Chapter 14
Representing USDL for Humans and Tools[*]

Keith Duddy, Matthias Heinrich, Steffen Heinzl, Martin Knechtel, Carlos Pedrinaci, Benjamin Schmeling, and Virginia Smith

Abstract This chapter deals with technical aspects of how USDL service descriptions can be read from and written to different representations for use by humans and tools. A combination of techniques for representing and exchanging USDL have been drawn from Model-Driven Engineering and Semantic Web technologies. The USDL language's structural definition is specified as a MOF meta-model, but some modules were originally defined using the OWL language from the Semantic Web community and translated to the meta-model format. We begin with the important topic of serializing USDL descriptions into XML, so that they can be exchanged between editors, repositories, and other tools. The following topic is how USDL can be made available through the Semantic Web as a network of linked data, connected via URIs. Finally, consideration is given to human-readable representations of USDL descriptions, and how they can be generated, in large part, from the contents of a stored USDL model.

[*] In Alistair Barros, Daniel Oberle (eds.): *Handbook of Service Description: USDL and its Methods*, Part III, Chapter 14, pages 357–384. Springer, New York, 2011.

Keith Duddy
Queensland University of Technology, GPO Box 2434, Brisbane, QLD 4001, Australia,
e-mail: `keith.duddy@qut.edu.au`

Steffen Heinzl, Benjamin Schmeling
SAP Research Darmstadt, Bleichstrasse 8, 64283 Darmstadt, Germany,
e-mail: `steffen.heinzl@sap.com`, e-mail: `benjamin.schmeling@sap.com`

Matthias Heinrich, Martin Knechtel
SAP Research Dresden, Chemnitzer Strasse 48, 01187 Dresden, Germany,
e-mail: `matthias.heinrich@sap.com`, e-mail: `martin.knechtel@sap.com`

Carlos Pedrinaci
Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK,
e-mail: `c.pedrinaci@open.ac.uk`

Virginia Smith
Hewlett-Packard Company, 8000 Foothills Blvd, Roseville, CA 95747, USA,
e-mail: `virginia.smith@hp.com`

14.1 Introduction

The previous part of the book has described the semantics of the USDL language in terms of its abstract syntax shown as UML *class diagrams* with natural language explanations, and examples. This part presents tools and approaches to the creation, representation and communication of actual service descriptions using a variety of formats. The class diagrams show the structure of information that the USDL language expresses, with classes representing the main concepts we wish to capture about services, attributes of these classes representing properties of the concepts, and references between the classes representing links between the concepts. The underlying specification in terms of which the USDL is defined is the Ecore language from the Eclipse Modeling Framework (EMF), which is derived from the Meta-Object Facility (MOF) [19] — an Object Management Group (OMG) standard for meta-data definition which shares the same semantics as the UML class modeling language. MOF and Ecore also use a subset of the graphical language of UML to provide diagrammatic representations of meta-models. The diagrams do not show every feature of UML, MOF or Ecore, but the concepts they do show share the same semantics. Ecore has an XML representation with filenames ending in `.ecore` which captures all of the details.

For readers not familiar with EMF, the following comparison with XML may assist. The Ecore language plays the same role as the XML Schema language for defining XML schemas — it provides the basic concepts in terms of which object-oriented models are defined: XML element types are roughly equivalent to classes, XML attribute types are similar to Ecore classes' attributes, and XML ref element types are similar to Ecore references. A document type X is defined by a schema definition `X.xsd`, in the same way as a language Y is defined by the Ecore model (also called a meta-model) `Y.ecore`.

A schema document (`.xsd` file) contains element type definitions that constrain what elements may appear in an XML document that validates against the schema, in the same way that a meta-model (serialized as an `.ecore` file) defines what objects may appear in a model instance (serialized as an `.xmi` file) that conforms to the meta-model.

Figure 14.1 shows the modeling hierarchy in the EMF technical space in which the USDL is defined. It has the Ecore language at the top layer, with its concepts of packages, classes, attributes and references. The next layer down contains the USDL meta-model, which identifies concepts about services in terms of Ecore classes, attributes and references. The lowest layer shows USDL service descriptions expressed in the USDL language, as instances of the classes defined in the USDL meta-model.

As you can see, there are two possible instantiations that come as standard within EMF: Java objects which are instances of Java classes generated according to standard mappings from the Ecore meta-models; and XMI files, which conform to the XML Schemas that are also generated from Ecore meta-models. In EMF the XMI mapping is used to create a default serialization from a set of in-memory Java objects to an XML document.

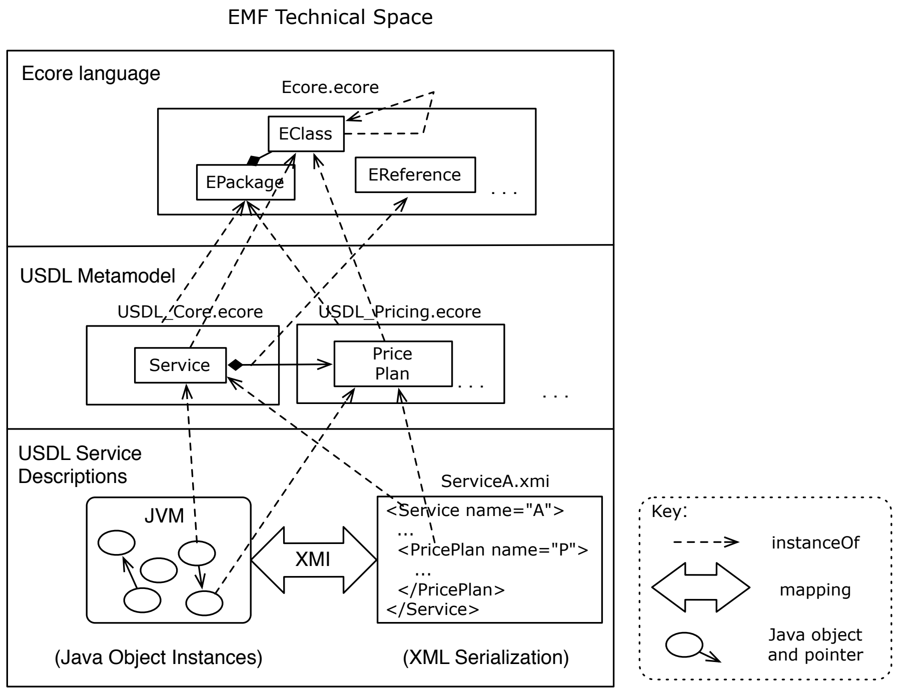


Fig. 14.1: Meta-model hierarchy for USDL.

In addition to the EMF code generators, there is a large developer community, both open-source and private, which has extended EMF with myriad tools for editing, serializing, and transforming models; for storing them in various kinds of databases; and for creating tools around them. EMF is a plugin for Eclipse, which is the world's most widely used Java developer environment, and therefore integrates the use of Ecore models in Java with hundreds of other frameworks and plugins.

The use of tools which manipulate models to produce textual syntaxes, code, editors, data stores and other models of various kinds is known generically as *Model-Driven Engineering (MDE)*. One of the earliest articulations of the concept of models as first class artifacts in software engineering was through the OMG's *Model Driven Architecture (MDA)* [26], introduced in 2000, which began by suggesting that abstract models of business functionality could be transformed into running code through a series of layered transformations: *Computation Independent Models* could be augmented by some additional information and transformed into *Platform Independent Models*, which could then be transformed using mappings to *Platform Specific Models* from which code could easily be derived by (re-)using best-practice patterns. Although MDA is only being used in this suggested form in a minority of cases, the basic concept of specifying higher-level abstractions of domain concepts as meta-models, and then using mappings to different formats, and code-generation tools to support the models, rather than writing code, is now well established. The approaches we call MDE also overlap with technologies from the *Domain Specific Language (DSL)* community, in which the concept of the domain language definition is very close to the abstraction of the meta-model, and many tools bridge the gap between textual and graphical DSLs and object-oriented models. The storage and manipulation of object models in programs which are derived from grammars representing textual syntaxes is now commonplace, and integrates tools designed for compiler construction, such as parser generators and parse-tree manipulation tools with other model transformation languages and tools.

Furthermore, there are automated mappings from Ecore to other representational formats such as non-XML textual languages, as well as Semantic Web languages such as OWL and RDF. However, it is more valuable in a Semantic Web environment to use human-guided mappings which match the concepts in the USDL to those represented in existing vocabularies so that service meta-data from USDL can be integrated into a larger web of Linked Data.

This chapter shows how model-driven techniques are used to manipulate the Ecore models of the USDL language in order to create a number of concrete representations of USDL service descriptions. This chapter is structured as follows: Section 14.2 explains the approach used to serialize USDL for interchange between tools that can place an entire service description into a single XML document, or can exchange an XML representation of one of the modules of USDL at a time. The use of the augmented XML framework, viz., the *Service Modeling Language (SML)*, is also considered for its document cross referencing, packaging and validation capabilities. This is followed in Section 14.3 by an analysis of the existing Semantic Web environment for established ontologies and vocabularies with which the USDL is well aligned. Section 14.3 then describes approaches across several Semantic Web technologies for representing USDL so that existing Linked Data approaches to modeling domain semantics can be matched and re-used, and to overcome some of the limitations of semantic representation in the object-oriented MOF specification. Finally, Section 14.4 demonstrates the use of template-based query approaches over EMF models to provide human-readable representations of USDL.

14.2 Serialization of USDL models

The approach used for the serialization of USDL models is based on the XML Metadata Interchange standard (XMI), which is defined by the Object Management Group (OMG). A few requirements are introduced that are necessary (or nice to have) for the goal of creating well-formed XML documents that describe a concrete USDL model (i.e., a service description). Based on the requirements, two ways of creating a serialization module for USDL are described. Leveraging XMI, it is possible to generate XML documents from the USDL models, serving as a concrete syntax for the USDL's abstract syntax. Finally, the use of the Service Modeling Language is considered for its capacity to cross-link models between documents, and to package a set of XML Schema and instance documents into a single XML file for distribution.

14.2.1 Model Requirements for an XML-based Concrete Syntax

When using XML as a concrete syntax, a number of requirements arise that we will categorize into two types: *technical* and *structural* requirements. Technical requirements are necessary for the serialization of our model into a single XML document, and for the ability to serialize the subset of a service description that pertains to a particular module. Structural requirements make the concrete syntax simpler and easier to read.

Technical Requirement 1:

The model to be serialized must define a single root element. The reason is that XML is a textual representation that is tree-based, whereas Ecore models are graph-based. From the root, navigation to all other elements that are part of the tree is possible.

Technical Requirement 2:

Each model class that is not the root element for a module must be contained by another model class. This implies that there is always a navigable path from the root element class to all other classes. This path must be available using Ecore containment references only. XMI already allows for XML path navigation to subclasses of any class that is already reachable from the root element by containment. Any class navigable from such subclasses is also transitively considered navigable from the root class, and its XMI serialization will form a correct nested element set.

Technical Requirement 3:

Non-containment references must be mapped by some textual linking mechanism. Otherwise already contained XML elements cannot be reused but have to be duplicated which would destroy the well-formed structure of the concrete model instance.

Different XML specifications provide different mechanisms for referencing other elements, such as XML Schema's ref mechanism or RDF's URIref mechanism. In general, there are three strategies to handle references. The first strategy identifies XML elements by their location in the XML subtree (often referred to as URI fragments), the second uses unique characteristics of an XML element such as an attribute with unique values, and the third introduces dedicated unique identifier attributes which are added to the XML element. The first two strategies have one significant disadvantage: If the identifier changes, the references in the document are broken. The third strategy on the other hand has no need to change identifiers even if the document structure or attribute values change. However, the challenge in using the third strategy is to generate unique identifiers across multiple documents.

Structural Requirement 1:

If a model consists of several modules (or packages in MOF terminology), each module should define a module root element. This allows for the definition of documents containing contents from only a single module, e.g., a USDL fragment containing only the pricing model could be serialized and exchanged.

Structural Requirement 2:

Model classes that are referenced (by non-containment references) should be top elements contained directly by the document root. The reason for this requirement is that referenced elements are reusable: once defined they can be referenced from different parts of the XML document. When enforcing this requirement, the semantics of the model should be taken into account, i.e., there may be reasons to not strictly follow this requirement.

Structural Requirement 3:

Imported elements should be part of the serialization model. There is no theoretical reason for this requirement, because elements that are referenced by URI in other documents can be located. However, it should be obvious and directly visible which documents need to be accessed to resolve all references. Unless we define an import element in the model which maps to an import schema for the document, all references will have to be checked and document dependencies calculated. This is why XML-based specifications such as WSDL and XML Schema introduce dedicated import elements.

14.2.2 The USDL Serialization Model

To allow the serialization into an XML-based syntax, a few extensions to the abstract USDL meta-model are needed. These extensions are used to address the above requirements and result in a concrete USDL meta-model.

We introduce two variants for the concrete USDL meta-model: a lightweight model that fulfills the technical requirements and a full-fledged model that also fulfills the structural requirements. The lightweight version is easily adaptable for tool builders since it directly builds upon available XMI tools. The full-fledged model also needs extensions in each module as we explain below.

14.2.2.1 Lightweight USDL Meta-model

Technical Requirement 1 has been addressed by the introduction of an additional root class in Ecore — the USDL3Document — which has a containment reference to all other classes either directly or indirectly. This class is in a Serialization Module which imports the other modules and makes references to classes in other modules.

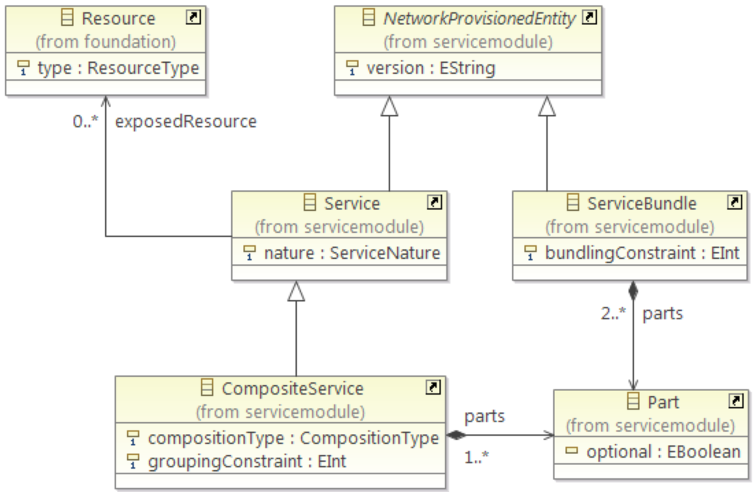


Fig. 14.2: Excerpt from the USDL meta-model.

Technical Requirement 2 can be met by defining references from the root element USDL3Document which contain either (i) all non-contained, concrete classes or (ii) all non-inheriting, non-contained classes.

The definition of the root class and its module can either be achieved manually or derived automatically by applying a model transformation to the standard USDL meta-model. Take for example the excerpt from the USDL model shown in Figure 14.2. The model shows the interrelations of a few classes. The ServiceBundle and the CompositeService classes contain the class Part, whereas the Resource class is only referenced by Service. CompositeService inherits from Service. Service andServiceBundle inherit from NetworkProvisionedEntity.

Following the approach (i) above (containing all non-contained, concrete classes), the classes that need to be contained by the newly introduced root element in this diagram are Resource, Service, Composite Service and ServiceBundle. Ap-

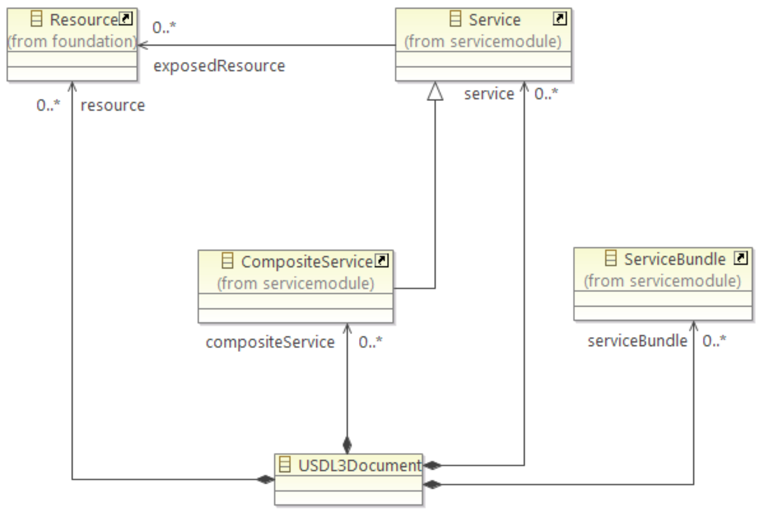


Fig. 14.3: Serialization module for the USDL excerpt.

plying the model transformation to the USDL meta-model excerpt results in the (automatically generated) Serialization module shown in Figure 14.3. The same model transformation can be applied to all USDL modules to create the lightweight serialization module for USDL.

Technical Requirement 3 is already addressed by virtue of the choice of the standard XMI mapping rules (see Section 14.2.4). One advantage of the lightweight model is apparent: The serialization module can simply be *added* to the other Ecore modules without modifying them.

14.2.2.2 Full-fledged USDL Meta-model

A full-fledged approach to USDL Serialization requires changing the standard meta-model to address the structural requirements as well. In order to meet Structural Requirement 1, a root element for each module is introduced which is used as a container for all non-contained classes of a module, e.g., a PricingElements class has been added to the Pricing Module. Technical Requirement 2 is solved in a slightly different way than the lightweight model, namely by adding containment references between the root class USDL3Document and each of the module roots. In addition, containment references are created between all classes that are not already reachable via containments from the USDL3Document root class via each module root class (and thus Structural Requirement 2 is met). For example the PricePlan class is not reachable from the module root, PricingElements, and therefore we add a containment reference from it to PricePlan. Consequently, the PricePlan is now reachable from the USDL3Document by navigating from USDL3Document to PricingElements to PricePlan.

Structural Requirement 3 has been addressed by introducing the Import class. The Import class provides a uri attribute that points to the imported USDL document. This convenience class is especially helpful for tools to preload referenced documents in order to navigate cross-document references.

The full-fledged USDL meta-model has the advantage that structural requirements are fulfilled and thus that the readability of the resulting USDL is better due to a better separation of the different modules inside the XML document (or documents — if serialized per module). A major disadvantage is that the existing USDL modules have to be extended with module root elements, such as PriceElements. Thus, the abstract syntax of USDL has to be made more complex in order to make the concrete syntax easier to understand.

14.2.3 Serialization Model

The Serialization model that is used for USDL (Version 3, Milestone 5) is a lightweight one with additional support of an Import mechanism. Figure 14.4 depicts the Serialization model in that module.

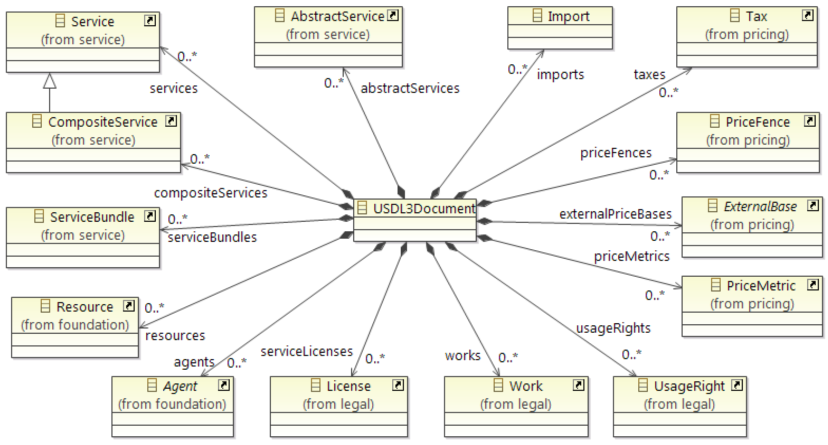


Fig. 14.4: Serialization module for USDL (Milestone 5).

14.2.4 Mapping the USDL Serialization Model to XML

After defining a Serialization Module — which meets our technical (and structural) requirements — this model needs to be mapped to XML. The mapping is based on the default Ecore serialization mechanism which itself is based on XMI. Hence, a short introduction to XMI follows.

The XML Metadata Interchange (XMI) specification (V2.0 was used for Ecore) defines a standard for exchanging any type of metadata that is compliant to the Meta Object Facility (MOF). Ecore has been built to comply to a subset the MOF standard, and thus satisfies this requirement. XMI defines XML Schema constructs for the purpose of identification, linking, object type hierarchies, and data typing. These predefined schema element patterns allow for the serialization of instances of any MOF-conformant object-oriented meta-model to XML, including standards such as UML 2.0 and BPMN 2.0.

Table 14.1: Overview of XMI Mapping from Ecore to XML

Ecore	XML
Instance of EClass	XML Element with xmi:id
Instance of EAttribute of Datatype	XML Attribute
Instance of EReference (containment)	Nesting XML Elements
Instance of EReference (non-containment)	XML Ref Attribute
Inheritance — A is a concrete subtype of B	`type` Attribute set to “A” in the XMLElement representing “B”

To reduce the effort of building tools on top of USDL, the default XMI mapping of Ecore to XML has been applied to USDL. A summary of the major parts of this mapping is shown in Table 14.1. This mapping fulfils Technical Requirement 3 since each generated element has an `xmi:id` that can be referenced. Listing 14.1 shows an example XMI serialization of a USDL service model instance, as specified by the Serialization Module and transformed USDL Modules, as documented in the USDL Version 3 Milestone 5 specification.

Listing 14.1: XMI serialization of a USDL service model instance.

```
<usdl3:USDL3Document xmlns:xmi="http://www.omg.org/XMI" ...>

 <services xmi:id="Service_178" version="1.0" nature="Manual" ...>

  <names xmi:id="Description_389"
      value="Lead Logistics - General Freight" type="name" />

  <provider xmi:id="Provider_2562" enactingAgent="Organization_432"/>

 </services>

 <agents xsi:type="foundation:Organization" xmi:id="Organization_432">

  ...

  </agents>

</usdl3:USDL3Document>
```

The `USDL3Document` is the root element. It may contain several service description which are in turn represented by a `services` element (named after the containment reference from the USDL3Document class to the Service class). The `services` element furthermore has an `xmi:id` attribute that allows other elements to reference the `services` element. The nature of the service is a simple EAttribute and therefore mapped to an XML attribute. Names are contained by the Service class and therefore realized by nesting the `names` element inside it. The Provider has an enacting agent. This is realized by including an `agents` element with type `Organization` and using the Organization's `xmi:id`.

It is common practice for Ecore modelers to apply plural nouns for representing multi-valued containment references, such as, "services," but when mapped to XML, this results in multiple nested elements, each of which is named after the reference. The resulting XML would read better with singular names — such as "service," but this would require the adaptation of the default XMI mapping.

14.2.5 Serialization and Exchange of USDL Models Using SML

The Service Modeling Language (SML) [21], which is a W3C recommendation, provides some useful tools for representing and exchanging USDL models. SML is not a domain language itself, i.e., it does not define the domain entities. However, SML does provide useful constructs for representing complex service descriptions. In addition, the corresponding SML Interchange Format (SML-IF) [20] provides a convenient and standardized way to represent and exchange self-contained USDL models.

14.2.5.1 SML Models

An SML model is a set of interrelated documents that describe a service (or other domain) model. This set of documents consists of model definition documents and model instance documents. The model definition documents contain information about the service (the service model), as well as constraints on the model that must be satisfied for the service to function properly. The model instance documents contain the data for the modeled instances.

There are two types of model definition documents: schema documents and rule documents. The model definition documents provide much of the information a model validator needs to decide whether a given model is valid. Schema documents define constraints on the structure and content of the instance documents in a model. SML uses XML Schema as the schema language and defines a set of extensions to

XML Schema to support references that may cross document boundaries. Rules are boolean expressions that additionally constrain the structure and content of documents in a model in ways that may not expressible within a given XML Schema. SML uses Schematron [13] and XPath [7] for rules. While the rules can be embedded in SML model schema documents, they can also be placed in separate documents so that the schema documents themselves are not altered.

SML schema documents are defined as a strict superset of XML Schema. All valid XML Schema documents are valid SML schema documents. SML does define an extension of XML Schema, namely the SML references. SML references are used to link from one element in a model to another element in the same model. The linked elements may reside in separate XML documents at runtime. In addition, SML references may be constrained to specific elements or element types. Extending the Ecore default mapping by defining the mapping of non-containment references to SML references instead of XML Ref attributes enhances the cognitive sufficiency of the USDL model by making the reference endpoint more explicit while still satisfying Technical Requirement 3. However, it is worth noting that, as long as the Schema extensions defined by SML are not used, then SML model documents can still be processed by currently available XML processors.

SML model instance documents are XML documents that together form a service's description (instance). They describe or support the description of the individual resources that the model portrays and must conform to the structure and constraints as defined in the model definition documents, as shown in Figure 14.5.

14.2.5.2 SML Interchange Format

The SML-IF specification defines a standard interchange format that preserves the content and interrelationships among the model documents. It also defines a constrained form of model validation to ensure interoperability when specific conditions are met and to increase the likelihood of interoperability in other cases. But, at a minimum, the SML-IF interchange format provides a well-defined standard for exchanging a set of model documents regardless of the validation process. An SML-IF document packages the set of SML model documents to be interchanged as a single XML document. Each model document appears as content in either the 'definitions' or 'instances' subsection of the SML-IF document, depending on whether the model document in question is a model definition document or a model instance document. Each model document can be represented in either of two ways, by embedding its content or by providing a reference to it.

14.2.5.3 USDL Models as SML Models

As mentioned, SML is agnostic of any domain model such as a service description model. USDL provides such a domain model. As specified previously in this chapter, USDL Version 3 Milestone 5 defines a concrete representation of the USDL

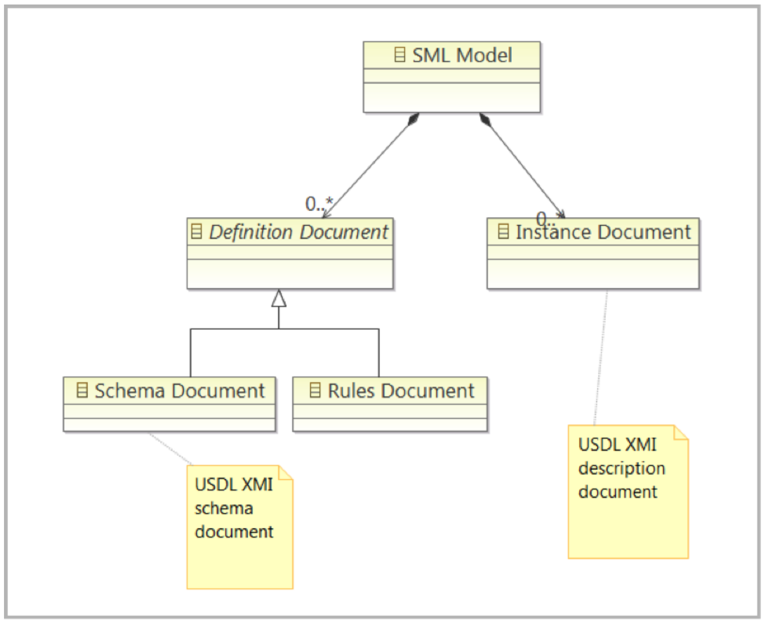


Fig. 14.5: The SML document structure.

model as an XMI schema and a specific model instance as an XMI document that validates to that schema. These documents form a valid SML model as described here and can be packaged into one SML-IF document. Note that in the case of full-fledged USDL serialization, a serialization may be in a single file, or each module may be represented in a separate XML document. In the latter case an SML-IF document provides a convenient way to package multiple documents into one deliverable XML document. The value of SML-IF is that a single USDL model made of multiple schema and instance documents can easily be passed around as a coherent whole thus satisfying the requirement of cognitive sufficiency.

The user could also manually expand the model description beyond the USDL meta-model to formalize specific business rules as SML rule documents. This can be done without altering the XMI schema for USDL. SML-IF provides a way for the USDL model to incorporate this additional model definition document. In addition, SML provides the capability to link references across schema documents which would assist in the implementation of Structural Requirement 1 and the full-fledged USDL serialization model by enabling cross document links. However, this capability would require an SML processor rather than a standard XML processor and, therefore, this capability comes at an additional cost of implementation.

14.3 Representing USDL as Linked Data

The W3C defines the Web as "an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (URI)" [14]. The Web is based on three main aspects, namely the identification of resources, enabling the interaction between agents (software or humans) via well-defined protocols, and formats that govern the representation of data and resources transmitted.

These principles have effectively governed the Web and still maintain the ability for extension to cope with new kinds of resources, or to enable more complex activities to be carried out. A good example is the work carried out on the Semantic Web towards providing machine interpretable semantic descriptions of resources, which could pave the way for the development of more intelligent agents. Most relevant is the use of RDF and OWL, which are based on pre-existing Web standards, to define domain-specific models of concepts in an effective and extensible manner.

The Linked Data principles were suggested[3] by Tim Berners-Lee in 2006 as a means of creating a Web of Data better suited for machine processing. These principles recommend that one should:

1. use URIs as names for things,
2. use HTTP URIs so that people can look up those names,
3. provide useful information, using the standards (RDF, SPARQL) when someone looks up a URI,
4. include links to other URIs so that they can discover more things.

Since these principles were proposed we have witnessed an outstanding growth in terms of data and vocabularies allowing people to freely expose and interlink large quantities of heterogeneous data. In fact, for raw data that can effectively be modeled in RDF, Linked Data principles are considered by well cited authors [4] as the best means for publishing to the Web.

The first principle ensures that resources are uniquely identified. The second principle ensures that their identification is such that HTTP can be used for obtaining information about the resource. The third principle establishes standard technologies for exposing data in a manner that is suitable for machine processing. Finally, the fourth principle aims to promote the interlinking of data. The same way hyperlinks connect Web documents into a single global information space, Linked Data uses hyperlinks to connect data into a single global data space. These links allow applications to navigate the Web of Data and, since the data is exposed through HTTP (see principle 2) and represented in some standard format (see principle 3), machines can obtain it, interpret it, and act accordingly in an automated manner.

USDL was originally modeled in Ecore and integrates a number of different perspectives on services (e.g., pricing, technical details, legal aspects, stakeholders in service provision, etc). In addition, the Pricing and Legal Modules were also modeled ontologically in OWL. In [15] Semantic Web tools are used to create and man-

[3] `http://www.w3.org/DesignIssues/LinkedData.html`

age instances of pricing plans for billing purposes. In work related to the USDL Legal and Service Level Modules [3] the German copyright legislation was modeled as an ontology from which licence rights models can be derived to describe the conditions for use of copyright material provided by a service. In the remainder of this section we shall thus focus on how all of USDL can adopt linked data principles, focusing mainly on the representation of USDL in RDF(S), and on the interlinking with external vocabularies and data sources.

14.3.1 Linked USDL

Creating a linked-data-ready version of USDL involves modeling USDL data and thus the USDL language (meta-model) itself in RDF(S) [5] or related standards, such as RDFa [1] and OWL [8]. USDL is composed of a number of modules some of which started out as Ecore models only, and need to be re-modeled in RDF(S)/OWL accordingly. Redescribing the whole USDL model again in the form of ontologies is beyond the scope of this book, so instead we shall focus on the main design decisions concerning the lightweight semantic representation of USDL in RDF(S)/OWL. Then some examples targeted at the reuse of existing vocabularies and instances are considered. Through this exercise it shall be seen not only that existing vocabularies cover a good part of USDL, but also that modeling USDL in this manner has a number of benefits from the use of Semantic Web tools and formalisms (e.g., temporal reasoning) and from compatibility with existing datasets.

14.3.1.1 Integrating USDL in the Web of Data Through Reuse

The fourth Linked Data principle is to include links to other data sources from the Web. These links are an essential means of generating a Web of Data as opposed to disconnected silos. Linked data simplifies data integration and interpretation, as well as enabling the discovery of related data that is not part of a USDL service description.

Most often there are three kinds of links contemplated [9]:

Relationship Links whereby entities from a data source are linked to entities from other data sources through relationships. For instance, *Service A* stored in a USDL repository can be described as being provided by *Company C* defined in an external *Companies Catalogue*. This type of link enables the reuse of data and establishes links across datasets.

Identity Links which indicate that two URIs refer to the same entity. This allows us to state, for instance, that *Company C* from the previous catalogue is actually the same as *Company X*, rated by customers in a certain social Web site. By means of these links, machines can incorporate different views about the same entity based on data coming from diverse sources.

Vocabulary Links which are established between entities and the vocabularies used to define them, thus allow machines to retrieve the definitions and interpret them. For instance, knowing that *Service A* is a ServiceBundle informs us that it has other *Services* as part of it. These links also allow us to indicate relationships (e.g., equivalence, subsumption) between concepts, and re-use relationships defined in different vocabularies. This helps integrate data from diverse datasets.

A fundamental activity in adapting USDL for the Web of Data therefore concerns the analysis of existing vocabularies and datasets, in order to i) identify reusable vocabularies to avoid reinventing the wheel and promote reuse and integration; and ii) identify possible relationships with USDL concepts and USDL data to support navigation across datasets and to simplify data integration.

14.3.2 Design Decisions

In this section we introduce some of the main decisions that have been adopted while creating Linked USDL. We first introduce general modeling decisions and we then cover choices made concerning capturing information of particular kinds, such as geospatial and temporal.

14.3.2.1 Classifications and SKOS Schemes

The main purpose of the USDL specification is to provide a schema, or type system, which defines the contents and structure of concrete service descriptions, e.g., *Service A*, its kind, e.g., *Service*, and a certain categorization of the Service kind, e.g., an *Automated Service*. When defined in terms of ontologies the USDL Ecore model could be replicated using sub-classing and meta-modeling, however, the use of a hierarchy of classes essentially establishes explicitly and *a priori* the subclasses available. Some parts of the model, however, consist of a set of enumerated values which act as classification categories, and are not expressed as class hierarchies.

The conceptual distinction introduced by a hierarchy of classes is sometimes appropriate as is the case for instance when capturing the relationship between *Service* and *Composite Service*, in which the latter has a grouping constraint which makes no sense for *Services* in general. However, in the Participants module, there are many different subtypes of Role, including BusinessOwner, Provider and Stakeholder, which do not introduce any extra data values or structural constraints. We consider that these subtypes would be better expressed as simple categories.

As a general decision for Linked USDL, we have captured the relationship between concepts via subsumption relationships whenever there was a structural and semantic difference like in the case of Service and CompositeService. For cases like the nature of services which are represented as an enumeration of values in Ecore, we use Simple Knowledge Organization System (SKOS) [18] schemes for defining the different categories. SKOS is a common lightweight data model represented in RDF, which supports the capture of knowledge organization systems such as classification schemes and taxonomies. Using SKOS, categorizations can be represented in a machine processable manner. We can easily integrate different perspectives simply by changing the definition of the concept categorized by means of a property whose range is *skos:Concept*. In cases where the USDL specification uses subclassing to introduce categories, such as the *Role* example above, we can also use SKOS, and apply simple taxonomies to the concept of agents, rather than having a fixed set of subsumed concepts. Indeed, this mechanism does not prevent users of USDL from providing their own vertical domain-specific categorizations through subsumption if they wish to.

14.3.2.2 Types as Properties

The USDL specification defines some kinds of things using a property indicating the type (e.g., dependencyType) and others by using a hierarchy of concepts (see for instance the different Roles above). In RDF(S) both these approaches could be best modeled using several properties, possibly in a hierarchy. For example, in the case of Dependency and DependencyType, where the relationship is binary, RDF properties are the natural choice as they allow capture of relationships between the properties where necessary. In this case we have therefore modeled all the DependencyTypes as properties, we have defined their range accordingly, and we have dropped the concept DependencyTarget since it becomes redundant with this modeling approach. We have thus defined a top level property dependsOn and a number of sub-properties including requires, includes, mirrors, etc.

14.3.2.3 Partonomy

Part-whole relations are very common structuring primitives of the universe, and indeed they are represented in the USDL Ecore model by containment references (black diamonds in the visual representation). For example, ServiceBundle and CompositeService have a number of constituent parts which can be either AbstractServices or NetworkProvisionedEntities. The existence of a part or parts in the model can be specified as a cardinality, or range of numbers, which if the lower bound is zero is *optional*, or if the lower bound is some other number, then that number of parts is *mandatory*. In the case of CompositeServices the Ecore has no way of directly specifying the CompositionType, and so an enumerated value in an attribute of Services specifies whether the sub-services are *data dependent*, *ordered*, or just an *aggregation*.

RDFS and OWL do not have specific construct for modeling part-whole relations but there are however a number of general purpose proposals for capturing these. The reader is referred to [23] and [24]. For the purposes of USDL we adopted the latter as it helps define both direct and transitive relations. We thus include hasPartTransitive as a transitive relation and hasPart, a sub-property of hasPart-

Transitive, as a normal property. Doing so allows us to simply capture the hasPart relations and if necessary be able to deduce the existence of transitive containment relationships automatically through reasoning over hasPartTransitive.

Additionally, as indicated earlier, USDL constrains the cardinality of parts, with the most common being single-valued, either mandatory or optional. These different kinds are captured through a hierarchy of properties refining hasPart and hasPartTransitive respectively. Notably we have included hasOptionalPart and hasOptionalPartTransitive as well as hasMandatoryPartTransitive and hasMandatoryPartTransitive. The current version does not insert the inverse relation isPartOf, but, should it be necessary, this would be an easy addition.

14.3.2.4 Agents and Roles

Given that the provisioning of services necessarily involves a number of individuals or organizations taking part, USDL provides a number of classes and relations covering this. In particular, Agent represents all the entities that can take active part in the provisioning of a Service. USDL identifies Organization, NaturalPerson, and ResourceAgent as the main kinds of Agents. This term appears in a number of vocabularies, notably in Dublin Core,[4] and FOAF [6] to name the main ones. The notion of Agent also concerns organizations which are covered in other vocabularies, for example by *gr:BusinessEntity* in GoodRelations [10].

Closely related to the notion of Agent, USDL includes the notion of Role. Role serves as the super type of all concrete USDL classes that represent roles found in a service network (e.g., service provider). Agents participating in the provisioning and delivery of a service perform distinct functions, which define their Role. Roles may either be bound to a concrete Agent or may be used as placeholders. The latter is necessary if the Service is in a stage where some Agents are yet to be determined. For example, a service description may specify that there needs to be a B2B gateway in order to deliver the service to a consumer. Which gateway provider will be chosen, however, depends on the message/interface standards supported by a consumer and the consumer's preferences.

In order to maximize reuse and integration across vocabularies we have adopted Reynolds' organization ontology [25], which covers all the core notions, provides basic modeling constructs for Roles and is already integrated with existing vocabularies such as FOAF and GoodRelations. The main design decisions in this ontology are i) capturing the relationship between Agents and Roles played in an organization through an n-ary relationship represented by the intermediate class Membership, and ii) integration with GoodRelations, FOAF and the vCard vocabulary [12]. By means of these alignments we enable the re-use of many FOAF profiles, GoodRelations descriptions, and other existing Web data.

[4] `http://dublincore.org/`

14.3.2.5 Geospatial Modeling

One aspect of the USDL Foundation module concerns location-related entities and relationships. In particular USDL specifies a super type for all location related entities, namely Location, and the subtypes PhysicalLocation, GeographicalPoint, PhysicalAddress, AdministrativeArea, and many others.

There has been a considerable amount of work devoted to creating ontologies and services in this area. Currently, perhaps the most reused vocabulary for geographic concepts is the W3C Basic Geo vocabulary, which facilitates the capture of GeographicalPoints on the basis of their latitude, longitude and altitude. In addition to this effort, the W3C Geo Incubator Group also devoted some effort to creating a simple and reusable vocabulary [16] for capturing some basic geometry relations, which we have adopted in Linked USDL.

There is also a range of complementary vocabularies, data sources, and services available on the Web with which it would be interesting to integrate for data reuse. It is worth noting the work by several organizations: firstly, the UK Ordnance Survey[5] as part of the data.gov.uk initiative for the public release of a large quantity of governmental data in the UK; the site Geonames.org, which comes with a large knowledge base of locations and services for accessing it; and the geospatial data set authored by the FAO.[6]

14.3.2.6 Temporal Modeling and Reasoning

USDL includes quite a few classes for representing time, including Time Instant, Time Interval, etc. Time representation and reasoning has been addressed quite often by researchers. Indeed, Semantic Web researchers already have several works on time representation. In particular, perhaps the most popular for Linked Data is OWL Time [11] which is hosted by W3C.

Time Ontology defines temporal entities and temporal relationships based on James Allen's interval temporal algebra [2]. It therefore identifies Instants, defines Intervals on the basis of beginning and end Instants and includes the typical temporal relationships between Instants and between Intervals (e.g., before, during, etc). Linked USDL supports the capture of most of the temporal aspects of USDL using OWL Time, and additionally supports the implementation of Allen's interval temporal algebra for reasoning about intervals and instants. Some issues like the precision of OWL Time (currently limited to seconds) and notions such as Recurrent Time and Time Pattern still need to be addressed.

[5] `http://www.ordnancesurvey.co.uk/`

[6] `http://www.fao.org/countryprofiles/geoinfo.asp?lang=en`

14.3.3 Services and Service Vocabularies

Modeling the central notion of Service in USDL with Linked Data did not require any particular decisions other than those mentioned above. However, we carefully reviewed the state of the art in service ontologies and vocabularies in order to identify the main alignments to be addressed. The main vocabularies used can be divided into those that address business aspects of services, such as e3Service [27] and GoodRelations, and those that tackle the technical aspects of services, for example, OWL-S [17], WSMO-Lite [28], and the Minimal Service Model [22].

In the current version of Linked USDL, we have performed the following alignments with GoodRelations since it is the most widely used vocabulary for (business) services on the Web:

- AbstractService is a subclass of *gr:Product Or Service Model* since it provides prototypical definitions of Services.
- Service and CompositeService are subclasses of *gr:Product Or Services Some Instances Placeholder* as they both identify a placeholder for instances of a service.
- The inter-service relationship enhances is equivalent to *gr:addOn*.

Possible additional alignments could be carried out with the e3 family of ontologies. However, at this stage these ontologies are not offered publicly on the Web in a resolvable manner which is a requirement for Linked Data.

14.3.4 Summary of USDL as Linked Data

This section has outlined the approach to mapping USDL to existing Linked Data and the Semantic Web resources. The structural specification of the USDL metamodel replicates a lot of vocabularies and relations specified in ontologies, for which there are meaningful and interlinked instances available on the Web through Linked Data. In fact the original form of some parts of USDL, notably the Pricing and Legal modules, were as OWL specifications, to facilitate the use of existing tools to create and manipulate service descriptions. By drawing equivalences between classes, attributes and relationships in the USDL specification with existing vocabularies, relations, and stores of data in the Linked Data Web, we can re-use both common concepts and instances of these which already exist, and use a broad range of tools for reasoning about the contents of USDL service descriptions.

14.4 USDL Documentation Generation using USDL-Doc

USDL service descriptions are a convenient way to capture various aspects of a service (e.g., technical, legal or operational aspects) in a structured manner which allows for further automatic processing. However, raw formats embedding structuring tags (e.g., XMI) typically fall short of providing a decent human-readable description. Therefore, we have implemented the USDL-Doc tool capable of transforming USDL service descriptions into HTML or PDF documents.

14.4.1 USDL-Doc Architecture

The USDL-Doc tool is a simplistic but powerful USDL editor add-on dedicated to mapping existing USDL service descriptions, stored as EMF models in XMI format, to HTML or PDF documents. Generated HTML/PDF documents may serve as service detail pages exhibited on the service marketplace, developer documentation, etc.

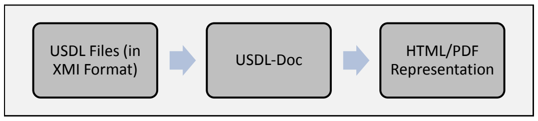


Fig. 14.6: Document Generation Workflow.

The document generation process is fully automated and follows the workflow depicted in Figure 14.6. Existing USDL service descriptions persisted in the XMI format are processed by the USDL-Doc tool which eventually produces HTML or PDF files.

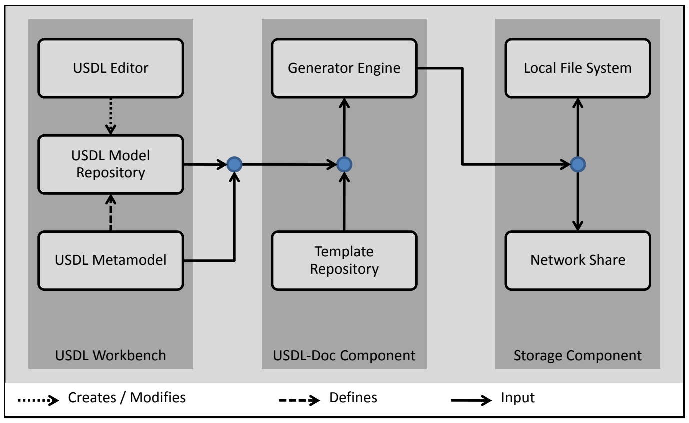


Fig. 14.7: Document Generation Platform Architecture.

In order to provide a fully automated workflow, the architecture illustrated in Figure 14.7 was devised. Essentially, there are three major building blocks: (i) the USDL workbench, (ii) the USDL-Doc component and (iii) the storage component. The USDL workbench is used by the USDL designer when creating and revising USDL descriptions by means of the USDL editor. The USDL editor creates USDL models which are compliant to the USDL meta-model. In essence, the USDL meta-model provides a fixed language vocabulary which can be combined to form arbitrary USDL models. Those models are stored in a dedicated USDL model repository which may be implemented in any form from a complex shared repository to a folder of XMI files on the local file system.

After USDL services are completely described, the USDL-Doc component may be triggered to produce self-contained HTML or PDF documents. The USDL-Doc component derives the documents taking into account the USDL service description instance itself as well as the associated USDL meta-model. Therefore, the generator engine — having access to the model and meta-model — can query the model in a declarative style. For example, a declarative query could ask for all natural persons belonging to a specific organization where the classes NaturalPerson and Organization are part of the USDL meta-model, and therefore nouns in language vocabulary implied by the meta-model. The template language to express these declarative queries is the Xpand language. Besides specifying dynamic queries, Xpand templates can also include static code blocks (e.g., HTML code). Hence, Xpand serves as a flexible code generation language targeting any kind of textual generated code (e.g., HTML, Java, C). The Xpand template language, together with the Xpand editor, is bundled in the openArchitectureWare (oAW) framework that also provides a generator engine executing the Xpand templates. In summary, the USDL-Doc component leverages the oAW template language Xpand and the oAW text generation engine to transform USDL models into HTML or PDF files.

These generated files are consequently transferred to the storage component. The storage component may put files into a variety of configured storage locations (including the file system, network shares and databases). Thus, generated documents can easily be shared and consumed.

14.4.2 HTML Generation Example using USDL-Doc

The following example will expose the technical details of the documentation generation workflow. Therefore, we have chosen a minimal example that illustrates all relevant aspects.

Let's assume we want to model a new organization and all of its representatives. The USDL meta-model already provides concepts representing people and organizations and their properties and relationships. Figure 14.8 shows a small fragment of the USDL meta-model (residing in the Participant module) depicting the classes Organization, NaturalPerson and the reference representatives. Consequently, an arbitrary USDL editor may instantiate these classes to create an object instance

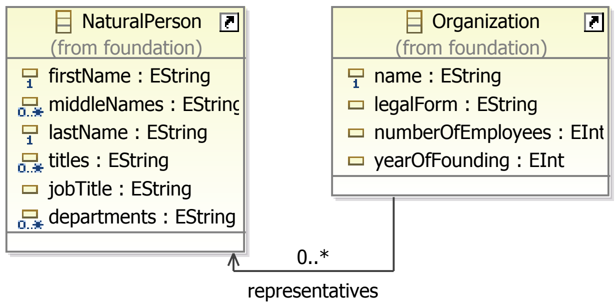
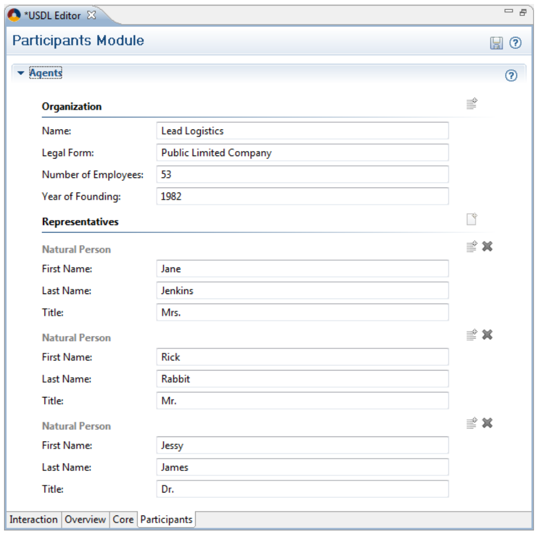


Fig. 14.8: Fragment of the USDL meta-model.

representing a new organization, its properties, and objects representing its associated people. Figure 14.9 shows such a USDL editor where the organization Lead Logistics (cf. running example in Section 8.7) is established and multiple natural persons are the associated representatives.



Organization

Name:	Lead Logistics
Legal Form:	Public Limited Company
Number of Employees:	53
Year of Founding:	1982

Representatives

Natural Person

First Name:	Jane
Last Name:	Jenkins
Title:	Mrs.

Natural Person

First Name:	Rick
Last Name:	Rabbit
Title:	Mr.

Natural Person

First Name:	Jessy
Last Name:	James
Title:	Dr.

Fig. 14.9: USDL Example Model.

In order to advertise the service specified in USDL, we can deploy the service description we have created in the editor to a broker of services in a service marketplace in XMI format. We also might want to offer the information about the service on a Web page to inform potential customers. Therefore, we initiate an USDL-to-HTML conversion by invoking the USDL-Doc tool. The tool processes the USDL service description file, along with the USDL meta-model, and uses the Xpand template to produce a human-readable representation.

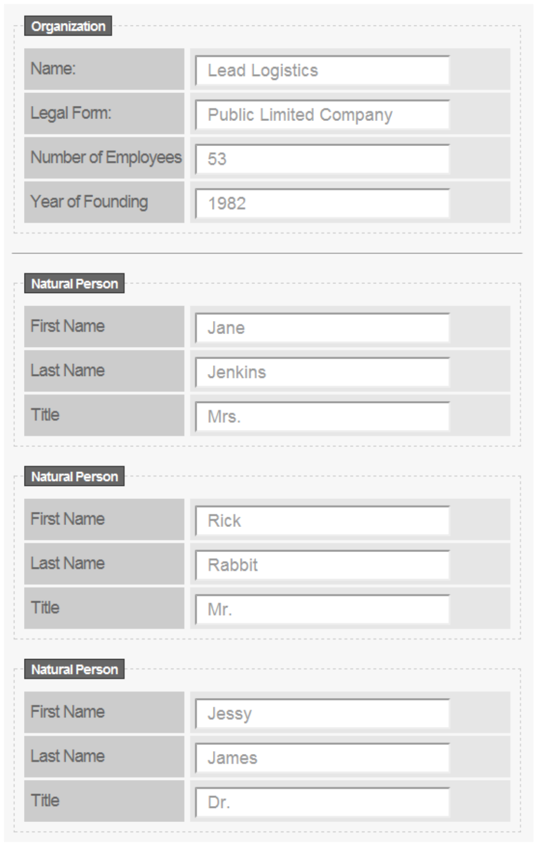
```
«IMPORT usdl»
...
«DEFINE expandOrganization FOR Organization»
    <html>
    <body>
    <div>
    <p>organization.name</p>
    <p>organization.legalForm</p>
    <p>organization.numberOfEmployees</p>
    <p>organization.yearOfFounding</p>
    </div>
    «EXPAND expandNaturalPerson FOREACH representatives»
    </body>
    </html>
«ENDDEFINE»

«DEFINE expandNaturalPerson FOR NaturalPerson»
    <div>
    <p>naturalPerson.firstName</p>
    <p>naturalPerson.lastName</p>
    <p>naturalPerson.title</p>
    </div>
«ENDDEFINE»
...
```

Fig. 14.10: Xpand Code Generation Template.

An example Xpand template is shown in figure 14.10. It defines the rules (e.g., `expandOrganization`, `expandNaturalPerson`) for how to map USDL model elements to HTML code. While the bold black font denotes dynamic code generation, the lighter font expresses static HTML code. To have a means to access the USDL model elements, the template has to declare the available types by importing the USDL meta-model. Once the generator engine is aware of the USDL types, various typed rules might be defined. In the example in Figure 14.10 there are two rules specified: `expandOrganization` and `expandNaturalPerson`. These rules are applied to all organizations and all natural persons defined in the USDL model. The `expandOrganization` rule actually creates a new HTML file and prints the organization properties to a div-block. Moreover, within the *expandOrganization* rule an `expandNaturalPerson` rule is called creating a dedicated div-block for each person. Note that Xpand language is capable of (i) defining declarative rules for specific types, (ii) accessing element properties using the dot-operator and (iii) navigating through the model to discover linked objects via references (e.g., representatives).

Finally, the generator engine processing the Xpand template creates an HTML page like the one depicted in Figure 14.11. The illustrated HTML generation is also feasible for PDF documents. It merely requires a PDF-specific Xpand template.



Name:	Lead Logistics
Legal Form:	Public Limited Company
Number of Employees	53
Year of Founding	1982

First Name	Jane
Last Name	Jenkins
Title	Mrs.

First Name	Rick
Last Name	Rabbit
Title	Mr.

First Name	Jessy
Last Name	James
Title	Dr.

Fig. 14.11: Generated HTML Page.

14.4.3 Summary of USDL-Doc

In this section, we have demonstrated the USDL-Doc tool which transforms USDL service descriptions into HTML or PDF documents. The resulting documents may address various needs of service providers such as providing developer documentation or marketing material.

14.5 Conclusion

This chapter has explained how tools can use the USDL meta-model to make representations of concrete service descriptions for use by humans and tools.

Firstly, we can see that additional meta-model machinery is needed to allow tree-based textual serialization of the USDL's Ecore models designed for describing the structure and constraints of a service description. The main purpose is for model interchange between tools, but human readability is also considered. A simple structural containment is introduced to allow tools using the XMI specification to create valid XML documents for whole service descriptions, and for fragments from particular modules. However, an additional class for an *import* mechanism are also used to facilitate the easy tracing of the documents in which parts of a USDL model are located. The use of SML is and its interchange format SML-IF are also discussed. This framework facilitates the packaging together of a coherent set of XML Schema and instance documents, the cross-linking of elements between these documents, and the potential to validate additional constraints that cannot be expressed via XMI.

The integration of USDL service descriptions within existing vocabularies and ontologies in the Semantic Web space has been explored in Section 14.3. The technologies that make up the Semantic Web have more flexible kinds of relationships between concepts than the minimal object-oriented typing of the MOF, and some of these are considered to maximize the ability to match concepts in USDL within a larger Linked Data ecosystem. The other idea put forward is that an initial effort to match the concepts in the USDL meta-model with similar concepts in existing ontologies will allow the USDL service descriptions available in a Semantic Web context to be linked across domains such as Agents and Geospatial data, and to be manipulated by a range of reasoning tools.

Finally, we consider the use of USDL-Doc tools which use the USDL meta-model in concert with service description instances to format USDL data for human comprehension and navigation.

References

1. B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and Processing. `http://www.w3.org/TR/rdfa-syntax/`, October 2008.
2. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
3. C. Baumann and C. Loës. Formalizing copyright for the internet of services. In G. Kotsis, D. Taniar, E. Pardede, I. Saleh, and I. Khalil, editors, *iiWAS'2010 - The 12th International Conference on Information Integration and Web-based Applications and Services, 8-10 November 2010, Paris, France*, pages 714–721. ACM, 2010.
4. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
5. D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, 2002. `http://www.w3.org/TR/rdf-schema`.
6. D. Brickley and L. Miller. FOAF Vocabulary Specification 0.98. http://xmlns.com/foaf/spec/, August 2010. Last Visited: July 2011.
7. J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. Recommendation, W3C, November 1999.

8. M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. `http://www.w3.org/TR/owl-ref/`, February 2004. Last Visited: March 2005.
9. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*, volume 1 of *Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool, 1st edition edition, 2011.
10. M. Hepp. Goodrelations: An ontology for describing products and services offers on the web. In *16th International Conference on Knowledge Engineering and Knowledge Management (EKAW2008)*, volume 5268 of *LNCS*, pages 332–347, Acitrezza, Italy, October 2008. Springer.
11. J. R. Hobbs and F. Pan. Time Ontology in OWL. Available at `http://www.w3.org/TR/owl-time/`, September 2006.
12. R. Iannella, H. Halpin, R. Iannella, B. Suda, and N. Walsh. Representing vcard objects in rdf. Member submission, W3C, January 2010.
13. ISO. Information technology – Document Schema Definition Languages (DSDL) – Part 3: Rule-based validation – Schematron . ISO/IEC 19757-3, June 2006.
14. I. Jacobs and N. Walsh. Architecture of the world wide web, volume one. Recommendation, W3C, December 2004.
15. T. Kiemes and D. Oberle. Generic modeling and management of price plans in the internet of services. In K.-P. Fähnrich and B. Franczyk, editors, *Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 1, 27.09. - 1.10.2010, Leipzig*, volume 175 of *LNI*, pages 533–538. GI, 2010.
16. J. Lieberman, R. Singh, and C. Goad. W3c geospatial vocabulary. Incubator group report, W3C, October 2007.
17. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. Member submission, W3C, 2004. W3C Member Submission 22 November 2004.
18. A. Miles and S. Bechhofer. SKOS Simple Knowledge Organization System Reference. Recommendation, W3C, August 2009.
19. Object Management Group. Meta Object Facility (MOF) Core Specification Version 2.4. OMG Document No. formal/2008-12-10, December 2008.
20. B. Pandit, V. Popescu, and V. Smith. Service modeling language interchange format, version 1.1. Recommendation, W3C, May 2009.
21. B. Pandit, V. Popescu, and V. Smith. Service modeling language, version 1.1. Recommendation, W3C, May 2009.
22. C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecký, and J. Domingue. iServe: a Linked Services Publishing Platform. In *Proceedings of Ontology Repositories and Editors for the Semantic Web at 7th ESWC*, 2010.
23. V. Presutti. Part whole design pattern.
24. A. Rector, C. Welty, N. Noy, and E. Wallace. Simple part-whole relations in owl ontologies. Editor's draft, W3C, 2005.
25. D. Reynolds. An organization ontology. `http://www.epimorphics.com/public/vocabulary/org.html`, October 2010.
26. Richard Soley. Model Driven Architecture. OMG Document No. formal/2000-11-05, November 2000.
27. P. van Eck, J. Gordijn, and R. Wieringa. Value-based design of collaboration processes for e-commerce. In *2004 IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE 04), 29-31 March 2004, Taipei, Taiwan*, pages 349–358. IEEE Computer Society, 2004.
28. T. Vitvar, J. Kopecky, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In M. Hauswirth, M. Koubarakis, and S. Bechhofer, editors, *Proceedings of the 5th European Semantic Web Conference*, LNCS, Berlin, Heidelberg, June 2008. Springer Verlag.