c) **hasValue**: Links a restriction class to either an individual or a data value. Describes a restriction for all individuals for which the property concerned has at least one value semantically equal to the specified.

## 4.10 Property Axioms

A ***property axiom*** defines characteristics of a property. In its simplest form, a property axiom just defines the existence of a property. This happens when the ODM class *"Property"* is instantiated. ODM supports the following constructs for property axioms:

a) **subPropertyOf**. Defines that a property is sub-property of another property.
b) **Domain** and **Range**. Domain is the class for which the property is defined, while range is the range of values that the property can take. The range of a property can be set to a datatype (e.g. String), to an Enumeration (e.g. one of {"Car", "Bicycle", "Motorbike"}), or to a class implying that an instance of this class can be the value of the property.
c) **equivalentProperty**. Defines two properties as equivalent to each other.
d) **inverseOf**. Describes a property as the inverse property of another one. The domain of one property is the range of the other and vice versa.
e) **global cardinality constraints**:
    a. **FunctionalObjectProperty**. A functional object property is an object property that can have only one (unique) value for each individual. Object properties may be stated to have a unique value. This characteristic is referred to as having a functional object property.
    b. **FunctionalDataTypeProperty**. A functional datatype property is a datatype property that can have only one (unique) value for each individual. Datatype properties may be stated to have a unique value. This characteristic is referred to as having a functional datatype property. A Functional Property (both DataType and Object property) cannot have more than one values for the same individual. As an example, think of the property "lastsFor"; this property should have exactly one value for a given individual (e.g. "BML_Editor").
    c. **InverseFunctionalProperty**. An inverse-functional property is an object property whose value uniquely determines some individual. Inverse-functional properties resemble the notion of a key in databases. For example, the value of the property "PrimaryContactPerson" of the class "DBEPartner" uniquely identifies an individual of this class (e.g. "MiguelVidal" identifies "Sun").
f) **logical property characteristics**:

**a. SymmetricProperty**. A symmetric property is an object property for which holds that if the pair (x, y) is an instance of the property, then the pair (y, x) is also an instance of the property. A property is defined as being symmetric by making it an instance of SymmetricProperty, a subclass of object property.

b. **TransitiveProperty**. A transitive property is an object property for which holds that if the pair (x, y) is an instance of the property, and the pair (y, z) is also instance of the property, then the pair (x, z) is also an instance of the property. A property is defined as being transitive by making it an instance of TransitiveProperty which is defined as a subclass of object property.

**c. DeprecatedObjectProperty**. A deprecated object property is an object property that is not used anymore and typically has been replaced by another object property.

**d. DeprecatedDataTypeProperty**. A deprecated object property is an object property that is not used anymore and typically has been replaced by another object property.

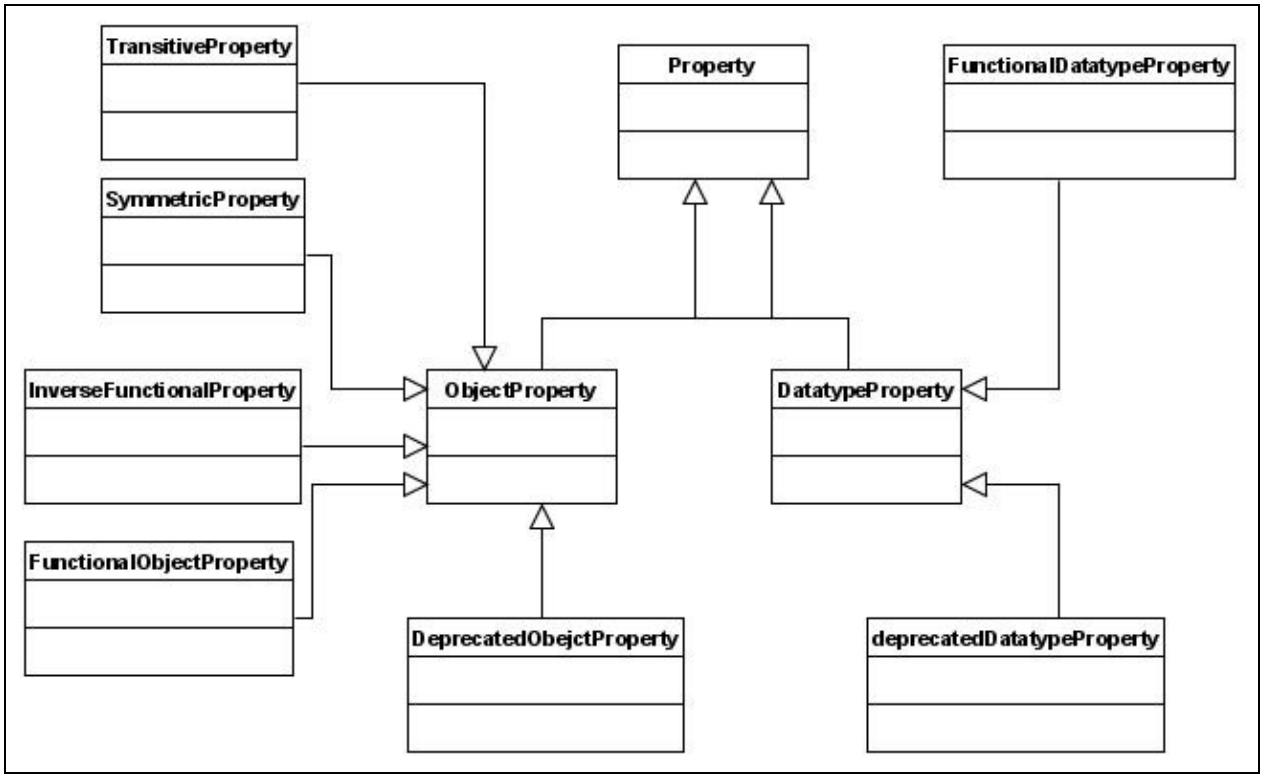


**Figure 13: ODM Constructs for Property Axioms**

## 4.11 Things or Individuals

Classes provide an abstraction mechanism for grouping resources with similar characteristics. Every ODM class is associated with a set of *individuals*, called the class extension. The individuals in the class extension are called the instances of the class. The ***Thing*** element is an abstract class used to represent all the instances (either instances of classes, or instances of Properties) in an ontology. As such, a *thing* may be *differentFrom* another *thing* (e.g. "BML Editor" is *differentFrom* "BML Definition" in the DBE Ontology

Example although both are instances of the class “DBE_Task”). In the same way, a thing may be *sameAs* another thing (e.g. “IBM” –instance of the class “DBEPartner”- is sameAs “PrimeContractor” –instance of the class “PMEB Participant”).

ODM defines three sub-classes of the class *Thing*:

1. **ClassThing:** This class is used to define instances of ODM classes. That is, it is used for representation of the individuals of a class. Each individual (instance of a class) should explicitly define its type (class).
2. **ObjectPropertyThing:** Represents the instances of the object properties of a class. As such, it relates *ClassThings* (individuals) to other *ClassThings* (individuals). Each ObjectPropertyThing must explicitly define its type (ObjectProperty).
3. **DataTypePropertyThing:** Represents the instances of the Datatype Properties of a class. As such, it relates a *ClassThing* to a *Literal* (data value). Each DataTypePropertyThing should explicitly define its type (datatype property).

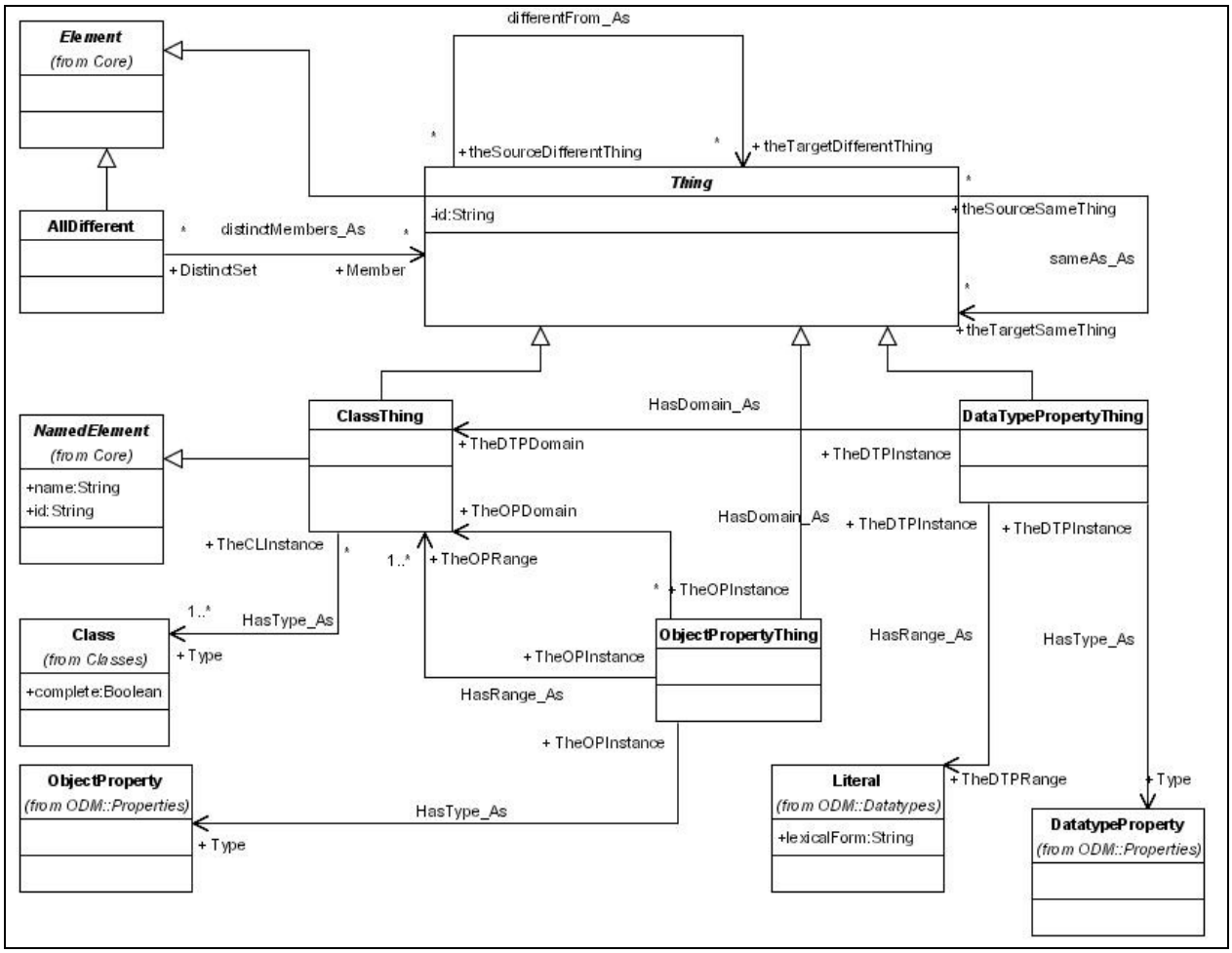


**Figure 14: Individuals in ODM**

### 4.11.1 Defining Distinct Groups of Individuals

When defining ontologies, one may want to define that some individuals in a list are all different from each other. Instead of saying that each individual is different from every

other, the ontology creator may use the *AllDifferent* construct to define that all the members of this set are different from each other.

## 4.12 Annotations

When defining ontologies, it is a common practice for the modeller to annotate the various concepts or things that he describes. For example, when defining a concept one may wishes to annotate this concept with a comment, a label, or whatever. To address this need, ODM introduces a set of concepts that allow the modeller to define his own annotation constructs. Figure 15 illustrates that part of the metamodel.

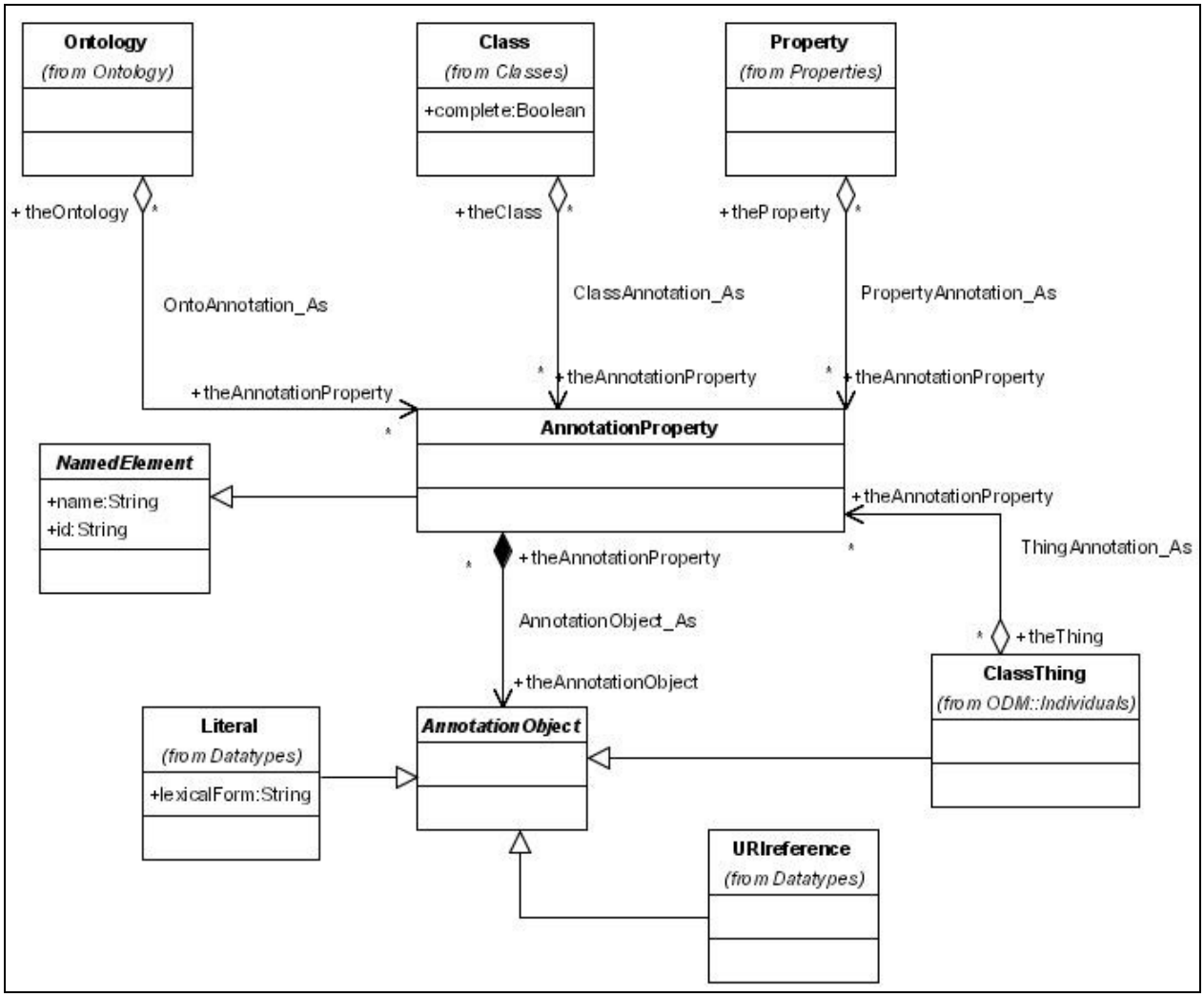


**Figure 15: ODM Annotations**

ODM allows annotations on classes, properties, individuals and ontologies by using the concept ***AnnotationProperty***. Instances of the class AnnotationProperty are associated with class definitions, properties, individuals and ontologies through the associations *ClassAnnotation*, *PropertyAnnotation*, *ThingAnnotation*, and *OntoAnnotation* correspondingly. The *AnnotationObject* of an annotation property (i.e. the "value" of the annotation property) must be either a data value (literal), or a URI reference, or an instance

of a class. OWL DL pre-defines some annotation properties like comment, label, versionInfo, seeAlso, and isDefinedBy. ODM opts not to pre-define annotation properties. Rather it provides the mechanism for defining annotation properties as shown in Figure 15. Tools that will be used for modelling with ODM may choose to import the owl pre-defined annotation properties as already defined instances of the ODM concept AnnotationProperty.

## 4.13 Data Types

When creating a metamodel using MOF one must define what the data types used at that level (M2) are in order to give a type to the attributes of the MOF Classes that he defines. Moreover, one should define a data typing scheme for the models of that metamodel, if non CORBA data types are to be used (which are the default data types used by OMG). Thus, in ODM there are data types for the M2 Level itself (*String*, *Boolean*, *Integer* classes stereotypes with the UML «DataType» stereotype) and also a data typing scheme for the data types that a model will use when defining ontologies.
The notion of data range has been mentioned repeatedly in this document in order to specify the range of data values that a property may take. ODM allows (in compatibility with OWL) two types of data range specifications:

a) **A Datatype specification.** It specifies a data typing scheme that is appropriate for referring to XML Schema data types. In particular ODM defines the Class ***DataType*** that will be used as the class of all data types that are defined in M1 layer. Each instance of the class "DataType" must provide a reference to the implementation/definition of the datatype. XML Schema Datatypes are expected to be defined once by ODM modelling tools as instances of this class and imported into ontologies when modelling. For example, the XMLShema String type must be defined as an instance of the ODM class "DataType" with the URIReference "http://www.w3.org/2001/XMLSchema#string" which indicates the definition of the type. Data values in ODM are instances of the ODM class ***Literal***. Literals may be plain or typed. Typed Literals have an associated URI Reference indicating the Definition/Implementation location of their type.
b) **An Enumeration specification**. In addition to the XML Schema Datatypes, ODM provides an additional construct for defining a range of data values, namely ***Enumeration***. This construct is a special *Element* that is a container of an ordered set of literals, named *Enumeration Literals*. An instance of this type is *oneOf* the *Enumeration Literals.*

The XML Schema data types that are expected to be used when modelling with ODM are the following:
The primitive data type xsd:string, plus the following data types derived from xsd:string:

- xsd:normalizedString,
- xsd:token,
- xsd:language,
- xsd:NMTOKEN,
- xsd:Name,
- xsd:NCName.

The primitive data type xsd:boolean.
The primitive numerical data types xsd:decimal, xsd:float, and xsd:double, plus all derived types of xsd:decimal

- xsd:integer,
- xsd:positiveInteger
- xsd:nonPositiveInteger

- xsd:negativeInteger,
- xsd:nonNegativeInteger,
- xsd:long,
- xsd:int,
- xsd:short,
- xsd:byte,
- xsd:unsignedLong,
- xsd:unsignedInt,
- xsd:unsignedShort,
- xsd:unsignedByte

The primitive time-related data types: xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, and xsd:gMonth. The primitive data types xsd:hexBinary, xsd:base64Binary, and xsd:anyURI.

In order to avoid definition of these data types by the user (modeller) it is suggested that the tools used for modelling ontologies will transparently define the data types that are needed.
What has been described in this section is depicted on the Figure 16.

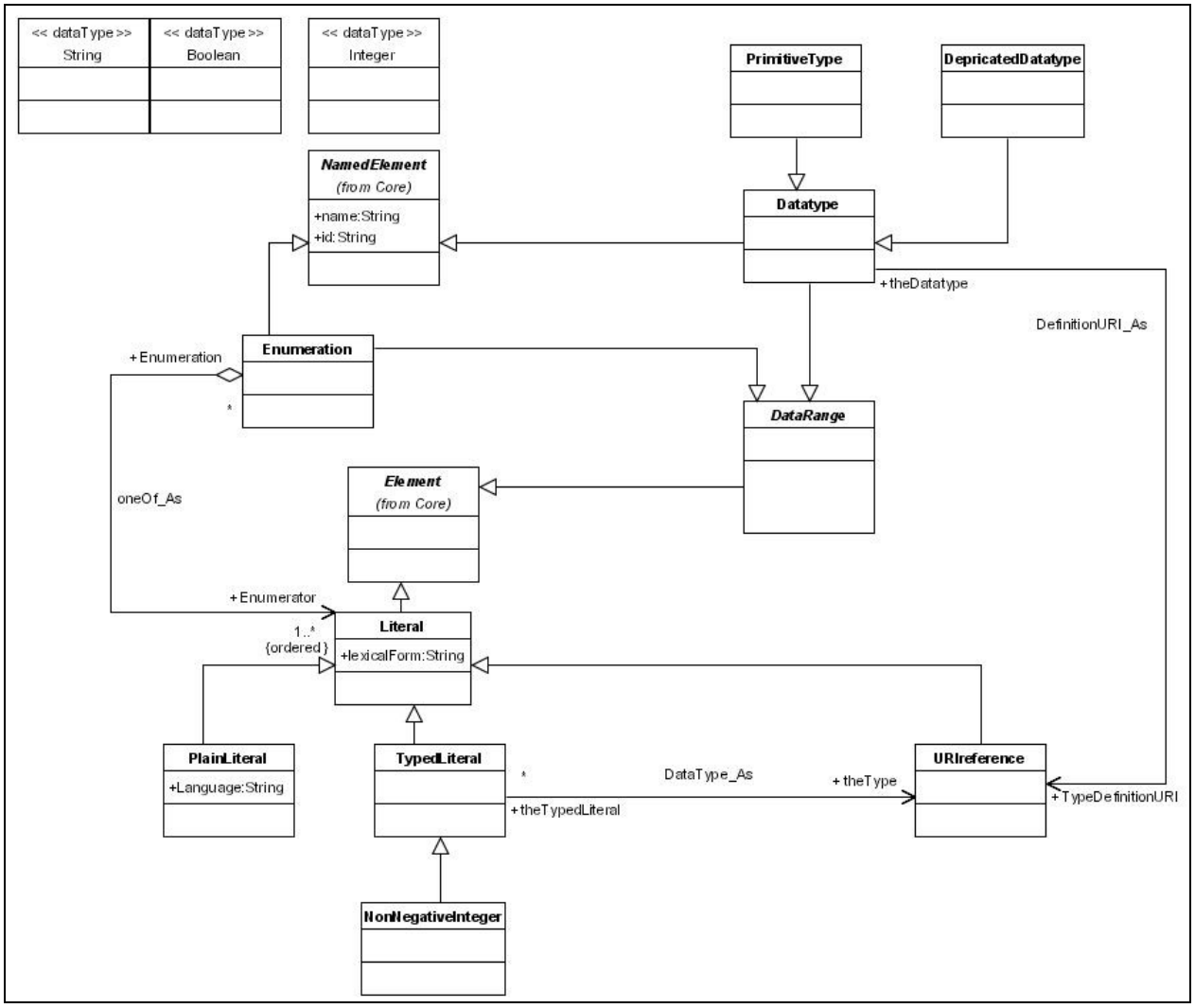


**Figure 16: ODM Data Types**

## 4.14 Summary of the Ontology Definition Metamodel

This chapter described the first version of the Ontology Definition Metamodel (ODM) developed in the DBE project for the representation of domain specific ontologies. The design goal of this metamodel was the compatibility with the Ontology Web Language (OWL) since it is the most widely accepted language for describing ontologies in the semantic web community. Hence, it is important for DBE to be able to handle (import and represent) existing ontologies described with this language.

Ontologies are a fundamental part of the knowledge base of DBE since they will be the main mechanism for the enabling semantic interoperability and representation of shared knowledge. Domain specific ontologies will be used to describe common terminology in specific business sectors. DBE users may also have their own ontologies (vocabularies). Domain and/or user defined ontologies will be extensively used by other modelling mechanisms (Business Modelling Language, Semantic Service Language, Service Description Language, etc.) in DBE, as well as by the recommender component for information retrieval reasons.

# 5. Representing Business Models

This chapter briefly presents a Knowledge Representation Model that is used to represent Business Models in DBE. Business Model representation in DBE is achieved though a set of metamodels that all together form the abstract syntax of the Business Modelling Language (BML). This chapter highlights only the architectural choices, the main achievements, and the future challenges in BML. More details regarding this language can be found in the BML Deliverable in [25].

To better explain and justify the architectural choices related to the Business Modelling Language (BML) the main BML objectives will be shown in this section, deriving them from the strategy that the Digital Business Ecosystem Project intends to apply, in order to improve e-business adoption by European SMEs. According to the project description, the two fundamental strategic objectives of the project are:

1. Enabling the competitiveness of small and medium enterprises that work in a global market of software production;
2. Giving a software solution that is able to adapt to the local needs of SMEs, supporting the use of ICT in the local innovation point [42].

To fulfil the first point, BML should be able to capture and represent enterprise business knowledge and make it available for the software engineer that has to develop software for that enterprise. More specifically, the BML framework should allow the business analyst to capture information related to the business profile of a company in order to build a business knowledge repository, expressed in machine readable language that is a source of fundamental information for the software requirements phase as much as for the design and implementation phase. This business knowledge should be represented in a machine readable format in order to grant compatibility with future development in the model driven software development methodology (Constraint 1). The second point clearly shows that BML has to sustain the design and development of an innovative service oriented architecture, with the specific task of delivering a mechanism to represent the business knowledge related to the actors involved in an e-business scenario; it means that BML should be expressive enough to be able to describe the SME in any e-business scenario (in terms of offered products or services description, agreements or contracts, organisational structure, business processes and business models) (Constraint 2). Another key statement, fundamental for the project strategy, is the involvement of open source communities and standardisation bodies. The project aims at using existing standards and contributing, through an interaction with communities, consortia and standardisation bodies, to their improvement, with specific reference to those frameworks and languages related to software development or e-business trade. Another key issue for the BML framework is that it should grant, as much as possible, the compatibility with a wide range of consolidated standards, with a specific regard to the e-business sector (Constraint 3). According to the DBE objectives, described in the previous paragraph, BML framework must grant a flexible mechanism through which territories and communities could derive their own languages and vocabularies tailored to their needs. At the same time the BML framework should grant coherence in time between the different developed domain models or vocabularies, allowing interaction among different local ecosystems (Constraint 4). These four macro constraints could be used to derive a set of specific requirements for the BML framework.

According to the first constraint identified, the BML has to be a general framework enabling business people to represent the business knowledge related to services and products offered and to the enterprises that stand behind such services or products, in order to allow

trade mechanisms based on semantically rich information models. The BML should be expressed in terms of business concepts that is, in terms of the actual concepts, actions and events that business people have to deal with as they run their businesses. BML should be independent of technological aspects, with the exception of such issues commonly accepted by business people as part of the terminology and standards of practice in each business community. BML has to handle many different kinds of information, such as the parameters that define the services and the firm as well as those that describe the processes SMEs shall perform in order to conclude transactions, the information on contracts among firms, and the agreement characteristics. BML should be capable of describing information exchange between enterprises about volume, discount policies, rules and parameters by which potential suppliers are admitted, payment methods, contract duration, warranty, import/export rules and limitations. That is, BML has to describe on the one hand processes and key information that allow trading of products and services, and on the other hand all the information describing the company that offers such product or service, in order to enrich and empower the search for e-business partners. In order to support users in the complex search for the right e-business partners to be engaged in an e-business transaction, it could be useful to allow automatic processes of search based on software agents and intelligent systems; it means that BML framework should impose that all the business knowledge should be also represented in a machine readable format. So, if on one side BML has to allow business people to easily write and understand business concepts, it has to grant, at the same time, a rigorous mapping to formal logics to make business knowledge accessible to software, in order to support search processes. Another important issue concerns security and privacy of sensitive information each enterprise has to publish and to share in the description of their DBE services. Privacy of information should be addressed through a mechanism that could allow each enterprise to define policy and rules of data management for each service published in the e-business environment, in order to grant the right level of trusted interchange of information.

The second constraint is mainly related with the software production methodology and with the effort in developing mechanisms that allow the production of software starting from models. What is asked to BML framework is to contribute to the software development process, allowing to the business analyst to express in a formal and well defined way all the knowledge necessary to represent a client company; the business analyst using the BML framework will be able to represent every characteristic of the company from its organisational aspects to its motivation, its processes and its business model expressing them in a language that is the same that business people use. This set of knowledge should be clear and understandable from the software designer's perspective and will represent the repository of knowledge regarding the client company, in all the processes of design and implementation of the software solution. One of the perspectives software development is trying to address is the capability to develop software starting from description of the model (Model Driven Development). If BML wants to support a similar effort, and this would represent a very significant result for the SME software development community, it is necessary that BML is expressed through a machine readable language.

The third constraint is less related to the DBE general objectives but helps to understand how the project wants to reach such objectives and what strategy it has to pursue in order to grant a sustainable effort in time. It is clear that the project has to deal with open source communities and with standardisation bodies in order to be effective and to aggregate a critical mass of researchers, developers and users from open source communities, software developer communities or existing or emerging e-business communities. According to the DBE strategy the BML framework should adopt existing and emerging standards, and should be able to contribute to these standards evolving and adapting them to the DBE needs. It needs to analyze the existing and the emerging standards in order to understand if they

could be adopted or improved or simply if it should be granted compatibility, if such standards are largely diffused and used but do not match DBE needs.

The fourth constraint is related to the DBE adoption and diffusion strategy and requires that the BML framework is able to provide each DBE regional implementation with domain models and vocabularies. BML framework should allow, through a simple mechanism, to create new vocabularies and domain models tailored on community needs, every time a new territory, a region or a community decides to develop a new DBE implementation. Such a mechanism should be also able to grant coherence among vocabularies and domain models developed in different regions and related to different business domains, in order to enable the widest vision of a global ecosystem as a network of regional ecosystems. This issue is fundamental to grant that SMEs belonging to different regional implementation can interact and trade. Only through this functionality will it be possible to overcome the barriers of the local community and to transform the e-business adoption from a “saving money” advantage into the great opportunity to accede to the global market and to trade worldwide. To achieve this complex objective it has been decided to use a metamodel representation. The metamodel has to contain a wide set of high level business concepts that are the primitives of the BML framework [Con. 5]. Each vocabulary or model developed in a specific domain will be developed starting from this core of business knowledge, since the BML framework will contain mechanisms to create new vocabularies, rules and business models in specific business domains starting from the BML metamodel. The separation of model from vocabulary is a fundamental key since generic model could enable multiple standards of business descriptions and specific industry sector domain description. The meta model, in order to support this scenario, should be agile enough to enable a simple maintenance process; at the same time the meta model should be stable enough and contain enough abstract primitives in order to develop whatever kind of business vocabulary or model is required. In this way we don’t lock ourselves or our users into a single schema and, at the same time, will provide mechanisms for future standards to be linked in and application designers to map different standards as needed. To do that it is necessary to design a stable metamodel since changes to the domain model level should have much less impact on application software. It is necessary to grant a sharp division between the core DBE infrastructure delivery and any component provided by the DBE that would normally be provided by external partners.

## 5.1 The BML Description

Starting from the expressed objectives, we started to work on the BML information model and the BML architecture. The metamodeling choice, clearly expressed in the previous paragraph, is demonstrating particularly useful to decouple these two problems since the information model could change during the evolution of the project. This allows to evolve and maintain the metamodel content independently from the general architecture of BML. New information to be modelled could at most imply the adding of new packages in the metamodel or the update of the existing ones and do not influence in any way the general architecture. In order to define the BML Architecture we developed a benchmarking of the successful modelling strategies, analysing existing and emerging standards both in the e-business world and in data modelling.

In order to define the BML Information Model we spent work on a cross confrontation strategy based mainly on three different source of analysis. The first analysis allowed for the definition of specific information that is necessary to support specific processes and software components of the digital ecosystem environment. This analysis was based on

interaction and meeting with project partners, in order to deeply understand in what way BML has to support the DBE environment. The second analysis examined existing enterprise modelling frameworks with the aim of identifying and suggesting typologies of information that could be useful in order to extend the model and to allow new and richer descriptions of enterprise characteristics that seem necessary or useful.

The third decisive analysis will be the on-field analysis that will be responsible for validation of the developed BML and, at same time, for supporting the selection of new categories of information that SMEs consider as fundamental elements in the BML, in order to describe properly their services or their organisation's main characteristics.

It should be specified that, in order to make the general methodology effective and to ensure that each analysis can produce a relevant contribution to the Business Information Model, an incremental and iterative approach has been applied, which will re-examine each element of the model at every cycle with respect to the different analysis.

### 5.1.1 The BML Architecture

The BML architecture has been designed using a MOF based approach. As a consequence the M3 level contains the MOF itself which, as explained, defines the set of elements used to define metamodels, such as Class, Attribute, Association and so on. M3 is the higher meta-level since MOF, as explained, is self describing and so doesn't need any other meta-language to define its primitives. MOF has been developed and is maintained by OMG.

The M2 level contains all the metamodels that are necessary in order to create business domain vocabularies and models in the DBE project. More specifically this level contains the Business Meta Model which defines the abstract syntax of the BML and the SBVR (Semantics of Business Vocabularies and Rules: an ongoing specification of OMG for modelling the semantics of businesses and the applying business rules) and Meta Model which constitute the concrete syntax of the BML.

More specifically the Business metamodel, provides the entire package that constitutes the BML information model and contains all the primitives necessary to describe a generic business domain. All the primitives in the Business Meta Model are quite abstract and independent from specific business domain, in order to be instantiated in whatever business domain.

The information at M2 level is considered long lifecycle information and since it requires considerable competencies in both the business and modelling fields, it can be modified only by DBE experts. At a level below, in the M1 level, knowledge about specific domains should be represented in different Business Domain Models. These Models will represent a repository of knowledge for each specific business domain or industrial sector, allowing the definition of products, services, price policies or contract information but also providing a description of the enterprise characteristics such as its business strategy, its business model or its business process.

Each element, defined in a Business Domain Model, should be an instance of a general business concept defined at M2 level, as described in the previous paragraph, in order to maintain coherence among different domains. The Business Domain Model contains medium range changing information and it should be developed and maintained by domain experts, while end users couldn't access or modify it directly.

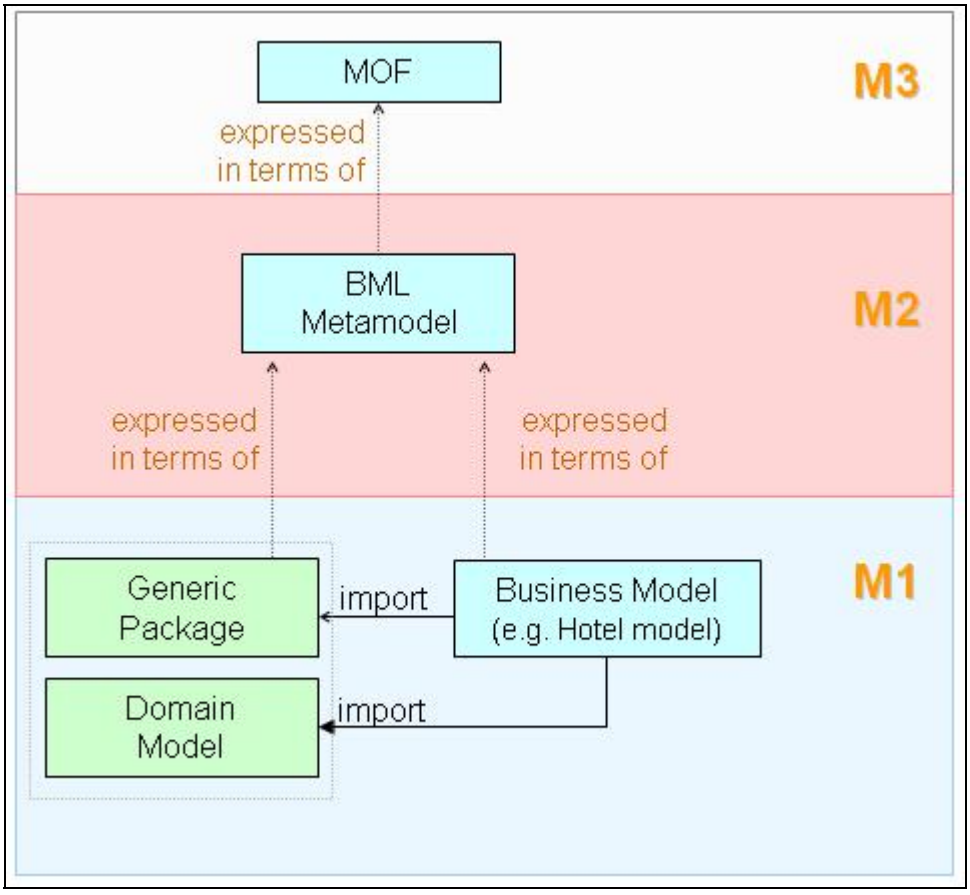


**Figure 17: BML Architecture**

### 5.1.2 The BML Metamodel

The BML Metamodel defines an abstract syntax for the BML language using the MOF v1.4 model. Its basis are the six interrogatives - What, How, Where, Who, When and Why - of the Zachman Framework for Enterprise Architecture [39]. For each interrogative of the Zachman Framework a corresponding package in the metamodel has been created. Moreover, an additional package containing the root meta-concepts inherited by the other packages has been created.

Currently, the metamodel is split up into seven packages:

- *Core*, containing the basic elements used to describe other packages meta-concepts characteristics;
- *BusinessOrganization*, containing meta-concepts to model each business entity aspect, including its relationships, resources, products and services;
- *BusinessProcess*, describing behavioural aspects of a business organisation, focusing on interaction with other organisations;
- *BusinessMotivation*, containing meta-concepts to describe the way in which an organisation makes choices and defines its actions;
- *BusinessLocation*, containing the metamodel to define spatial references in a business model;
- *BusinessEvent*, describing the occurrences able to influence business behaviours;
- *BusinessObject*, containing meta-concepts to define primitive and user-defined datatypes, typically used for declaring the types of the class attributes.

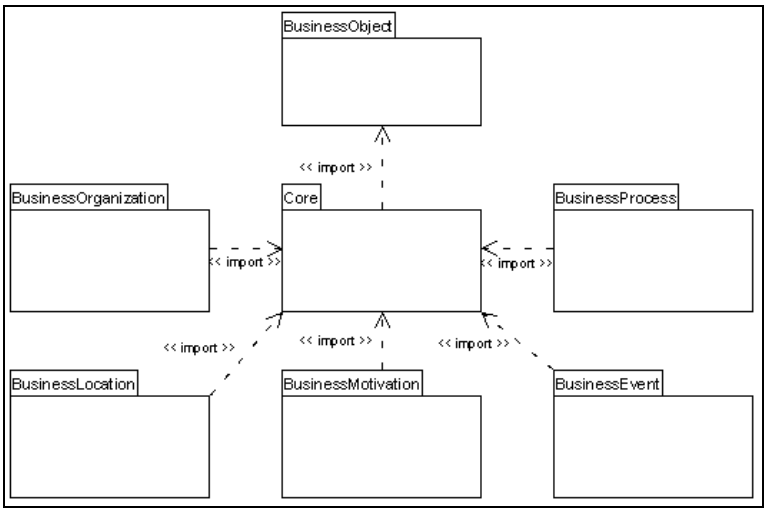


**Figure 18: BML Metamodel Packages**

Figure 18 shows the relationships between Core and other BML packages. The former imports meta-concepts defined in BusinessObject package. The other packages import the Core package, since they extend the Core classes in order to define their own meta-concepts.

The BML package structure, including packages that are relatively independent of one another, can provide the required range of expressivity and flexibility in order to reach BML purposes. In more detail, the Core package contains the basic classes extended by the other six packages. These elements are generic enough to be defined as a separate set within the BML metamodel. Many of the elements of this Core package are defined using the ebXML [44] core component library, in order to grant compatibility with this e-business standard and to avoid the re-inventing of terms and definition.

The BusinessOrganization package aims at describing the organisation as a whole, specifying the entities involved in the business, their resources and how they can interact. In particular, this package contains all the primitives for describing an organisation in terms of business entities or units useful for representing any complex organisational structure or form of collaboration between different organisations. Furthermore, the BusinessOrganization package contains meta-concepts for describing products and services an organisation offers to its customers or partners.

The BusinessProcess package is used to define the behavioural elements of an organization. This means that it contains all the meta-concepts needed to describe how the organization actually performs its business. In other words, the package specifies all information that needs to be captured during the analysis of the definition of a business process. One of the purposes of this package is the description of the dependencies existing between partner processes, modelling the business actions and objects that create, consume and exchange business information. The BusinessProcess package encompasses two main areas, closely related each other: an agreement area and a behavioural area. The former concerns the reciprocal engagements created between two different collaborative parties; the latter is related to the organisational working activities needed to perform a specific business activity. The agreement area, due to future research activities in the DBE project, could become a separate package, depending on the complexity and depth of the new exploring strategy. BML business processes are seen as a composition of tasks; if a task is a multi-partner activity, it represents a collaboration. In the same way, collaboration is composed of transactions, defined as documents exchange between different partners. This process

structure is compliant with the UMM Business Requirements View [40] definitions allowing a strong mapping to ebXML, which uses the UMM metamodel to define the behavioural aspects of a business process.

The BusinessMotivation package aims at describing the elements an organisation analyses and settles in order to make choices and define its action. This means that the package allows for defining and representing business plans in a structured way, highlighting what the organisation wants to achieve and what it needs in order to do it. Nevertheless, the real carrying out of this kind of plans is beyond this metamodel purpose; therefore, the package contains all and only the elements able to describe why an organization performs a given action. The BusinessMotivation package contains the metamodel elements for describing ends, means, assessments and influences used by an organization in order to justify and plan its behaviours. These generic terms summarise a great variety of concepts, such as vision, mission, strategy, goals and so on.

The BusinessEvent package contains the metamodel for describing the events that influence the business behaviours, including rules about their temporal ordering or partial ordering in the business activity cycle. Due to its definition, the package is closely related to the BusinessProcess package.

The BusinessLocation package contains the metamodel for describing geographic locations, business sites, geographic areas, volumes, and perimeters, political subdivisions and boundaries, and logical connections between them.

The BusinessObject package contains the business data type constructors. It allows the modeller to create the types needed in order to model a particular domain or business. The package also contains the business data types used at M2 level, created as instances of primitive and structured MOF data types. However, due to its inherent nature, the BML metamodel doesn't allow for the definition of the data types' operational features. These aspects are typical of programming language, so they are not relevant in a business-oriented context.

### 5.1.3 BML and SBVR

Rules play a very important role in defining business semantics: they can influence or guide behaviour and support policies, responding to environmental situations and events. This means that rules represent the primary means by which an organisation can direct its business, defining the operative way to reach its objectives and perform its actions. Consequently, a rule-based approach can be described as a way "for identifying and articulating the rules which define the structure and control the operation of an enterprise" [41]; it represents a new way to think about the enterprise and its rules, in order to enable a complete business representation made by and for business people. As stated above, OMG issued the BSBR RFP, in order to create a standard "to allow business people to define the policies and rules by which they run their business in their own language, in terms of the things they deal with in the business, and to capture those rules in a way that is clear, unambiguous and readily translatable into other representations" [38].

*Semantics of Business Vocabulary and Business Rules* (SBVR) [43] answers to the OMG BSBR RFP and, in particular, it aims at enabling:

- business vocabularies construction and rule formalisation, based on vocabularies, using a *natural language* (Structured English);
- making business rules accessible by software tools, including tools that support the business experts in creating, finding, validating, and managing business rules and tools

that support the information technology experts in converting business rules into implementation rules for automated systems.

Since SBVR is strongly focused on rules, its approach can also be defined as a fact-oriented approach. Indeed, foremost industry experts agree on saying that “rules build on facts”; therefore, SBVR uses facts in order to define vocabulary entries and rules. SBVR has been chosen as a linguistic metamodel for representing the BML abstract syntax, thus allowing to use the SBVR Structured English as notation for BML models in a natural language and rule-based paradigm. There are several reasons for such a choice. First of all, SBVR is created in order to reach full compliance with the MDA standard. It uses CIMs and PIMs and defines mapping rules between them; using a structured linguistic form, transforming SBVR models into a machine-readable form is a very simple process. Moreover, SBVR provides a set of capabilities useful in order to transform any SBVR vocabulary into a MOF/XMI implementation that supports repository services and document interchangeability between tools. Another important reason to choose SBVR as a linguistic metamodel for BML is represented by the strong focus toward business people and their language in a pure business perspective. SBVR is thought of as a framework aiming at allowing business people to represent their business knowledge, in terms of shared concepts and guidance. Furthermore, business vocabularies and related rules, both expressed in natural language, are conceptually very close to a CIM: they allow business environment representation, avoiding technical details, and are created by and for business people (i.e. domain practitioners).

Another important remark is represented by the compliance with MOF. SBVR specifications define a metamodel and allow to instantiate it, in order to create different vocabularies and to define the related business rules. It is also possible to complete these models with data suitable to describe a specific organisation. Conceptually, this corresponds to place an artefact in each level of the traditional four-level MOF architecture. Actually, creating SBVR models in natural language and placing them in such metadata architecture is a more complex process; however, it is possible to assert that SBVR is created to work with models, documents and linguistic structures at each MOF level. Moreover, as stated above, the SBVR approach provides the means (i.e. mapping rules) to translate natural language artefacts into MOF-compliant artefacts; this allows exploiting all the advantages related to MOF (repository facility, interchangeability, tools, etc.) that satisfy BML objectives related to its use within a distributed environment. From an enterprise perspective, choosing SBVR implies two main kinds of benefits. First of all, using it enables business people to directly choose what they want to represent and to create the suitable rules expressing this knowledge (obviously, it is essential to provide them with the right tools). However, even if model creation is performed by IT modellers, communications problems must be overcame, since modellers and business people are forced to use the same language, based on a shared and accepted vocabulary. In this way, it is possible to avoid misunderstanding and different interpretations of the same terms and concepts. The second benefit is related to requirements gathering for information systems aimed to support business activities. As formerly stated, the main DBE project recipients are SMEs and, in particular, those involved in software development. SBVR can offer support in this perspective, since one of its design objectives is to capture these requirements in the language of business people in a rigorous and unambiguous way, avoiding mismatching between what an enterprise wants its system to do and what it really can do. Moreover, due to its compliance with the MDA approach, SBVR can offer support in automating software production: starting from SBVR models, it should be possible to create diagrams, classes and code in an automated way. This implies that SBVR could be very powerful in order to support SMEs producing software to exploit reuse and automation.

Adopting SBVR as notation for BML implies creating a BML Vocabulary and defining the related rules, accordingly with the SBVR syntax. This translation can be realised starting from the converse process, clearly explained in SBVR specification. It defines the way to obtain a graphical (UML) representation of an SBVR-compliant vocabulary. Obviously, once the BML Vocabulary and Rules is created, it is possible to generate a MOF-compliant representation, applying the MOF/XMI mapping rules provided by SBVR. Note that a great difference exists between these two UML diagrams and they should not be confused. While the BML Metamodel can be called a “Business Object Model”, its MOF representation generated from the vocabulary can be defined as fact-oriented, since it reflects the SBVR fact-orientation.

From a more practical perspective, BML Vocabulary and Rules define standard terms, facts and rules specialising SBVR meta-concepts. In the same way, BML is used in order to create business models: a modeller specialises BML terms and facts in order to define concepts proper of a given business or domain. This means that passing from the UML/MOF to the SBVR structured language the way to create a model changes radically. In the former case, an artefact is generated passing through a MOF level (i.e. creating instances). Conversely, in the latter case, a specialisation mechanism is adopted. Only by adding data (populating the model) is it possible to speak about instances. Conceptually, this corresponds to collapsing different MOF levels into one, solving some important problems: business people speak about concepts using other concepts, regardless the number of levels introduced; moreover, they relate concepts among them regardless if they are in different levels. This means that, from a business perspective, forcing to use a four layered structure is very limiting.

# 6. Representing Semantic Service Models

This chapter describes the first version of the Semantic Service Language. This language is used to define service description models for capturing the semantics of SME business offerings (services). Instances of these models will be the Semantic Service Descriptions that are published in the DBE Knowledge Base (as part of the Service Manifests) in order to advertise the SME Services. Semantic Descriptions of Services are one of the main inputs to the Semantic Service Discovery Process carried out by the Recommender Service.

Semantic service discovery, as envisioned in DBE, requires the existence of semantic description of services and service providers, description of technical details of services, knowledge about the user needs and preferences, usage history data, and so on. All these requirements have been identified in the scope of DBE and corresponding tasks have been defined to address each one of them. The task "Service Ontologies" is intended to address the requirement for semantic description of DBE Services. To this end, this section presents a first version of the ***Semantic Service Language*** (hereafter SSL).

This language is an integral part of the DBE Knowledge Representation Framework and thus it will follow the adopted metamodelling approach. The abstract syntax (metamodel) of the language provides the required modelling elements (and their relationships, constraints, etc.) that the user (modeller) will have available in order to define custom models for semantically describing his services. These models can then be instantiated and populated with data that refer to specific services and be published into the DBE Knowledge Base to advertise these services in the ecosystem.

This language is part of the Business Modelling Language and its role is to semantically describe SME services from the outer-most perspective (i.e. what the candidate consumers will see).

The service specific conceptualisation includes both the business and technical description of a service. In particular, the service specific conceptualisation describes:

1. ***What a service is:*** This is the business level description of the service and provides semantic information about the special service properties like the business sector that the service belongs to, the geographic area that it applies to, charging schemes that can be used, etc. It may also provide descriptions of the business level interaction needed to consume a service. This part will be covered by the Semantic Service Language and the corresponding metamodel will be described in this section.

2. ***How it works****:* This kind of description refers to composite services. It will technically describe the internal composition and communication logic of composite services as well as the formal messages that are needed to be exchanged in a service execution (probably under conditions). This kind of description will be produced by the service composer and it will be defined by the Service Composition Metamodel [31] (based on the specification of BPEL [19]).

3. ***How is it used:*** This part of the service specific conceptualisation describes the technical details (i.e. available operations, message formats, exceptions, etc.) of a service from the outer-most perspective (i.e. how a service implemented by a software component and accessible through the network is consumed by other software components). It is actually the technical specification of the service interface, and it will be defined using the Service Description Language [30].

## 6.1 Approaches to Semantic Descriptions of Services

While promising to revolutionise eCommerce and enterprise-wide integration, current standard technologies for Web services (e.g. WSDL [13]) provide only syntactic-level descriptions of their functionalities, without any formal definition to what the syntactic definitions might mean. In many cases, Web services offer little more than a formally defined invocation interface, with some human oriented metadata that describes what the service does, and the organisation that developed it (e.g. through UDDI descriptions). Applications may invoke Web services using a common, extendable communication framework (e.g. SOAP [14]). However, the lack of machine readable semantics necessitates human intervention for semantically driven automated service discovery and composition within open systems, thus hampering their usage in complex business contexts [16].

Semantic Web Services (SWS) relax this restriction by augmenting Web services with rich formal descriptions of their capabilities, thus facilitating automated discovery, composition, dynamic binding, and invocation of services within an open environment. A prerequisite to this, however, is the emergence and evolution of the Semantic Service Description frameworks, which will provide the infrastructure for the semantic annotation and interoperability of Web Services. Such frameworks will augment services with rich formal descriptions of their capabilities, such that they can be utilized by applications or other services without human assistance. Thus, Semantic Web Services have the potential to change the way that services are consumed and provided on the Web.

Current efforts in developing Semantic Web Service infrastructures can be characterised along three orthogonal dimensions: *usage activities, architecture and service ontology.* Usage activities define the functional requirements, which a framework for Semantic Web Services ought to support. The architecture of SWS describes the components needed for accomplishing the activities defined for SWS, whereas the service ontology aggregates all concept models related to the description of a Semantic Web Service.

### 6.1.1 Web Services

A Web Service is a software program identified by an URI, which can be accessed via the internet through its exposed interface. The interface description declares the operations which can be performed by the service, the types of messages being exchanged during the interaction with the service, and the physical location of ports, where information should be exchanged. For example, a Web service for calculating the exchange rate between two money currencies can declare the operation getExchangeRate with two inputs of type string (for source and target currencies) and an output of type float (for the resulting rate). A binding then defines the machine and ports where messages should be sent. Although there can be many ways of implementing Web services, we basically assume that they are deployed in Web Servers such that they can be invoked by any software component independently of their implementations. In addition Web services can invoke other Web services.

Figure 19 illustrates a usage scenario of a Service Oriented Architecture defined by three phases; *Publish*, *Find*, and *Bind*/Interact; and three entities: the *service requestor*, which invokes services; the *service provider* which responds to requests; and the *service registries* where services can be published or advertised. A service provider publishes a description of

a service it provides to a service registry. This description (or advertisement) includes information about the provider of the service (e.g. company name and address); the service itself (e.g. name, category); and the URL of its service interface definition (e.g. WSDL description).

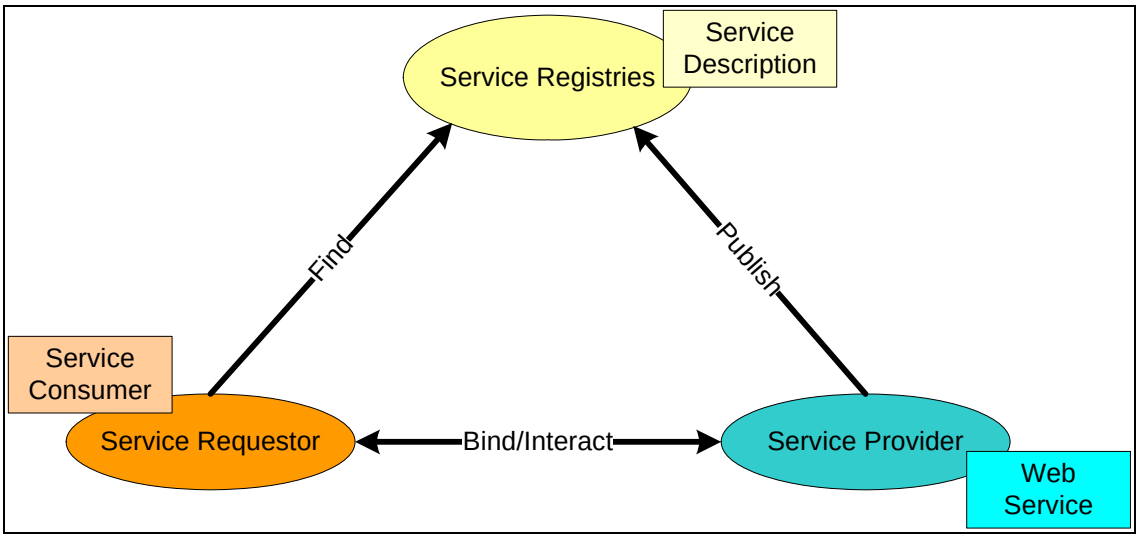


**Figure 19: Service Oriented Architecture – SOA**

When a developer realises a need for a new service, he finds the desired service either by constructing a query, or browsing the registry. The developer then interprets the meaning of the interface description (typically through the use of meaningful label or variable names, comments, or additional documentation) and binds to (i.e. includes a call to invoke) the discovered service within the application it is developing. This application is known as the service requestor. At this point, the service requestor can automatically invoke the discovered service (provided by the service provider) using appropriate service communication protocols (e.g. SOAP).

Key to the interoperability of Web services is an adoption of a set of enabling standard protocols. Figure 20 shows the most important XML-based standards that have been proposed to support service oriented architectures.

XML schema (XML-S) [17] provides the underlying framework for defining the Web Services Standards, and also the variables, messages, data types etc. that are exchanged between services. SOAP is W3C's recommended XML-data transport protocol, used for data exchange over web-based communications protocols (http).

Two levels of abstraction are used to describe Web services; the first defines atomic method calls, or operations, in terms of input and output messages (each of which contain one or more parameters defined in XML-S). Operations define the way in which messages are handled e.g. whether an operation is a one-way operation, request-response, solicit-response or notification. The second abstraction maps operations and associated messages to physical endpoints, in terms of ports and bindings. Ports declare the operations available with corresponding inputs and outputs. The bindings declare the transport mechanism (usually SOAP) being used by each operation. WSDL also specifies one or more network locations or endpoints at which the service can be invoked. As services become available, they may be registered with a UDDI registry [18] which can subsequently be browsed and queried by other users, services and applications. UDDI web service discovery is typically

human oriented, based upon yellow or white-page queries (i.e. metadata descriptions of service types, or information about the service providers). UDDI service registrations may also include references to WSDL descriptions, which may facilitate limited automation of discovery and invocation. However, as no explicit semantic information is normally defined, automated interpretation and understanding of the WSDL description is limited to cases where the provider and requester assume pre-agreed specification of protocols, and operation naming and meaning.

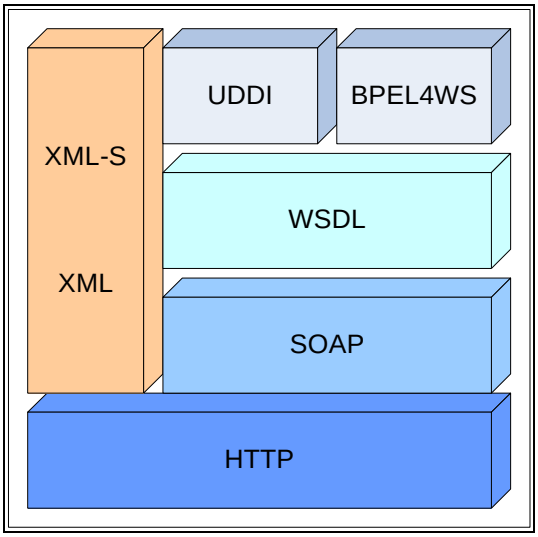


**Figure 20: Web Service Enabling Standards**

A service might be defined as a workflow describing the choreography of several operations. Such a workflow may determine: the order of operation execution; what operations may be executed concurrently; and alternative execution paths (if conditional operators are included in the workflow modelling language). Conversely, workflows are required to orchestrate the execution of several simple services that may be composed together for forming a more complex service. Various choreography and orchestration languages have been proposed such as BPEL4WS [19], and are currently in the finalisation and/or standardisation phase.

### 6.1.2 Semantic Web Services

Semantic descriptions of Web services are necessary in order to enable their automatic discovery, composition and execution across heterogeneous users and domains. As shown in the previous section, existing technologies for Web services provide only descriptions at the syntactic level, making it difficult for requesters and providers to interpret or represent non-trivial statements such as the meaning of inputs and outputs or applicable constraints. This limitation may be relaxed by providing a rich set of semantic annotations that augment the service description. A Semantic Web Service is defined as the semantically described service through sophisticated description models that can be enhanced with ontologies enabling machine interpretability of its capabilities as well as integration with domain knowledge. Many people believe that the deployment of Semantic Web Services will rely on the further development and combination of Web Services and Semantic Web enabling technologies (i.e. RDF, RDFS, and OWL). To this end, several initiatives (e.g. http://dip.semanticweb.org or http://www.swsi.org) have started in both industry and academia, which are investigating solutions for the main issues regarding the infrastructure for SWS.

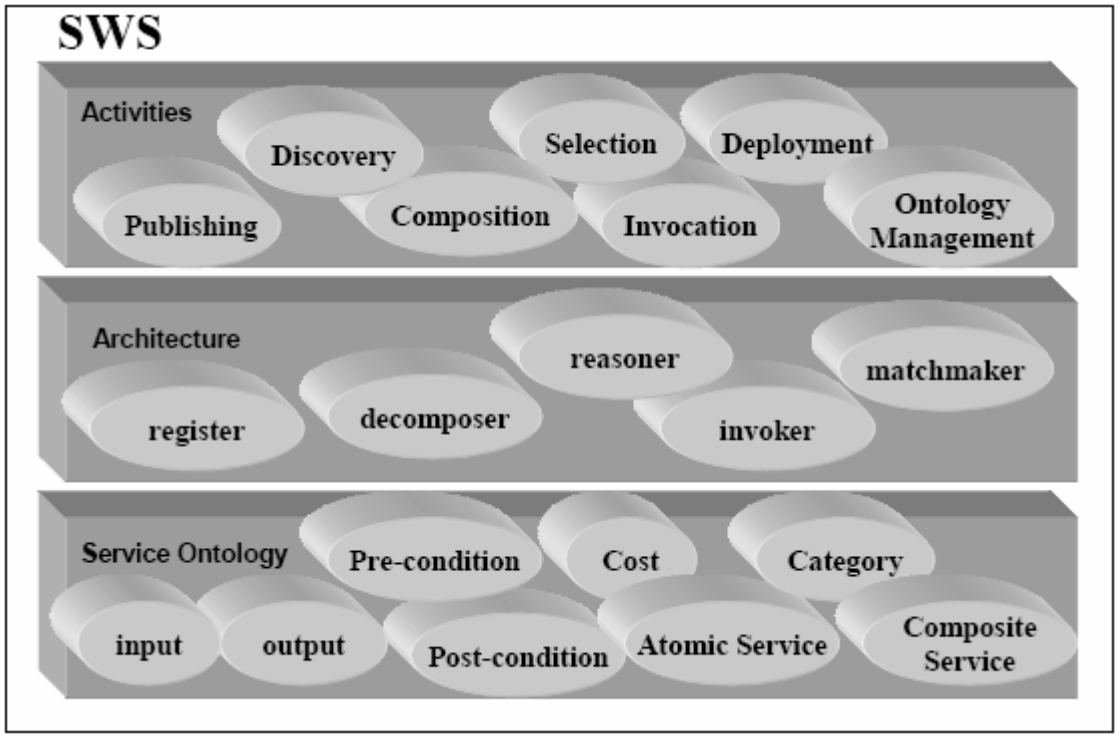


**Figure 21: Dimensions of Semantic Web Service Infrastructures**

The authors of [16] believe that Semantic Web Service infrastructures can be characterised by three orthogonal dimensions: usage activities, architecture and service ontology as illustrated in Figure 21. These dimensions relate to the requirements for SWS at business, physical and conceptual levels. Usage activities define the functional requirements, which a framework for Semantic Web Services ought to support. The architecture of SWS defines the components needed for accomplishing these activities.

The service ontology dimension aggregates all concept models related to the description of a Semantic Web Service, and constitutes the knowledge-level model of the information describing and supporting the usage of the service.

This dimension is fundamental in defining SWS as it represents the capabilities of a service itself and the restrictions applied to its use. The service ontology essentially integrates at the knowledge-level the information which has been defined by Web services standards, such as UDDI and WSDL with related domain knowledge. This would include: functional capabilities such as inputs, output, pre-conditions and post-conditions; non-functional capabilities such as category, cost and quality of service; provider related information, such as company name and address; task or goal-related information; and domain knowledge definitions, for instance, the type of the inputs of the service. This information can be divided into several ontologies. However, the service ontology used for describing SWS will rely on the expressivity and inference power of the underlying ontology language supported by the Semantic Web. Three main approaches have been driving the development of Semantic Web Service frameworks: IRS-II [20], OWL-S [21], and the WSMF [22].

### The IRS-II Approach

The Internet Reasoning Service - IRS-II [20] is a Semantic Web Services framework, which allows applications to semantically describe and execute Web services. IRS-II is based on the UPML (Unified Problem Solving Method Development Language) framework [26], which distinguishes between the following categories of components specified by means of an appropriate ontology:

A) **Domain models.** These describe the domain of an application (e.g. vehicles, a medical disease).

B) **Task models.** These provide a generic description of the task to be solved, specifying the input and output types, the goal to be achieved and applicable preconditions.

C) **Problem Solving Methods (PSMs).** These provide abstract, implementation independent descriptions of reasoning processes which can be applied to solve tasks in a specific domain.

D) **Bridges.** These specify mappings between the different model components within an application.

A key feature of IRS-II is that Web service invocation is capability driven. The IRS-II supports this by providing a task centric invocation mechanism. An IRS-II user simply asks for a task to be achieved and the IRS-II broker locates an appropriate PSM and then invokes the corresponding Web service. IRS-II was designed for ease of use. Developers can interact with IRS-II through the IRS-II browser, which facilitates navigation of knowledge models registered in IRS-II as well as the editing of service descriptions, the publishing and the invocation of individual services. Application programs can be integrated with IRS-II by using the Java API. These programs can then combine tasks that can be achieved within an application scenario.

### The OWL-S Approach

OWL-S (previously DAML-S [21]) consists of a set of ontologies designed for describing and reasoning over service descriptions. The OWL-S approach originated within communities

with an AI background and has previously been used to describe agent functionality within several Multi-Agent Systems as well as with a variety of planners to solve higher-level goals.

OWL-S combines the expressivity of description logics (in this case OWL) and the practical feasibility found in the emerging Web Services Standards, to describe services that can be expressed semantically, and yet grounded within a well-defined data typing formalism. It consists of three main ***upper ontologies***:

A) **The Profile.** The Profile is used to describe services for the purposes of discovery; service descriptions (and queries) are constructed from a description of functional properties (i.e. inputs, outputs, preconditions, and effects - IOPEs), and non-functional properties (human oriented properties such as service name, etc, and parameters for defining additional meta data about the service itself, such as concept type or quality of service). In addition, the profile class can be sub-classed and specialised, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.

B) **Process Model.** The process model describes the composition or orchestration of one or more services flow of control and execution sequence. This is used both for reasoning about possible compositions (such as validating a possible composition, determining if a model is executable given a specific context, etc) and controlling the enactment/invocation of a service. Three process classes have been defined: the composite, simple and atomic process. The atomic process is a single, black-box process description with exposed IOPEs. Inputs and outputs relate to data channels, where data flows between processes. Preconditions specify facts of the world that must be asserted in order for an agent to execute a service. Effects characterise facts that become asserted given a successful execution of the service, such as the physical side-effects that the execution the service has on the physical world. Simple processes provide a means of describing service or process abstractions – such elements have no specific binding to a physical service, and thus have to be realised by an atomic process (e.g. through service discovery and dynamic binding at run-time), or expanded into a composite process. Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. These process workflows are constructed using a number of different composition constructs, including: Sequence, Unordered, Choice, If-then-else, Iterate, Repeat-until, Repeat-while, Split, and Split+join. The profile and process models provide semantic frameworks whereby services can be discovered and invoked, based upon conceptual descriptions defined within Semantic Web (i.e. OWL) ontologies

C) **Grounding.** The grounding provides a pragmatic binding between this concept space and the physical data/machine/port space, thus facilitating service execution. The process model is mapped to a WSDL description of the service, through a thin grounding. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types. Additional properties pertaining to the binding of the service are also provided (i.e. the IP address of the machine hosting the service, and the ports used to expose the service).

### The WSMF Approach

The Web Service Modeling Framework (WSMF) [22] provides a model for describing the various aspects related to Web services. Its main goal is to fully enable e-commerce by applying Semantic Web technology to Web services. WSMF is centred on two complementary principles: a strong de-coupling of the various components that realise an e-commerce application; and a strong mediation service enabling Web services to communicate in a scalable manner. Mediation is applied at several levels: mediation of data structures; mediation of business logics; mediation of message exchange protocols; and mediation (e.g. concept and data mapping, etc.) of dynamic service invocation.

WSMF consists of four main elements: *ontologies* that provide the terminology used by other elements; *goal repositories* that define the problems that should be solved by Web services; *Web services descriptions* that define various aspects of a Web service; and *mediators* which bypass interoperability problems.

WSMF implementation has been assigned to two main projects: Semantic Web enabled Web Services (SWWS) [24], and WSMO (Web Service Modeling Ontology) [23]. SWWS will provide a description framework, a discovery framework and a mediation platform for Web Services, according to a conceptual architecture. WSMO will refine WSMF and develop a formal service ontology and language for SWS. WSMO service ontology includes definitions for goals, mediators and web services. A web service consists of a capability and an interface. The underlying representation language for WSMO is *F-logic*. The rationale for the choice of F-logic is that it is a full first order logic language that provides second order syntax while staying in the first order logic semantics, and has minimal model semantics. The main characterising feature of the WSMO architecture is that the goal, web service and ontology components are linked by four types of mediators as follows:

A) **OO mediators** link ontologies to ontologies,

B) **WW mediators** link web services to web services,

C) **WG mediators** link web services to goals, and finally, and

D) **GG mediators** link goals to goals.

Since within WSMO all interoperability aspects are concentrated in mediators the provision of different classes of mediators based on the types of components connected facilitates a clean separation of the different mediation functionalities required when creating WSMO based applications.

## 6.2 The Semantic Service Language (SSL)

In this section we present the Semantic Service Metamodel that will be used in the DBE environment. Throughout this section examples will be used to illustrate how the various concepts of the metamodel are used in practice. Our design goal is the use of MOF for the definition of a metamodel that will be used in connection with other models of the DBE Knowledge Representation Framework (i.e. Metamodels of BML, SDL, ODM, etc. etc.). Since MOF is used for the definition of the metamodel, terminology of this technology will be extensively used and a familiarity of the reader with that technology (as well as with the MOF metadata architecture) is assumed.

The Semantic Service Language (SSL) is used to define semantic service description models for capturing the semantics of specific types (e.g. hotel booking, flight reservation, etc.) of business offerings (services). These semantics (i.e. instances of SSL models) will be used to advertise a specific service of a particular type to the outside world (the candidate service consumers).

SSL is used to semantically model classes of services from the external business point of view. That is, to describe what the candidate consumers of the services will see. Typically, the semantics captured with SSL will be the main input in the semantic service discovery process. These semantics may or may not have been defined in the internal business model of the service provider(s). In any case, in the SSL description of a service, the service provider will include the semantics that are supposed to advertise a service in the best way.

SSL provides a mechanism to specify different "profiles" of a service. A profile of a service defines how a service should be described for a particular purpose (e.g. for a particular target group). It is possible that a real world service is described by more than one service profile.

In short, SSL describes three kinds of semantics:

a) **The special features of a service** that it is believed will be valuable to the candidate service consumer. Such features may refer to special business characteristics ranging from supported payment methods and applied charging schemes, to product or resource features, as well as to other related information that may be influencing to the decision of the candidate service consumer (e.g. description of the weather information in a resort booking service).

   It is possible that these special features of a service may have been modelled in the internal business model of the service provider. SSL will leverage the existence of such information in the business model of the service provider by referring to this information. Moreover, it can be used as a mechanism that integrates into a single advertisement various semantic characteristics related to a service and are spread in various parts of the business model description like business organization, business processes, business rules, etc.

   This operation of SSL will be particularly important in case of a composite service. In such a case, the semantic description of a composite service (that will be developed by the (automatic or manual) composer during the service composition process) can select and integrate the desirable features of the constituent services into a coherent description that has a specific meaning for the composite service.

   SSL is integrated with the Ontology Definition Metamodel (ODM) in order to allow SSL models to make use of concepts defined in domain specific ontologies increasing the semantic interoperability among SMEs.

b) **Business Level Interaction needed to Consume a Service**. In order for a service to be consumed, two business parties must have an interaction during which they will exchange resources and/or information. SSL allows the specification of this interaction in a high level of abstraction. That is, it does not specify what exactly the messages and the message formats will be; rather it describes what (information and resources) the consumer has to give to the provider, and what (information and resources) the service provider will give back to the consumer and under which conditions. To do so, SSL defines the notion of service functionality in order to describe logical units of business level interaction between provider and consumer. Finally, it can specify special conditions that must hold in order for such an interaction to be started. This information will be taken into account by the SDL compiler for the automatic creation of the service's technical interfaces.

c) **Contacts Responsible for the Provision of a Service**. An SSL description of a service may describe one or more information structures for referring to agents or individuals responsible for the service (or some aspect of the service). Contact information is a specific type of semantic information that most of the approaches to semantic service description (OWL-S, WSMF, etc.) have included.

The following sections describe the SSL metamodel in much more detail by presenting all the primitives (and their semantics) provided by the language.

### 6.2.1 Model Organization of the SSL Metamodel

The SSL metamodel has been organized into 4 different packages as Figure 22 depicts.

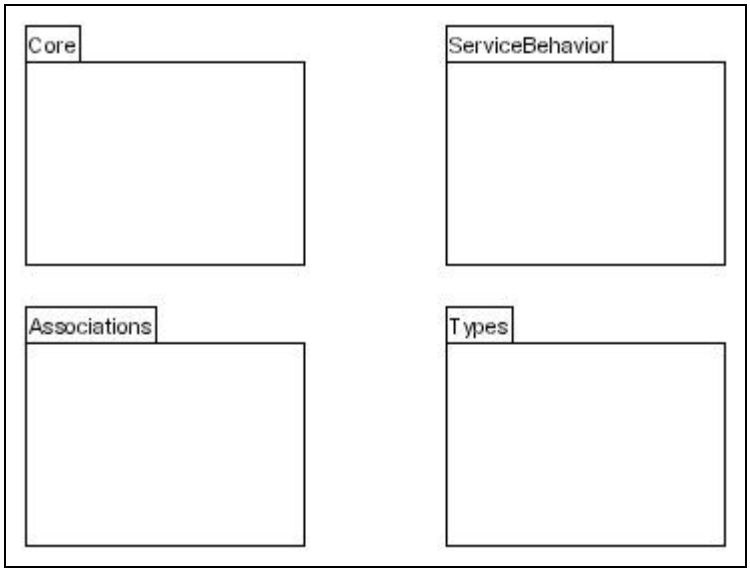


**Figure 22: Package View of the SSL Metamodel**

The ***core*** package defines the basic concepts used in the SSL metamodel and defines the structure of a semantic service description by introducing the concept of service profile (i.e. the profile of a service) associated with many other concepts. The package ***Types*** contains the appropriate concepts that allow the modeller to describe the types of the model constructs that he defines as well as to integrate M1 models with Ontologies. That is, to re-use ontology concepts in the semantic description of a service. The business level interaction required to consume a service is described using the constructs defined in the ***ServiceBehavior*** package of the SSL metamodel, while the ***Associations*** package provides the mechanism for relating the various constructs of the user models with each other. In what follows, we will gradually describe the SSL metamodel starting from the most self-contained concepts and going to more advanced ones.

### 6.2.2 Semantic Service Package and Profile

The root concept in the SSL metamodel is the abstract class ***ServiceConcept***. Most of the elements used in this metamodel are sub-classes of this class. Each service concept has a *name,* an *id* and a *Documentation* attribute. That is, at the M1 level all their instances must be given a name, an id and optionally a documentation text. The name and the documentation are given by the modeller when instantiating these concepts (creating models for semantic service descriptions), while the id is used for model management purposes (i.e. it is not semantic information) and will be provided by the BML Editor tool.

Each DBE Service will be described by two or three points of view. The first perspective describes a service from a semantic point of view, the second from an external technical point of view, while a third perspective may be desired to describe the internal logic of composite services. These descriptions along with the BML description of the service provider will constitute the Service Manifest. The SSL will be used to describe a service from the semantic point of view (perspective). This perspective, in the SSL metamodel is modelled as a ***SemanticPackage***.

The abstract class ***NameSpace*** is used for model management reasons and in particular for scoping modelling elements of the kind *ServiceConcept*. A namespace is a special element (a container) that contains *service concepts*. As defined in the SSL metamodel, each element, created by the modeller at the M1 layer, belongs to one and only one namespace. In the current version of the metamodel only a Semantic Package and a Service Profile can be used as a namespace for other elements. The idea is that the modeller defines a Semantic Package (which is also a namespace) and inside this package creates one or more models (named service profiles) that each one of them includes other concepts that are used to semantically describe a real world service. This real world service is not a specific one provided by a specific agent. Rather it is a class of services in the real world. For example, a Hotel Reservation Service is not a specific service in the real world (e.g. the hotel reservation offered by the Ibis hotel in London). Rather it is a model for describing with a specific way (i.e. advertising specific kinds of features, and characteristics) a type of services (i.e. hotel reservation services).

A (valid) ServicePackage must contain at least one ***ServiceProfile***. A service profile is a model for describing semantically a class of services. A semantic package may contain more than one such profile, since it is possible that a service provider may want to describe its service with many different ways according to the customer group that it targets to. Consider for example, that a hotel owner wants to describe its Hotel Reservation service with two different ways: One way for Tourist Agents (for B2B collaborations), and one for end customers (for B2C purposes). These two semantic descriptions of the real world hotel reservation service need to be based on different description models, since different features of the service should be advertised in each case. For example, the cancellation policy that is applied to end customers is of different type than those applied to Tourism Agents or the Semantic Description for the Tourism Agents may need to mention what the commission is for the Agent, or what are the policies governing allotment issues, etc.

Whenever the *Service Profile* class is instantiated, a name must be given by the modeller. The name is typically describing the real world service that this profile is going to model (e.g. “HotelReservationService”, “CarRental”, “CourierService”, “Shipping”, etc.

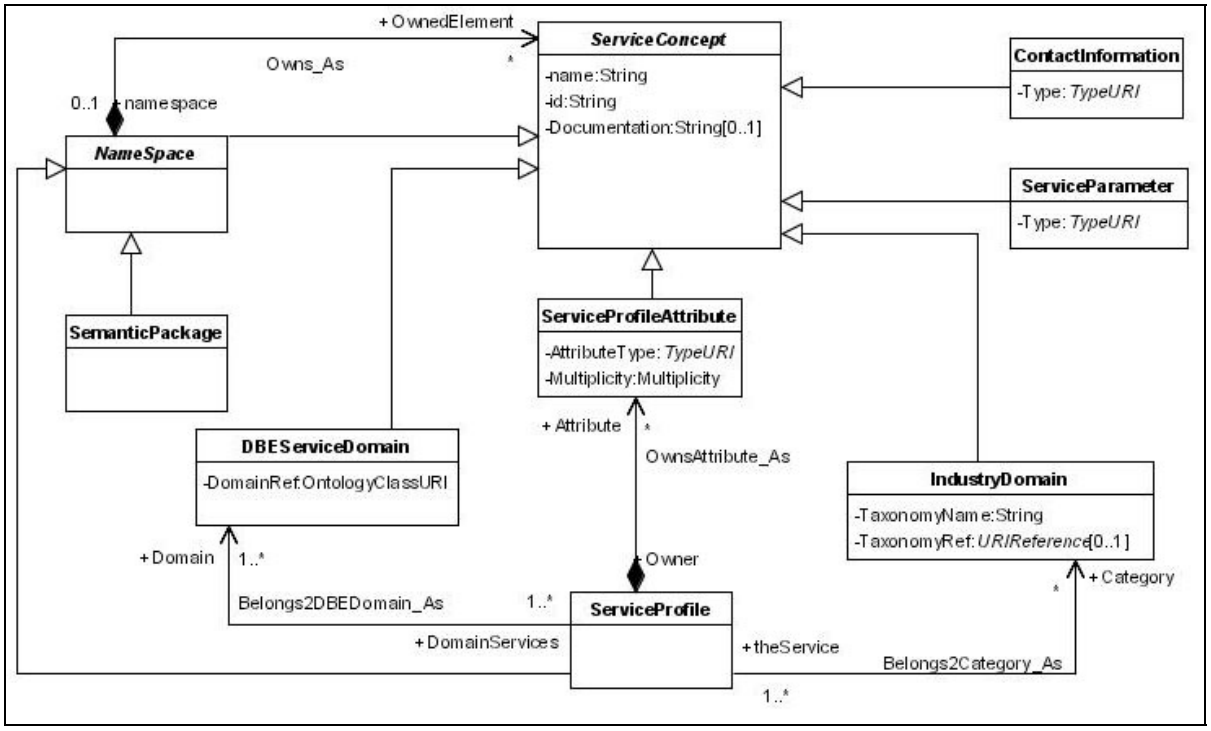


**Figure 23: SSL Main Concepts**

### 6.2.3 Classification of Service Profiles

Each service described in the DBE should belong to at least one *DBE Service Domain*. DBE has its own business classification scheme in order to consistently classify businesses and services. This scheme is provided in the form of a taxonomy that contains a hierarchically organised set of business domains (business sectors) like *"Tourism"*, *"Manufacturing"*, *"Transportation"*, *"Finance"*, *"Insurance"*, etc, and it will be maintained and extended by DBE Regional Catalysts or some special authority assigned with this task.

Although one could think of several reasons for classifying services this way (e.g. as a means to scope user queries), the most important reason for this classification is the definition of a service in connection to its specific domain conceptualisation. This service classification acts in helping or even guiding the user (the service provider) using domain specific ontologies when defining models for describing his services semantically. A possible scenario could be that when the modeller specifies the domain of the service profile that is creating for a service that he offers, then the BML editor will work in connections with the ontologies that have been defined for this domain to help the user to use appropriate concepts (but also allowing the user to use its own-defined concepts and not enforcing him to only use existing ones). As shown in Figure 23, a *Service Profile* may belong to more than one *DBE Service Domains*.

Another way to classify services in SSL is through the use of the ***IndustryDomain*** Concept. As in many cases in the real world, several classification schemes (taxonomies) for industrial companies and/or activities may exist. Such taxonomies exist outside the DBE environment and it may be desirable by service providers to classify their services according to those taxonomies. Thus, an instance of the class *Service Profile* may be associated with one or more *Industry Domains.* An *Industry Domain* instance has a name which is the name of the domain that the service belongs to (e.g. "Pet supply stores"), a *TaxonomyName* which is the name of the used taxonomy (e.g. "NAICS" for the North American Industry

Classification System), and a *TaxonomyReference* pointing to a particular entry of that taxonomy (e.g. “http://www.census.gov/epcd/naics02/#56172”).

Figure 24[3] illustrates an SSL model for a Hotel Reservation Service and is used in presenting the primitives of the SSL metamodel.

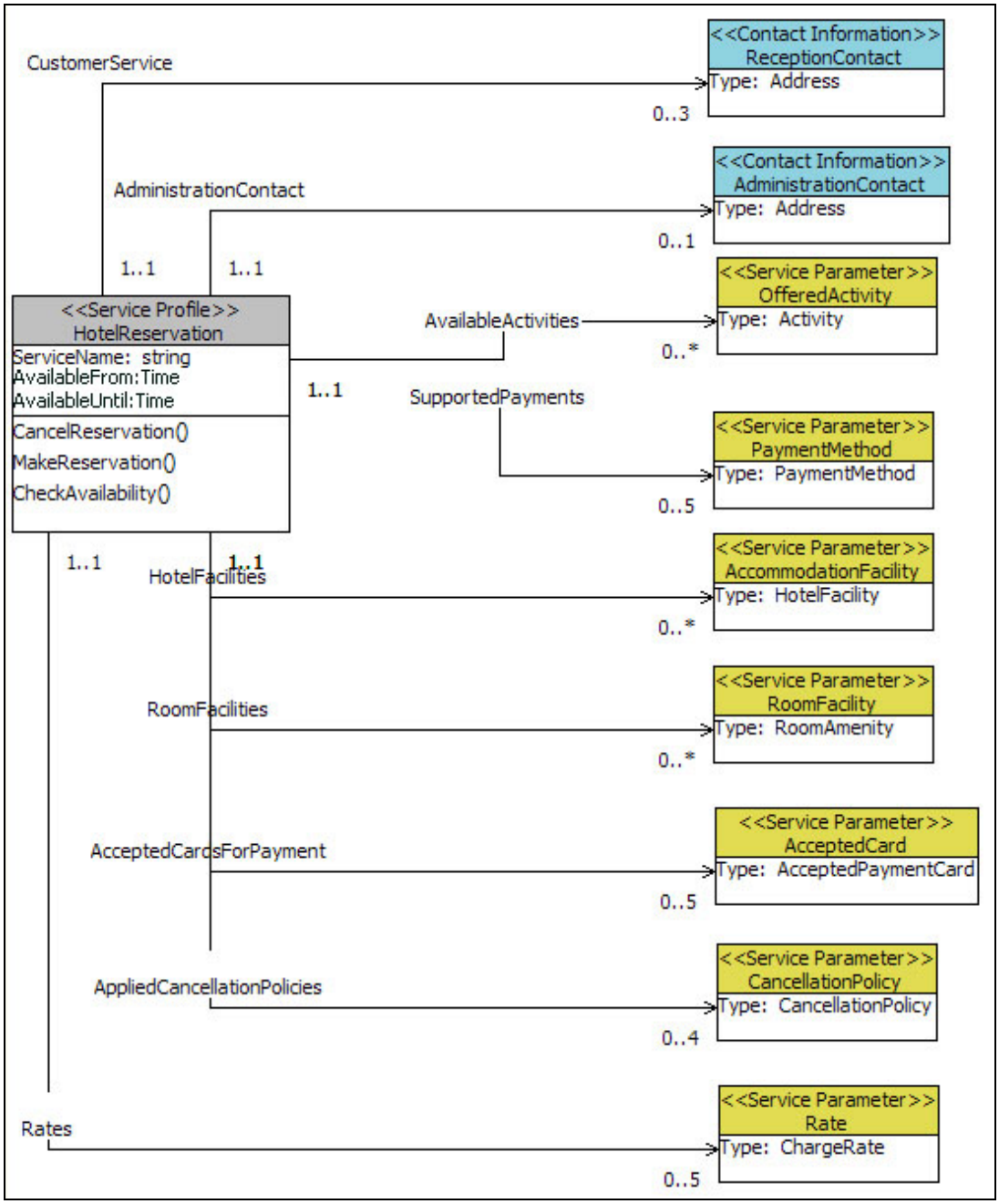


**Figure 24: An SSL Model for a Hotel Reservation Service**

[3] This figure is also included in Appendix B. It has been reused here in order to facilitate in the understanding of the SSL primitives presented in this chapter.

### 6.2.4 Service Profile Attributes

A Service Profile instance may have a number of ServiceProfileAttributes. This primitive is used to model particular characteristics, not only of a service, but more specifically, of a particular profile of that service. By using this primitive one can model for example the hours that a service is offered (as in the Hotel Reservation example in Figure 24), or a special discount offered to a certain target group (i.e. to those that the particular description -service profile- is targeting, etc. For this reason, a service attribute cannot be associated with more than one service profile. From a model management perspective, the lifecycle of a service profile attribute is closely related to its owner profile.

Each Service Profile Attribute should be given a name, a type and a multiplicity range. The name of an attribute is a string value (e.g. “Service_Name” in Figure 24). The type of an attribute is a reference to a data type (e.g. String, Integer, etc.), or to an enumeration of literals (e.g. a set of data values), or to a concept defined in some ontology (e.g. a service attribute named “Payment” could be of type “Discount” where the semantics of this concept may have been defined in domain specific ontology). The multiplicity of an attribute defines the minimum and maximum number of slots (i.e. instances of this attribute) that can be owned by a service profile instance.

### 6.2.5 Service Profile Associated Concepts

Service attributes are attached to service profiles in order to represent special characteristics of services described only in this particular profile (i.e. this particular view of the service). The association semantics between a service profile and its service profile attributes are predefined in the SSL metamodel. That is, the modeller cannot define his/her own (user-defined) semantics on what the information that a service profile attribute captures means for a particular service profile. Such an option is many times desirable in a modelling process since it allows decoupling and re-use of the modelled concepts.

Consider for example that a modeller wishes to define two service profiles to describe the same real world service (the Hotel Reservation Service in Figure 24) for two different target groups: travel agents and individual tourists. In such a case it is expected that the two service profiles will not be disjoint, but they will make use of the same common semantics. For example both user groups should know what the available hotel facilities are. From a modelling point of view, the model element “AccommodationFacility” is able to describe a facility provided by a hotel. Assume that the hotel owner wants to make a special offer only for individual tourists and give for free all the offered hotel facilities (e.g. pool, sauna, and parking). In order to re-use the model element “AccommodationFacility” for this second profile the modeller should have the ability to connect this model element to the second profile and to define that the connection semantics are “FreeFacilities”. SSL allows the modelling of semantic service characteristics that can be re-used by different service profiles with user-defined semantics.

SSL defines the concept **ServiceParameter** to model semantic characteristics of services that can be presented in many different service profiles. A service parameter is defined by the modeller by giving it a name and a type. The name of the parameter is a string value like the “AccommodationFacility”, or the “RoomFacility” in the hotel reservation example presented in Figure 24. The type of a service parameter can be either a data type (like String, Integer, etc.) or an enumeration of literals (e.g. {“ColourTV”, “A/C”, “MiniBar”}, or a concept defined in some ontology (e.g. “HotelFacility” from a tourism ontology). In the Hotel Reservation example presented in Figure 24 (as well as in Appendix B) we have

defined several service parameters whose type has been defined in the Tourism Ontology Example presented in Appendix A. The SSL metamodel requires that the type of a service parameter is a class definition in some ontology and not some individual (instance). In such a case (i.e. when an ontology concept is used) the semantics and the structure of a service parameter has been defined in some domain (or user ontology) and it is re-used in the SSL Model. Thus, domain knowledge can be integrated with user models for semantic service descriptions.

Another modelling primitive that describes a real world service characteristic which can be presented in more than one service profiles is the **ContactInformation**. A Service Profile may be associated with one or more concepts that model contact information structures referring to organizational units or individuals responsible for the service or some aspect of the service. As with the service parameter, the ContactInformation class does not specify any obligatory structure for representing contacts. Instead, the user can define its own structure for representing contact information at M1 level (when defining a specific Model for describing a service). The type of a ContactInformation can be either a primitive data type, or an enumeration, or a complex structure defined as concept in some ontology. A Service profile may be associated with more than one contact information instances. For example, the Hotel Reservation Service in Figure 24, defines two types of contact information: The first is for contacting the Hotel's reception, while the other for contacting the Hotel's administrator (e.g. owner). Contact information is a specific type of semantic information that most of the approaches to semantic service description (OWL-S, WSMF, etc.) have included. That is the reason for defining a special class for this concept, rather than assuming that it would be an instance of a service parameter or attribute.

Re-use of modelling concepts in different profiles is certainly a valuable ability that needs to be provided by SSL as a modelling language. It should be stressed however that re-use of the same modelling concept is not only useful in different profiles, but also in the same profile. As in many modelling mechanisms (e.g. in UML), two modelling concepts may be associated in more than one ways. In SSL one could associate a service profile with a service parameter or contact information with more than one ways. For example, in the Hotel Reservation Profile an alternative could be to define one single ContactInformation named "HotelContact" and to associate this concept with the HotelReservation service profile with two associations: one named "AdministrativeContact", and one named "ReceptionContact". This way, the same modelling concept is re-used in the same service profile with different user-defined association semantics.

### 6.2.6 Associating Service Concepts

The classes ServiceParameter and ContactInformation in the SSL metamodel (Figure 23) are not explicitly associated with the class ServiceProfile. The reason for this, as already stated, is that SSL allows user-defined relationships to be established between service profiles and these concepts. To allow the modellers to define their own relationships between these concepts, SSL defines the concept **Association** that is used to associate service concepts.

An *association* represents a binary user-defined relationship between two service concepts. Each association is a group (composition) of two endpoints named AssociationEnds. Each association end has a type that declares the service concept that it is connected to it. In every instance of the class association there are two instances of the class association end named source and target. The source association end represents the first service concept in a binary relationship, while the target association end represents the second part of the relationship.

As shown in Figure 25, each instance of the class association should be given a name and an id. The name of the association describes the meaning (semantics) of this relationship between the source and target concepts. For example, the association "SupportedPaymentMethods" in the Hotel Reservation Example (Figure 24) associates the Service Parameter "PaymentMethod" and the service profile "HotelReservation" with a particular user-defined semantic relationship: SupportedPaymentMethods (i.e. the payment methods that are supported for the customers that this profile is targeting to).

When defining an association between two service concepts, the modeller can give also a multiplicity range for each associated concept. The multiplicity range defines what is the minimum and optionally the maximum number of instances of the associated service concept that can be associated with one instance of the service concept connected to the opposite association end. For example, the multiplicity of the association "SupportedPaymentMethods" at the association end of type PaymentMethod in the Hotel Reservation Example, is defined as 0..5. That is, a semantic service description (following this service profile) MAY include at most 5 supported payment methods (instances of the concept "PaymentMethod"). Figure 25 shows the aforementioned primitives of the SSL metamodel.

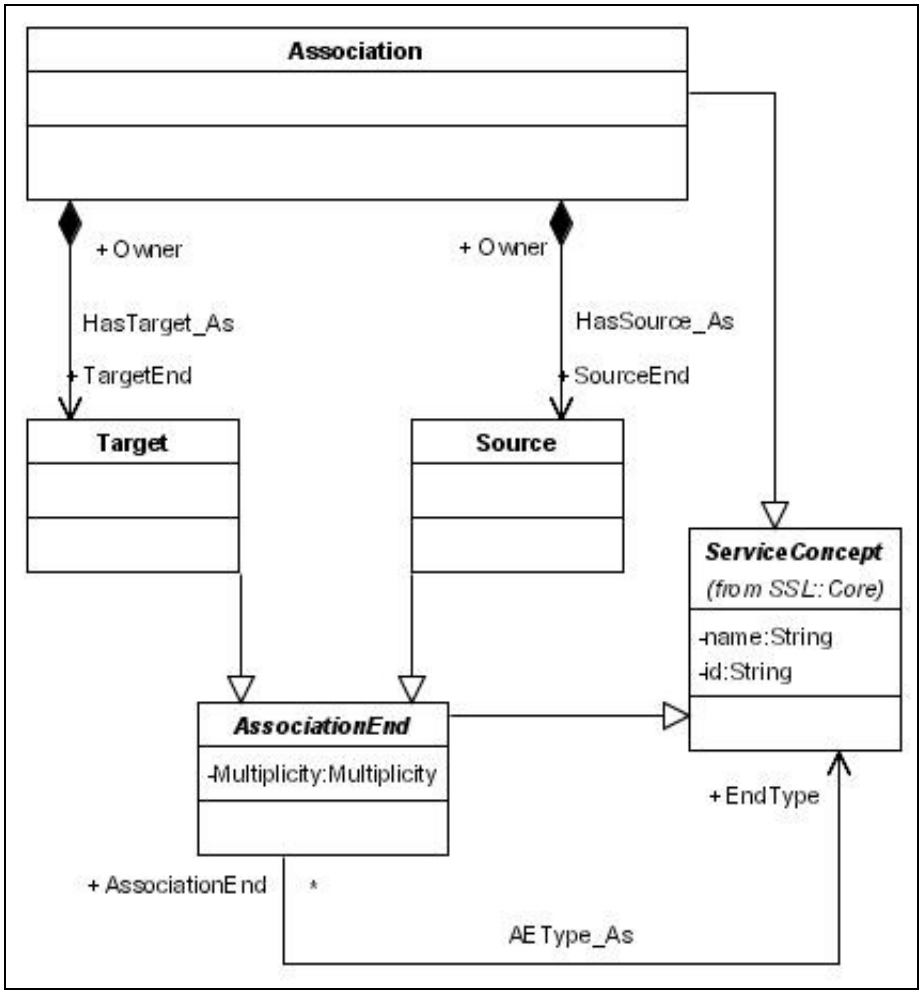


**Figure 25: Associations in SSL**

In the current version of the metamodel an instance of the class association may be used to connect:

a. Two instances of the class *Service Profile*. For example, one could connect two service profiles with an association named “sameAs”, or “DerivedFrom”, etc.

b. An instance of the class *Service Profile* and an instance of the class *Service Parameter*.

c. An instance of the class *Service Profile* and an instance of the class *Contact Information*.

### 6.2.7 Describing the Service Behaviour

An essential component of the profile is the specification of the business level interaction needed to consume a service. This is done by modelling the functionalities that a service provides to the customers and the specification of the conditions that must be satisfied for a successful execution (consumption) of these functionalities. SSL defines the concept ***ServiceFunctionality*** to describe logical functions that are performed by a service described by a *service profile*.

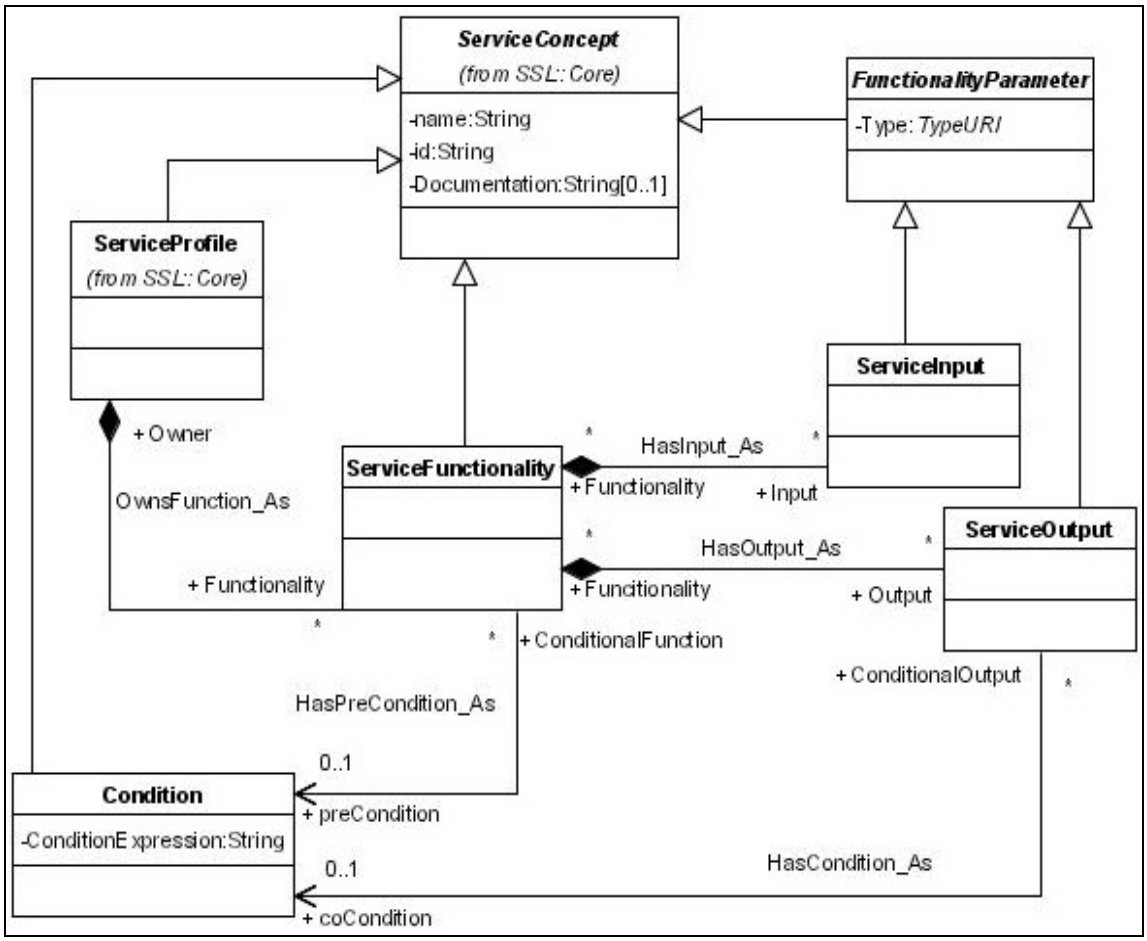


**Figure 26: Service Behaviour Modelling**

For example, the Service Profile defined for the Hotel Reservation Service in Figure 24 defines three functionalities offered by this service: a) *CheckAvailability* to allow the service consumer to check whether there are available rooms for reservations for a specific time period, b) *MakeReservation* to allow the service consumer to book rooms for a specific period of time, and c) *CancelReservation* to allow the user to cancel a booking.

It is important to mention that the primitive ServiceFunctionality is used to model the behaviour of a service from the external point of view. That is, to describe what business level interaction a consumer needs to have with the service provider in order to consume this service. The internal description of how a ServiceFunctionality is implemented is outside the scope of SSL since it refers to the internal business processes of the organisation. The objective of SSL is to describe to the candidate customers what they need to offer and what they will receive when consuming this service. When modelling a ServiceFunctionality the modeller can specify the following parts:

- **ServiceInput**: Represents the information required by the service consumer in order for the *service functionality* to be performed. The type of a *service input parameter* will be typically a concept defined in some domain ontology. It should be stressed that no specific schema for an input document needs to be provided. What is needed is to abstractly define what the consumer has to give. Examples of SSL ServiceInput are business concepts like "TravelPlan", "BankAccount", "Order", etc. The meaning of these business concepts will typically be defined into some domain ontologies.

- **ServiceOutput:** Represents the information produced when the service has performed a specific functionality. As with the *service input* parameter, the type of a *service output parameter* will typically be a concept defined in some ontology. Furthermore, a service output parameter may be associated with one ***condition***. At the current version of the SSL meta-model, an instance of the class *condition* is a description of the expected state of the world in order for the *service functionality* to produce this specific *service output* information. An output parameter which is associated with a condition is called a *conditional output parameter*.

SSL defines that Service Functionalities can be associated with one condition, which represents the expected state of the world in order for the functionality to be successfully carried out. In that case, the associated condition is called **precondition**. A pre-condition for example may define that in order to make a room reservation you need to have a valid credit card. It is important to stress that a service functionality condition (precondition or not) has nothing to do with the format (or the schema) of the input or output. It does not define constraints on the format of data. When used as a precondition it defines constrains on the status of the resources (information, or other tangible resources) provided by the consumer during the business level interaction (service consumption). On the other hand, when used with the functionality output it describes the conditions under which the specific resource (information or tangible resource) will be given.

SSL does not specify that the service functionalities defined for a service profile should correspond to the same operation implemented by a software component. It is important to understand that a service profile describes semantically the business level interaction needed to consume a service and not its programmatic interface. In case that a service will be implemented by a software component, then a single service functionality described in the Service Profile of a service may be implemented using several operations with different (probably specified in more detail) input and output message types in the SDL model. For instance, the "MakeReservation" functionality described in the hotel reservation example could be implemented by four SDL operations: One for getting the CheckIn, CheckOut, NumberOfAdults, and RoomType, parameters, one for getting the Guest related information, one for getting the billing information (credit card number details), and one for responding with the result of the reservation request.

SDL operations could have reference to the SSL functionalities that they realize in order to declare what functionality they implement. In general, SSL neither enforces nor prohibits

consistency between SSL functionalities and SDL operations. In the software generation process SSL could drive the SDL specification. That is, it could be used as input to (semi-) automatic SDL definition. Such a task will be done by the SSL compiler component.

### 6.2.8 SSL Type Referencing Scheme

When creating a metamodel using MOF one must define the data types that are used at that level (M2) in order to provide a type for attributes of the MOF Classes that one defines. Moreover, it is necessary to define a data typing scheme for the models of that metamodel, if non CORBA data types are to be used (which are the default data types used by OMG). SSL makes use of the data typing scheme defined in ODM. Moreover, as already mentioned several times in this document, SSL modelling elements may have as type a concept defined in some domain ontology. SSL provides a type referencing scheme that allows the modeller to define as types of modelling elements the following:

a) XML Schema Datatypes
b) Ontology Concepts
c) Enumerations[4]

The following Figure shows how the SSL data typing scheme has been defined.

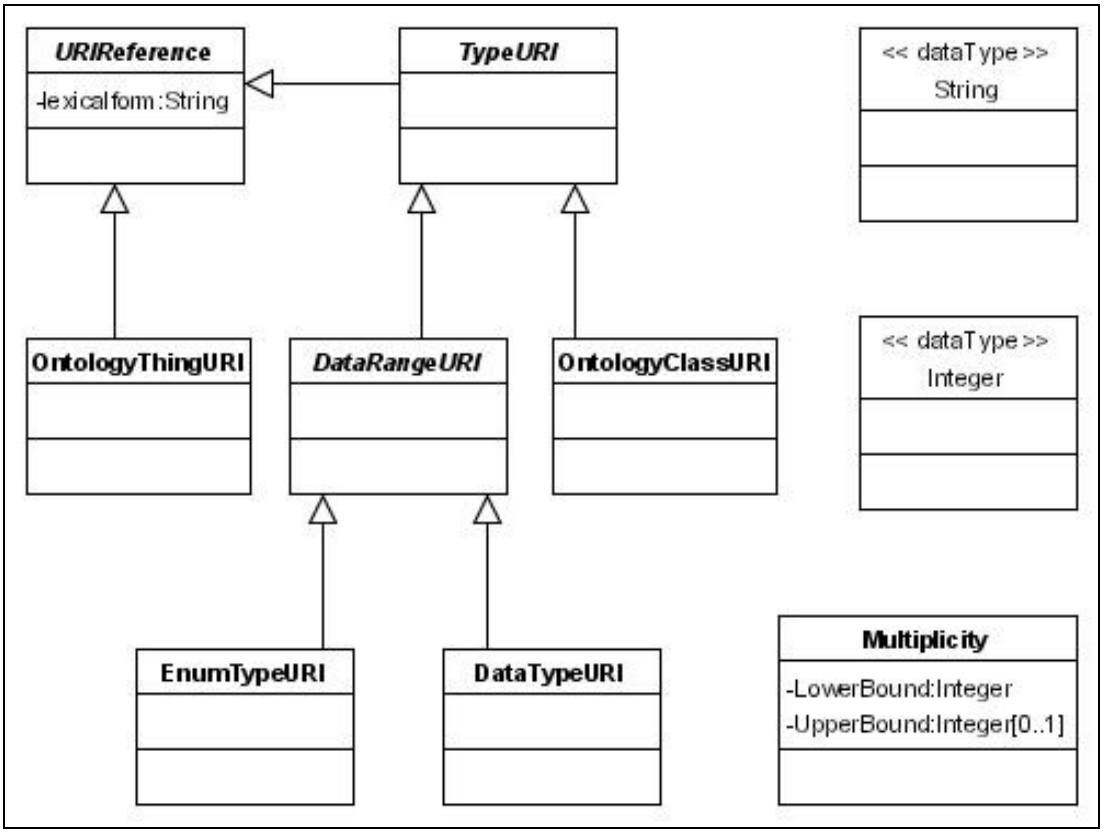


**Figure 27: SSL Type Referencing Scheme**

Examples of using all the aforementioned types are provided in the appended Hotel Reservation Example.

[4] Enumerations cannot be defined outside ontologies at the moment.

## 6.3 Summary of the Semantic Service Language

This chapter described the first version of the metamodel of the Semantic Service Language developed in the scope of the "Service Ontologies" task of the DBE project. This metamodel is used in conjunction with the other metamodels of the DBE Knowledge Representation Framework (i.e. ODM, BML, SCM, and SDL). In particular, the domain expert, or the SME provider will describe the concepts that describe a domain of discourse using the ODM. Then, a SME Service Provider may create a model to semantically describe its service's by using the SSL as well as Domain Ontology Concepts.

Besides its integrations with the ODM, SSL needs also to be integrated with the rest parts of BML as well as with the SDL (Service Description Language) in order to work into its full potential. Such a process is under way and it will be finalised in the next releases of these metamodels in the scope of the future work in the KB, BML, and SDL work packages (WP14, WP15, and WP16 respectively).

# 7. Representing Data

The DBE Knowledge Representation Framework follows the MOF metadata architecture according to which four information abstraction layers are defined:

a) The information or instance layer (M0)

b) The metadata or model layer (M1)

c) The meta-metadata or metamodel layer (M2)

d) And the meta-meta-metadata or meta-metamodel layer (M3)

In the definition of this architecture the focus was metadata modelling and organisation. The MOF is used as the primary meta-modelling mechanism (at the M3 layer) for defining metamodels (at the M2 layer) which are in turn used to specify the structure of Models (at the M1 layer). A model standing at one of these three layers defines the meta-data and the structure of the models in the lower layer. That is, a model at M3 (such as MOF) defines the meta-data and the structure of models at M2 layer (metamodels); a model at M2 layer defines the metadata and the structure of the models at M1 layer, and so on.

For all of these kinds of meta-information the MOF metadata architecture uses MOF and XMI to specify their structure and the representation. However, since it was not in its objectives, the architecture does not specify how data in the Information layer are represented.

Although it would be possible to distinguish data from meta-data and to represent them using standard XML, this approach would be in contradiction to the technical requirement for Knowledge integrity enforcement. That is, meta-data (i.e. models that specify the structure of data and the constraints that should be enforced) would be represented in MOF and XMI (following the MOF metadata architecture) and data would be represented with standard XML without having an XML schema against to which would be validated. Thus, it would be very difficult to check the integrity of data against their models.

In order to enforce (to some degree) knowledge integrity, it would be necessary to create XML schemas from M1 models and then to generate XML documents conforming to these schemas as instances of the M1 models. Such an approach increases extremely the complexity of the system making its maintenance a difficult task. Moreover, it causes a number of other problems that make its adoption impossible.

The most important of these problems refer to knowledge management and access. Since both data and metadata are exploitable knowledge that will be managed by the DBE Knowledge Base, it is ineffective to use different technologies for encoding and representing its contents. Moreover, the knowledge access would not be achieved by the same mechanism build for accessing knowledge represented with MOF [32].

To overcome these problems, a native MOF-based Representation Model is needed in order to represent the Instances of the DBE modelling languages. Similar approaches have been followed in other MOF-based standards like the Common Warehouse Metamodel (CWM) [33] in order to allow exchange of both data and metadata using XMI.

To address this need we have developed the DBE Instantiation Metamodel that allows for uniform, native representation of DBE model instances (data) to be created. Figure 28 illustrates the simple DBE instantiation metamodel.

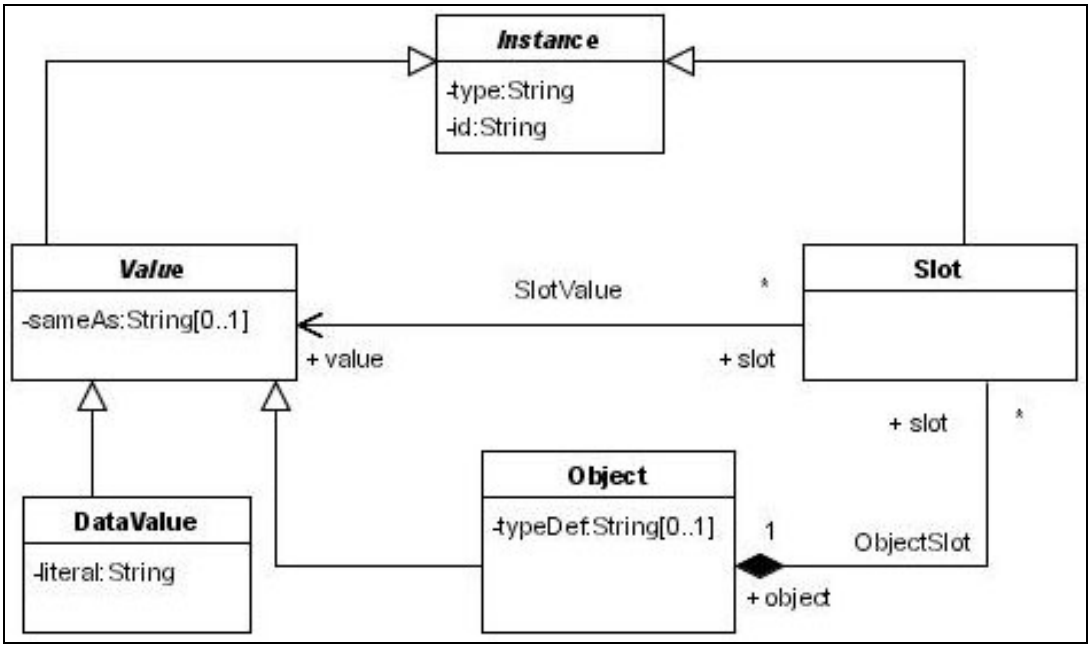


**Figure 28: The DBE Instantiation Metamodel**

Every element that is represented with this metamodel is an ***Instance*** of some modelling element (e.g. instance of an SSL *ServiceParameter*, instance of a BML BusinessItem, etc.). Each instance should be given an *id* and a *type*. While the id is used for model management reasons, the type is used for both model management and semantic representation reasons. The type of an instance element refers to the model element that is instantiated by this element. For example, an instance of service parameter (of an SSL model) named "HotelFacility" will have as type the id of this M1 model element (i.e. the "HotelFacility" service parameter).

The class Instance is an abstract class that is not instantiated. Only the concrete classes of the Instantiation Metamodel are instantiated. These are:

- ***Object***: An object represents an instance of a class defined in some model. Such classes are for example the instances of the SSL ServiceProfile meta-class, the instances of the SSL ServiceParameter and ContantInformation meta-class, the BML BusinessItem and Asset etc. Since the DBE Knowledge Representation Framework gives the ability of integration of Ontology Concepts into SSL and BML models, an instance of such a class should keep a reference not only to the model class, but also to the ontology class that it is related to. This is done through the attribute *typeDef* that keeps a reference to the ID of that ontology concept.
- ***Slot***: A slot represents an instance of an attribute of some entity defined in some model. In a model attributes are owned by classes; accordingly, slots are owned by objects. Moreover, slots also represent instances of associated entities. For example, an instance of ContactInformation class that is associated with a Service Profile class will be represented as a slot of the object instance of the Service Profile. Slots have a ***value***. The value of a slot can be either a single data value in case that its type is a simple attribute (i.e. an attribute that has as type a primitive type) or an object in case that its type is a complex attribute (i.e. an attribute that has as type another class), or an association end (which has as type another class). Values may keep a reference to an individual or literal into some ontology for reasoning purposes

- ***DataValue***: A DataValue represents a literal that is the instance of a primitive type (e.g. string) and is assigned to a slot.

The DBE instantiation metamodel is currently implemented into the DBE Knowledge Base and it will be tested once the DBE Authoring Tool will be developed. Thus, minor modifications may be needed (as will all the other Knowledge Representation Models of DBE) in order to better support data representation.

# 8. Conclusions

In this document we have described the Knowledge Representation Framework of DBE and how the various knowledge representation models of this framework are inter-connected. The DBE Knowledge Representation Framework specifies the structure and the dependencies of the Knowledge Base contents. The knowledge that is kept in the DBE Knowledge Base will be exploited by the inhabitants of the ecosystem through other system components like the Semantic Discovery Tool, the Recommender, the Fitness Landscape in the EvE, etc.

The high level objectives of the DBE Knowledge Representation Framework and its models are:

a) To enable semantic and syntactic interoperability among SMEs by describing both the semantics and the syntax of their organisation and the description of their offerings.

b) To support SMEs in advertising themselves and discovering candidate partner SMEs and services in order to constitute larger value chains and thus to operate in new business opportunity spaces.

c) To allow exploitation of the business semantics in the (semi-) automatic generation of software that implements business services.

These high level objectives impose several technical requirements for the DBE Knowledge Representation Framework and its models that have been discussed in early sections of this document. For the satisfaction of these requirements the Knowledge Representation Framework of the DBE Knowledge Base follows the OMG's MOF metadata architecture where four different abstraction layers (M3, M2, M1, and M0) are defined for modelling, representing and organizing data and metadata. Thus, the Knowledge Representation Models of DBE have been defined as MOF models (i.e. M2 metamodels) and allow the definition (by users) of models (M1) as well as instances of these models (M0).

The core metamodel of the DBE Knowledge Representation Framework (DBE KRF) is the Ontology Definition Metamodel (ODM) that is used to represent domain ontologies that capture community accepted semantics (concepts and relationships among them) of a particular business domain and they are the reference point of every specific model or semantic description. The ODM has been developed in accordance to the corresponding Request for Proposals (RFP) issued by OMG and is compatible with the Ontology Web Language (OWL - DL) in order to allow the DBE to be an open environment that can re-use existing ontologies from the semantic web community can be reused in DBE.

For the modelling and description of specific businesses and services, specific modelling languages providing the required modelling primitives to the business experts have been developed. Such languages are the Business Modelling Language (BML) and the Semantic Service Language (SSL).

The approach followed to the Business Modelling in DBE has been inspired by the corresponding vision of the Object Management Group. This vision specifies the components of a complete architecture for modelling business organizations. DBE has opted to extend this architecture by specifying service modelling mechanisms for modelling the services offered by a business organization. Thus BML and SSL will be integrated languages of the same business modelling mechanism.

The abstract syntax of these languages (their meta-models) has been defined using MOF and is constituent parts of the DBE Knowledge Representation Framework. With these

languages the business analysts can define (or reuse) models (M1 Layer) of business and service descriptions whose instances will provide the specific business and service descriptions. These descriptions (M0 data) are particularly important in DBE and thus the DBE Knowledge Base should be able to host and manipulate it in an effective way. Since the MOF meta-data architecture does not specify how M0 information is encoded and represented a specific Knowledge Representation Model (instantiation metamodel) has been developed as specific mechanism for this purpose. This mechanism is the DBE Instantiation Metamodel that is used to represent the instances of BML and SSL models (i.e. the BML and SSL descriptions).

Furthermore, the DBE Knowledge Representation Framework will be extended to represent the biosphere of the ecosystem and in particular the Regulatory Framework (the Knowledge Base Model for the Regulatory Framework will be defined in WP32). The Regulatory Framework would be exploited in the service evolution and recommendation processes. Representation of User Profiles (defined in WP7) is also an important extension that will be done to the DBE Knowledge Representation Framework since user profiles are important knowledge for several advanced functions of the system. User Profiles will be defined by the partners FZI under the work package 7 (User Profiling Mechanism) in collaboration with TUC so that user profiles to be integrated into the entire Knowledge Representation Framework of DBE and exploited by the recommender component.

# 9. Glossary

| Term | Description |
|---|---|
| ADDTF | Analysis and Design Domain Task Force: A Special Interest Group in the OMG organization that focuses on System Analysis and Modelling as well as on Software Engineering technology. |
| API | Application Programming Interface: Is a technology that facilitates exchanging messages or data between two or more different software applications |
| BEIDTF | Business Enterprise Integration Domain Task Force: A Special Interest Group in the OMG organization that focuses on Business Modelling and Integration. |
| BML | Business Modelling Language |
| BOM | Business Organization Metamodel: It is one component of the OMG's Business Modelling Architecture and provides for modelling the whole company, its resources, its organizational units, the relationships among them, etc. |
| BPDF | Business Process Definition Metamodel: It is one component of the OMG's Business Modelling Architecture and provides for describing processes. Processes are described in terms of input, activity, output, including process precedence and rules governing process branching, looping, and synchronization. |
| BPEL4WS | Business Process Execution Language for Web Services, in its Version 1.1, is a specification produced by a consortium that joins together BEA, IBM, Microsoft, SAP AG and Siebel Systems, whose main aim is the definition of an interoperable and integrated model that should facilitate the expansion of automated process integration in business-to-business interactions. Such specification provides a language that specifies business processes and business interaction protocols extending the Web Services interaction model and enabling it to support business transactions. |
| BSBR (or SBVR) | Business Semantics for Business Rules (or Semantics of Business Vocabularies and Business Rules): It is one component of the OMG's business modelling architecture and provides a general linguistic metamodel that is mapped to formal logics, especially first order predicate logic, modal logic, basic arithmetic, and set theory. The BSBR (or SBVR) metamodel has two parts: a Vocabulary metamodel and a Business Rules metamodel and is mapped to MOF for model interchange. |
| CIM | Computational Independent Model: The most abstract layer in the MDA architecture. Models of this layer describe a modelled system from the business point of view (i.e. requirements, abstract functions, etc.) without any assumption on the implementation (software or not) details. |

| | |
|---|---|
| DBE KRF | DBE Knowledge Representation Framework: The Framework of the interconnected DBE Knowledge Representation Models and/or Languages that capture the knowledge of the ecosystem. |
| EvE | Evolution Environment: It is the "environment" of the DBE platform where the services are evolved in order to reach the best fitness point. |
| Fitness Landscape | Graphical visualisation of the fitness of a population of slightly different individuals drawn as the height of a surface over a plane defined by (two) independent parameters encoded in the DNA of the species. |
| IDL | Interface Definition Language: An OMG standard for describing the interfaces of objects. It is widely used in CORBA architectures. |
| IR | Information Retrieval: Technology for retrieving personalized information from large collections of unstructured, semi-structured, or structured data. |
| KB | Knowledge Base: Is the part of the DBE system where the DBE knowledge is stored and managed. Such knowledge refers to ontologies, business and service descriptions, etc. |
| MDA | Model Driven Architecture: An approach (proposed by OMG) to IT system specification that separates the specification of system functionality for the specification of the implementation of that functionality on a specific technology. |
| MOF | Meta Object Facility: A generalized facility and for specifying abstract information about very concrete object systems. |
| MOF Repository | A Repository for storing, managing and retrieving meta-data (models) and meta-meta-data (metamodels) that have been described with MOF. |
| OCL | Object Constraint Language: OMG's standard for expressing constraints and well-formedness rules on object models. The last release is also considered suitable for querying object models. |
| ODM | Ontology Definition Metamodel: A MOF model (metamodel) developed in DBE for ontology representation. |
| OMG | Object Management Group: International standardization body |
| OWL | Ontology Web Language: The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full. |
| OWL-S | Ontology Web Language – Services: It is a Web Service |

| | |
|---|---|
| | ontology based on OWL syntax and primitives. It is intended to enable greater access to content and services on the web as it is created in order to provide Web Service producers with a core set of mark-up language constructs allowing them to describe properties and capabilities of their web services in unambiguous, computer-interpretable form. |
| PIM | Platform Independent Model of a modelled system |
| PSM | Platform Specific Model of a modelled system |
| QML | Query Metamodel Language: A MOF-Based language developed in DBE that allows uniform access of DBE Knowledge kept in the DBE Knowledge Base. |
| RDF | Resource Description Framework. RDF is a general framework for how to describe any Internet resource such as a Web site and its content. It provides interoperability between applications that exchange machine-understandable information on the Web. RDF descriptions are often referred to as metadata, or "data about data". These can include the authors of the resource, date of creation or updating, the organization of the pages on a site. i.e. the sitemap, information that describes content in terms of audience or content rating, key words for search engine data collection, subject categories, etc. |
| RDF-S | RDF-Schema: A language for defining specific formal schemas in the RDF framework. |
| Recommender | A DBE (autonomous) Core Service that will provide users (SMEs) with personalized knowledge by exploiting their profiles |
| RFP | Request For Proposals: OMG documents asking for proposals on a new standard specification. |
| SCM | Service Composition Metamodel: The DBE metamodel used to model the internal workflow and communication logic of composite services. At the moment BPEL4WS is used for this purpose |
| SDL | Service Description Language: A MOF model (metamodel) that provides technical description of the programmatic interface of a service |
| Semantic Discovery Tool | A DBE tool providing the appropriate Graphical User Interface for formulating semantic–based queries for service discovery. |
| Semantic Registry | The component of the DBE Knowledge Base that hosts the published services (in the form of Service Manifest Documents). |
| Service DNA | Service DNA contains information related to a generic DBE service in term of its BML/SSL Model and/or its SDL Model. The service DNA does not contain any data (M0 in MOF architecture) and in this perspective it is a container structure like the Service Manifest but without the data layer. Service DNA has no dependency with technical services and the information it contains is abstracted respect to any real situation. The main purpose of the service DNA is to preserve service model with |

| | |
|---|---|
| | reuse purpose. |
| Service Manifest (SM) | The Service Manifest is a two-fold formal description of a specific DBE Service, and contains both the models and data (comprising both business and technological information) of a single specific real word service owned by a specific SME. |
| SME | Small and Medium Enterprise: Independent enterprise with less than 250 dependent. |
| SOAP | Simple Object Access Protocol: It is a lightweight protocol for exchanging information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols. |
| SR | Semantic Registry: It is the component of the Knowledge Base that hosts the service descriptions published in the DBE environment and available for discovery and consumption. |
| SSL | Semantic Service Language: A MOF-based language for semantically describing SME services in DBE. |
| SWS | Semantic Web Service: The semantically described service through sophisticated description models that can be enhanced with ontologies enabling machine interpretability of its capabilities as well as integration with domain knowledge. |
| UDDI | Universal Description, Discovery & Integration: An OASIS standard for the description, publishing, and discovery of web services. |
| UML | Unified Modeling Language: A method for specifying, visualizing, and documenting the artefacts of an object-oriented system under development; as well as for business modelling. |
| User Profiling Mechanism | A DBE mechanism used to trace user's actions (and transactions) in order to inspect his preferences on desirable services, and partners. |
| W3C | World Wide Web Consortium: International Standardization body that has defined and is maintaining many IT related standards like HTML, XML, XML-Schema, OWL, etc. |
| WSDL | Web Service Description Language: It is a standard that provides a model and an XML format for describing Web Services. This specification defines a language for describing the abstract functionality of a service as well as a framework for describing the concrete details of a service description. |
| XMI | XML Metadata Interchange: An SMIF (see SMIF description) standard specification based on XML. |
| XML | eXtensible Mark-up Language. XML is a flexible way to create common information formats and share both the format and the |

| | |
|---|---|
| | data on the World Wide Web, intranets, and elsewhere. |
| XML –Schema | An XML schema is a description of the type of an XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntax constraints imposed by XML itself. An XML schema provides a view of the document type at a relatively high level of abstraction. |
| XSL-T | A W3C standard for transforming XML documents into other XML documents or other formats. This was conceived as part of XSL but has been found to have wider applications |

# 10. Bibliography


[1] The World Wide Web Consortium, “Ontology Web Language”, http://www.w3.org/TR/owl-ref/

[2] Dragan Đuric, Dragan Gaševic, Vladan Devedžic“A MDA-based Approach to the Ontology Definition Metamodel”.

[3] Cranefield, S., “UML and the Semantic Web”, In Proceedings of the International Semantic Web Working Symposium, Palo Alto, 2001, www.semanticweb.org/SWWS/program/full/paper1.pdf

[4] The Object Management Group, ODM RFP, http://www.omg.org/cgi-bin/apps/doc?ad/03-09-06.pdf

[5] The Object Management Group, The Meta-Object Facility Specification http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf

[6] Tim Berners-lee. Semantic Web Road Map. W3C Design Issues. URL http://www.w3.org/DesignIssues/Semantic.html, Oct. 1998.

[7] Ora Lassila and Ralph R.Swick. Resource Description Framework (RDF) Model and Syntax Specification. Feb. 1999.

[8] Pan, J., Horrocks, I., “Metamodeling Architecture of Web Ontology Languages”, In Proceedings of the First SemanticWeb Working Symposium (SWWS'01), Stanford, July 2001, pp 131-149.

[9] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C Recommendation, URL http://www.w3.org/TR/rdf-schema, Mar. 2000.

[10] W3C, XML 1.0 (third edition), W3C Recommendation. http://www.w3.org/TR/2004/REC-xml-20040204

[11] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. can Harmelen, and I. Horrocks. Enabling knowledge representation on the Web by extending RDF Schema, Nov. 2000.

[12] Pan, J., Horrocks, I., “Metamodeling Architecture of Web Ontology Languages”, In Proceedings of the First SemanticWeb Working Symposium (SWWS'01), Stanford, July 2001, pp 131-149.

[13] Christensen, E. Curbera, F., Meredith, G., Weerawarana, S. Web Services Description Language (WSDL), W3C Note 15. http://www.w3.org/TR/wsdl. (2001)

[14] W3C. SOAP 1.2, W3C Recommendation. http://www.w3.org/TR/soap12-part0/ (2003)

[15] W3C. XSL-Transformations 1.0, W3C Recommendation, http://www.w3.org/TR/xslt

[16] Cabral, L. Domingue, J. Motta, E. Payne, T. and Hakimpour, F. Approaches to Semantic Web Services: An Overview and Comparisons. In Proceedings First European Semantic Web Symposium (ESWS2004)

[17] Biron, P. V., Malhotra, A. (eds.): XML Schema Part 2: Datatypes, W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-2/. (2001)

[18] UDDI Consortium. UDDI Specification. http://www.uddi.org/specification.html (2000)

[19] BPEL4WS Consortium. Business Process Execution Language for Web Services, http://www.ibm.com/developerworks/library/ws-bpel

[20] Motta, E., Domingue, J., Cabral, L., Gaspari, M.: IRS-II: A Framework and Infrastructure for Semantic Web Services. In: Fensel, D., Sycara, K., Mylopoulos, J. (volume eds.): The SemanticWeb - ISWC 2003. Lecture Notes in Computer Science, Vol. 2870. Springer-Verlag, Heidelberg (2003) 306–318

[21] OWL-S Coalition: OWL-S 1.0 Release. http://www.daml.org/services/owl-s/1.0/.(2003)

[22] Fensel, D., Bussler, C. The Web Service Modeling Framework WSMF. Eletronic Commerce: Research and Applications. Vol. 1. (2002). 113-137

[23] The SDK WSMO working group, Web Services Modeling Ontology, http://www.wsmo.org/

[24] SWWS Consortium. Report on Development of Web Service Discovery Framework. October 2003. http://swws.semanticweb.org/public_doc/D3.1.pdf

[25] ISUFI, DBE Deliverable: D16.1 - BML First Release: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID=16&expandFolder=16&folderID=0

[26] Omelayenko, B., Crubezy, M., Fensel, D., Benjamins, R., Wielinga, B., Motta, E., Musen, M., Ding, Y..: UPML: The language and Tool Support for Makiing the Semantic Web Alive. In: Fensel, D. et al. (eds.): Spinning the Semantic Web: Bringing the WWW to its Full Potential. MIT Press (2003) 141–170

[27] The Object Management Group, MDA Guide Version 1.0.1: http://www.omg.org/docs/omg/03-06-01.pdf, 2003

[28] Object Management Group, Analysis and Design Domain Task Force (ADTF): http://www.omg.org/adtf/

[29] Object Management Group, Stanley Hendryx & Associates: Architecture of Business Modeling: http://www.omg.org/docs/br/03-11-01.pdf

[30] SOLUTA.net, DBE Deliverable, D16.1 – Service Description Models and Language Definition: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID=16& expandFolder=16&folderID=0

[31] TCD, DBE Deliverable, D17.2 – Composer: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID =16& expandFolder=16&folderID=0

[32] TUC, DBE Deliverable, D17.1 – Recommender: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID =16& expandFolder=16&folderID=0

[33] The Object Management Group, Common Warehouse Metamodel (CWM) Specification v 1.1: http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf

[34] Studer R, Benjamins VR, Fensel D (1998) Knowledge Engineering: Principles and Methods. IEEE Transactions on Data and Knowledge Engineering 25(1-2):161-197

[35] Stanford University: Travel Ontology: http://protege.stanford.edu/plugins/owl/owl-library/travel.owl

[36] Open Travel Alliance: http://www.opentravel.org/2004A/index.cfm

[37] Object Management Group, XML Metadata Interchange Specification (v1.2): http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf

[38] OMG, Business Semantics of Business Rules – Request For Proposal: OMG technical document br/2003-06-03

[39] Zachman Framework for Enterprise Architecture

[40] UN/CEFACT: Modeling Methodology (UMM) User Guide, (2003) CEFACT/TMG/N093

[41] The Business Rules Group: Defining Business Rules, What Are They Really? Final Report, revision 1.3, (2000)

[42] P. Dini, T. Kuusisto; A. Corallo, P. Ferronato, N. Rathbone: Toward a Semantically Rich Business Modelling Language for the Automatic Composition of Web Services. (2003) Business Research Forum. Proceedings of eBRF 2002.

[43] The Business Rules Team: Semantics of Business Vocabularies and Rules (SBVR), Revised submission to BEI RFP br/2003-06-03, http://www.omg.org/docs/bei/05-03-01.pdf

[44] Organization for the Advancement of Structured Information Standards (OASIS), electronic business XML (ebXML), http://www.ebxml.org/

# 11. Appendix A: ODM Example

In this appendix we describe a sample ontology that describes the concepts (some of them) of the Tourism business domain. This sample ontology has been defined using the ODM. Most of the concepts defined in this ontology have been borrowed from the travel owl sample ontology provided by the Stanford University [35] and from the OTA (Open Travel Alliance) specifications [36].

The travel owl ontology defines a small set of concepts for the tourism domain describing destinations, activities that can be performed in each destination, etc. The Open Travel Alliance is dedicated to helping the travel industry take full advantage of the almost universal access to the Internet. Its main product is a set of specifications for allowing interoperability between tourism companies. OTA members are organizations that represent all segments of the travel industry, along with key technology and services suppliers.

The developed sample tourism ontology has been imported into the DBE Knowledge Base and its use (in defining SSL models) has been demonstrated in the first annual review of the project. The following diagrams present the conceptualization specified by this sample ontology in a graphical notation.

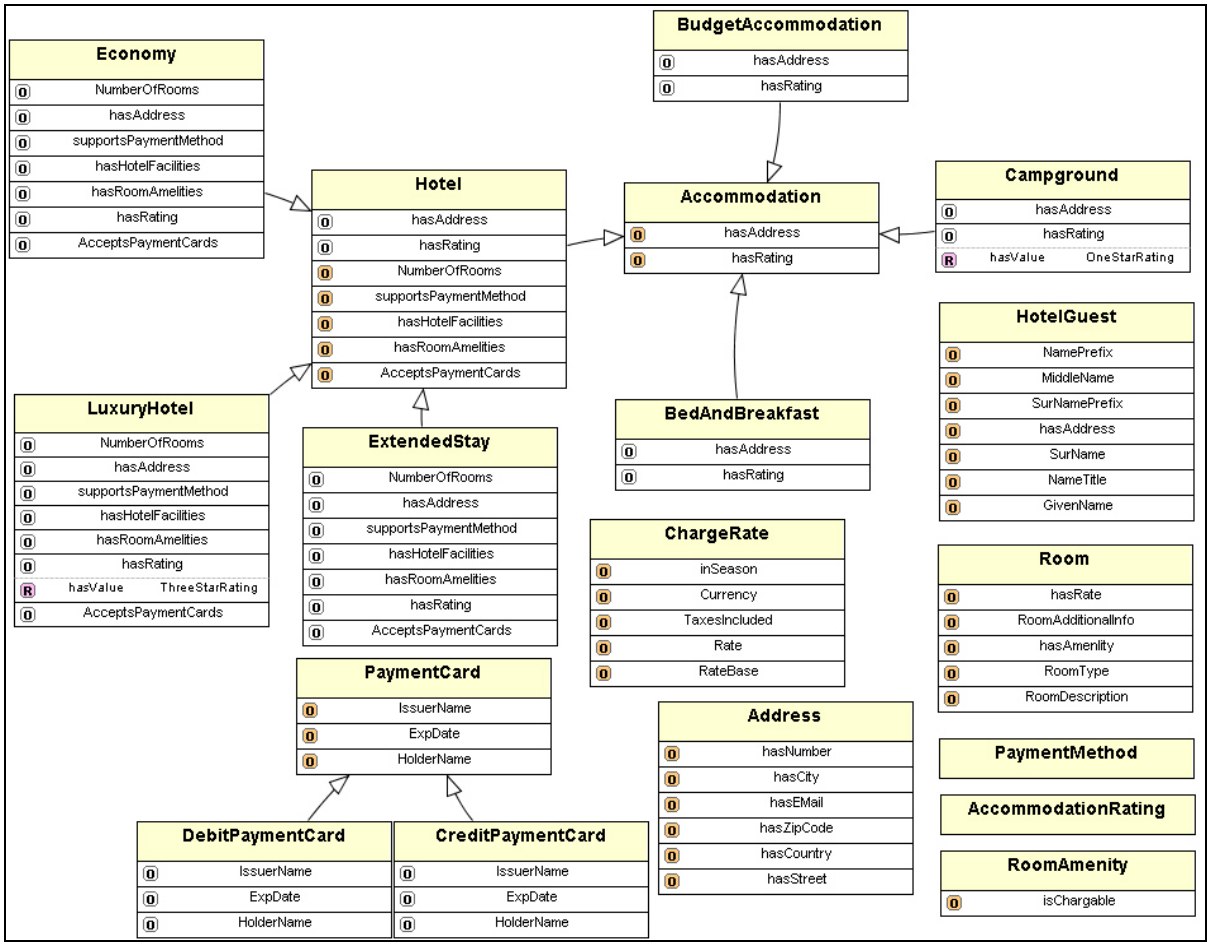


**Figure 29: Tourism Ontology – Diagram 1**

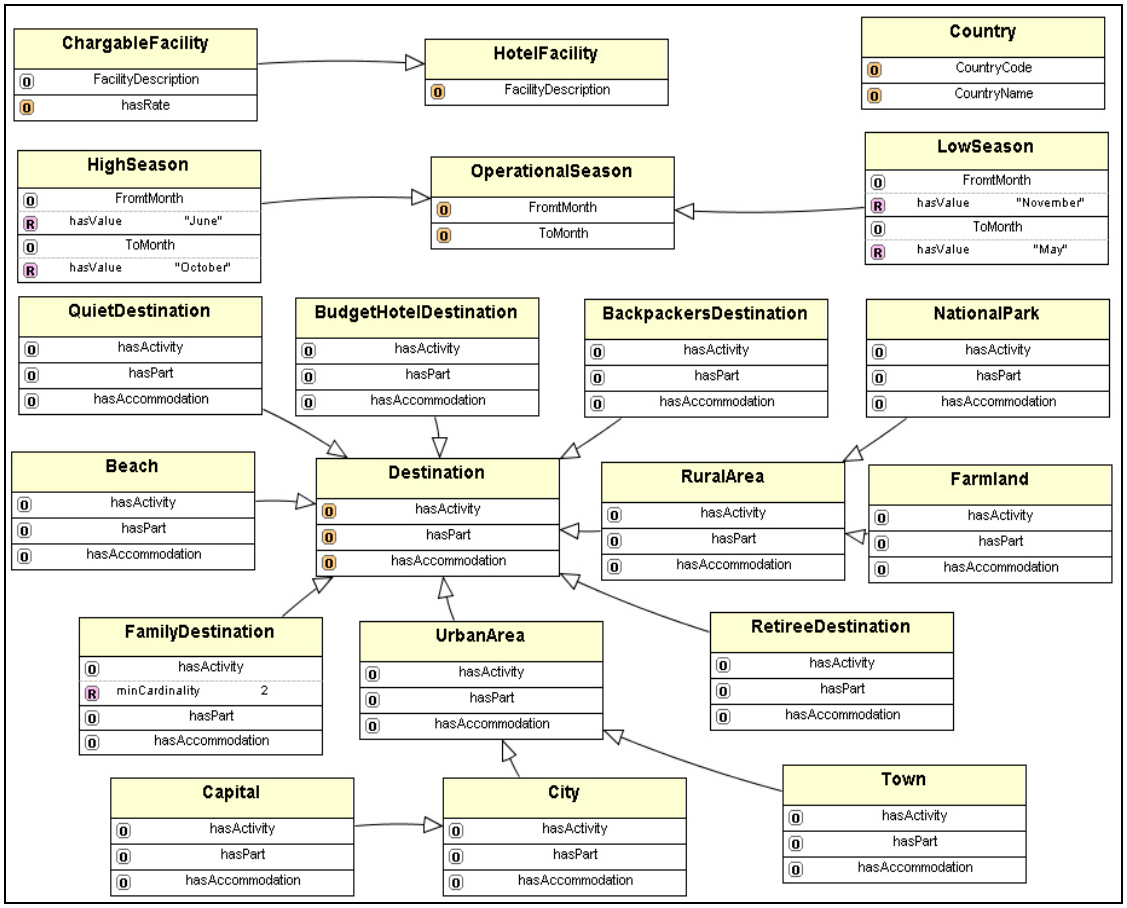


**Figure 30: Tourism Ontology – Diagram 2**

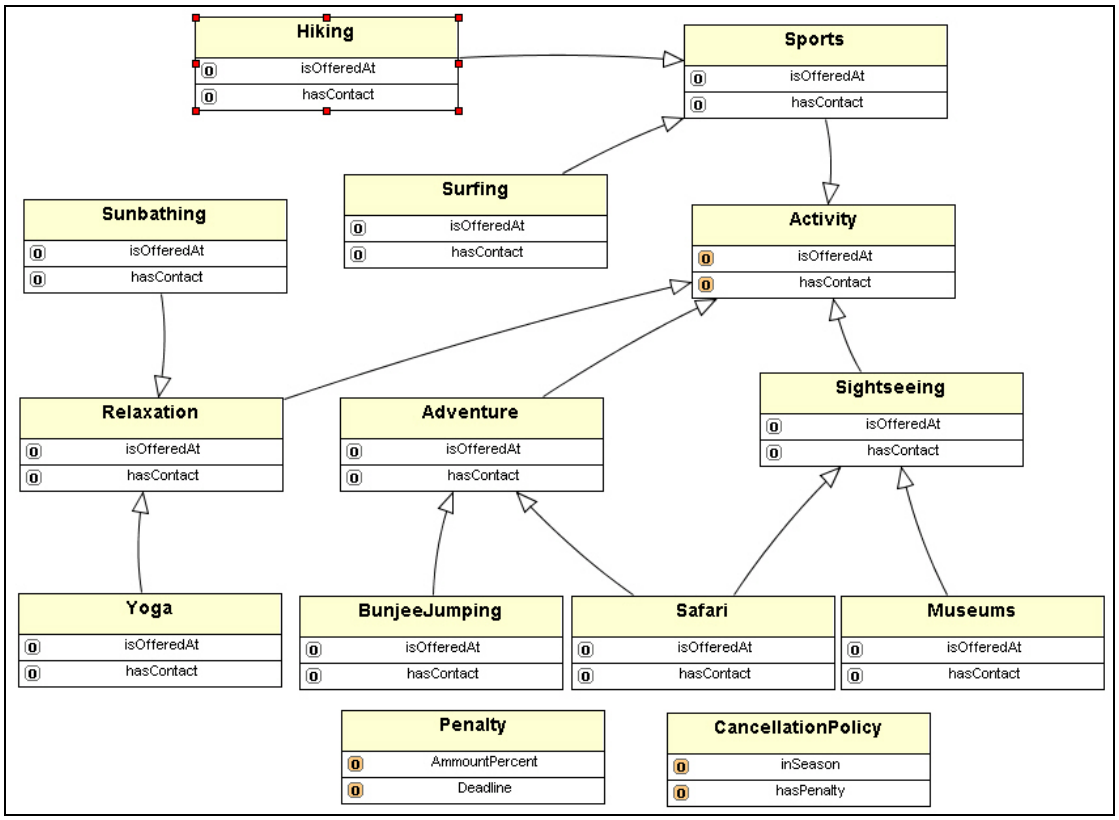


**Figure 31: Tourism Ontology - Diagram 3**

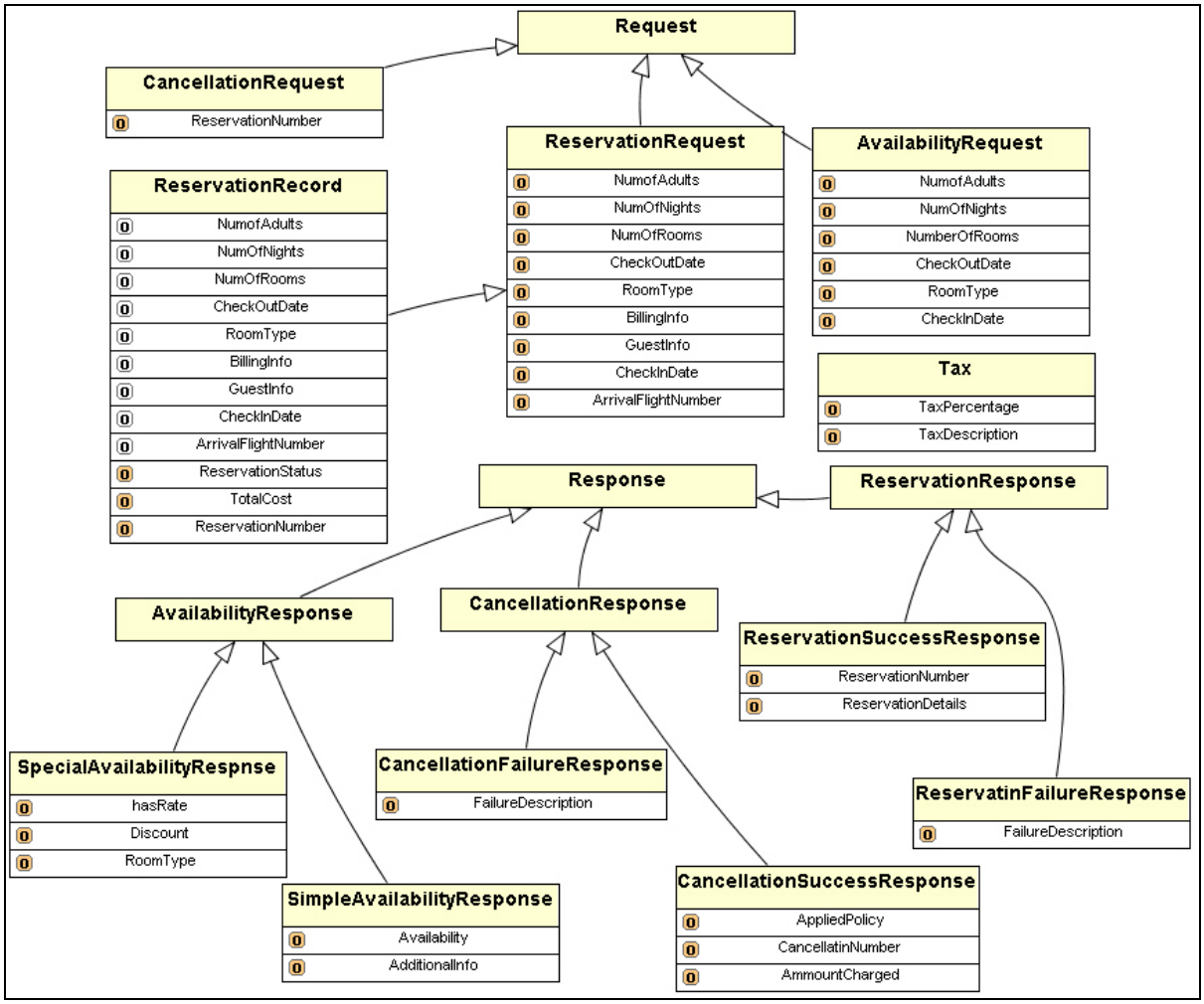


**Figure 32: Tourism Ontology - Diagram 4**

# 12. Appendix B: SSL Example

This appendix presents an example of how SSL can be used to define a model (service profile) for semantically describing Hotel Reservation Services. As explained in this document, the DBE Knowledge Representation Framework allows business and service models to make use of domain conceptualization in order to maximize semantic interoperability. Thus, the SSL model defined in this appendix makes use of concepts defined in tourism ontology defined in Appendix A. As with the tourism ontology, the presented SSL model has been used in the demonstration during the first annual review of the project. The following diagram shows this SSL model for Hotel Reservation Services in a UML – Like notation and it has been developed with the BML – Editor.

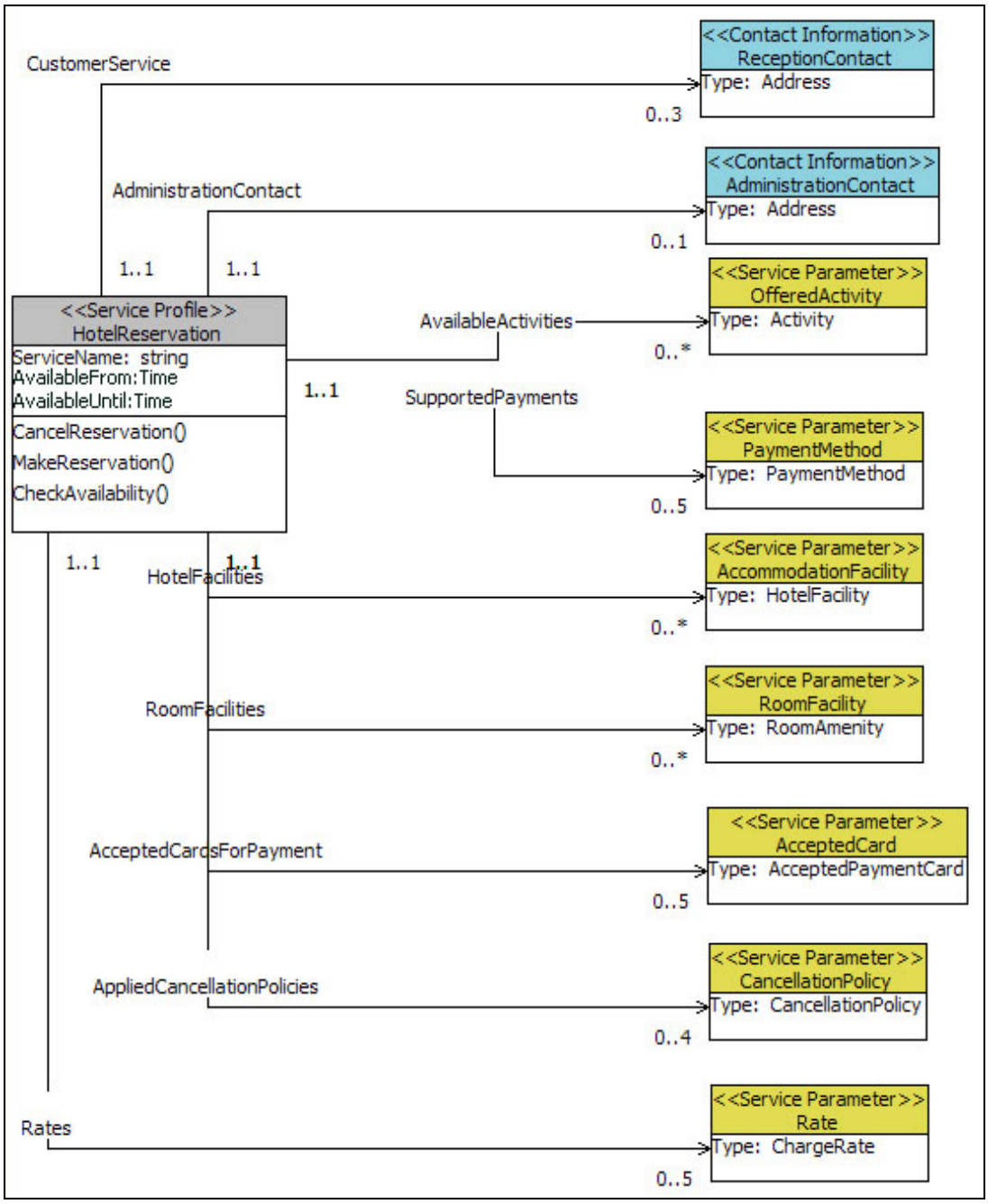


**Figure 33: SSL Model for Hotel Reservation Services**

# Appendix C: ODM and OWL Language Mappings

In this appendix we describe a mapping from ODM to OWL concepts and vice versa. The purpose of this mapping is to help users who are familiar with OWL to become familiar with ODM. Another purpose is for helping the creation of a UML profile that will be used as notation for defining ontologies with ODM.

As mentioned in the introductory sections of this document, ODM is not a one-to-one representation of the OWL data model in MOF. The effort was to be as close as possible; however, due to the formalisation required in MOF, some extra constructs were required for ensuring MOF compliance. On the other hand, OWL has pre-defined instances of some of its classes (e.g. the predefined ontology properties), something that is not allowed in MOF since it is not possible to define at M2 level both classes and their instances.

In what follows, we give the mappings a) from ODM to OWL and b) from OWL to ODM.

| ODM CLASS OR ASSOCIATION | OWL CONCEPT |
|---|---|
| Ontology | owl:Ontology |
| OntologyProperty | owl: OntologyProperty |
| AnnotationProperty | owl: AnnotationProperty |
| AnnotationObject | N/A |
| Class | owl:Class |
| complementOf_As | owl:complementOf |
| intersectionOf_As | owl:intersectionOf |
| unionOf_As | owl:unionOf |
| oneOf_As | owl:oneOf |
| disjointWith_As | owl:disjointWith |
| equivalentClass_As | owl:equivalentClass |
| subClassOf_As | rdfs:subClassOf |
| DeprecatedClass | owl:DeprecatedClass |
| Restriction | owl:Restriction |
| allValuesFrom_As | owl:allValuesFrom |
| someValuesFrom_As | owl:someValuesFrom |
| hasValue_As | owl:hasValue |
| onProperty_As | owl:onProperty |
| maxCardinality_As | owl:maxCardinality |
| minCardinality_As | owl:minCardinality |
| cardinality_As | owl:cardinality |
| Value | N/A |

| | |
|---|---|
| ValueRange | N/A |
| Thing | owl:Thing |
| AllDifferent | owl:allDifferent |
| distinctMembers_As | owl:distinctMembers |
| sameAs_As | owl:same_As |
| hasType_As | owl:type |
| differentFrom_As | owl:differentFrom |
| Property | N/A |
| DatatypeProperty | owl:DatatypeProperty |
| ObjectProperty | owl:ObjectProperty |
| hasDomain_As | rdfs:domain |
| hasDataRange_As | rdfs:range |
| hasRange_As | rdfs:range |
| inverseOf_As | owl:inverseOf |
| subPropertyOf_As | owl:subPropertyOf |
| equivalentProperty | owl:equivalentProperty |
| SymmetricProperty | owl: SymmetricProperty |
| TransitiveProperty | owl: TransitiveProperty |
| InverseFunctionalProperty | owl: InverseFunctionalProperty |
| FunctionalObjectProperty | owl:FunctionalProperty |
| FunctionalDatatypeProperty | owl:FunctionalProperty |
| DeprecatedDatatypeProperty | owl:DeprecatedProperty |
| DeprecatedObjectProperty | owl:DeprecatedProperty |
| PrimitiveType | N/A |
| DeprecatedDatatype | owl:DeprecatedClass |
| Datatype | rdfs:Datatype |
| DataRange | rdfs:DataRange |
| Enumeration | rdf:List |
| Literal | owl:Literal |
| PlainLiteral | N/A |
| TypedLiteral | N/A |
| URIreference | N/A |
| DefinitionURI_As | N/A |
| Datatype_As | N/A |

| | |
|---|---|
| NonNegativeInteger | N/A |

**Table 1: ODM to OWL Mapping**

| OWL | ODM CLASS OR ASSOCIATION |
|---|---|
| owl:allDifferent | allDifferent |
| owl:allValuesFrom | allValuesFrom_As |
| owl:AnnotationProperty | AnnotationProperty |
| owl:*backwardCompatibleWith* | N/A |
| owl:cardinality | cardinality_As |
| owl:Class | Class |
| owl:complementOf | complementOf_As |
| owl:DatatypeProperty | DatatypeProperty |
| owl:DeprecatedClass | DeprecatedClass |
| owl:DeprecatedProperty | DeprecatedDatatypeProperty<br>DeprecatedObjectProperty |
| owl:DataRange | DataRange |
| owl:differentFrom | differentFrom_As |
| owl:disjointWith | disjointWith_As |
| owl:distinctMembers | distinctMembers_As |
| owl:equivalentClass | equivalentClass_As |
| owl:equivalentProperty | equivalentProperty |
| owl:FunctionalProperty | FunctionalObjectProperty<br>FunctionalDatatypeProperty |
| owl:hasValue | hasValues_As |
| owl:*imports* | N/A |
| owl:*incompatibleWith* | N/A |
| owl:intersectionOf | intersectionOf_As |
| owl:inverseFunctionalProperty | inverseFunctionalProperty |
| owl:inverseOf | inverseOf_As |
| owl:maxCardinality | maxCardinality_As |
| owl:minCardinality | minCardinality_As |
| owl:Nothing | N/A |
| owl:ObjectProperty | ObjectProperty |

| | |
|---|---|
| owl:oneOf | oneOf_As |
| owl:onProperty | onProperty_As |
| owl:Ontology | Ontology |
| owl:OntologyProperty | OntologyProperty |
| owl:*priorVersion* | N/A |
| owl:Restriction | Restriction |
| owl:sameAs | sameAs_As |
| owl:someValuesFrom | someValuesFrom_As |
| owl:SymmetricProperty | SymmetricProperty |
| owl:Thing | Thing |
| owl:TransitiveProperty | TransitiveProperty |
| owl:unionOf | unionOf_As |
| owl:*versionInfo* | N/A |
| rdf:type | HasType_As |
| rdfs:Datatype | Datatype |
| rdfs:domain | hasDomain_As |
| rdfs:Literal | Literal |
| rdfs:range | hasDataRange_As<br>hasRange_As |
| rdfs:subClassOf | subClassOf_As |
| rdfs:subPropertyOf | subPropertyOf_As |
| | |

**Table 2: OWL to ODM Mapping**