OSLC Core Version 3.0: Link Guidance Version 1.0

Project Note 01
16 December 2021

This stage:

https://docs.oasis-open-projects.org/oslc-op/link-guidance/v1.0/pn01/link-guidance.html (Authoritative)
https://docs.oasis-open-projects.org/oslc-op/link-guidance/v1.0/pn01/link-guidance.pdf

Previous stage:

N/A

Latest stage:

https://docs.oasis-open-projects.org/oslc-op/link-guidance/v1.0/link-guidance.html (Authoritative)
https://docs.oasis-open-projects.org/oslc-op/link-guidance/v1.0/link-guidance.pdf

Latest editor's draft:

https://oslc-op.github.io/oslc-specs/notes/link-guidance.html

Open Project:

OASIS Open Services for Lifecycle Integration (OSLC) Open Project

Editor:

Jim Amsden (jamsden@us.ibm.com), IBM

Related work:

This specification is related to:

- *OSLC Core 2.0 Appendix C: Guidance on Links & Relationships*. Finalized: http://open-services.net/bin/view/Main/OslcCoreSpecAppendixLinks
- *OSLC Core Version 3.0. Part 1: Overview*. OASIS Standard: https://open-services.net/spec/core/latest

Abstract:

In this informative note we offer advice to OSLC Technical Committees on modeling links and relationships between OSLC resources.

Status:

This is a Non-Standards Track Work Product. The patent provisions of the OASIS IPR Policy do not apply.

This document was last revised or approved by the Project Governing Board of the OASIS Open Services for Lifecycle Integration (OSLC) Open Project on the above date. The level of approval is also listed above. Check the “Latest stage” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work

Citation format:

When referencing this specification the following citation format should be used:
[OSLC-Link-v1.0]
OSLC Core Version 3.0: Link Guidance Version 1.0. Edited by Jim Amsden. 16 December 2021. OASIS Project Note 01. https://docs.oasis-open-projects.org/oslc-op/link-guidance/v1.0/pn01/link-guidance.html. Latest stage: https://docs.oasis-open-projects.org/oslc-op/link-guidance/v1.0/link-guidance.html.

Notices

Table of Contents

1. Relationships in OSLC

The Core workgroup's guidance to OSLC domain workgroups is that relationships are uni-directional and in most cases relationships should be modeled as links. We offer two ways to express relationships, a link and an anchor. Let's define those terms and others that we will use in this document:

1. Relationship: a relationship is said to exist between two resources if there is something that connects them; they work together, belong together, are similar or should be considered together. There may be different types of relationships.
2. Link: a link is a URI reference from one resource, the subject or source, to another resource, said to be the object or target. In RDF and in OSLC we use links to model relationships and like relationships, links are uni-directional.
3. Anchor: an anchor is a mechanism for attaching property-values to a link, thus annotating a relationship with values that are about that relationship.

Now that we've defined our terminology, let's move on to expressing relationships.

2. Expressing relationships for OSLC

The RDF data model gives us a number of value-types to express relationships, some more complex than others. We expect that the majority of links will be expressed with the most simple option, URI Reference or as it is known in conversation, a Link. In this simple case, a Link refers to a RDF assertion where the predicate is the URI. It is the subject, predicate, object that represents the relationship. For more complex cases, when a relationship must be annotated with property-values, we use what is called an Anchor, which is a link plus information that annotates or labels that link. Read the sub-sections below for details and examples of Links and Anchors.

2.1 Link

2.1.1 Link from subject to object

Use this when you need a simple link and you do not need to annotate the link with property values. Most relationships should be represented this way.

In RDF terminology, a link is called a URI Reference. In OSLC resource constraints, you express a link as a property with OSLC valueType of Resource as described in [OSLCCore3]. The following example shows a property `terms:subscribesTo` that you might find inside a customer resource that links to a resource that is "subscribedTo" by the customer.

EXAMPLE 1: Relationship expressed as simple link (i.e. URI reference)

```
<rdf:RDF
     xmlns:terms="http://example.com/terms/"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <terms:Customer rdf:about="http://example.com/customers/4321">
         <terms:subscribesTo rdf:resource="http://example.com/magazines/Field_and_Stream" />
    </terms:Customer>
</rdf:RDF>
```

The constraints on the terms:subscribesTo property would include `oscl:valueType oslc:Resource` as shown in the example below.

EXAMPLE 2: Constraints on property subscribesTo

```
<#dcterms-title> a  oslc:Property ;
  oslc:name                 "subscribesTo" ;
  oslc:occurs               oslc:Zero-or-many ;
  oslc:propertyDefinition   terms:subscribesTo ;
  oslc:valueType            oslc:Resource.
```

2.1.2 Don't make assumptions about the target of links

Relationships in OSLC resources are at their simplest an RDF property whose object is a URI. In some cases within one system, it is necessary to require a closed link, i.e. require the object of a link be of one or more specific resource types. In general, it is better to be open like the web and not place restrictions on the type of resource linked to. The property's purpose and name should clearly reflect the scenarios it is supporting. Since the usage of these relationship properties may exist for a long period of time, specification authors should use great care in determining these.

As resources evolve over time, and they adapt to different situations, different types will be exposed as targets to existing link types. Well behaved clients should gracefully handle resource types they don't expect when exercising links in resources. Specifications should allow links to be open ended (have any type, specifying Range any), and use text in the property description to suggest expected types, without being limiting. For example: “Change request affects a plan item. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case." Some implementations may only work well if the link object comes from a set of "known" or "expected" types. The following graceful degradation sequence is suggested for providers when an unexpected type is encountered:

1. If possible, tolerate the unexpected link target type and allow the client's request to proceed normally.
2. Else, if possible, allow the client's request to proceed and include an informational message with the response.
3. Else, fail the client's request with the most appropriate 4xx Bad Request HTTP status code.

2.1.3 Store link information in one place

By "place" we mean the resource that contains the link assertion. In most cases, link information should be captured in one place to avoid information redundancy and the associated information maintenance issues.

There are two general sources of information redundancy:

1. Inverse properties
2. Duplicate assertions

Inverse properties create significant information redundancy. For example, assertions A ex:affects B and B ex:isAffectedBy A require both assertions to be updated in order to maintain information integrity. Inverse properties are also usually unnecessary because it is often easy to specify queries that specify the subject or object, allowing the relationship to be "navigated" in either direction. One of these assertions, perhaps the "isAffectedBy", should not exist. The predicate to keep is a design choice, depending partly on the direction of the relationship that is most often known or requested by users, and partly on the expected ordering of development of the resources involved. It is often better to have a link stored in the resource that is developed later (the 'downstream' artifact), so that the previously completed end of the link does not need to be updated when creating the link.

Duplicated triples have the same problem as any redundancy, and should be avoided whenever possible. This redundancy may be required to provide the information independently from different servers, but it must be handled with care to avoid data integrity issues.

There are times when duplicate triples are necessary, and there is a preferred way to use them. For example, a request to GET resource A might respond with triple "A ex:affects B", and a request to GET resource B might also respond with the same "A ex:affects B" triple. If they are on different servers then B's assertion might get out of date. Clients may judge the authority of any RDF statement based on its origin or provenance.

Storing the link in one place simplifies updates.

2.2 Anchor

2.2.1 Local Resource holds link from subject to object plus property-values that annotate the relationship

Relationships are expressed as RDF triples and triples do not have property values - only resources have property values. Links are used to express relationships. Many relationships can be represented by a single link, but some relationships need to be annotated with property values. For example, if I am trying to represent the subscriber relationship between a customer and a magazine, I may need to record an expiration date for the relationship. One way to do this is to use an RDF concept known as reification. Reification is a way to make a statement about a statement. See RDF Syntax Specification: Reifying Statements for more information on this concept (reference: RDF Syntax).

Assuming the subscriber scenario, the example below shows how you would use reification to express the annotation.

EXAMPLE 3: Link as reified statement in Turtle format

```
<rdf:RDF
     xmlns:terms="http://example.com/terms/"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <terms:Customer rdf:about="http://example.com/customers/4321">
         <terms:subscribesTo rdf:resource="http://example.com/magazines/Field_and_Stream" />
    </terms:Customer>
    <terms:Customer rdf:about="http://example.com/customers/4321">
         <terms:subscribesTo rdf:resource="http://example.com/magazines/Cat_Fancy" />
    </terms:Customer>

    <rdf:Statement rdf:about="#n1">
        <terms:expirationDate>2010-06-03</terms:expirationDate>
        <terms:annualPriceUSD>23.95</terms:annualPriceUSD>
        <terms:delivery rdf:resource="http://example.com/terms/online" />
        <rdf:subject rdf:resource="http://example.com/customers/4321"/>
        <rdf:object rdf:resource="http://example.com/magazines/Field_and_Stream"/>
        <rdf:predicate rdf:resource="http://example.com/terms/subscribesTo" />
   </rdf:Statement>

    <rdf:Statement rdf:about="#n2">
        <terms:expirationDate>2010-01-22</terms:expirationDate>
        <terms:annualPriceUSD>15.95</terms:annualPriceUSD>
        <terms:delivery rdf:resource="http://example.com/terms/mail" />
        <rdf:subject rdf:resource="http://example.com/customers/4321"/>
        <rdf:object rdf:resource="http://example.com/magazines/Cat_Fancy"/>
        <rdf:predicate rdf:resource="http://example.com/terms/subscribesTo" />
   </rdf:Statement>

</rdf:RDF>
```

However, there is a better RDF/XML form, made possible by the rdf:ID attribute. OSLC implementations should prefer this next form because it is easier to read and easier to parse.

EXAMPLE 4: Link as reified statement in RDF/XML format (preferred form)

```
<rdf:RDF
    xmlns:terms="http://example.com/terms/"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <terms:Customer rdf:about="http://example.com/customers/4321">
         <terms:subscribesTo rdf:ID="n1"
             rdf:resource="http://example.com/magazines/Field_and_Stream" />
    </terms:Customer>

    <terms:Customer rdf:about="http://example.com/customers/4321">
         <terms:subscribesTo rdf:ID="n2"
             rdf:resource="http://example.com/magazines/Cat_Fancy" />
    </terms:Customer>

    <rdf:Description rdf:about="#n1">
        <terms:expirationDate>2010-06-03</terms:expirationDate>
        <terms:annualPriceUSD>23.95</terms:annualPriceUSD>
        <terms:delivery rdf:resource="http://example.com/terms/online" />
   </rdf:Description>

    <rdf:Description rdf:about="#n2">
        <terms:expirationDate>2010-01-22</terms:expirationDate>
        <terms:annualPriceUSD>15.95</terms:annualPriceUSD>
        <terms:delivery rdf:resource="http://example.com/terms/online" />
   </rdf:Description>

</rdf:RDF>
```

And here's how to express the same within the JSON format specified by OSLC Core. Because we already represent valueType Resource as a JSON object, we can simply add annotating property-values directly into that. In this case, adding `terms:expirationDate` to the `terms:subscribesTo` object.

EXAMPLE 5: Link as reified statement in OSLC JSON format

```
{
   "prefixes" : {
      "terms": "http://example.com/terms/",
      "oslc": "http://open-services.net/ns/core#",
      "rdf" : "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   },
   "rdf:about" : "http://example.com/customers/4321",
   "rdf:type" : { "rdf:resource" : "http://example.com/terms/Customer" },
   "terms:subscribesTo" : [{
       "rdf:resource" : "http://example.com/magazines/Field_and_Stream",
       "terms:expirationDate" : "2010-06-03"
       "terms:annualPriceUSD" : 23.95,
       "terms:delivery" : { "rdf:resource" : "http://example.com/terms/online" }
   },
   {
       "rdf:resource" : "http://example.com/magazines/Cat_Fancy",
       "terms:expirationDate" : "2010-01-22"
       "terms:annualPriceUSD" : 15.95,
       "terms:delivery" : { "rdf:resource" : "http://example.com/terms/mail" }
   }]
}
```

Although the approaches above work to allow anchors to represent links with properties, reification can have entailment and inferencing issues. When there is a need to reify a statement in order to make statements about the stagement (called hearsay), this may be an indication that there is a missing class in the RDF vocabulary. Rather than reifying the statement, the modeler can create a separate class and create specific relationships to that class. That may actually improve the model. The example above could be modeled by introducing a Subscription to capture information about a customer and their magazine subscriptions. This is sometimes called an associative object.

EXAMPLE 6: Using an associative object using Turtle syntax

```
@prefix terms: <http://example.com/terms/> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http:terms:Subscription>
        a                     rdf:Class.

<#s1>
        a                     terms:Subscription;
        terms:magazine        <http://example.com/magazines/Cat_Fancy> ;
        terms:subscriber      <http://example.com/customers/4321> ;
        terms:annualPriceUSD  "15.95" ;
        terms:delivery        terms:online ;
        terms:expirationDate  "2010-01-22" .

<#s2>
        a                     terms:Subscription ;
        terms:magazine        <http://example.com/magazines/Field_and_Stream> ;
        terms:subscriber      <http://example.com/customers/4321> ;
        terms:annualPriceUSD  "23.95" ;
        terms:delivery        terms:online ;
        terms:expirationDate  "2010-06-03" .
```

In this example, the Subscription subject has all the required information including the magazine, subscriber, anual price, etc. There is no need to reify any statements. A simple query will provide the list of magazines a customer subscribes to.

2.3 Anti-patterns

There are also some anti-patterns that clients and servers should avoid.

2.3.1 Use of Blank Nodes

Blank nodes may be a convenient shorthand to use in examples - especially in Turtle where the [...] syntax makes blank nodes easy to use. However, blank nodes may make queries and updates much harder, since there is no way to reference a specific blank node later. Introducing an explicit resource with its own URI is usually better practice.

Below is an example of a Blog Entry resource with a blank node for an uploaded-file resource.

EXAMPLE 7: Blog Entry with blank node for file

```
@prefix dcterms:  <http://purl.org/dc/terms/> .
@prefix blog:     <http://open-services.net/ns/bogus/blog#> .

<http://example.com/blogs/entry/5>
  a blog:BlogEntry ;
  dcterms:title "New Wink release available" ;
  blog:attachment [
    a blog:UploadedFile ;
    dcterms:title "wink-0.9.6.jar"
  ] .
```

It would be better to represent the uploaded-file resource with an explicit URI. Note that having an explicit URI does not mean the resource has to be sent in a different HTTP request; in this example, we use a hash URI to have the uploaded-file resource returned in the same HTTP request as its parent blog entry resource.

EXAMPLE 8: Blog Entry with hash URI for file

```
@prefix dcterms:  <http://purl.org/dc/terms/> .
@prefix blog:     <http://open-services.net/ns/bogus/blog#> .

<http://example.com/blogs/entry/5>
  a blog:BlogEntry ;
  dcterms:title "New Wink release available" ;
  blog:attachment <#file> .

<#file>
  a blog:UploadedFile ;
  dcterms:title "wink-0.9.6.jar" .
```

Feedback should be directed to the OSLC Core Workgroup mailing-list. TBD - put link here.

Appendix A. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Project Governing Board:

James Amsden, IBM (co-chair)
Andrii Berezovskyi, KTH (co-chair)
Axel Reichwein, Koneksys

Techical Steering Committee:

James Amsden, IBM
Andrii Berezovskyi, KTH
Axel Reichwein, Koneksys

Additional Participants:

Dave Johnson
Arthur Ryman

Appendix B. References

B.1 Informative references

[OSLCCore3]

Jim Amsden; Andrii Berezovskyi. *OSLC Core Version 3.0. Part 1: Overview*. OASIS. OASIS Standard. URL: https://open-services.net/spec/core/latest