

# O-Telos-RDF: A Resource Description Format with Enhanced Meta-Modeling Functionalities based on O-Telos

Wolfgang Nejdl, Hadhami Dhraief, Martin Wolpers  
Computer Engineering – Knowledge Based Systems  
University of Hannover  
Appelstr. 4, 30167 Hannover, Germany  
email: nejdl,dhraief,wolpers@kbs.uni-hannover.de

## ABSTRACT

In this paper we will describe the formalization of an RDF(S) variant we call O-Telos-RDF, which provides enhanced functionalities for meta-modeling and reified statements. This formalization is based very closely on the formalization of the modeling language O-Telos, which is based on a semantic network model quite similar to RDF(S), and has been used as a (meta) modeling language in various application areas during the last 10 years.

## 1. INTRODUCTION

RDF(S) [5] is based on the simple, yet quite powerful model of semantic networks, where every statement is expressed as a binary predicate on ground arguments, with the nodes in the graph denoting arbitrary resources and literals which are used as subjects and objects of statements, and the arcs denoting specific relationships between two of these nodes.

RDF(S) falls short however in exploiting both the simplicity and the expressivity of the underlying semantic network formalism. The RDF(S) description (see [5] and [3]) is quite difficult to read and lacks a formal description of semantics (though [4] has later given quite a good formalization). Additionally, meta-modeling and reification is cumbersome and restricted in RDF(S), even though it has been envisioned as one of the important application areas of RDF(S).

In this paper we will use the (meta-) modeling language O-Telos as a basis for an alternative resource description format we call O-Telos-RDF, which retains the basic principles of RDF(S), but extends its meta-modeling and reification functionalities. The modeling language O-Telos has been defined and axiomatized in [7], based on its predecessor Telos [10, 9], and implemented in the deductive object oriented database system ConceptBase [6]. It has been used in a variety of modeling contexts (see [8] for an overview) during the last 10 years.

The extended functionalities of O-Telos-RDF are based on O-Telos' use of a semantic network (i.e. binary predicates) for modeling in a uniform way all kinds of information (information about objects and specific relationships between these objects, informa-

tion about classes and relations, information about metaclasses, etc.) This capability is also present in many aspects of RDF(S), but unfortunately not in all, as we will see in this paper. Using semantic networks for modeling has a long tradition, starting with Tarskis use of binary relations [12], and continuing with Abrials semantic datamodel based on such binary relations [1].

We will use the paper by Conen and Klapsing [4] as a starting point for the description of our RDF(S) variant, O-Telos-RDF, and retain basically all original O-Telos axioms from [7], only modifying them when necessary to make them suitable for a resource description format comparable to RDF(S). We will use RDF(S) terminology whenever possible. Our order of presenting the different O-Telos-RDF concepts reflects the order of RDF(S) concept presentation in [4] which will make it easier for knowledgeable RDF(S) users to compare RDF(S) and the O-Telos-RDF variant. We will also use an XML serialization very similar to the original RDF(S) serialization in order to necessitate as few changes in current RDF parsers as possible.

## 2. O-TELOS-RDF CONCEPTS

O-Telos-RDF graphs are semantic networks, which consist of nodes and (directed) arcs. Nodes represent all kinds of concepts (such as classes, metaclasses etc.) and individuals (such as specific WWW pages and literals), arcs represent specific (property) relationships.

Both nodes and arcs are labeled with atomic values or URIs as names. They are both first class entities, and therefore are referenced by IDs (unique within the current namespace), usually in the form of URIs, which either contain the node or arc label (if it is an atomic value), or use the label directly as ID (when the labels are URLs or instances of primitive types like literals). By referencing these IDs, arcs can connect any two other entities (i.e. two nodes, one node and one arc or two arcs), in contrast to RDF(S), where arcs only connect resources (i.e. nodes). As in RDF(S) [4], all nodes have different labels, and represent different entities. Arc labels are not unique, two arcs between two specific entities have different labels, however.

In the following we will discuss the basic O-Telos-RDF concepts building on this framework in more detail, give examples using both tuple and XML-syntax, and compare the O-Telos-RDF concepts with the original O-Telos axioms they are based on, as well as with the usual RDF(S) concepts. We will use the namespace concept of XML and use the namespace "otelos:" throughout this paper, referring to the URI "http://www.kbs.uni-hannover.de/otelos/2001/06/otelos-rdf-schema#" which contains the O-Telos-RDF definitions of this paper.

To allow easy comparison of O-Telos-RDF with RDF(S), we fol-

low the structure of [4] quite closely (distinguishing basic concepts (corresponding to RDF concepts) and schema level concepts (corresponding to RDFS concepts)), even though O-Telos-RDF does not distinguish between schema level and object level statements.

## 2.1 Basic O-Telos-RDF Concepts and Predicates

### 2.1.1 Statement

All nodes and arcs in an O-Telos-RDF graph are represented by statements of the general form  $s(sid, x, l, y)$  where  $sid$  represents the statement ID (a unique identifier of the statement),  $x$  and  $y$  represent identifiers of (possibly other) statements and  $l$  is called the label of the statement. All statement identifiers  $sid$  are URIs or as URI-like as possible and are unique globally (except when exactly the same statements are made in two different places).

We call  $x$  the subject,  $y$  the object and  $l$  the predicate of a statement. All statements in O-Telos-RDF are implicitly reified, and can be identified by their statement ID. We therefore have the first axiom:

AXIOM 2.1. *Statement identifiers uniquely identify statements:*

$$\begin{aligned} & \forall sid, x1, x2, l1, l2, y1, y2 \\ & s(sid, x1, l1, y1) \wedge s(sid, x2, l2, y2) \\ & \Rightarrow (x1 = x2) \wedge (y1 = y2) \wedge (l1 = l2) \end{aligned}$$

We can define auxiliary predicates subject, predicate and object as follows:

$$\begin{aligned} & \forall sid, c, p, r \ s(sid, c, p, r) \\ & \Leftrightarrow subject(sid, c) \wedge predicate(sid, p) \wedge object(sid, r) \end{aligned}$$

**Example 1:** This example gives a first impression of how the resulting statements look like. The exact definitions of the used constructs will follow in later sections. We want to express that a resource “LectureUnit1” has a property “title” with the value “Lecture Unit 1”.

The XML-serialization (which we keep as similar to RDF(S) as possible - see appendix A for the basic XML serialisation) looks as follows:

```
<otelos:Description ID="LectureUnit1">
  <title>Lecture Unit 1</title>
</otelos:Description>
```

The statements for this serialization are

```
s(sid1,sid1,LectureUnit1,sid1)
s(sid2,sid1,title,"Lecture Unit 1")
s(sid3,sid1,type,otelos:individual)
s(sid4,sid2,type,otelos:property)

where

sid1=ns:LectureUnit1
sid2=ns:LectureUnit1_title
sid3=ns:LectureUnit_type_otelos:individual
sid4=bns:LectureUnit1_title_type_otelos:property
```

and “ns:” stands for the current namespace specifying where these metadata can be found (e.g. “http://www.kbs.uni-hannover.de/ai1/metadata.html#”).<sup>1</sup> The statement identifiers are generated automatically from the other arguments (we’ll discuss how later), so they do not have to be included in the XML serialization.

<sup>1</sup>In the following, we will abbreviate these URLs to save space, leaving out hostname and directories, and use forms like “http://.../metadata.html#”.

Tuples  $sid3$  and  $sid4$  specify membership of  $sid1$  and  $sid2$  in the (predefined) O-Telos-RDF classes  $otelos:individual$  and  $otelos:property$  respectively (similar to  $rdfs:class$  and  $rdf:property$ ), and are themselves members of the O-Telos-RDF class  $otelos:type$ . As we will see in the following sections, membership in these predefined classes is specified by the syntactic form of the statements, and we will not include these type-statements in our later examples, as they can always be reconstructed from the axiomatic membership definition (2.3, 2.5 and 2.10) for these predefined classes.

**Example 2:** This example states that a web page with the URI “http://.../Definitions.html” has a title called “Definitions”.

```
<otelos:Description
  about="http://.../Definition.html">
  <title>Definitions</title>
</otelos:Description>
```

The corresponding statements are

```
s(sid1,sid1,http://.../Definitions.html,sid1)
s(sid2,sid1,title,"Definitions")
```

where

```
sid1=http://.../Definitions.html
sid2=http://.../Definitions.html_title
```

Additionally, as already mentioned in example 1,  $sid1$  is instance of  $otelos:individual$  while  $sid2$  is instance of  $otelos:property$ . Furthermore these two type-statements are instances of  $otelos:type$ .

#### 2.1.1.1 Comparison to RDF(S) and O-Telos.

Axiom 2.1 corresponds to the O-Telos axiom 1 (uniqueness of object identifiers). O-Telos-RDF statement identifiers are constructed as URIs or URI-like identifiers in order to be able to reference them easily in other statements. In contrast to O-Telos object identifiers, they are not invisible. Their exact form will be discussed in the following sections. In contrast to RDF(S), there is no difference between “named” and “unnamed” resources, all statements have unique IDs. For Web-Pages, these are the usual URLs.

Axiom 1 of [4] states that an RDF(S) statement connects two nodes with a label. The label has to be RFC 2396 [2] conform, statements do not have a unique ID. In O-Telos-RDF, labels can also be atomic values, while statement IDs are unique and usually conform to RFC 2396.

Analogously to RDF(S), we call  $x$  the subject,  $y$  the object and  $l$  the predicate of a statement, although we have not defined them as predefined properties (which we could, though).

All statements in O-Telos-RDF are implicitly reified, and can be identified by their statement ID. Thus we don’t have to reify a statement explicitly which contrasts with the axioms 7, 8 and 9 of [4]. Instead, when one statement talks about another statement, we have to enforce the existence of the statement talked about. This is basically the same as a foreign key constraint in relational database theory.

In order to explicitly distinguish between statements and explicit reifications thereof similar to RDF(S) (and the [4] axioms), we could define a class  $otelos:reified\_statement$ , with three properties  $otelos:subject$ ,  $otelos:predicate$  and  $otelos:object$ .

#### 2.1.2 Individual

Nodes in an O-Telos-RDF graph are represented by statements of the form  $s(o,o,o,o)$  or  $s(ns:l,ns:l,ns:l)$ , where  $o$  and  $ns:l$  are URIs,  $ns$  is the current namespace and  $l$  is an atomic label. We call these statements individuals, they can be used as subjects and objects in statements.

AXIOM 2.2. *The set of all statements, abbreviated as `otelos:statement` is represented by the individual `s(otelos:statement,otelos:statement,statement,otelos:statement)`, where “`otelos:`” is the namespace for the O-Telos-RDF definitions.*

Serialized as XML, this individual is simply declared as

```
<otelos:Description ID="statement"/>
  or, more RDF(S)-like
<otelos:Description ID="statement"/>
  <type s="otelos:individual"/>
</otelos:description>
```

though the type-statement (which will be explained in section 2.2.1) is not necessary, as membership of statements in the set of individuals is defined by the syntactic form of the statement.

AXIOM 2.3. *The set of all individuals is represented by `otelos:individual` or (in longer form) `s(otelos:individual,otelos:individual,individual,otelos:individual)`.*

O-Telos-RDF individuals can represent both classes and instantiated objects (as well as metaclasses, metametaclasses, etc.), there is no syntactic distinction between these different abstraction levels. This makes unrestricted metamodelling hierarchies possible in O-Telos-RDF.

AXIOM 2.4. *If the label of an individual is an atom, it is unique within its namespace. Together with its namespace, or if the label is already an URI, it is unique globally. Therefore the statement ID of an individual is unique globally in all cases.*

In the following, we will use the statement ID of an individual as an abbreviated name for that individual, i.e. we will be talking about `otelos:statement`, `otelos:individual`, etc. As the statement ID is unique and human readable, we can use this ID also to reference all other statements, which are not individuals.

**Example 3:** The following example declares an individual with the name “LectureUnit1”:

```
<otelos:Description ID="LectureUnit1"/>
  with the corresponding statement
s(sid1,sid1,LectureUnit1,sid1)
  where
sid1=ns:LectureUnit1
```

### 2.1.2.1 Comparison to RDF(S) and O-Telos.

In O-Telos-RDF all individuals are resources in the usual RDF(S) sense, and can be used as subjects and objects in statements. O-Telos states the existence of statements in its axioms 18 and 24 which correspond to axiom 2.2 of O-Telos-RDF.

Axioms 1 and 2 of [4] state the existence of Resource which is equal to the O-Telos-RDF concept `otelos:individual` (see axiom 2.3), which corresponds to the O-Telos axioms 19 and 25.<sup>2</sup>

Axiom 4 of [4] states that a named Resource is identified by an URI. O-Telos-RDF broadens the scope of the identifiers with axiom 2.4 so that all individuals are identified by an URI, and further statements can be made about each individual using this URI. Labels of individuals are unique in O-Telos-RDF (corresponding to O-Telos axiom 2).

<sup>2</sup>If we view something as a resource which can be referenced by an identifier, “resource” would correspond to “statement” in O-Telos-RDF, though.

### 2.1.3 Property

Arcs are represented by statements of the form `s(sid,x,p,y)` or by the special case `s(ns:p,otelos:statement,p,y)`, with `sid ≠ x`, `sid ≠ y`, `ns:p ≠ otelos:statement` and `ns:p ≠ y`, and `p` different from the three reserved labels `type`, `subClassOf` and `subPropertyOf`. We call these statements properties:

AXIOM 2.5. *Definition of properties:*

$$\forall sid, x, p, y$$

$$s(sid, x, p, y) \wedge (sid \neq x) \wedge (sid \neq y) \wedge (p \neq subClassOf)$$

$$\wedge (p \neq subPropertyOf) \wedge (p \neq type)$$

$$\Leftrightarrow type(sid, otelos : property)$$

AXIOM 2.6. *The set of all properties is denoted by the statement `(otelos:property,otelos:statement,property,otelos:statement)`, abbreviated as `otelos:property`.*

#### 2.1.3.1 Object Scoped Properties.

In the first case `s(sid,x,p,y)`, `p` is an atomic value, and is unique within the current namespace in conjunction with the source object `x`, which is the unique identifier of another statement. Then the `sid` is of the form `x_p`, which is unique in the current namespace, and potentially unique globally (except if another namespace specifies a property with the same label for the same `x`). This naming of the statement identifiers does not guarantee globally unique statement IDs, but is in line with the object scoped way of looking at properties.<sup>3</sup> If the property `s(sid,x,p,y)` is meant to represent the property definition for `p`, then (using RDF(S) terminology), `x` is the domain of `p` and `y` is the range of `p`.

We then call `p` an *object scoped* property. This definition corresponds to the class-centric way of defining properties used in frame-based languages like O-Telos.

AXIOM 2.7. *Names of “object scoped” properties are unique in conjunction with the source object:*

$$\forall sid1, sid2, x, p, y1, y2$$

$$s(sid1, x, p, y1) \wedge s(sid2, x, p, y2)$$

$$\Rightarrow (sid1 = sid2) \vee (p = type) \vee (p = subClassOf)$$

$$\vee (p = subPropertyOf)$$

**Example 4:** The following description defines `LectureUnit`, which has a property named `title` of type `literal`.

```
<otelos:Description ID="LectureUnit">
  <title s="otelos:literal">
</otelos:Description>
```

with the main statements

```
s(sid1,sid1,LectureUnit,sid1)
s(sid2,sid1,title,otelos:literal)
```

<sup>3</sup>Note, that duplicate property statements are also possible in RDF(S) between different namespaces. In any case, even if we defined the statement identifiers in a globally unique way (which would be possible by prefixing the ID with the current namespace), integration of statements from different namespaces would still need to be dealt with care, as globally unique statement IDs would only handle uniqueness of statements, but still allow inconsistency of properties (such as multiple values for single valued properties, etc.)

with

```
sid1:ns:LectureUnit
sid2:ns:LectureUnit_title
```

sid1 is a member of the set otelos:individual, sid2 is a member of the set otelos:property. Both membership-statements are member of the set otelos:type.

### 2.1.3.2 Globally Scoped Properties.

In the second (special) case  $s(ns:p, otelos:statement, p, y)$ ,  $p$  is an atomic value, and is a unique label within the current namespace  $ns$ . We then call  $p$  a *globally scoped* property, because it can be used as property for all kinds of statements. This definition corresponds to the property-centric way of defining properties in RDF(S).

AXIOM 2.8. For “globally scoped” properties, axiom 2.7 is extended, so that the names of attributes are unique even without conjunction with the source object:

$$\begin{aligned} & \forall sid1, x1, x2, p, y1, y2 \\ & s(ns : p, x1, p, y1) \wedge s(sid1, x2, p, y2) \\ & \Rightarrow ((sid1 = ns : p) \wedge (x1 = x2) \wedge (y1 = y2)) \\ & \vee (p = type) \vee (p = subclassOf) \\ & \vee (p = subPropertyOf) \end{aligned}$$

### 2.1.3.3 Comparison to RDF(S) and O-Telos.

O-Telos-RDF axiom 2.7 corresponds to O-Telos axiom 3. Axioms 2.5 and 2.6 correspond to O-Telos axioms 22 and 26.

RDF(S) regards properties as a projection of the second argument of the RDF-statement. Therefore [4] state with their axioms 2 and 3 that each label of a statement is a resource and that it is identified by an URI. This holds for O-Telos (axiom 3) and O-Telos-RDF (axiom 2.7) as well, with the exception of literals and some property statements (type etc.) which yield IDs that are not RFC 2396 conform.

In contrast to O-Telos and to RDF, O-Telos-RDF allows both globally or object scoped properties, though the globally scoped properties are just a special case of the object scoped properties. So, as can be seen in example 4, the definition of properties is object centered and is included in the class definition rather than written outside of the class definition in separate statements.

If different domains (as in RDF(S)) have to be expressed for a property, this property has to be an object scoped property. When using object scoped properties, different ranges are possible (in contrast to the current RDF(S) specification).

### 2.1.4 Literals and Other Primitive Types

Literals  $l$  are represented as individuals  $s(l, l, l)$ . The set of all literals is denoted by the individual otelos:literal. Other primitive types (like the ones defined in the recent XML schema specification, or Integer, Boolean, etc. from O-Telos) can be introduced in the same way.

While the statement IDs for these primitive individuals are no URIs, this representation allows us to use these individuals consistently with current RDF(S)/XML statements, by simply including the value in a statement.

#### 2.1.4.1 Comparison to RDF(S) and O-Telos.

Axioms 5 and 6 of [4] declare a literal as an object that is not a resource. These literals are instances of rdfs:Literal. In contrast, O-Telos (and O-Telos-RDF) declare predefined classes, such as Literal (String), Boolean, Integer, etc.

## 2.2 Schema Definition Concepts and Predicates

### 2.2.1 type

Statements of the form  $s(sid, x, type, y)$  denote class membership of  $x$  in  $y$ . We talk about  $x$  being an instance of  $y$ , where  $x$  and  $y$  represent either two individuals or two properties.

AXIOM 2.9. Type statements can be written using the auxiliary predicate  $type(x, y)$ :

$$\begin{aligned} & \forall sid, x, y \\ & s(sid, x, type, y) \Rightarrow type(x, y) \end{aligned}$$

AXIOM 2.10. All statements, where subject and object are different from the statement ID and which use the label “type”, are instances of the set otelos:type, represented by the statement  $s(otelos:type, otelos:statement, type, otelos:statement)$ :

$$\begin{aligned} & \forall sid, x, c \\ & s(sid, x, type, c) \wedge (sid \neq x) \wedge (sid \neq c) \\ & \Leftrightarrow type(sid, otelos : type) \end{aligned}$$

AXIOM 2.11. The label “type” of these statements is unique in conjunction with the source object  $x$  and the destination object  $y$ . The statement identifier  $sid$  has therefore the form  $x\_type\_y$ :

$$\begin{aligned} & \forall sid1, sid2, x, p, y \\ & s(sid1, x, p, y) \wedge s(sid2, x, p, y) \wedge (p = type) \\ & \Rightarrow (sid1 = sid2) \end{aligned}$$

AXIOM 2.12. Property statements can be written with the name of the property (instead of a new label) by using the auxiliary predicate  $P(x, m, y)$ :

$$\begin{aligned} & \forall sid1, sid2, x, l, y, c, m, d \\ & s(sid1, x, l, y) \wedge s(sid2, c, m, d) \wedge type(sid1, sid2) \\ & \Rightarrow P(x, m, y) \end{aligned}$$

AXIOM 2.13. Properties  $P$  of a subject  $x$  are always expressed as a property statement, which is an instantiation of a property definition for the class  $c$  of which  $x$  is a member of:

$$\begin{aligned} & \forall x, l, y, c, m, d, sid1, sid2 \\ & type(x, c) \wedge P(x, m, y) \wedge s(sid1, c, m, d) \\ & \Rightarrow \exists s(sid2, x, l, y) \wedge type(sid2, sid1) \end{aligned}$$

AXIOM 2.14. In case  $x$  is an instance of two classes  $c$  and  $d$ , which both define a property  $m$ ,  $x$  has also to be an instance of a class  $g$ , which is a subclass of both  $c$  and  $d$ , and which also defines property  $m$ :

$$\begin{aligned} & \forall x, m, y, c, d, sid1, sid2, e, f \\ & (type(x, c) \wedge type(x, d) \wedge s(sid1, c, m, e) \wedge s(sid2, d, m, f)) \\ & \Rightarrow \exists g, sid3, h type(x, g) \wedge s(sid3, g, m, h) \\ & \wedge subclassOf(g, c) \wedge subclassOf(g, d) \end{aligned}$$

The axiom 2.14 handles multiple inheritance/instantiation in case two classes of an object both define a common property, by demanding a third class the object is an instance of, which defines this property and thus makes instantiating the property for  $x$  unique.

**Example 5:** The following example shows the use of type-statements. Let us assume, that there is an individual called `LectureUnit` that has a property of type `String` labeled `title`. Another individual called `LectureUnit1` is an instance of `LectureUnit` and instantiates the property `title` with the value “Lecture Unit 1”.

Please note that the following examples use the namespaces `rdf:`, `s:`, `t:` and `otelos:` instead of the URIs with the statement-declarations. We abbreviated the URIs in order to enhance the readability of the examples.

```
<otelos:OTELOS
  xmlns:t="http://www.kbs.uni-hannover.de/otelos
           /2001/06/example-5"
  xmlns:otelos="http://www.kbs.uni-hannover.de
               /otelos/2001/06/otelos-rdf-schema">

  <otelos:Description ID="LectureUnit">
    <title s="otelos:literal">
    </otelos:Description>

  <otelos:Description ID="LectureUnit1">
    <type s="t:LectureUnit"/>
    <title>Lecture Unit 1</title>
    </otelos:Description>

  <otelos:Description ID="LectureUnit1_title">
    <type s="t:LectureUnit_title"/>
    </otelos:Description>

</otelos:OTELOS>
```

We have the main statements

```
s(sid1,sid1,LectureUnit,sid1)
s(sid2,sid1,title,otelos:Literal)
s(sid3,sid3,LectureUnit1,sid3)
s(sid4,sid3,title,"Lecture Unit 1")
```

and the type statements for `LectureUnit1` and `LectureUnit1_title`

```
s(sid5,sid3,type,sid1)
s(sid6,sid4,type,sid2)
```

We do not need to specify type statements for membership in `otelos:individual`, `otelos:property` and `otelos:type` because they result from the respective axioms.

```
sid1=ns:LectureUnit
sid2=ns:LectureUnit_title
sid3=ns:LectureUnit1
sid4=ns:LectureUnit1_title
sid5=ns:LectureUnit1_type_ns:LectureUnit
sid6="ns:Lecture Unit 1_type_otelos:Literal"
```

### 2.2.1.1 Comparison to RDF(S) and O-Telos.

Instantiations of classes using “type” are done in the same way in RDF(S) as in O-Telos-RDF axioms 2.11 and 2.9 (corresponding to the O-Telos axioms 4 and 5). Furthermore, objects can instantiate the attributes defined in their classes in the same way in RDF(S) and O-Telos-RDF (axioms 2.12 and 2.13, corresponding to O-Telos axioms 7, 8 and 9). Axiom 2.10 corresponds to the O-Telos axioms 27 and 20.

In contrast to RDF(S) [4], which only has three abstraction hierarchies (`rdfs:class`, `classes`, which are instances of `rdfs:class` and instances of these classes) and represents property instantiation implicitly (i.e. not by explicit statements), O-Telos and

hence O-Telos-RDF allow arbitrarily many abstraction hierarchies, because their representation does not distinguish between meta-classes, classes and objects, and instantiation is represented explicitly for properties as well as for individuals.

This instantiation of properties is defined in axioms 2.12 and 2.13 (corresponding to O-Telos axioms 7, 8 and 9), and leads to instantiated property statements, which have a (possibly) different label than the property definitions they instantiate, and which can be instantiated again if needed.

Axiom 2.14 (corresponding to O-Telos axiom 17) handles multiple inheritance/instantiation. Such an axiom is not necessary in RDF(S) because properties are defined separately from classes anyway, and a given property can have just one range restriction.

Differently to RDF(S), `otelos:type` stands on its own and is different from `otelos:property` (axiom 2.10, corresponding to O-Telos axiom 20).

Axiom 2.12 (corresponding to O-Telos axioms 7 and 8) defines how to state properties in an RDF(S)-like way. We can therefore translate RDF(S)-like property statements like  $P(x,m,y)$  into the corresponding O-Telos-RDF statements by generating a unique object identifier and a label for the statement representing the instantiated property statement, plus an additional statement for the instantiation relationship.

### 2.2.2 Domain and Range

Domain and range properties like those in RDF(S) are not needed, as domain and range restrictions are already included in all property definitions. For compatibility reasons, we can define auxiliary predicates domain and range like

$$\forall sid, c, p, r \\ s(sid, c, p, r) \Leftrightarrow domain(p, c) \wedge range(p, r)$$

or alternatively, as the (globally scoped) properties `otelos:domain` and `otelos:range`, defined by the statements

```
s(otelos:domain,otelos:property,domain,
   otelos:statement)
s(otelos:range,otelos:property,range,
   otelos:statement)
```

**AXIOM 2.15.** *Subjects and objects in a statement representing an instantiated property are typed by the corresponding property definition:*

$$\forall sid1, sid2, x, l, y \\ s(sid1, x, l, y) \wedge type(sid1, sid2) \\ \Rightarrow \exists c, p, r s(sid2, c, p, r) \wedge type(x, c) \wedge type(y, r)$$

If no domain or range restrictions are desired for a property `p`, it can be defined as a globally scoped property in the following way:

```
s(ns:p,otelos:statement,p,otelos:statement)
```

### 2.2.2.1 Comparison to RDF(S) and O-Telos.

In contrast to O-Telos-RDF, RDF(S) has to state domain and range using separate properties.

[4] define the domain property in rules 19 to 21. It has a cardinality of zero or more which complies with O-Telos-RDF. Furthermore, [4] define the range property in rules 22 to 26. RDF(S) restricts the range property to one range constraint per property. In contrast, each O-Telos-RDF property definition has exactly one range constraint, but for different domains different property definitions with range constraints are possible (axiom 2.15). This corresponds to O-Telos and its axiom 14.

### 2.2.3 Class

As individuals can represent both objects and classes and all individuals can be instantiated, there is no need to introduce a separate class concept. All RDF(S) classes are represented as O-Telos-RDF individuals, as are their members. Class membership is defined using type-statements.

Individuals as well as properties are instantiated explicitly (in contrast to RDF(S), where instantiation is only explicit for individuals). Furthermore, all statements (including the ones representing instantiated individuals and properties) have a label and a unique ID. This makes it possible to build up arbitrary instantiation hierarchies in a metamodelling context (objects, classes, metaclasses, metametaclasses etc). Even though RDF(S) schema concepts are expressed as RDF(S) concepts, this is not possible in conventional RDF(S), as instantiated property statements neither have a label nor statement ID, which makes it impossible to uniquely reference these statements as first class objects.

For compatibility reasons, we could define an individual `otelos:class`, which denotes the set of all individuals, which are/will be instantiated (which is actually done in O-Telos), or define `otelos:class` as a synonym for `otelos:individual`.

#### 2.2.3.1 Comparison to RDF(S) and O-Telos.

Even though RDF(S) defines the concept `rdfs:Class`, no special axioms are defined in [4]. Rather, axioms are defined for the special properties related to `rdfs:Class`, namely `rdf:type`, `rdfs:subClassOf`, `rdfs:range` and `rdfs:domain`. This is similar to the formalization in O-Telos-RDF, which does not define the class concept at all, but has similar restrictions on the properties related to this concept.

### 2.2.4 subClassOf

Statements of the form `s(sid,x,subClassOf,y)` denote a subclass relationship between `x` and `y`. We talk about `x` being a subclass of `y`, where `x` represents a class and `y` its superclass. `x` and `y` are individuals.

AXIOM 2.16. *The label “subClassOf” in subClassOf-statements is unique in conjunction with the source object `x` and the destination object `y`, thus sids are of the form `x_subClassOf-y`:*

$$\begin{aligned} &\forall sid1, sid2, x, l, y \\ &s(sid1, x, l, y) \wedge s(sid2, x, l, y) \wedge (l = subClassOf) \\ &\Rightarrow (sid1 = sid2) \end{aligned}$$

AXIOM 2.17. *We can also define an auxiliary predicate `subClassOf(x,y)`:*

$$\begin{aligned} &\forall sid, c, d \\ &s(sid, c, subClassOf, d) \Rightarrow subClassOf(c, d) \end{aligned}$$

AXIOM 2.18. *The set of all subClassOf-statements is represented by the statement `s(otelos:subClassOf,otelos:individual, subClassOf,otelos:individual)`.<sup>4</sup>*

<sup>4</sup> Actually, because O-Telos uses the same label “isA” (corresponding to our “subClassOf”) for subclassing both individuals and properties, the direct translation from O-Telos would be the more general statement `s(otelos:subClassOf,otelos:statement,subClassOf,otelos:statement)`. However, in line with RDF(S) conventions, we will distinguish between the two concepts `subClassOf` and `subPropertyOf`.

AXIOM 2.19. *All statements with a label “subClassOf” whose sid is not equal to subject and object are members of the set `otelos:subClassOf`:*

$$\begin{aligned} &\forall sid, c, d \\ &s(sid, c, subClassOf, d) \wedge (sid \neq c) \wedge (sid \neq d) \\ &\Leftrightarrow type(sid, otelos : subClassOf) \end{aligned}$$

The `subClassOf` relationship is a partial order on the statement identifiers. This relationship is reflexive as well as transitive, and does not contain cycles, but uses reflexivity to state equality.

AXIOM 2.20. *Reflexivity of subClassOf:*

$$\begin{aligned} &\forall sid \\ &type(sid, otelos : statement) \Rightarrow subClassOf(sid, sid) \end{aligned}$$

AXIOM 2.21. *Transitivity of subClassOf:*

$$\begin{aligned} &\forall sid1, sid2, sid3 \\ &subClassOf(sid1, sid2) \wedge subClassOf(sid2, sid3) \\ &\Rightarrow subClassOf(sid1, sid3) \end{aligned}$$

AXIOM 2.22. *No cycles, statement of equality:*

$$\begin{aligned} &\forall sid1, sid2 \\ &subClassOf(sid1, sid2) \wedge subClassOf(sid2, sid1) \\ &\Rightarrow (sid1 = sid2) \end{aligned}$$

AXIOM 2.23. *Class membership is inherited upwardly to the superclasses:*

$$\begin{aligned} &\forall x, c, d \\ &type(x, d) \wedge subClassOf(d, c) \Rightarrow type(x, c) \end{aligned}$$

#### 2.2.4.1 Comparison to RDF(S) and O-Telos.

Axiom 2.28, 2.19, 2.17 and 2.16 correspond to O-Telos axioms 28, 21, 6 and 4, and define `subClassOf` similarly to RDF(S).

Contrary to RDF(S), the `subClassOf`-relationship is defined as a partial order, which is reflexive as well as transitive. However (as in RDF(S)) `subClassOf` does not contain cycles, and uses reflexivity just to state equality (2.20, 2.21 and 2.22, corresponding to O-Telos axioms 10, 11, 12).

Class membership inherits upwardly to the superclasses in all formalisms (RDF(S) [4] in rule 13, O-Telos-RDF in axiom 2.23, O-Telos in axiom 13).

### 2.2.5 subPropertyOf

For compatibility reasons to RDF(S), we define `otelos:subPropertyOf` separately from `subClassOf`, even though it has all axioms from `subClassOf`, plus additional ones which are relevant for `subPropertyOf` only.

Statements of the form `s(sid,x,subPropertyOf,y)` denote a subproperty relationship between `x` and `y`. We talk about `x` being a subproperty of `y`, where `x` represents a property and `y` its superproperty. `x` and `y` are properties.

AXIOM 2.24. *As for “subClassOf”, the label “subPropertyOf” in statements is unique in conjunction with the source object*

$x$  and the destination object  $y$ :

$$\begin{aligned} & \forall sid1, sid2, x, y, l \\ & s(sid1, x, l, y) \wedge s(sid2, x, l, y) \wedge (l = subPropertyOf) \\ & \wedge type(x, otelos : property) \wedge type(y, otelos : property) \\ & \Rightarrow (sid1 = sid2) \end{aligned}$$

AXIOM 2.25. The auxiliary predicate  $subPropertyOf(c,d)$  is defined as follows:

$$\begin{aligned} & \forall c, d, sid \\ & type(c, otelos : property) \wedge type(d, otelos : property) \\ & \wedge s(sid, c, subPropertyOf, d) \Rightarrow subPropertyOf(c, d) \end{aligned}$$

The  $subPropertyOf$  relationship is a partial order on the statement identifiers. This relationship is reflexive as well as transitive, and does not contain cycles, but uses reflexivity to state equality:

AXIOM 2.26. Reflexivity of  $subPropertyOf$ :

$$\begin{aligned} & \forall sid \\ & type(sid, otelos : statement) \Rightarrow subPropertyOf(sid, sid) \end{aligned}$$

AXIOM 2.27. Transitivity of  $subPropertyOf$ :

$$\begin{aligned} & \forall sid1, sid2, sid3 \\ & subPropertyOf(sid1, sid2) \wedge subPropertyOf(sid2, sid3) \\ & \wedge type(sid1, otelos : property) \wedge type(sid2, otelos : property) \\ & \Rightarrow subPropertyOf(sid1, sid3) \end{aligned}$$

AXIOM 2.28. No cycles, statement of equality:

$$\begin{aligned} & \forall sid1, sid2 \\ & subPropertyOf(sid1, sid2) \wedge subPropertyOf(sid2, sid1) \\ & \wedge type(sid1, otelos : property) \wedge type(sid2, otelos : property) \\ & \Rightarrow (sid1 = sid2) \end{aligned}$$

AXIOM 2.29. Property membership is inherited upwardly to the superproperties:

$$\begin{aligned} & \forall x, c, d \\ & type(x, d) \wedge subPropertyOf(d, c) \wedge type(c, otelos : property) \\ & \wedge type(d, otelos : property) \Rightarrow type(x, c) \end{aligned}$$

AXIOM 2.30. All  $subPropertyOf$ -statements are members of the set  $otelos:subPropertyOf$ , which is represented by the statement  $s(otelos:subPropertyOf, otelos:property, subPropertyOf, otelos:property)$ <sup>5</sup>, thus  $sid$  are of the form  $x\_subPropertyOf\_y$ :

$$\begin{aligned} & \forall sid, c, d \\ & (s(sid, c, subPropertyOf, d) \wedge (sid \neq c) \wedge (sid \neq d) \\ & \wedge type(c, otelos : property) \wedge type(d, otelos : property) \\ & \Leftrightarrow type(sid, otelos : subPropertyOf)) \end{aligned}$$

<sup>5</sup>In principle, we could allow the application of  $subPropertyOf$  for instances of  $otelos:type$ ,  $otelos:subClassOf$  and  $otelos:subPropertyOf$ . This is actually done in O-Telos, as it does not distinguish between  $subClassOf$  and  $subPropertyOf$ .

AXIOM 2.31. Subclasses, which define properties with the same name as properties of their classes, refine these properties:

$$\begin{aligned} & \forall sid1, sid2, c, d, e, f, m \\ & subClassOf(d, c) \wedge s(sid1, c, m, e) \wedge s(sid2, d, m, f) \\ & \Rightarrow subClassOf(f, e) \wedge subPropertyOf(sid2, sid1) \end{aligned}$$

AXIOM 2.32. Furthermore, O-Telos-RDF requires subproperties of properties to refine subject and object of the properties as well.

$$\begin{aligned} & \forall sid1, sid2, c, d, e, f, m, l \\ & subPropertyOf(sid2, sid1) \wedge s(sid1, c, m, e) \wedge s(sid2, d, l, f) \\ & \Rightarrow subClassOf(d, c) \wedge subClassOf(f, e) \end{aligned}$$

**Example 6:** The following example will further clarify things. Two resources are related by a property. Also, these resources are further subclassed, therefore the property has to be subclassed as well.

```
<otelos:OTELOS
  xmlns:t="http://www.kbs.uni-hannover.de/otelos
    /2001/06/example-6"
  xmlns:otelos="http://www.kbs.uni-hannover.de
    /otelos/2001/06/otelos-rdf-schema">
  <otelos:Description ID="Lecture"/>
  <otelos:Description ID="LectureUnit">
    <parentCourse s="t:Lecture"/>
  </otelos:Description>
  <otelos:Description ID="Seminar">
    <subClassOf s="t:Lecture"/>
  </otelos:Description>
  <otelos:Description ID="SeminarUnit">
    <subClassOf s="t:LectureUnit"/>
    <parentCourse s="t:Seminar"/>
  </otelos:Description>
  <otelos:Description
    ID="SeminarUnit_parentCourse">
    <subPropertyOf
      s="t:LectureUnit_parentCourse"/>
  </otelos:Description>
</otelos:OTELOS>
```

This example declares a lecture unit with a property parentCourse of type lecture. The subclass of LectureUnit called SeminarUnit must refine the property parentCourse in order to use it. Thus its value has to be a subclass of Lecture, e.g. Seminar.

### 2.2.5.1 Comparison to RDF(S) and O-Telos.

O-Telos-RDF treats  $otelos:subPropertyOf$  in basically the same way as  $otelos:subClassOf$  (like O-Telos, which does not distinguish these two concepts at all). This is also consistent with RDF(S), so [4] defines  $rdfs:subPropertyOf$  very similar to  $rdfs:subClassOf$ . In RDF(S) a statement with properties for subject and object and a label with the value  $subPropertyOf$  declares the subject as subproperty of the object ([4], rule 15), as is done in O-Telos-RDF, axioms 2.24 and 2.25, and in O-Telos, axioms 4 and 6. O-Telos and O-Telos-RDF declare a special set for the  $subPropertyOf$ -statements (O-Telos-RDF, axiom 2.30, O-Telos axioms 28 and 21).

In RDF(S) this relationship is transitive ([4], rule 16) and without cycles ([4], rule 18). As with `subClassOf`, O-Telos-RDF (axioms 2.26, 2.27 and 2.28) and O-Telos (axioms 10, 11 and 12) define `subPropertyOf` as a partial order.

Property membership in O-Telos-RDF is inherited upwardly to the superclasses (axiom 2.29, corresponds to O-Telos axiom 13). [4] has to define this by introducing new statements for superproperties (rule 17), as no instantiation statements for properties exist in RDF(S):

$$\forall s, p1, p2, o \\ s(s, p1, o) \wedge \text{subPropertyOf}(p1, p2) \Rightarrow s(s, p2, o)$$

Axioms 2.31 and 2.32 (corresponding to O-Telos axioms 15 and 16) do not exist in RDF(S).

## 2.3 Utility Concepts

### 2.3.1 Sequences and Bags

RDF(S) has additional utility constructs `rdf:Seq` and `rdf:Bag`, used to express sequences and bags. In our opinion, `rdf:Bag` is not really necessary, as multi-valued properties are expressed simply by more than one instantiated property statement.

As for `rdf:Seq`, the disadvantage of this construct is, that (similar to the case of the Java Vector class) it introduces untyped relation ends, if it is used as an explicit class. Because O-Telos-RDF has the capability of defining properties with the domain `otelos:property`, we will use this alternative way in O-Telos-RDF to express sequences.

To express sequences in O-Telos-RDF, we include an additional type called `otelos:ordinal`. `otelos:ordinal` is a subclass of `otelos:literal`. It specialises the literals by using only integer numbers prefixed by “\_” as labels, starting with 1, 2, etc. The respective statements use the label for `sid`, subject, predicate and object. For example the statement `s(_1,_1,_1,_1)` represents the ordinal “\_1”.

Furthermore we declare the new property `otelos:sequence` with the statement `s(otelos:sequence,otelos:statement,sequence,otelos:ordinal)`

**Example 7:** Let us assume that we have a multivalued property `sid1`, and that `sid2`, `sid3` and `sid4` are instantiations of that property. Furthermore, we want to represent a sequencing between these three property statements.

We have the following property-statements:

```
s(sid1,c,p,d)
s(sid2,x,p1,y1)
s(sid3,x,p2,y2)
s(sid4,x,p3,y3)
s(sid5,sid2,type,sid1)
s(sid6,sid3,type,sid1)
s(sid7,sid4,type,sid1)
```

Then, the following statements specify the sequencing:

```
s(sid8,sid2,number,\_1)
s(sid9,sid3,number,\_2)
s(sid10,sid4,number,\_3)
```

`sid4`, `sid5` and `sid6` are of type `otelos:sequence`, specified by the corresponding type-statements:

```
s(sid11,sid8,type,otelos:sequence)
s(sid12,sid9,type,otelos:sequence)
s(sid13,sid10,type,otelos:sequence)
```

### 2.3.1.1 Comparison to RDF(S) and O-Telos.

In order to state sequences, RDF(S) specifies the container `rdf:Seq`. Instances of `rdf:Seq` group together the resources of a sequence by using simple ordinal properties. O-Telos-RDF simplifies this approach by using the possibility to state properties about properties. Thus O-Telos-RDF avoids the introduction of untyped containers like `rdf:Seq` which is preferable for precise semantical modelling.

## 2.4 Modelling the lecture Artificial Intelligence

The following example is taken from a model of our lectures. For simplicity reasons it is a small part of the larger model only.

On the class level it states that lecture units belong to a lecture. Lecture units have a title and a description. A lecture unit consists of theory pages, examples, etc. We also include simple subclasses respectively instances like AI lecture units, etc.

We will start with the appropriate XML-serialisation using RDF(S):

In the following example the URIs of the namespaces `rdf:`, `rdfs:`, `s:`, `t:` and `otelos:` are abbreviated with their names.

```
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02
              /22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01
              /rdf-schema#"
  xmlns:s="http://www.kbs.uni-hannover.de/otelos
           /2001/06/example-schema#">

  <rdf:Description ID="Lecture">
    <rdf:type resource="rdfs:Class"/>
  </rdf:Description>

  <rdf:Description ID="LectureUnit">
    <rdf:type resource="rdfs:Class"/>
  </rdf:Description>

  <rdf:Description ID="title">
    <rdf:type resource="rdf:Property"/>
    <rdfs:range resource="rdfs:Literal"/>
    <rdfs:domain resource="s:LectureUnit"/>
  </rdf:Description>

  <rdf:Description ID="description">
    <rdf:type resource="rdf:Property"/>
    <rdfs:range resource="rdfs:Literal"/>
    <rdfs:domain resource="s:LectureUnit"/>
  </rdf:Description>

  <rdf:Description ID="parentCourse">
    <rdf:type resource="rdf:Property"/>
    <rdfs:range resource="s:Lecture"/>
    <rdfs:domain resource="s:LectureUnit"/>
  </rdf:Description>

  <rdf:Description ID="theoryPage">
    <rdf:type resource="rdf:Property"/>
    <rdfs:range resource="s:TheoryUnit"/>
    <rdfs:domain resource="s:LectureUnit"/>
  </rdf:Description>

  <rdf:Description ID="parentUnit">
    <rdf:type resource="rdf:Property"/>
    <rdfs:range resource="s:LectureUnit"/>
    <rdfs:domain resource="s:TheoryUnit"/>
```

```

</rdf:Description>

<rdf:Description ID="TheoryUnit">
  <rdf:type resource="rdfs:Class"/>
</rdf:Description>

<rdf:Description ID="AIIecture">
  <rdf:type resource="s:Lecture"/>
</rdf:Description>

<rdf:Description ID="LectureUnit1">
  <rdf:type resource="s:LectureUnit"/>
  <title>Lecture Unit 1</title>
  <description>
    Introduction to intelligent agents
  </description>
  <parentCourse resource="s:AIIecture"/>
  <theoryPage
    resource="http://.../Definitions.htm"/>
  <theoryPage
    resource="http://.../Characterisation.htm"/>
  <theoryPage
    resource="http://.../Structure.htm"/>
  <theoryPage resource="http://.../Types.htm"/>
</rdf:Description>

<rdf:Description
  about="http://.../Definitions.htm">
  <rdf:type resource="s:TheoryUnit"/>
  <parentUnit resource="s:LectureUnit1"/>
</rdf:Description>

<rdf:Description
  about="http://.../Characterisation.htm">
  <rdf:type resource="s:TheoryUnit"/>
  <parentUnit resource="s:LectureUnit1"/>
</rdf:Description>

<rdf:Description
  about="http://.../Structure.htm">
  <rdf:type resource="s:TheoryUnit"/>
  <parentUnit resource="s:LectureUnit1"/>
</rdf:Description>

<rdf:Description about="http://.../Types.htm">
  <rdf:type resource="s:TheoryUnit"/>
  <parentUnit resource="s:LectureUnit1"/>
</rdf:Description>
</rdf:RDF>

```

In O-Telos-RDF, this example looks as follows:

```

<otelos:OTELS xml:lang="en"
  xmlns:otelos="http://www.kbs.uni-hannover.de
    /otelos/2001/06/otelos-rdf-schema#"
  xmlns:t="http://www.kbs.uni-hannover.de/otelos/
    /2001/06/example-otelos-schema#">

  <otelos:Description ID="Lecture"/>

  <otelos:Description ID="LectureUnit">
    <title s=otelos:literal/>
    <description s=otelos:literal/>
    <parentCourse s=t:Lecture/>
    <theoryPage s=t:TheoryUnit/>
  </otelos:Description>

```

```

<otelos:Description ID="TheoryUnit">
  <parentUnit s="t:LectureUnit"/>
</otelos:Description>

<otelos:Description ID="AIIecture">
  <type s="t:Lecture"/>
</otelos:Description>

<otelos:Description ID="LectureUnit1">
  <type s="t:LectureUnit"/>
  <title>Lecture Unit 1</title>
  <description>
    Introduction to intelligent agents
  </description>
  <parentCourse s="t:AIIecture"/>
  <theoryPage1 s="http://.../Definitions.htm"/>
  <theoryPage2
    s="http://.../Characterisation.htm"/>
  <theoryPage3 s="http://.../Structure.htm"/>
  <theoryPage4 s="http://.../Types.htm"/>
</otelos:Description>

<otelos:Description ID="LectureUnit1_title">
  <type s="t:LectureUnit_title"/>
</otelos:Description>

<otelos:Description
  ID="LectureUnit1_description">
  <type s="t:LectureUnit_description"/>
</otelos:Description>

<otelos:Description
  ID="LectureUnit1_parentCourse">
  <type s="t:LectureUnit_parentCourse"/>
</otelos:Description>

<otelos:Description
  ID="LectureUnit1_theoryPage1">
  <type s="t:LectureUnit_theoryPage"/>
</otelos:Description>

<otelos:Description
  ID="LectureUnit1_theoryPage2">
  <type s="t:LectureUnit_theoryPage"/>
</otelos:Description>

<otelos:Description
  ID="LectureUnit1_theoryPage3">
  <type s="t:LectureUnit_theoryPage"/>
</otelos:Description>

<otelos:Description
  ID="LectureUnit1_theoryPage4">
  <type s="t:LectureUnit_theoryPage"/>
</otelos:Description>

<otelos:Description
  about="http://.../Definitions.htm">
  <type s="t:TheoryUnit"/>
  <parentUnit s="t:LectureUnit1"/>
</otelos:Description>

<otelos:Description
  about="http://.../Characterisation.htm">
  <type s="t:TheoryUnit"/>
  <parentUnit s="t:LectureUnit1"/>

```

```

</otelos:Description>

<otelos:Description
  about="http://.../Structure.htm">
  <type s="t:TheoryUnit"/>
  <parentUnit s="t:LectureUnit1"/>
</otelos:Description>

<otelos:Description
  about="http://.../Types.htm">
  <type s="t:TheoryUnit"/>
  <parentUnit s="t:LectureUnit1"/>
</otelos:Description>

</otelos:OTELOS>

```

Membership in `otelos:individual`, `otelos:property`, `otelos:type` etc. is automatically based on the syntactic form of the statements, so we do not declare any explicit type-statements for these classes.

All Web Pages are explicitly defined by statements. Note, that if we want to avoid doing this in the XML description, an intelligent parser could automatically generate these statements for each URL in the XML text file. Similarly, we could define “resource=” as a synonym for “s=” to retain current RDF/XML serialization as much as possible.

Another noticeable difference are the explicit declarations of the type property for the properties of the instance “Lecture-Unit1”. In contrast to RDF(S) the axiomatization of O-Telos-RDF enforces these explicit type declaration. This ensures that property-statements either instantiate the properties declared by their class(es) or declare new properties.

These differences are the basis for O-Telos-RDF’s capability of more expressive metamodelling and reification. Furthermore, we also see that O-Telos-RDF descriptions are class centric while RDF(S) descriptions are property centric.

The statements of the RDF(S)-representation are generated by the SiRPac [11] parser. The prefix `online:` represents constructs without an URI, `genid` represents automatically generated IDs

(Note that we replaced the URIs in the example by the respective namespaces in order to improve the readability.):

Nr.	Statement (subject,predicate,object)
1	s(online:Lecture,rdf:type,rdfs:Class)
2	s(online:LectureUnit,rdf:type,rdfs:Class))
3	s(online:title,rdf:type,rdf:Property))
4	s(online:genid2,resource,rdfs:Literal)
5	s(online:title,rdfs:range,online:genid2)
6	s(online:genid5,resource,s:LectureUnit)
7	s(online:title,domain,online:genid5)
8	s(online:description,type,rdf:Property)
9	s(online:genid8,resource,rdfs:Literal)
10	s(online:description,range,online:genid8)
11	s(online:genid11,resource,s:LectureUnit)
12	s(online:description,domain,online:genid11)
13	s(online:parentCourse,type,rdf:Property)
14	s(online:genid14,resource,s:Lecture)
15	s(online:parentCourse,range,online:genid14)
16	s(online:genid17,resource,s:LectureUnit)
17	s(online:parentCourse,domain,online:genid17)
18	s(online:parentUnit,type,rdf:Property)
19	s(online:genid20,resource,s:LectureUnit)
20	s(online:parentUnit,range,online:genid20)
21	s(online:genid23,resource,s:TheoryUnit)

Nr.	Statement (subject,predicate,object)
22	s(online:parentUnit,domain,online:genid23)
23	s(online:theoryPage,type,rdf:Property)
24	s(online:genid26,resource,s:TheoryUnit)
25	s(online:theoryPage,range,online:genid26)
26	s(online:genid29,resource,s:LectureUnit)
27	s(online:theoryPage,domain,online:genid29)
28	s(online:TheoryUnit,type,rdfs:Class)
29	s(online:AILecture,type,s:Lecture)
30	s(online:LectureUnit1,type,s:LectureUnit)
31	s(online:LectureUnit1,online:title,Lecture Unit 1)
32	s(online:LectureUnit1,online:description, Introduction to intelligent agents)
33	s(online:genid34,resource,s:AILecture)
34	s(online:LectureUnit1,online:parentCourse, online:genid34)
35	s(online:genid37,resource,http://.../Definitions.htm)
36	s(online:LectureUnit1,online:theoryPage, online:genid37)
37	s(online:genid40,resource, http://.../Characterisation.htm)
38	s(online:LectureUnit1,online:theoryPage, online:genid40)
39	s(online:genid43,resource,http://.../Structure.htm)
40	s(online:LectureUnit1,online:theoryPage, online:genid43)
41	s(online:genid46,resource,http://.../Types.htm)
42	s(online:LectureUnit1,online:theoryPage, online:genid46)
43	s(http://.../Definitions.htm,type,s:TheoryUnit)
44	s(online:genid50,resource,s:LectureUnit1)
45	s(http://.../Definitions.htm,online:parentUnit, online:genid50)
46	s(http://.../Characterisation.htm,type,s:TheoryUnit)
47	s(online:genid54,resource,s:LectureUnit1)
48	s(http://.../Characterisation.htm,online:parentUnit, online:genid54)
49	s(http://.../Structure.htm,type,s:TheoryUnit)
50	s(online:genid58,resource,s:LectureUnit1)
51	s(http://.../Structure.htm,online:parentUnit, online:genid58)
52	s(http://.../Types.htm,type,s:TheoryUnit)
53	s(online:genid62,resource,s:LectureUnit1)
54	s(http://.../Types.htm,online:parentUnit, online:genid62)

The O-Telos-RDF statements generated from the XML-representation are appended below. For simplicity, statement IDs are only referenced symbolically, but they can be derived from the rules stated in the earlier sections.

```

s(sid1,sid1,Lecture,sid1)

s(sid2,sid2,LectureUnit,sid2)
s(sid3,sid2,title,otelos:literal)
s(sid4,sid2,description,otelos:literal)
s(sid5,sid2,parentCourse,sid1)
s(sid6,sid2,theoryPage,sid7)

s(sid7,sid7,TheoryUnit,sid7)
s(sid8,sid7,parentUnit,sid2)

s(sid9,sid9,AILecture,sid9)
s(sid10,sid9,type,sid1)

```

```

s(sid11,sid11,LectureUnit1,sid11)
s(sid12,sid11,type,sid2)
s(sid13,sid11,title,"Lecture Unit 1")
s(sid14,sid13,type,sid3)
s(sid15,sid11,description,
  "Introduction to intelligent agents")
s(sid16,sid15,type,sid4)
s(sid17,sid11,parentCourse,sid9)
s(sid18,sid17,type,sid5)
s(sid19,sid11,theoryPage1,sid27)
s(sid20,sid19,type,sid6)
s(sid21,sid11,theoryPage2,sid31)
s(sid22,sid21,type,sid6)
s(sid23,sid11,theoryPage3,sid35)
s(sid24,sid23,type,sid6)
s(sid25,sid11,theoryPage4,sid39)
s(sid26,sid25,type,sid6)

s(sid27,sid27,"http://.../Definition.htm",sid27)
s(sid28,sid27,type,sid7)
s(sid29,sid21,parentUnit,sid11)
s(sid30,sid29,type,sid8)

s(sid31,sid31,
  "http://.../Characterisation.htm",sid31)
s(sid32,sid31,type,sid7)
s(sid33,sid31,parentUnit,sid11)
s(sid34,sid33,type,sid8)

s(sid35,sid35,"http://.../Structure.htm",sid35)
s(sid36,sid35,type,sid7)
s(sid37,sid35,parentUnit,sid11)
s(sid38,sid37,type,sid8)

s(sid39,sid39,"http://.../Types.htm",sid39)
s(sid40,sid39,type,sid7)
s(sid41,sid39,parentUnit,sid11)
s(sid42,sid41,type,sid8)

```

Note also, that we left out the statements that all literals are instances of `otelos:literal`. Similarly, we left out, that all type statements are instances of `otelos:type`, all individuals are instances of `otelos:individual` and all properties are instances of `otelos:property`. These statements can be deduced from the respective axioms. Furthermore the statement-IDs of each statement are generated automatically so that they are not stated in the XML-representation. If necessary, the ID of each statement can be generated easily by the user. This is necessary if one wants to make a statement about another statement.

### 3. DISCUSSION

In this paper we described the formalization of an RDF(S) variant called O-Telos-RDF, which provides enhanced functionalities for meta-modeling and reified statements. The formalization is based very closely on the formalization of the modeling language O-Telos, which is based on a semantic network model similar to RDF(S). Axioms and constraints are clearly defined. In the longer report, we include all these axioms in an appendix, together with their counterparts in O-Telos and RDF(S). This exact formalization will hopefully contribute to an understanding of both O-Telos-RDF and (in this context) the current RDF(S) formalization.

Compared with RDF, O-Telos-RDF shows its advantages in allowing easier reification of statements, and in metamodelling applications, where more than three abstraction hierarchies are needed.

Furthermore, the class centric approach to property definition allows the definition of properties with the same name for different domains, which have different ranges (not possible in RDF(S)).

### 4. REFERENCES

- [1] J. R. Abrial. Data semantics. In Klimbie and Koffeman, editors, *Data Base Management*. North-Holland Publ., 1974.
- [2] T. Berners-Lee, R. Fielding, U.C. Irvine, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. In *Internet Official Protocol Standards (STD 1)*, RFC. The Internet Society, rfc 2396 edition, 1998. <http://www.ietf.org/rfc2396.txt>.
- [3] D. Brickley and R. V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Technical report, World Wide Web Consortium (W3C), 2000. <http://www.w3.org/TR/rdf-schema>.
- [4] Wolfram Conen and Reinhold Klapsing. A logical interpretation of RDF. In *Linköping Electronic Articles in Computer and Information Science, Semantic Web Area*, volume 5. Linköping University Electronic Press, December 2000. <http://www.ep.liu.se/ea/cis/2000/013/>.
- [5] W3C Working Group. W3C Resource Description framework (RDF) Model and Syntax Specification. <http://www.w3.org/TR/REC-rdf-syntax/>, February 1999.
- [6] M. Jarke, R. Gellersdörfer, M. Jeusfeld, M. Staudt, and S. Eherer. ConceptBase - a deductive object base for meta data management. *Journal on Intelligent Information Systems*, 4(2):167 – 192, 1995.
- [7] M. Jeusfeld. *Änderungskontrolle in deduktiven Objektbanken*. Infix-Verlag, St. Augustin, Deutschland, 1992.
- [8] M. Jeusfeld and ConceptBase Team. Application papers. Technical report, Chair of Computer Science V - Information Systems, RWTH Aachen, 2000. <http://www-i5.informatik.rwth-aachen.de/CBdoc/cblit.html#cb-app>.
- [9] B.M. Kramer, V.K. Chandhri, M. Koubarakis, T. Topaloglon, H. Wang, and J. Mylopoulos. Implementing Telos. *SIGART Bulletin*, 2(3), June 1991.
- [10] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: A language for representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4), 1990.
- [11] Janne Saarela. *SiRPAC - simple RDF parser & compiler*. World Wide Web Consortium (W3C), 2001. <http://w3c.org/RDF/Implementations/SiRPAC>.
- [12] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3), September 1941.

## APPENDIX

### A. BASIC XML-SERIALISATION

The basic XML-serialisation stated here resembles very closely the basic XML-serialisation of RDF, given in the RDF Syntax and Model Specification [5], in order to necessitate only minor changes in current RDF parsers. It is used to express the O-Telos-RDF statements in XML, and in its current form takes care of the primitive types “Literal” and “Statement” (and their subclasses). Its main purpose is the grouping of multiple statements for the same resource into a description element using XML syntax.

As in the RDF Syntax and Model Specification the OTELOS element is a simple wrapper that marks the beginning and end of O-Telos-RDF statements in an XML document.

The RDF	EBNF	of the basic O-Telos-serialisation looks as follows:
1)	OTELOS	::= [ <code>&lt;otelos:OTELOS&gt;</code> ] description* [ <code>&lt;/otelos:OTELOS&gt;</code> ]
2)	description	::= <code>&lt;otelos:Description</code> idAboutAttr? <code>&gt;</code> propertyElt* <code>&lt;/otelos:Description&gt;</code>   <code>&lt;otelos:Description</code> idAboutAttr? <code>/ &gt;</code>
3)	idAboutAttr	::= idAttr   aboutAttr
4)	aboutAttr	::= <code>'about='</code> statement-reference <code>''''</code>
5)	idAttr	::= <code>'ID='</code> IDsymbol <code>''''</code>
6)	propertyElt	::= <code>&lt;</code> label <code>&gt;</code> value <code>&lt;/</code> label <code>&gt;</code>   <code>&lt;</code> label statementAttr <code>/&gt;</code>
7)	label	::= name
8)	value	::= description   string
9)	statement-Attr	::= <code>'s='</code> statement-reference <code>''''</code>
10)	statement-reference	::= string, interpreted as statement-ID, includes NSprefix
11)	IDsymbol	::= (any legal XML name symbol)
12)	name	::= (any legal XML name symbol)
13)	NSprefix	::= (any legal XML namespace prefix)
14)	string	::= (any XML text, with <code>&lt;</code> , <code>&gt;</code> , and <code>&amp;</code> escaped)