Typed Lists in OWL and SADL

Andrew Crapo

March 2018
GE Global Research, Niskayuna, NY 12309, USA

Abstract. Typed lists are currently missing from the standardized structures of the Web Ontology Language (OWL). However, the need to give order to a group of instances or data values is ubiquitous in model building, and it is useful to be able to specify the type of members that are expected to belong to such an ordering. Typed lists are easily supported using OWL constructs once the empty list terminator used in the Resource Description Framework (RDF) untyped list construct is rejected. This paper explains why such a terminator is undesirable, describes a straightforward implementation of typed lists in OWL and in the controlled-English Semantic Application Design Language (SADL), and illustrates with examples.

Keywords: Ontology; knowledge capture; semantic models.

1 Introduction

A list is an ordered set. The need for lists when building models is ubiquitous and can be driven by a variety of objectives including maintaining temporal or spatial ordering of things in the modeled world. For example, an aircraft's flight plan is not just a set of locations. It is a set of locations that are target destinations in a specific, geographic-based order. Similarly, the steps in a procedure must often be executed in the specified sequence; it isn't wise to bake the cake before combining the ingredients.

The basic building blocks of the Web Ontology Language (OWL) [1] do not explicitly support ordering. For example, an ontology might define the property *child* whose domain and range are the *Person* class. After defining *George*, *Mary*, *John*, and *Susan* as instances of the *Person* class, one might make the following statements:

- *George child Mary*
- *George child John*
- *George child Susan*

How could one best capture the children in age-order? One solution, of course, is to use a list. It would be nice if the list were typed, meaning that it is expected that all of the elements of the list will be of the specified type. Then we could, for example, define a property *children* with range *Person List*. In this paper, we demonstrate one approach accomplishing this objective. We express our models in the Semantic Application Design Language (SADL) [2,3], which translates directly to OWL.[1]

2 Prior Art

The Resource Description Framework (RDF) [4] includes the *rdf:List* construct [5], which is generally avoided in OWL models because *rdf:List* is used in the OWL serialization and using it in a domain model renders that ontology OWL full [6]. It is instructive to examine the structure of *rdf:List* as many Semantic Web approaches to typed lists take a similar approach, as will ours with one important exception.

The basic approach of *rdf:List* is that for each list element there is an unnamed instance of *rdf:List* with a *first* property pointing to the element and a *rest* property pointing to another instance of *rdf:List*, which constitutes the rest of the list. The list ends with an instance of *rdf:List* with *first* pointing to the last member of the list and *rest* pointing to the terminator *rdf:nil*, a special instance of *rdf:List*. This is illustrated in Fig. 1. The dark blue nodes are blank nodes, instances of *rdf:List*. References to the first (or any) instance of *rdf:List* would use the internal identifier of the first dark blue [blank] node.

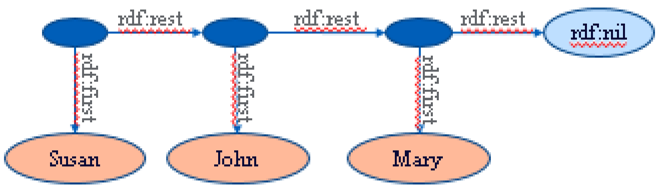

Fig. 1. Illustration of the *rdf:List* construct.

3 The Case Against the Terminator

The use of the special terminator, *rdf:nil*, is reminiscent of the ubiquitous *null* used in relational databases. However, in the graph-based modeling approach of RDF and OWL, it is an anomaly and, upon careful examination, is found to be problematic. Referring to the example in the introduction, using *rdf:nil* to terminate a list is analogous to saying that *Susan* has no children with the statement

Susan child nil

Since the range of *child* is *Person*, we may infer that *nil* is an instance of *Person*, but a special instance which stands for "there-is-no-Person who is child of Susan". If one takes the approach of adding a triple to the graph for each subject and property for which there is no value, it follows that every class that is the range of a property must have a member *nil*. But is that the same *nil* for all classes or a different *nil* for each

[1] SADL currently translates to OWL 1 plus qualified cardinality. OWL 2 support is limited because SADL uses Apache Jena.

class? And more fundamentally, since we are building on set theory, what does it mean to say that every set (class) contains a member which is used to indicate that "there is no member"? If it doesn't exist, how can it be a member of the set? And if it does exist, and is a member of the class, then it should conform to all of the axioms of the class, e.g., if *Person* must have a *birthdate* then the *nil* member of the *Person* class must have a *birthdate*. Or perhaps *nil* is the same as *owl:Nothing*? But *owl:Nothing* is a class, the empty set, not an instance of every class, i.e., an instance of *owl:Thing*.

Such an approach would get even worse if we think about properties of type *owl:DatatypeProperty*. Suppose we have a genealogy ontology with the property *deathDate*, domain *Person*, range *xsd:date*. For every living person, we would need to insert into our graph model a statement like

George deathDate nil

Every living *Person* known to the model would presumably have such a statement. Does that mean that all of the living people have the same deathDate? A reasoner might be inclined to so reason.

One of the nice properties of graph models is that we don't have to add nodes to say that there is no node. In other words, we do not put things into our graph model to indicate that there isn't anything in that part of the graph model. Why should it be any different for lists? We do not need a terminator for each list construct. The fact that the last blank node in the list is not the subject of a triple with *rest* as predicate is sufficient, just as it is sufficient that an instance of a living *Person* is not the subject of a triple with predicate *deathDate*.

Note that in model-driven user interfaces, we sometimes populate menus from the model, and may wish to have *None* as a menu choice indicating that we will not insert a triple for this subject and property into the graph store, or that we wish to remove any existing triple(s) for the identified subject and property. In this situation, *None* is not an instance of the property's range but rather is an instruction to the model manager. It is important to distinguish instructions about what to insert or remove or not insert from what is actually inserted into the graph model. Similarly, we may wish to have some means of expressing, in graph patterns in queries or rules, the condition that a particular triple pattern does not exist in the model. In the query or rule language one might choose to express this in an English-like representation as one of the following, using the grammar keywords *None* or *known*.

Susan child None
child of Susan is None
Susan child not known
child of Susan is not known

Once again, however it is expressed, creating a triple pattern with a given subject and predicate and a special object indicating that the pattern does not exist in the graph model is not at all the same as inserting a triple into the graph model that means that the triple isn't in the model.

4 Creating an Extensible Typed List Class in SADL

Eliminating the special *nil* terminator and following the normal approach of not putting something in the graph if it isn't in the graph goes most of the way towards enabling typed lists in in decidable flavors of OWL. We define a List class as follows:[2]

```
uri "http://sadl.org/sadllistmodel" alias sadllist.

List is a class
   described by ^first,
   described by rest with values of type List,
   described by lengthRestriction with values of type int,
   described by minLengthRestriction with values of type int,
   described by maxLengthRestriction with values of type int.
```

Fig. 2. Definition of sadllist:List class, super class of all typed lists.

The property *first* is an *rdf:Property* so its values may be either instances of classes or primitive data values—numbers, strings, dates, etc. The property *rest* is an *owl:ObjectProperty* and follows in the tradition of *rdf:rest,* pointing to the next List instance. The additional properties allow us to specify constraints on the number of elements that a list may contain[3].

In SADL syntax, a typed list is specified by placing the keyword *List* after the type. Typed lists may be named or unnamed. In either case, a class is created which is a subclass of *sadllist:List*, the class defined above in Fig 2. The class has a property restriction on *first* that all of its values must come from the type of the list. It has a property restriction on *rest* that all of its values must come from the class itself, this specific subclass of *sadllilst:List*.

Table 1 shows an example of an unnamed, typed list specified as the range of a property. The SADL statement is shown in the first row followed by the equivalent OWL in the second row. The unnamed list of type *Person* is identified in OWL with the rdf:nodeID *A0*. Values of *first* are restricted to instances of the *Person* class. Values of *rest* are restricted to the class locally identified as *A0*. Note that the example of Table 1 places the list in context because unnamed typed lists only exist in some larger context.

[2] The List class definition is expressed in SADL.

[3] See http://sadl.sourceforge.net/sadl3/SadlConstructs.html#TypedLists for more information.

Table 1. Unnamed typed list as object property range.

children describes **Person** with values of type **Person** List.
```
<owl:ObjectProperty rdf:ID="children">
   <rdfs:range rdf:nodeID="A0"/>
   <rdfs:domain rdf:resource="#Person"/>
</owl:ObjectProperty>
<owl:Class rdf:nodeID="A0">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:allValuesFrom rdf:nodeID="A0"/>
         <owl:onProperty rdf:resource="sadllist#rest"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:allValuesFrom rdf:resource="#Person"/>
         <owl:onProperty rdf:resource="sadllist#first"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf rdf:resource="sadllist#List"/>
</owl:Class>
```

Table 2 shows an example of a named typed list with a minimum length restriction. In the SADL syntax, if only a minimum length is specified, an “*” must be used to indicate that there is no maximum length. As far as the OWL representation of the list is concerned, there are two differences between Table 1 and Table 2 First the subclass of *sadllist:List* in Table 2 has an *rdf:ID* containing the user-specified class name, e.g., *ChildrenList*. Secondly, the class has an additional *owl:hasValue* restriction on property *sadllist:lengthMinRestriction*, capturing the minimum length of *1*.

Table 2. Named typed list.

ChildrenList is a type of **Person** List length 1-*.

```
   <owl:Class rdf:ID="ChildrenList">
     <rdfs:subClassOf>
       <owl:Restriction>
         <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
         >1</owl:hasValue>
         <owl:onProperty rdf:resource="sadllist#lengthMinRestriction"/>
       </owl:Restriction>
     </rdfs:subClassOf>
    <rdfs:subClassOf>
     <owl:Restriction>
       <owl:allValuesFrom rdf:resource="# ChildrenList "/>
       <owl:onProperty rdf:resource="sadllist#rest"/>
     </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
     <owl:Restriction>
       <owl:allValuesFrom>
         <owl:Class rdf:ID="Person"/>
       </owl:allValuesFrom>
       <owl:onProperty rdf:resource="sadllist#first"/>
     </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="sadllist#List"/>
</owl:Class
```

The examples in Tables 1 and 2 restrict all members of the List subclass to come from a user-defined class. However, the members of the list could come from a primitive data type such as *xsd:string*, *xsd:date*, etc.. In Table 3 we illustrate this point with an unnamed list of type *xsd:int*, again expressed in the context of a property range.

Table 3. Named typed list as datatype property range.

Test is a class, described by **grades** with values of type int List.
`<owl:ObjectProperty rdf:ID="grades"> <rdfs:domain rdf:resource="#Test"/> <rdfs:range rdf:nodeID="A2"/> </owl:ObjectProperty> <owl:Class rdf:nodeID="A2"> <rdfs:subClassOf> <owl:Restriction> <owl:allValuesFrom rdf:nodeID="A2"/> <owl:onProperty rdf:resource="sadllist#rest"/> </owl:Restriction> </rdfs:subClassOf> <rdfs:subClassOf> <owl:Restriction> <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#int"/> <owl:onProperty rdf:resource="sadllist#first"/> </owl:Restriction> </rdfs:subClassOf> <rdfs:subClassOf rdf:resource="sadllist#List"/> </owl:Class>`

```
<owl:ObjectProperty rdf:ID="grades">
   <rdfs:domain rdf:resource="#Test"/>
   <rdfs:range rdf:nodeID="A2"/>
</owl:ObjectProperty>
<owl:Class rdf:nodeID="A2">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:allValuesFrom rdf:nodeID="A2"/>
         <owl:onProperty rdf:resource="sadllist#rest"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:allValuesFrom rdf:re-
source="http://www.w3.org/2001/XMLSchema#int"/>
         <owl:onProperty rdf:resource="sadllist#first"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf rdf:resource="sadllist#List"/>
</owl:Class>
```

Notice that although the type of the list is *xsd:int*, the property whose range is the *int List* is an *owl:ObjectProperty* since its range is the *Test* class.

Table 4 shows a named list of type *xsd:int*. The OWL representation of the *List* subclass is very similar to the construct of Table 2, differing only in the restrictions on *first* and *rest*, and, of course, the class name.

Table 4. Named list of type xsd:int.

Grades is a type of int List.

```
<owl:Class rdf:ID="Grades">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:allValuesFrom rdf:resource="#Grades"/>
         <owl:onProperty rdf:resource="sadllist#rest"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:allValuesFrom rdf:re-
source="http://www.w3.org/2001/XMLSchema#int"/>
        <owl:onProperty rdf:resource="sadllist#first"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf rdf:resource="sadllist#List"/>
</owl:Class>
```

So far we have only given examples of *sadllist:List* subclasses. Of course, typed lists are only useful if we can actually construct and use them. Table 5 shows an example of creating an instance of an unnamed typed list and populating it with members of the specified type. The definition of the unnamed, typed list *A0* is omitted as it is exactly the same as in Table 1.

Table 5. Instance of an unnamed typed list with members (list definition as in Table 1).

JoesChildren is the **Person** List [John, Sue, Craig].

```
<rdf:Description rdf:ID="JoesChildren">
   <rdf:type rdf:nodeID="A0"/>
   <sadllist:first rdf:resource="#John"/>
   <sadllist:rest rdf:parseType="Resource">
      <rdf:type rdf:nodeID="A0"/>
      <sadllist:first rdf:resource="#Sue"/>
      <sadllist:rest rdf:parseType="Resource">
         <rdf:type rdf:nodeID="A0"/>
         <sadllist:first rdf:resource="#Craig"/>
      </sadllist:rest>
   </sadllist:rest>
</rdf:Description>
```

The list in OWL is very similar if it is an instance of a named subclass of List, as shown in Table 6.

Table 6. Instance of a named typed list with members.

```
JoesChildrenAlt is the ChildrenList [John, Sue, Craig].
```

```
<TestList:ChildrenList rdf:ID="JoesChildrenAlt">
   <sadllist:first rdf:resource="#John"/>
   <sadllist:rest>
      <TestList:ChildrenList>
         <sadllist:first rdf:resource="#Sue"/>
         <sadllist:rest>
            <TestList:ChildrenList>
               <sadllist:first rdf:resource="#Craig"/>
            </TestList:ChildrenList>
         </sadllist:rest>
      </TestList:ChildrenList>
   </sadllist:rest>
</TestList:ChildrenList>
```

Note that each SADL declaration of an unnamed subclass of *sadllist:List* of a particular type, e.g., *Person List*, results in the creation of a separate class declaration. Without an identifying URI, it is not possible, in theory, to conclusively determine if these subclasses are the same class or different classes. Of course, if two unnamed lists of the same type have different length restrictions then we can know that they are not the same class. However, it would be possible, in translating from SADL to OWL, to assume that unnamed, typed lists without length restrictions or with the same length restrictions are the same class because the scope would be limited to the current model, by virtue of being blank nodes, and because any changes to the SADL file which modified the list in one location but not in another would result in retranslation, after which the assumption would not hold and they would be different classes.

As a final example, consider an instance of the *Grades* class defined in Table 4. The list, shown in Table 7, has 4 members, each of which is of type xsd:int as required in the Table 4 definition of the class. Note that the order of the members of the list is determined by the relationships of the graph model, not by the order in which the values appear in the serialization.

Table 7. Example of a named instance of the named sadllist:List subclass Grades.

```
Test3Grades is the Grades [95,86,67,99].
```

```
<Test:Grades rdf:ID="Test3Grades">
   <sadllist:rest>
      <Test:Grades>
         <sadllist:rest>
            <Test:Grades>
               <sadllist:rest>
                  <Test:Grades>
                     <sadllist:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">99</sadllist:first>
                  </Test:Grades>
               </sadllist:rest>
               <sadllist:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">67</sadllist:first>
            </Test:Grades>
         </sadllist:rest>
         <sadllist:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">86</sadllist:first>
      </Test:Grades>
   </sadllist:rest>
   <sadllist:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">95</sadllist:first>
</Test:Grades>
```

5 Conclusion

SADL supports typed lists, both named and unnamed, using OWL DL constructs. The crux of the SADL implementation of typed lists in a decidable fragment of OWL is the absence of an equivalent to the *rdf:nil* terminator. This allows the definition of a *sad-llist:Li*st class which can be subclassed and given property restrictions to generate named or unnamed lists whose members are constrained by restriction to be from any specified class or data type. In addition, the SADL implementation adds properties that allow the minimum, maximum, or exact number of list element expected to be present in an instance of the *sadllist:List* subclass to be specified.

References

1. Smith, M. K., Welty, C. and McGuinness, D. L., editors.: OWL Web Ontology Language Guide, https://www.w3.org/TR/owl-guide/, last accessed 2018/04/04.
2. Crapo, A.: Semantic Application Design Language (SADL), http://sadl.sourceforge.net/, last accessed 2018/04/04.
3. Crapo, A. and Moitra, A. Toward a Unified English-like Representation. International Journal of Semantic Computing, *7*(3), 215-236 (2013).
4. Hayes, P. J. and Patel-Schneider, P. F., editors. RDF 1.1 Semantics, https://www.w3.org/TR/rdf11-mt/, last accessed 2018/04/04.
5. Brickley, D. and Guha, R., editors. RDF Schema 1.1, https://www.w3.org/TR/rdf-schema/#ch_collectionvocab, last accessed 2018/04/04.
6. Drummond, N., Rector, A., Moulton, G., Stevens, R., Horridge, M., Wang, H., and Seidenberg, J. Sequences in OWL (2006), https://protege.stanford.edu/conference/2006/submissions/slides/7.1_Drummond.pdf, last accessed 2018/04/04.