Defining N-ary Relations on the Semantic Web

W3C Working Group Note 12 April 2006

This version:
http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/
Latest version:
http://www.w3.org/TR/swbp-n-aryRelations
Previous version:
http://www.w3.org/TR/2004/WD-swbp-n-aryRelations-20040721/
Editors:
Natasha Noy, Stanford University
Alan Rector, University of Manchester
Contributors:
Pat Hayes, IHMC
Chris Welty, IBM Research

Also see Acknowledgements.

Copyright © 2006 W3C® (MIT, ERCIM, Keio), All Rights Reserved. W3C liability, trademark and document use rules apply.

Abstract

In Semantic Web languages, such as RDF and OWL, a property is a *binary* relation: it is used to link two individuals or an individual and a value. However, in some cases, the natural and convenient way to represent certain concepts is to use relations to link an individual to more than just one individual or value. These relations are called *n-ary relations*. For example, we may want to represent properties of a relation, such as our certainty about it, severity or strength of a relation, relevance of a relation, and so on. Another example is representing relations among multiple individuals, such as a buyer, a seller, and an object that was bought when describing a purchase of a book. This document presents ontology patterns for representing n-ary relations in RDF and OWL and discusses what users must consider when choosing these patterns.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document is a Working Group Note, produced by the Semantic Web Best Practices and Deployment Working Group, part of the W3C Semantic Web Activity. This document is one of a set of documents providing an introduction and overview of ontology design patterns produced by the SWBPD Working Group's Ontology Engineering and Patterns Task Force.

As of the publication of this Working Group Note the SWBPD Working Group has completed work on this document. Changes from the previous Working Draft are summarized in an appendix. Comments on this document may be sent to public-swbp-wg@w3.org, a mailing list with a public archive. Further discussion on this material may also be sent to the Semantic Web Interest Group mailing list, semantic-web@w3.org, also with a public archive.

Publication as a Working Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. This document is informative only. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

General issues

In Semantic Web languages, such as RDF and OWL, a property is a *binary* relation: instances of properties link two individuals. Often we refer to the second individual as the "value" or to both both individuals as "arguments" [See note on vocabulary].

Issue 1: If property instances can link only two individuals, how do we deal with cases where we need to *describe* the instances of relations, such as its certainty, strength, etc?

Issue 2: If instances of properties can link only two individuals, how do we represent relations among more than two individuals? ("n-ary relations")

Issue 3: If instances of properties can link only two individuals, how do we represent relations in which one of the participants is an ordered list of individuals rather than a single individual?

The solutions to the first two problems are closely linked; the third problem is fundamentally different, although it can be adapted to meet issue one in special cases. Note that we don't use RDF reification in these patterns; the reasons for this decision are discussed in the final section.

Data descriptions used in this document

The data format used in this document is Turtle [Turtle], used to show each triple explicitly. Turtle allows URIs to be abbreviated with prefixes:

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
:book1  dc:title  "Defining N-ary Relations on the Semantic Web" .
```

Use case examples

Several common use cases fall under the category of n-ary relations. Here are some examples:

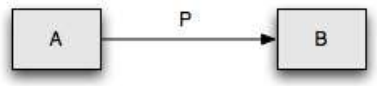
1. *Christine has breast tumor with high probability*. There is a binary relation between the person `Christine` and diagnosis `Breast_Tumor_Christine` and there is a qualitative probability value describing this relation (`high`).
2. *Steve has temperature, which is high, but falling*. The individual `Steve` has two values for two different aspects of a `has_temperature` relation: its `magnitude` is `high` and its `trend` is `falling`.
3. *John buys a "Lenny the Lion" book from books.example.com for $15 as a birthday gift.* There is a relation, in which individual `John`, entity `books.example.com` and the book `Lenny_the_Lion` participate. This relation has other components as well such as the purpose (`birthday_gift`) and the amount (`$15`).
4. *United Airlines flight 3177 visits the following airports: LAX, DFW, and JFK.* There is a relation between the individual flight and the three cities that it visits, `LAX`, `DFW`, `JFK`. Note that the order of the airports is important and indicates the order in which the flight visits these airports.

Another way to think about the use cases is how they might occur in the evolution of an ontology.

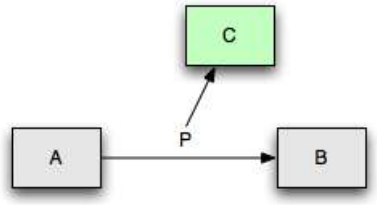
1. We discover that a relation that we thought was binary, really needs a further argument - a common origin of use case 1.
2. We discover that two binary properties always go together and should be represented as one n-ary relation - a common origin for use case 2
3. From the beginning, we realize that the relation is really amongst several things - a common origin for use case 3
4. The nature of the relation is such that one or more of the arguments is fundamentally a sequence rather than a single individual - use case 4.

Representation patterns

As we describer earlier, in Semantic Web Languages, properties are binary relations. Each instance of a property links an individual to another individual or a value as shown below.



We would like to have another individual or simple value `C` to be part of this relation instance:



`'P'` now refers to an instance of a relation among `'A'`, `'B'`, and `'C'`. (There might be other individuals '`D`', '`E`', and '`F`'. However, for simplicity, we will illustrate most of our use cases assuming a single additional individual. We can handle more individuals in exactly the same way.)

One common solution to this problem (pattern 1) is to represent the relation as a class rather than a property. Individual instances of such classes correspond to instances of the relation. Additional properties provide binary links to each argument of the relation. We can model examples 1, 2, and 3 above using this pattern. For instance, in the example 1 the instance of a new class `Diagnosis_Relation` would represent the fact that Christine has been diagnosed with a breast tumor with high probability. Similarly, in the example 3 the instance of a class `Purchase` would represent the fact that John bought the book "Lenny the Lion" from books.com for $15.

The second solution (pattern 2) is to represent several individuals participating in the relation as a collection or an ordered list. We use this solution when the order of the arguments of the n-ary relation is important in the model, as in the example 4 above.

Vocabulary for n-ary relations in RDF and OWL

The task force plans to produce a suggested vocabulary for describing that a class represents an n-ary relation and for defining mappings between n-ary relations in RDF and OWL and other languages. A note on this vocabulary is forthcoming.

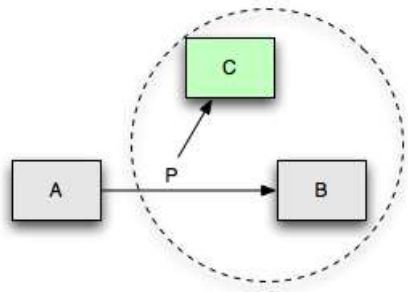
Pattern 1: Introducing a new class for a relation

We present a pattern where we create a new class and *n* new properties to represent an *n*-ary relation. An instance of the relation linking the *n* individuals is then an instance of this class. We consider three use cases for this pattern, illustrated by examples 1-3 above.

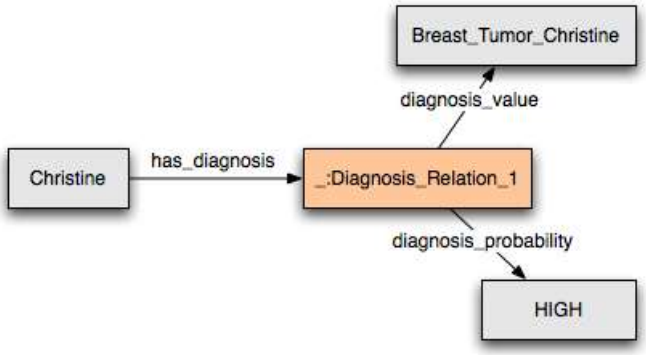
Ontologically the classes created in this way are often called "reified relations". Reified relations play important roles in many ontologies[3] (e.g. Ontoclean/DOLCE, Sowa, GALEN). However, the RDF and Topic Map communities have each used the word "reify" to mean other things (see the note below). Therefore, to avoid confusion, we do not use the term "reification" in this document.

Use Case 1: additional attributes describing a relation

In the first use case, we need to represent an additional attribute describing a relation instance (example 1, *Christine has breast tumor with high probability*). We create an individual that represents the relation instance itself, with links from the subject of the relation to this instance and with links from this instance to all participants that represent additional information about this instance:



For the example 1 above (*Christine has breast tumor with high probability*), the individual `Christine` has a property `has_diagnosis` that has another object (`_:Diagnosis_Relation_1`, an instance of the class `Diagnosis_Relation`) as its value:



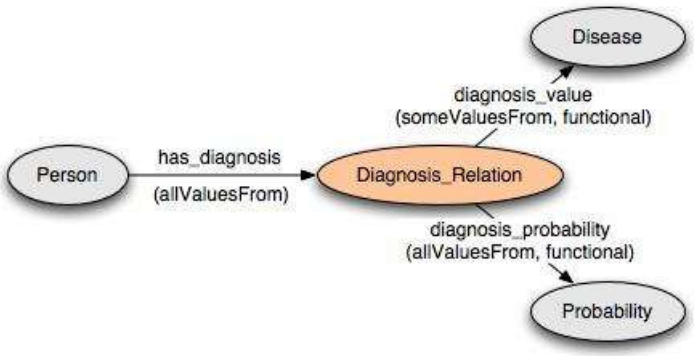
The individual `_:Diagnosis_Relation_1` here represents a single object encapsulating both the diagnosis (`Breast_Tumor_Christine`, a specific instance of `Disease`) and the probability of the diagnosis (`HIGH`)[3]. It contains all the information held in the original 3 arguments: who is being diagnosed, what the diagnosis is, and what the probability is. We use blank nodes in RDF to represent instances of a relation.

```
:Christine
      a       :Person ;
      :has_diagnosis _:Diagnosis_Relation_1 .

:_Diagnosis_relation_1
      a       :Diagnosis_Relation ;
      :diagnosis_probability :HIGH;
      :diagnosis_value :Breast_Tumor_Christine .
```

Each of the 3 arguments in the original n-ary relation—who is being diagnosed, what the diagnosis is, and what the probability is—gives rise to a true binary relationship. In this case, there are three: `has_diagnosis`, `diagnosis_value` and `diagnosis_probability`.[4]

The class definitions for the individuals in this pattern look as follows:



The additional labels on the links indicate the OWL restrictions on the properties. We define both `diagnosis_value` and `diagnosis_probability` as functional properties, thus requiring that each instance of `Diagnosis_Relation` has exactly one value for `Disease` and one value for `Probability`.

In RDFS, which does not have the OWL restrictions or functional properties, the links represent `rdfs:range` constraints on the properties. For example, the class `Diagnosis_Relation` is the range of the property `has_diagnosis`.

Here is a definition of the class `Diagnosis_Relation` in OWL, assuming that both properties—`diagnosis_value` and `diagnosis_probability`—are defined as functional (we provide full code for the example in OWL and RDFS below):

```
:Diagnosis_Relation
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:someValuesFrom :Disease ;
                owl:onProperty :diagnosis_value
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom :Probability_values ;
                owl:onProperty :diagnosis_probability
              ] .
```

In the definition of the `Person` class (of which the individual `Christine` is an instance), we specify a property `has_diagnosis` with the range restriction going to the `Diagnosis_Relation` class (of which `Diagnosis_Relation_1` is an instance):

```
:Person
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom :Diagnosis_Relation ;
                owl:onProperty :has_diagnosis
              ] .
```

Note that in discussing this pattern, we are not making any suggestion on the best way to represent probability pf an event. We simply use it as an example here.

RDFS code for this example

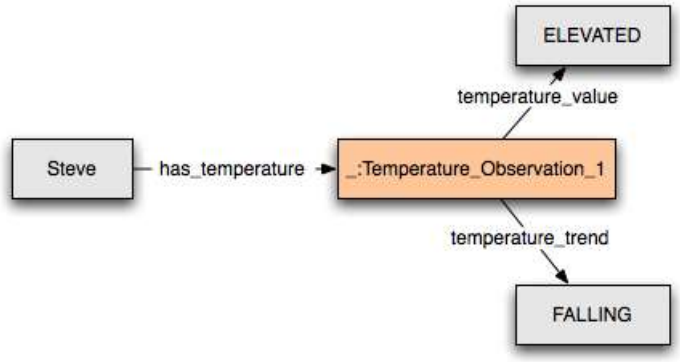
[RDFS]

OWL code for this example

[N3] [RDF/XML]

Use Case 2: different aspects of the same relation

We have a different use case in the example 2 above (*Steve has temperature, which is high, but falling*): In the example with the diagnosis, many will view the relationship we were representing as in a fact still a *binary* relation between the individual `Christine` and the diagnosis `Breast_Tumor_Christine` that has a probability associated with it. The relation in this example is between the individual `Steve` and the object representing different aspects of the temperature he has. In most intended interpretations, this instance of a relation cannot be viewed as an instance of a binary relation with additional attributes attached to it. Rather, it is a relation instance relating the individual `Steve` and the complex object representing different facts about his temperature. Such cases often come about in the course of evolution of an ontology when we realize that two relations need to be collapsed. For example, initially, we might have had two properties—`has_temperature_level` and `has_temperature_trend`—both relating to people. We might then have realized that these properties really are inextricably intertwined because we need to talk about "temperatures that are elevated but falling."



The RDFS and OWL patterns that implement this intuition are however the same as in the previous example. A class `Person` (of which the individual `Steve` is an instance) has a property `has_temperature` which has as a range the relation class `Temperature_Observation`. Instances of the class `Temperature_Observation` (such as `_:Temperature_Observation_1` in the figure) in turn have properties for `temperature_value` and `temperature_trend`.

RDFS code for this example

[RDFS]

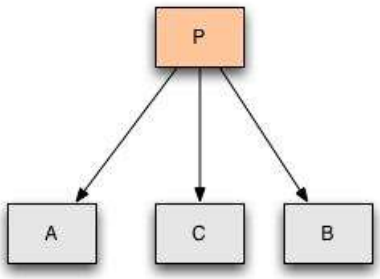
OWL code for this example

[N3] [RDF/XML]

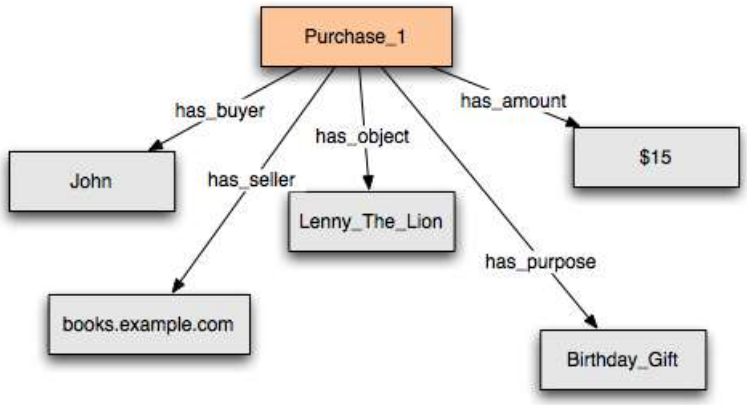
Use Case 3: N-ary relation with no distinguished participant

In some cases, the n-ary relationship links individuals that play different roles in a structure without any single individual standing out as the subject or the "owner" of the relation, such as `Purchase` in the example 3 above (*John buys a "Lenny the Lion" book from books.example.com for $15 as a birthday gift*). Here, the relation explicitly has

more than one participant, and, in many contexts, none of them can be considered a primary one. In this case, we create an individual to represent the relation instance with links to all participants:



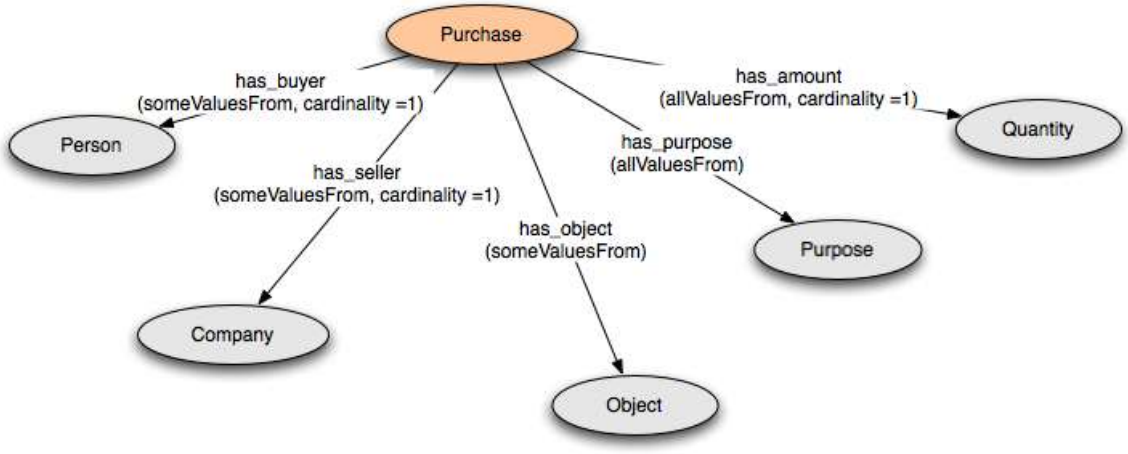
In our specific example, the representation will look as follows:



`Purchase_1`[5] is an individual instance of the `Purchase` class representing an instance of a relation:[6]

```
:Purchase_1
      a       :Purchase ;
      :has_buyer :John ;
      :has_object :Lenny_The_Lion ;
      :has_purpose :Birthday_Gift ;
      :has_amount 15 ;
      :has_seller :books.example.com .
```

The following diagram shows the corresponding classes and properties. For the sake of the example, we specify that each purchase has exactly one `buyer` (a `Person`), exactly one `seller` (a `Company`), exactly one `amount` and at least one `object` (an `Object`).



The diagram refers to OWL restrictions. In RDFS the arrows can be treated as `rdfs:range` links.

The class `Purchase` is defined as follows in OWL (see the RDFS file below for the definition in RDFS):

```
:Purchase
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom :Purpose ;
                owl:onProperty :has_purpose
              ] ;
      rdfs:subClassOf
```

```
        [ a       owl:Restriction ;
          owl:cardinality 1 ;
          owl:onProperty :has_buyer
        ] ;
rdfs:subClassOf
        [ a       owl:Restriction ;
          owl:onProperty :has_buyer ;
          owl:someValuesFrom :Person
        ] ;
rdfs:subClassOf
        [ a       owl:Restriction ;
          owl:cardinality 1 ;
          owl:onProperty :has_seller
        ] ;
rdfs:subClassOf
        [ a       owl:Restriction ;
          owl:onProperty :has_seller ;
          owl:someValuesFrom :Company
        ] ;
rdfs:subClassOf
        [ a       owl:Restriction ;
          owl:onProperty :has_object ;
          owl:someValuesFrom :Object
        ] .
```

Note that representation of OWL restrictions themselves follows this pattern: an OWL restriction is essentially a ternary relation between a class, a property, and a restriction value. In this case, an instance of the `Restriction` class is similar to the instance of `Purchase`.

RDFS code for this example

[RDFS]

OWL code for this example

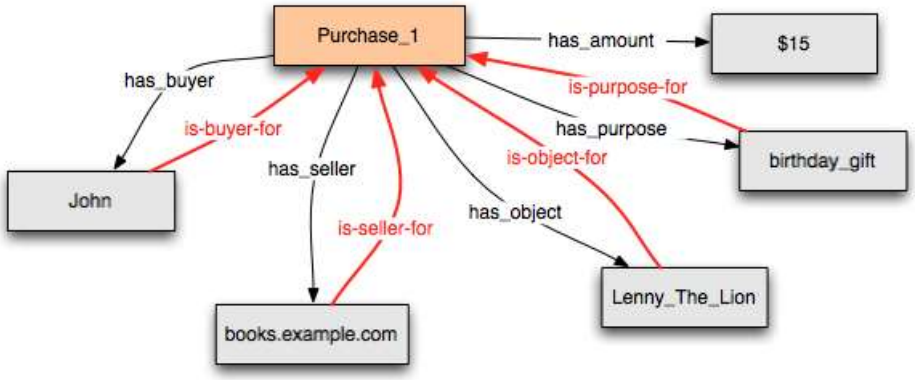
[N3] [RDF/XML]

Considerations when introducing a new class for a relation

- In our example, we did not give *meaningful names* to instances of properties or to the classes used to represent instances of n-ary relations, but merely label them `_:Temperature_Observation_1`, `Purchase_1`, etc. In most cases, these individuals do not stand on their own but merely function as auxiliaries to group together other objects. Hence a distinguishing name serves no purpose. Note that a similar approach is taken when reifying statements in RDF.
- Creating a class to represent an n-ary relation limits the use of many OWL constructs and creates a *maintenance problem*. The problem arises when we want to have local range or cardinality restrictions on some role in the n-ary relation that depend on the class of some other role. For example, we might want to say that we buy only instances of a class `Book` from companies in the category `Bookseller` (cf. use case 3). Expressing this constraint requires a special subclass of the n-ary relation class that represents the combination of restrictions. For instance, we will have to create a class `Book_Purchase` with the corresponding range restrictions for the property `seller` (`allValuesFrom Bookseller`) and `object` (`allValuesFrom Book`). We end up having to build an explicit lattice of classes to represent all the possible combinations.
- OWL allows definition of *inverse properties*. Defining inverse properties with n-ary relations, using any of the patterns above, requires more work than with binary relations. In order to specify inverse properties for n-ary relations, we must specify an inverse for each of the properties participating in the n-ary relation (with the proper constraints). Consider the example of `John` buying the `Lenny_The_Lion` book. We may want to have an instance of an inverse relation pointing from the `Lenny_The_Lion` book to the person who bought it. If we had a simple binary relation `John buys Lenny_The_Lion`, defining an inverse is simple: we simply define a property `is_bought_by` as an inverse of `buys`:

  ```
  :is_bought_by
        a       owl:ObjectProperty ;
        owl:inverseOf :buys .
  ```

  With the purchase relation represented as an instance, however, we need to add inverse relations between participants in the relation and the instance relation itself:



For example, the definitions of the inverse relations for `buyer` and `object` of a purchase, look as follows:

```
:is_buyer_for
      a       owl:ObjectProperty ;
      owl:inverseOf :has_buyer .
:is_object_for
      a       owl:ObjectProperty ;
      owl:inverseOf :has_object .
```

In the definition of the class `Person`, we include an `allValuesFrom` restriction on the property `is_buyer_for`, to restrict the values for this property to instances of the class `Purchase`:

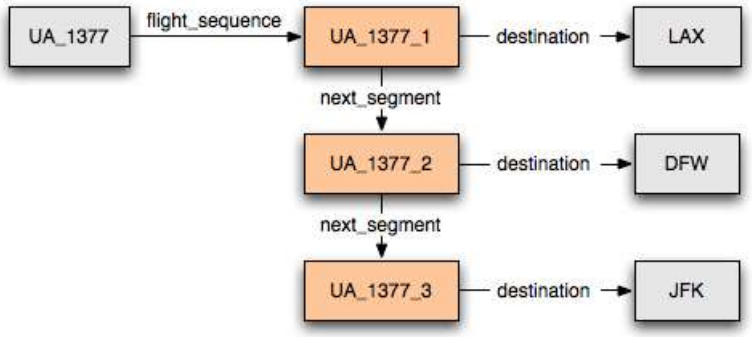
```
:Person
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:onProperty :is_buyer_for ;
                owl:allValuesFrom :Purchase
              ] .
```

Note that the value of the inverse property `is_buyer_for` for the individual `John`, for example, is the individual `Purchase_1` rather than the `object` or `recipient` of the purchase.

Pattern 2: Using lists for arguments in a relation

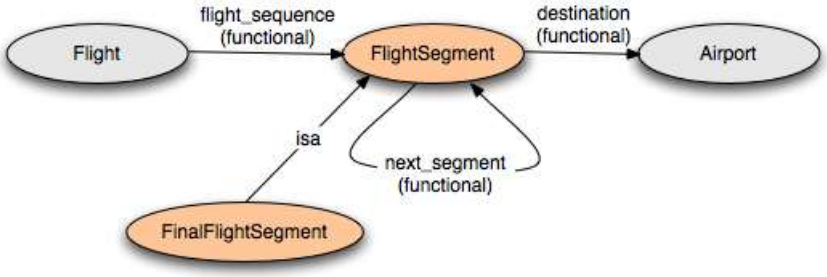
Some n-ary relations do not naturally fall into either of the use cases above, but are more similar to a list or sequence of arguments. The example 4 above (*United Airlines flight 3177 visits the following airports: LAX, DFW, and JFK*) falls into this category. In this example, the relation holds between the flight and the airports it visits, in the order of the arrival of the aircraft at each airport in turn. This relation might hold between many different numbers of arguments, and there is no natural way to break it up into a set of distinct properties relating the flight to each airport. At the same time, the order of the arguments is highly meaningful.

In cases where all but one participant in a relation do not have a specific role and essentially form an ordered list, it is natural to connect these arguments into a sequence according to some relation, and to relate the one participant to this sequence (or the first element of the sequence). We represent the example below using an ordering relation (`nextSegment`) between instances of the `FlightSegment` class. Each flight segment has a property for the destination of that segment. Note that we add a special subclass of flight segment, `FinalFlightSegment`, with a maximum cardinality of 0 on the `nextSegment` property, to indicate the end of the sequence.



RDF supplies a vocabulary for lists — the collection vocabulary, which can also be used in cases where a group of arguments to the relation have no special role. We do not use the RDF collection vocabulary in this example, because it is less practical to use a generic ordering relation when we are representing something more specific. In this example, we represent a temporal order among constituents.

We can represent the ontology for this example in OWL. Note that using the `rdf:List` vocabulary in OWL would have put the ontology in OWL Full (see the corresponding section of the OWL Guide for the comparison of OWL Full and OWL DL). The following ontology is in OWL Lite:




```
:Flight
        a owl:Class .

:flight_sequence
   a owl:ObjectProperty , owl:FunctionalProperty ;
   rdfs:domain :Flight ;
   rdfs:range :FlightSegment .

 :FlightSegment
    a owl:Class ;
    rdfs:subClassOf owl:Thing ;
    rdfs:subClassOf
         [ a owl:Restriction ;
             owl:cardinality "1";
             owl:onProperty :destination
         ] ;
   rdfs:subClassOf
         [ a owl:Restriction ;
             owl:allValuesFrom :Airport ;
             owl:onProperty :destination
          ] .

  :next_segment
      a owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :FlightSegment ;
      rdfs:range :FlightSegment .


:FinalFlightSegment
   a owl:Class ;
   rdfs:comment "The last flight segment has no next_segment";
   rdfs:subClassOf :FlightSegment ;
   rdfs:subClassOf
         [ a owl:Restriction ;
             owl:maxCardinality "0";
             owl:onProperty :next_segment
          ] .

  :Airport
      a owl:Class .

  :destination
      a owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :FlightSegment .
```

RDFS code for this example

[RDFS]

OWL code for this example

[N3] [RDF/XML]

N-ary relations and reification in RDF

It may be natural to think of RDF reification when representing n-ary relations. We do not want to use the RDF reification vocabulary to represent n-ary relations in general for the following reasons. The RDF reification vocabulary is designed to talk about *statements*—individuals that are instances of `rdf:Statement`. A statement is a object, predicate, subject triple and reification in RDF is used to put additional information about this triple. This information may include the source of the information in the triple, for example. In n-ary relations, however, additional arguments in the relation do not usually characterize the statement but rather provide additional information about the relation instance itself. Thus, it is more natural to talk about instances of a diagnosis relation or a purchase rather than about a statement. In the use cases that we discussed in the note, the intent is to talk about instances of a relation, not about statements about such instances.

Additional Background

Note on vocabulary: Relations and instances of relations, Properties and Property instances

We usually think of semantic web languages as consisting of triples of the form "Individual1-Property-Individual2" (Traditionally, these have been termed "object-attribute-value" triples, but we do not use this language here because it conflicts with RDF usage.)

However, formally, we interpret properties as representing relations, i.e. sets of ordered pairs of individuals. Each instance of a relation is just one of those ordered pairs. The "Property" in each triple is fundamentally different from the individuals in the triple. It merely indicates to which relation the ordered pair consisting of the two individuals belongs. We normally name individuals; we do not normally name the ordered pairs.

Anonymous vs named instances in these patterns

Often in cases such as use case 1, we wish to regard two instances of the relation that have the same argument as equivalent. We can capture this intuition by using RDF blank nodes (e.g., `_:Diagnosis_relation`) to represent relation instances. In use case 2, we wish to consider the possibility that there might be two distinct purchases with identical arguments. In that case, the node should be named, e.g. `Purchase_1`.

Notes

1. "Reified relations" play an important role or have a special status in a number of ontologies, e.g. see Sowa, J. Knowledge Representation. Morgan Kaufmann, 1999; Welty, C. and Guarino, N. Supporting ontological analysis of taxonomic relationships. Data and Knowledge Engineering, 39 (1). 51-74.
2. For simplicity, we represent each disease as an individual. This decision may not always be appropriate, and we refer the reader to a different note (*to be written*). Similarly, for simplicity, in OWL we represent probability values as a class that is an enumeration of three individuals (`HIGH`, `MEDIUM`, and `LOW`):

   ```
   :Probability_values
         a       owl:Class ;
         owl:equivalentClass
                 [ a       owl:Class ;
                   owl:oneOf (:HIGH :MEDIUM :LOW)
                 ] .
   ```

   There are other ways to represent partitions of values. Please refer to a note on Representing Specified Values in OWL [Specified Values]. In RDF Schema version, we represent them simply as strings, also for simplicity reasons.

3. RDF has a property `rdf:value` that is appropriate in examples such as the Diagnosis example here. While `rdf:value` has no meaning on its own, the RDF specification encourages its use as a vocabulary element to identify the "main" component of a structured value of a property. Therefore, in our example, we made `diagnosis_value` a subproperty of `rdf:value` property instead of making it a direct instance of `rdf:Property` to indicate that `diagnosis_value` is indeed the "main" component of a diagnosis.
4. Note that we used a named individual for an instance of the class `Purchase` (`Purchase_1`) rather than an anonymous blank node here. In this example, there might be two distinct purchases with exactly the same arguments.
5. For simplicity, we will ignore the fact that the amount is expressed in $ and will use a simple number as the value for the property. For a discussion on how to represent units and quantities in OWL, please refer to a different note (*to be written*)

References

[Specified Values]
Representing Specified Values in OWL: "value partitions" and "value sets", Alan Rector, Editor, W3C Working Draft, 3 August 2004, http://www.w3.org/TR/swbp-specified-values/ .

[OWL Overview]
OWL Web Ontology Language Overview, Deborah L. McGuinness and Frank van Harmelen, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/ . Latest version available at http://www.w3.org/TR/owl-features/ .

[OWL Guide]
OWL Web Ontology Language Guide, Michael K. Smith, Chris Welty, and Deborah L. McGuinness, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-guide-20040210/ . Latest version available at http://www.w3.org/TR/owl-guide/ .

[OWL Semantics and Abstract Syntax]
OWL Web Ontology Language Semantics and Abstract Syntax, Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/ . Latest version available at http://www.w3.org/TR/owl-semantics/ .

[RDF Primer]

RDF Primer, Frank Manola and Eric Miller, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ . Latest version available at http://www.w3.org/TR/rdf-primer/ .

[RDF Semantics]
RDF Semantics, Pat Hayes, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ . Latest version available at http://www.w3.org/TR/rdf-mt/ .

[RDF Vocabulary]
RDF Vocabulary Description Language 1.0: RDF Schema, Dan Brickley and R. V. Guha, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ . Latest version available at http://www.w3.org/TR/rdf-schema/ .

[Turtle]
"Turtle - Terse RDF Triple Language, Dave Beckett.

Changes

- Merged patterns 1 and 2 into one pattern with different use cases. The same use cases remain, but they are described as different use cases for the same pattern.
- Removed consideration bullet talking about logical equivalence of patterns 1 and 2 (since they are a single pattern now).
- Added more discussion to General issues and Use cases
- Added pattern 2 (using rdf:Lists)
- Added the flight example
- Changed the wording under "Representation Pattern"
- Use blank nodes for relation instances in pattern 1 and pattern 2
- Added a section on N-ary relations and reification in RDF
- Added a section on Additional background
- Added references
- Changed some of references to "relation" to "relation instance" or "instance of relation"
- Removed examples in abstract syntax
- Added Acknowledgements

Acknowledgements

The editors would like to thank the following Working Group members for their contributions to this document: Pat Hayes, Jeremy Carroll, Chris Welty, Michael Uschold, Bernard Vatant. Frank Manola, Ivan Herman, Jamie Lawrence have also contributed to the document.

This document is a product of the Ontology Engineering and Patterns Task Force of the Semantic Web Best Practices and Deployment Working Group.