Data Types in UML and OWL-2

Jesper Zedlitz
Christian-Albrechts-Universität
Kiel, Germany
Email: jze@informatik.uni-kiel.de

Norbert Luttenberger
Christian-Albrechts-Universität
Kiel, Germany
Email: n.luttenberger@email.uni-kiel.de

***Abstract*—Both OWL-2 and UML static class diagrams lend themselves very well for conceptual modeling of complex information systems. To ease the choice between either of these languages it worthwhile to clarify the differences and similarities in the representation of different kinds of datatypes (primitive types, enumerations, complex datatypes, and generalization of datatypes) in static UML data models and OWL-2 ontologies. Where similarities allow a transformation of datatypes from one language into the other, we describe a possible transformation.**

Keywords*—*UML; OWL; datatypes; conceptual modeling

I. INTRODUCTION

Though nowadays the Web Ontology Language (OWL) is mostly considered as a language for knowledge representation, it can also be used as a language for conceptual modeling of complex information systems, i.e., as a language for representing the entities of a certain domain and for expressing the meaning of various, usually ambiguous terms and to identify the relationships between these. In this respect, OWL can be seen as a direct "competitor" to static Unified Modeling Language (UML) class diagrams, which are—e.g., in the ISO 191xx series of standards—often used for this purpose. Known approaches for UML-to-OWL (and reverse) transformations having this "conceptual modeling focus" mostly neglect the subtle problem of datatype mapping, i.e., the mapping of the OWL type system to the UML type system and reverse. This aspect should however not be ignored, especially as OWL-2 comes with an elaborate support for datatype properties. In this paper, we focus on these datatypes: We highlight the different representation of datatypes in UML and OWL-2 and present possibilities and limitation of transforming datatype definitions written in language into the other language.

The paper is organized as follows. We start with an overview of existing approaches for transformations between UML and OWL-2 that take datatypes into account. Section III gives an general overview of different kind of datatypes. The next section shows how these datatypes can be represented in UML and OWL-2. In Section V, we present how datatypes defined in one language can be transformed into the other language. How we use the transformation described before is shown in Section VI. Section VII concludes and points out fields of future work.

II. RELATED WORK

Several publications discuss the relation of UML and OWL in general [1] [2] and transformations of UML class diagrams into OWL ontologies [3] [4]. The revision of OWL-2 made it necessary to rework the transformations from UML to OWL-2 [5] [6]. However, datatypes play only a minor role in these publications. The approaches have in common that UML attributes with a primitive type are transformed into OWL-2 data properties. Enumerations become enumerated datatypes (`owl:oneOf` resp. `DataOneOf`) and vice versa.

Tschirner et al. [7] note that datatypes can be structured. However, this fact and its impact on a transformation is not further discussed in the article. It is noteworthy that—in difference to all other approaches—enumerations are transformed into sets of individuals instead of set of literals.

A special kind of extensible enumeration defined in the ISO 19103 standard "Codelist" and its transformation from UML to OWL-2 has been discussed in [8].

III. INTRODUCTION TO DATATYPES

In general, a datatype consists of three components: the value space, the lexical space, and a well-defined mapping from the lexical into the value space. The value space is the—possible infinite—set of values that can be represented by the datatype. The lexical space describes the syntax of the datatype's values. The mapping is used to map syntactically correct values to elements of the value spaces. It is possible that—even infinite—many syntactically different values are mapped to the same element of the values space.

Primitive datatypes do not have an internal structure. Examples of primitive types are character strings, logical values, and numbers.

Enumerations are a special kind of datatypes with no internal structure. In contrast to general primitive types the lexical space and the value space of an enumeration are equal-sized, well-defined finite sets. The mapping from lexical to value space is a one-to-one mapping. An example for an enumeration datatype are the English names of the days of the week which consist of seven possible values.

In contrast to primitive data types **complex data types** have an internal structure. These are some examples for complex data types:

- a person's name consisting of given name and family name
- a physical measurement consisting of value and unit of measurement
- an address consisting of street name, house number, postal code and city name

Generalization of datatypes can be defined similarly to the generalization of element types. If a datatype A generalizes a datatype B each date that is instance of B (i.e., its lexical representation belongs to the lexical space of B and its value belongs to the value space of B) is also instance of datatype A. For example the integers generalize natural numbers. Each natural number is also an integer.

IV. Representation of Datatypes

A. Unified Modeling Language (UML)

Besides a few pre-defined primitive types UML allows the definition of additional datatypes in class diagrams. These can be primitive types, complex datatypes, and enumerations. In UML, datatypes—similar to classes—can have owned attributes (as well as operations which are not discussed here). Therefore, they can be used to describe structure. Figure 1 shows examples for the three kind of datatypes.

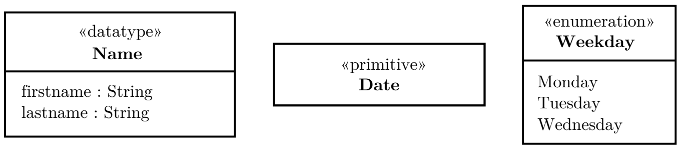


Fig. 1. Examples for datatypes in UML. Left: user-defined datatype with two components. Center: user-defined primitive datatype. Right: Enumeration with three allowed values.

In contrast to instances of classes "any instances of that data type with the same value are considered to be equal instances."[9, p. 63] Although the graphical representations of datatypes in general (instances of *DataType*) as well as primitive types (instances of *PrimitiveType* and enumerations (instances of *Enumeration*) in particular look similar to the representation of classes (instances of *Class*) they are different elements of the meta model as shown in Figure 2.

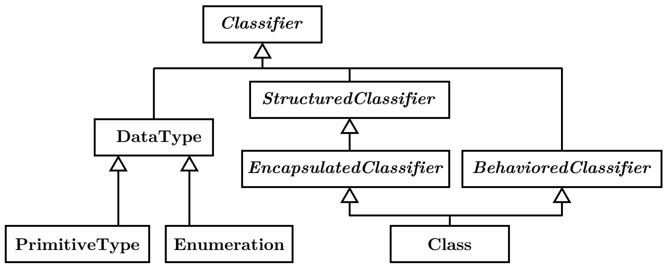


Fig. 2. Extract from the UML meta model, showing the difference between classes and datatypes.

In UML, generalizations are defined for *Classifier* and therefore also for *DataType*. Thus inheritance/generalization relations between datatypes can be defined in a UML class diagram.

B. Web Ontology Language (OWL-2)

In OWL-2 three different kinds of datatypes can be distinguished:

1) `rdfs:Literal` as base datatype
2) datatypes of the OWL-2 datatype map, which is basically a subset of the XML Schema datatypes [10].
3) datatypes that have been defined within an ontology using `DatatypeDefinition`

The value space of the base datatype `rdfs:Literal` is the union of the value spaces of all other datatypes. The OWL-2 datatype map adopts the value space, lexical space, and the restrictions for user-defined datatypes from the XML Schema specification. Sets of values (instances of datatypes)—so called *Data Ranges*—can be defined by combining datatypes via common set-theoretic operations. A set of values consisting exactly of a pre-defined list can be described by using `DataOneOf`. A `DatatypeRestriction` allows to define a set of values by restricting the value space of a datatype with *constraining facets*. The OWL-2 datatype map defines which restrictions are allowed. For example a number datatype can be restricted by: less equal, greater equal, equal, and greater.

An OWL-2 datatype is defined by assigning an Internationalized Resource Identifier (IRI) to a `DataRange` using a `DatatypeDefinition` axiom. According to the OWL-2 DL specification this IRI must have been declared as the name of a datatype.

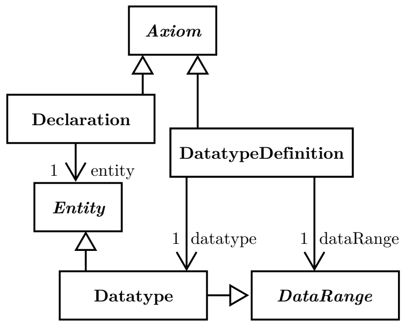


Fig. 3. Extract relevant for datatypes from the OWL-2 meta model.

The abstract syntax (see Figure 3) shows that a datatype is linked indirectly (via an instance of `DatatypeDefinition`) with its value space (an instance of a subclass of `DataRange`. Therefore, it is possible to use a datatype with no assigned values space. By definition this datatype has the value space of `rdfs:Literal`.

Subclasses of `DataRange` (e.g., `DataUnionOf`) which are used for the definition of value sets (and therefore datatypes) have references to `DataRange`. `Datatype` is a subclass of `DataRange`, too. Thus, arbitrarily nested constructions of datatype-defining elements are possible.

V. Transformation of Datatypes

A. Primitive Types

Three cases have to be considered for the UML $\rightarrow$ OWL-2 transformation of primitive types:

1) The datatype is one of the four pre-defined datatypes "Boolean", "Integer", "String", or "UnlimitedNatural".
2) The datatype is one of the XML Schema datatypes.
3) The definition of the (user-defined) datatype is part of the UML-model.

Since OWL-2 uses the datatype-definitions from XML Schema a datatype in case (1) can be transformed into its corresponding datatype from XML Schema. Primitive types can be recognized by the fact that they are contained in a packet “UMLPrimitiveTypes”.

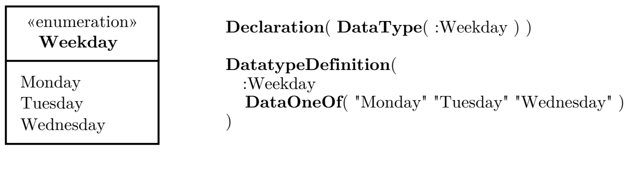
The transformation in case (2) is even more obvious because a datatype is used that is also present in OWL-2. The name of the package containing the primitive types depends on the UML type library used. A common package name is “XMLPrimitiveTypes”. This name can be used to recognize primitive types falling under case (2). The XML Schema datatype can be referenced in the ontology by adding the XSD namespace to the type’s name.

For user-defined datatypes in case (3) a new datatype is defined in the ontology by using a `Datatype` axiom. OWL-2 datatypes—like all OWL-2 model elements—are identified by unique IRIs. Therefore, an appropriate IRI must be generated during the transformation. In UML, elements (including datatypes) are uniquely identified by their name and package hierarchy. Therefore, a combination of package and datatype name can be used for the IRI.

For the transformation OWL-2 $\rightarrow$ UML primitive types are difficult. OWL-2 offers a variety of possibilities to define new datatypes. However, some primitive types—and probably the most common ones—can be transformed. The primitive types of OWL-2 derive from the XML Schema datatypes. There are established UML-libraries for the XML datatypes. Therefore, it is sufficient to include such a library into the transformation process. An instance of a primitive type contained in the library can be looked up by the IRI of the OWL-2-datatype and references as necessary.

B. Enumerations

As mentioned in Section II, several authors have already discussed how to transform enumerations: In OWL-2 the data range `DataOneOf` is suitable for defining a datatype with a fixed pre-defined value space. Each lexical value of the `DataOneOf` data range is transformed into an *EnumerationLiteral* instance and vice-versa. OWL-2 as well as UML support the specification of datatypes for the elements of an enumeration: An OWL-2 *Literal* instance has a *datatype* attribute, an UML *EnumerationLiteral* instance has a *classifier* attribute referencing the datatype.



```
Declaration( DataType( :Weekday ) )

DatatypeDefinition(
  :Weekday
  DataOneOf( "Monday" "Tuesday" "Wednesday" )
)
```

Fig. 4. Example for the transformation of an enumeration.

For the transformation OWL-2 $\rightarrow$ UML one has to consider the fact that in OWL-2 the data range `DataOneOf` can be used without a `DatatypeDefinition` which assigns a name to it. Since an UML *Enumeration* necessarily needs a name it can be generated based on the literals contained in the data range.

C. Complex Data Types

OWL-2 datatypes consist of exactly one literal and are therefore not further structured. Since OWL-2 is built upon the Resource Description Framework (RDF) there is the theoretical possibility to use a blank node and the RDF-instruction `parseType="Resource"` to implement complex data as shown in this listing:

```
<rdf:RDF xml:base="http://example.com/persons/"
   xmlns="http://example.com/persons/"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

   <owl:Ontology rdf:about="http://example.com/persons/"/>

   <owl:Class rdf:about="Person" />

   <owl:NamedIndividual rdf:about="Timmi">
      <rdf:type rdf:resource="Person"/>
      <hasName rdf:parseType="Resource">
         <first>Timmi</first>
         <last>Tester</last>
      </hasName>
   </owl:NamedIndividual>
</rdf:RDF>
```

However, neither the OWL-1 nor the OWL-2 specification mention `parseType="Resource"`. Therefore, it is probably not a valid construct for OWL-2. Even if this notation was valid for OWL-2 and an element type could be assigned to such an anonymous individual, the definition of the element type would be indistinguishable from the definition of a “normal” element type.

The UML $\rightarrow$ OWL-2 transformation of complex datatypes, i.e., datatypes with owned attributes, is similar to the transformation of UML classes with owned attributes into OWL-2 classes and properties. There are two characteristics of UML datatypes that have to be considered:

1) Values do not have an identity.
2) Every value exists only once.

Since the transformation is similar to the transformation of classes the instances of the resulting element in the ontology will be individuals. In OWL-2 every (typed) individual must have a name. Therefore, the semantics for characteristic (1) is changed: In UML, the instance of the datatype does not have an identity. The corresponding individual in OWL-2 is assigned with an IRI by which it can be referenced (and also identified).

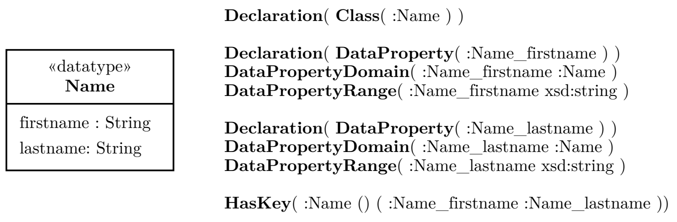
Characteristic (2) requiring that every value must exist not more than once can be ensured by using `HasKey` axioms. For every UML datatype D with owned attributes $a_1 \ldots a_n$ that is transformed into a OWL-2 class C with data property $dp_1 \ldots dp_n$ the following axiom is added to the ontology:

HasKey(C () ($dp_1 \ldots dp_n$))

This axiom ensures that every occurrence of an individual with the same values for $dp_1 \ldots dp_n$ is one and the same individual.

D. Generalization of datatypes

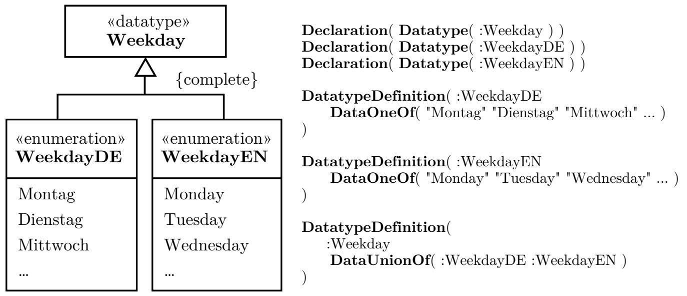
In general, the transformation of a datatype generalization in a UML class diagram is not possible since OWL-2 has no support for inheritance/generalization of datatypes. In the special case of a complete generalization of datatypes with



```
Declaration( Class( :Name ) )

Declaration( DataProperty( :Name_firstname ) )
DataPropertyDomain( :Name_firstname :Name )
DataPropertyRange( :Name_firstname xsd:string )

Declaration( DataProperty( :Name_lastname ) )
DataPropertyDomain( :Name_lastname :Name )
DataPropertyRange( :Name_lastname xsd:string )

HasKey( :Name () ( :Name_firstname :Name_lastname ))
```

Fig. 5. Example for the transformation of a complex datatype.

no internal strcuture (e.g., enumerations) a transformation is possible: While the generalization of UML classes can be transformed into an OWL-2 `ObjectUnionOf` class expression this is not possible for datatypes. As the name suggest, an `ObjectUnionOf` can only be used for classes. Instead an instance of `DataUnionOf` is used. The sub-datatypes combined in the `DataUnionOf` constitute a new data range. Using a `DatatypeDefinition` axiom a name is assigned to this set of datatypes. This name is the name of the super-datatype from UML. Figure 6 shows an example for such a transformation.



```
Declaration( Datatype( :Weekday ) )
Declaration( Datatype( :WeekdayDE ) )
Declaration( Datatype( :WeekdayEN ) )

DatatypeDefinition( :WeekdayDE
    DataOneOf( "Montag" "Dienstag" "Mittwoch" ... )
)

DatatypeDefinition( :WeekdayEN
    DataOneOf( "Monday" "Tuesday" "Wednesday" ... )
)

DatatypeDefinition(
  :Weekday
    DataUnionOf( :WeekdayDE :WeekdayEN )
)
```

Fig. 6. Example for the transformation of a generalization relation between datatypes.

VI. Implementation

We have implemented the transformations presented in this paper as part of two model-to-model transformations written in Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Relations language [11]. In contrast to other approaches working on transfer formats (e.g., XML Metadata Exchange (XMI) and a XML-based syntax for OWL-2) our transformation is specified using the meta-models of UML and OWL-2 only. Therefore, the transformation is independent of any concrete syntax.

QVT-Relations is a declarative model-to-model transformation language. In addition to a textual syntax it also has a visual syntax. An example of the visual syntax is shown in Figure 7. During a transformation execution so called *trace classes* and their instances are automatically created to record what occurred that execution. These characteristics of QVT-Relations make it possible to analyze the transformation easily. Additional to manual/visual automated tests can be performed since the output of one transformation is again a model that can serve as the input of the other transformation.

VII. Conclusion and Future Work

In this paper we have focused on the datatypes of static data models—often neglected when working on transformation between UML and OWL-2. We showed differences and similarities in the representation of datatypes in UML and OWL-2. Where similarities allow a transformation of datatypes from one language into the other we have described a possible transformation and highlighted the tricky points and/or limitations.

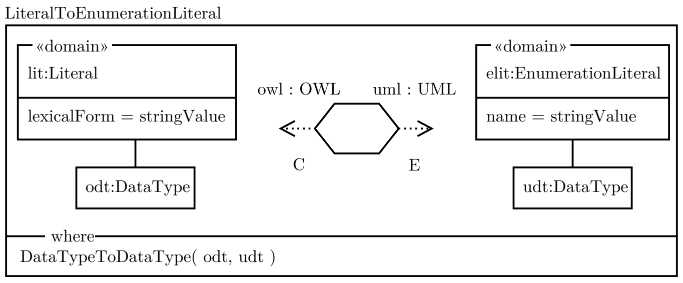


Fig. 7. Example of a QVT-Relations rule from the OWL-2 $\rightarrow$ UML transformation that maps OWL-2 *Literals* to UML *EnumerationLiterals*.

References

[1] G. Schreiber, “A UML Presentation Syntax for OWL Lite,” 2002. [Online]. Available: http://www.swi.psy.uva.nl/usr/Schreiber/docs/owl-uml/owl-uml.html

[2] K. Kiko and C. Atkinson, “A Detailed Comparison of UML and OWL,” Mannheim, 2008, technical report.

[3] L. Hart, P. Emery, B. Colomb, K. Raymond, S. Taraporewall a, D. Chang, Y. Ye, E. Kendall, and M. Dutra, “OWL Full and UML 2.0 Compared,” 2004. [Online]. Available: http://www.omg.org/docs/ontology/04-03-01.pdf

[4] OMG, “Ontology Definition Metamodel,” Object Management Group, 2009. [Online]. Available: http://www.omg.org/spec/ODM/1.0/

[5] S. Höglund, A. Khan, Y. Liu, and I. Porres, “Representing and Validating Metamodels using the Web Ontology Language OWL 2. TUCS Technical Report No. 973,” Turku 2010. [Online]. Available: http://tucs.fi/publications/attachment.php?fname=TR973.full.pdf

[6] J. Zedlitz, J. Jörke, and N. Luttenberger, “From UML to OWL 2,” in *Proceedings of Knowledge Technology Week 2011*, D. Lukose, A. R. Ahmad, and A. Suliman, Eds., Berlin/Heidelberg, 2012, pp. p. 154–163.

[7] S. Tschirner, A. Scherp, and S. Staab, “Semantic access to INSPIRE,” in *Terra Cognita 2011. Proceedings of the Terra Cognita Workshop on Foundations, Technologies and Applications of the Geospatial Web*, R. Grütter, D. Kolas, M. Koubarakis, and P. D., Eds., 2011, pp. p. 75–87. [Online]. Available: http://ceur-ws.org/Vol-798/proceedings.pdf

[8] J. Zedlitz and N. Luttenberger, “Transforming Between UML Conceptual Models and OWL 2 Ontologies,” in *Proceedings of the Terra Cognita Workshop on Foundations, Technologies and Applications of the Geospatial Web, in conjunction with the 11th International Semantic Web Conference (ISWC 2012)*, D. Kolas, M. Perry, R. Grütter, and M. Koubarakis, Eds., 2012, pp. p. 15–26. [Online]. Available: http://ceur-ws.org/Vol-901/paper2.pdf

[9] OMG, “Unified Modeling Language, Superstructure Version 2.4,” 2011. [Online]. Available: http://www.omg.org/spec/UML/2.4/Superstructure

[10] XMLSchema-2, “XML Schema Part 2: Datatypes,” 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-2/

[11] OMG, “Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.1,” 2011. [Online]. Available: http://www.omg.org/spec/QVT/1.1/PDF/