

A common ground

the whole-parts information set

Didier PH Martin
Talva Corp., St-Basile, Canada
martind@netfolder.com
<http://www.netfolder.com>

Abstract:

Is there any similarities between topic maps, site maps, tables of content, parts catalogs and directory services? The answer is yes. Learn how Xlink and RDF could be used to solve this enigma and hear more about whole-part information sets.

Introduction

In the recent months I made several experimentations with the new XML frameworks introduced by W3C. More particularly, I made some experiments with and . This paper is about one of these experiments.

Whole-parts structures are used in different domains such as document or web resources categorization, illustrated parts catalogs, network or enterprise resources management, etc. Throughout this paper will see how two different XML frameworks could be re-united with a common data structure: the whole-parts information set.

The whole-parts pattern

We tend to reduce the universe's complexity by chunking it into parts. We also tend to build things from parts. Hence, when we structure, compose or analyze, we intuitively use the whole-part pattern.

The whole part pattern helps with the aggregation of things, of components, that together form a semantic unit. The whole encapsulates its constituent components, the parts and provides a new emergent behavior. It presents a new interface to the component collection and this often leads us to infer that the whole is more than the sum of its parts. This pattern can be applied to different situations when information units need to be aggregated and when we need to provide a single interface to the aggregated collection.

The whole-parts pattern is applicable to the organization of some types of relationship:

The assembly-part relationship such as, for instance, the relationship between an airplane and its parts. The parts are tightly coupled and form a structure.

The container-contents relationship where the parts are loosely coupled and in which the aggregated object represents a container.

The collection-members relationship in which loosely coupled elements are grouped together in accordance to certain ownership criteria.

In the whole-parts pattern, two agents play a different role:

The whole represents an aggregation of the parts and forms a semantic grouping of its parts.

The parts are the elements which compose the whole or the elements that are contained by the whole. Their structure may constitute the whole.

For instance, each object of an illustrated parts catalog is a part used to contribute to the whole. In the case of an illustration, when the whole receives a request to zoom in or zoom out, the appropriate request is then forwarded to the parts by the whole. The whole presents a single interface to the parts collection.

The whole-parts pattern offers an economical way to manipulate collection of objects.

The whole-parts information set

A whole-parts information set is a model that contains several collection of objects. At the minimum, each object has a collection of parts related to it. Preferably, an object is also described by a set of properties. A whole-parts structure is hierarchical. Each collection is not necessarily homogeneous but may be homogeneous if required. An object may be part of more than one structure. For instance, a document collection may be aggregated by more than one topic. A topic may be simultaneously member of the “markup technologies” topic and part of the “meta-language” topic. In the system, a document collection, aggregated under a particular subject, can be included in more than one facet.

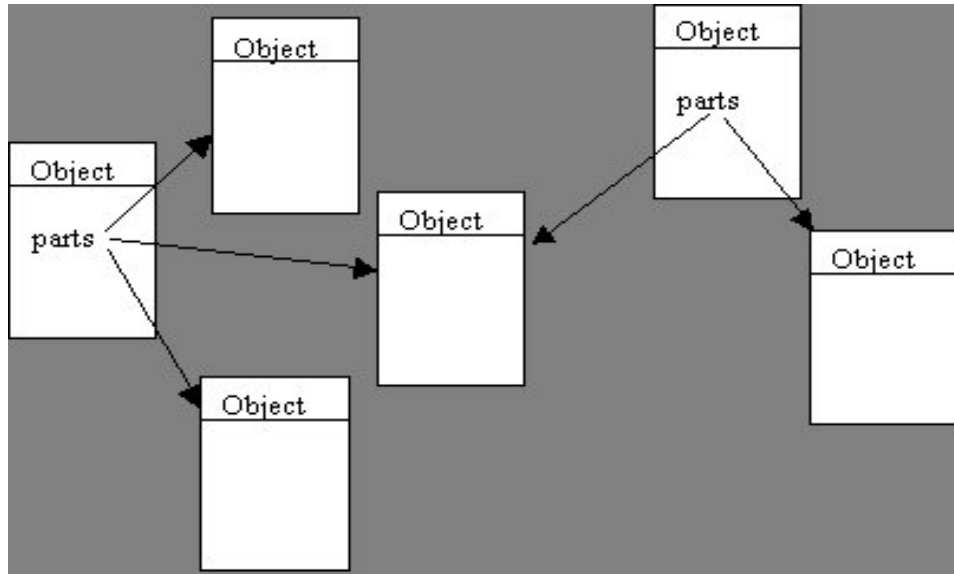


Figure 1. Facets aggregation

To allow an object to be a member of several collections, this object has to be packaged as a separate unit. For instance, to encode a hierarchy of topics, it is preferable to define each topic as a separate unit, and then, to assemble these topics in a new unit for a particular context.

The basic elements of a whole-parts information set contain two node types.

Property nodes

Pointers to other object nodes

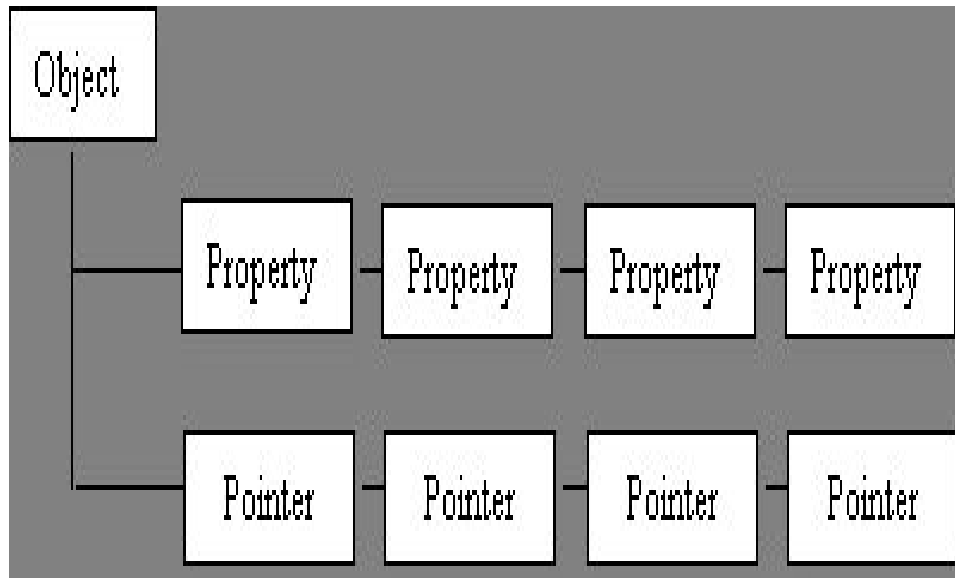


Figure 2. Whole-parts information unit

In the architecture, each GROVE's node contains a set of properties. The GROVE architecture is intrinsically a particular implementation of the whole-parts pattern. In the GROVE paradigm, an object contains a set of properties. So, if our whole-parts information set would be implemented with a GROVE engine, the pointers collection would be simply a node's property.

The GROVE architecture do not provide any interfaces definition. In the remaining text, we'll take a different approach and we will define the whole-parts information set interfaces.

Because a "whole" node contains more than one property node, these property nodes are structured as a list. The named-node-list interface as defined by the W3C DOM recommendation can be re-used in our context. We choused the named-node-list interface because we need to access a particular property node by its property name. However, the property named-node-list has no imperatives to maintain a particular sequence. Each property node can be randomly accessed by a property name

Also, a "whole" object is a collection of pointers to parts. A part can be a "whole" object that may or may not contain parts. Hence, a whole-parts structure is intrinsically recursive. The parts collection aggregated by the "whole" object is abstracted as a node-list as defined by the W3C DOM recommendations. The node-list interface is more adapted to enumeration than to random access. A pointer node may refer to an other object located in an external information set or to an other "whole" object node located in the same information set.

Finally, the "whole" objects collection can be organized in a named-node-list. "Whole" object nodes are accessed randomly by their name. Figure 3 below illustrates the whole-parts elements data types or more particularly the interface type used to interact with them

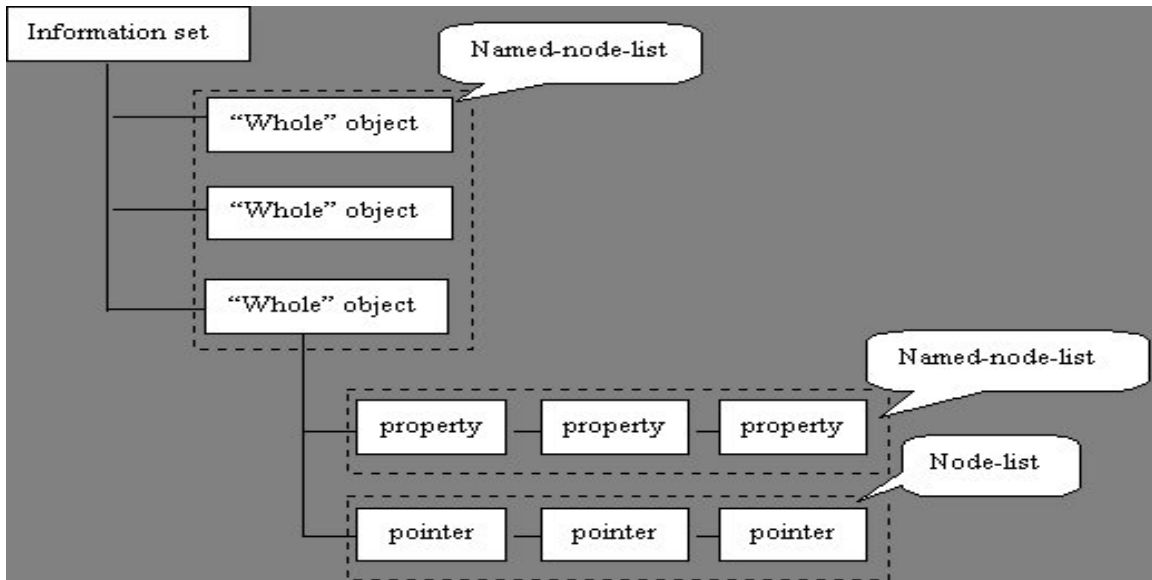


Figure 3. Whole-parts information set structure

Now that the different collections have their interfaces defined, we can further refine the model by defining new interfaces for each “whole”, property and pointer objects.

We’ll use here the CORBA interface definition language to define the object’s interfaces but an other interface definition language such as the DCOM IDL may be used as well.

```

Interface object
{
readonly attribute NamedNodelist properties;
readonly attribute nodeList pointers;
}
interface property
{
attribute string name;
attribute string value;
}
interface pointer
{
    attribute string title;
    attribute string href;
    attribute string role;
}

```

The object interface represent the “whole” object, the property and pointer interfaces represent respectively an element of the properties collection and an element of the pointers collection.

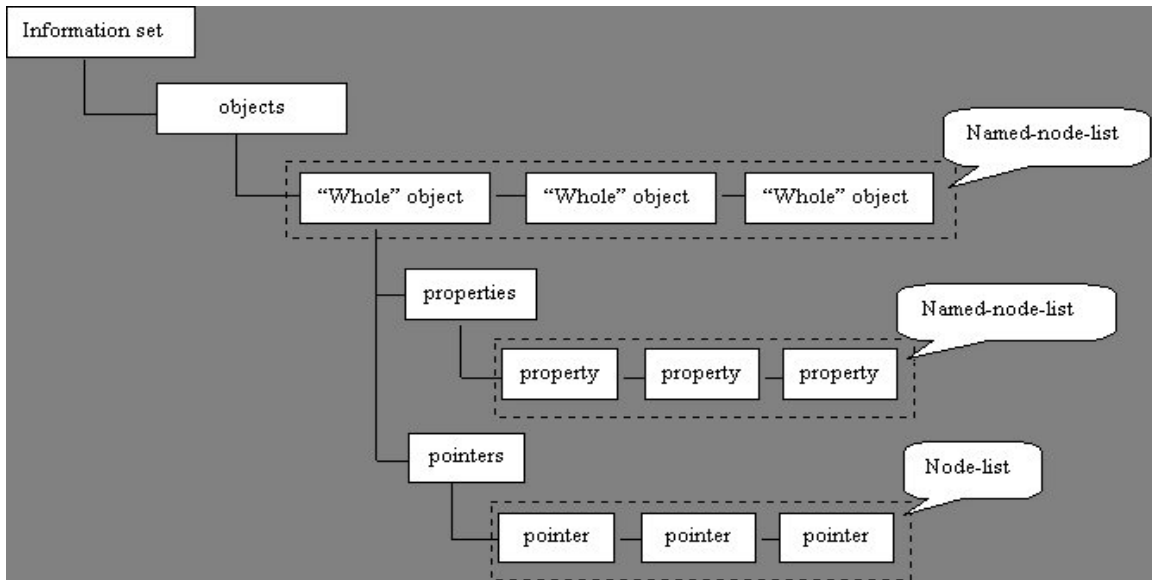


Figure 4. Whole-parts detailed information set structure

An interpreter can be used to parse and interpret any format representing a whole-parts pattern. If, for instance, a whole-parts structure is serialized as an XML document, then the interpreter will parse and interpret the XML document. The result of this interpretation is a whole-parts information set.

We'll examine two XML frameworks we used to encode some whole-parts pattern.

- RDF
- XLink

RDF

The RDF basic element, the `<rdf:description>` element is based on the concept of frame. In contrast to records as found in relational data bases, a frame is a free association of {subject, property, value} tuples. And, in contrast to relational data base fields, a frame slot (i.e. a field) can contain a collection and is not restricted solely to singleton values.

An RDF element contains a property set encoded as sub-elements. For instance, the following expression is equivalent to Figure 5 displayed below.

```

<rdf:description id="SVG">
<description> Scalable Vector Graphic recommendations</description>
<institution>W3C</institution>
<language>English</language>
<stage>working draft</stage>
</rdf:description>

```

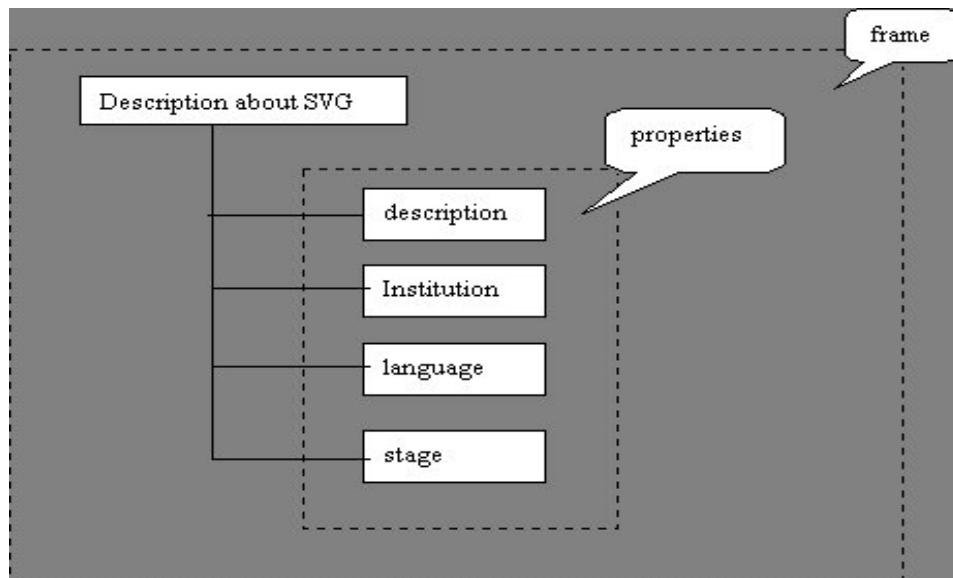


Figure 5. RDF frame information set fragment

The concept of is a powerful concept for knowledge management.. The XML implicit containment relationship between the resource element (i.e. `<rdf:description>`) and its related properties re-enforce the logical containment relationship between the resource and its properties. In fact, to see an `<rdf:description>` element as a kind of record (i.e. a frame) is a lot easier to understand than to reduce it to a collection of tuples.

If the RDF language is used to encode a whole-parts structure, the `<rdf:description>` element should contain both the property collection and the pointers collection. We just have to remember that the property collection is the property set associated to the “whole” object and that the pointers collection is the parts collection. This last collection is a collection of pointers to parts that are also considered as “whole” objects. This is because a parts can be itself composed of parts and then also be a “whole” object.

The object keeps a collection of pointers to its parts in an `<rdf:bag>` element. The `<rdf:bag>` element is a collection element to be used when we need to contain a item collection of the same type. The following expression represents a collection of web resources aggregated by a topic named “XML”.

```
<rdf:description id="XML" >
  <Synopsis>Markup Language or meta language used to implement
  specific domain languages</Synopsis>
  <rdf:bag>
    <rdf:li resource="http://www.xml.com"/>
    <rdf:li resource="http://www.xml.org"/>
    <rdf:li resource="http://www.w3.org/TR/WD-xml-lang.html"/>
  </rdf:bag>
</rdf :description>
```

The `<rdf:description>` element contains a “synopsis” property and a “resource” collection. The synopsis property is a property associated to the XML topic (i.e. an `<rdf:description>` element) and the resource collection is, in our context, a pointers’ collection. This kind of RDF element could be, for instance, the result of a request made to a news feed server. A response file from the news feed server can contain more than one topic. Each topic is encoded as an `<rdf:description>` element.

As an other example, let's now take an illustrated parts catalog in which each part is encoded as an `<rdf:description>` element.

When the pointed resources do not contain any meta information or properties, then the following expression could be used.

```
<rdf:description id="motor" about="http://www.widget.com/motor.svg">
  < name>fuel injection motor</ name >
  <quantity>1</quantity>
  <PartNumber>1343-3g5-3e78</ PartNumber >
  <rdf:bag>
<rdf:li>
  <rdf:description about="http://www.widget.com/motor.svg #crankshaft">
    <name>Crankshaft</name>
    <quantity>1</quantity>
    <PartNumber>123-345-3c34</ PartNumber >
  </rdf:description>
<rdf:li>
  <rdf:description about="http://www.widget.com/motor.svg #overhead_cam">
    <name>Overhead cam</name>
    <quantity>1</quantity>
    <PartNumber>1f3-355-7c23</ PartNumber >
  </rdf:description>
</rdf:li>
<rdf:bag>
</rdf:description>
```

In the last example, the `<rdf:description>` element points to an SVG document which provides an illustration describing a motor. The motor's properties, "name", "quantity" and "PartNumber", are defined as elements located immediately under the `<rdf:description>` element. Then, the parts illustrated in the SVG document are listed in a bag collection. Each part is treated as a separate resource and possess its own property set.

The last example has the benefit of being compact but the "parts" are intimately attached to a particular "whole". In the first example we followed closely our whole-parts information set pattern and an interpreter would not need too much work to transform the rdf format into a whole-parts information set. In the second example, the interpreter has more work to isolate the parts from the whole. A very sophisticated interpreter can do that. If, however only a simple interpreter can be realized, then, the following expression will be easier to be interpreted and to be transformed into a whole-parts information set.

```
<rdf:description id="motor" about="http://www.widget.com/motor.svg">
  <name>fuel injection motor</ name >
  <quantity>1</quantity>
  <PartNumber>1343-3g5-3e78</ PartNumber >
  <rdf:bag>
    <rdf:li resource="http://www.widget.com/motor.svg #crankshaft"/>
    <rdf:li resource="http://www.widget.com/motor.svg #overhead_cam"/>
  </rdf:bag>
</rdf:description>

<rdf:description about="http://www.widget.com/motor.svg #crankshaft">
  <name>Crankshaft</name>
  <quantity>1</quantity>
  <PartNumber>123-345-3c34</ PartNumber >
</rdf:description>
```

```

<rdf:description about="http://www.widget.com/motor.svg #overhead_cam">
  <name>Overhead cam</name>
  <quantity>1</quantity>
  <PartNumber>1f3-355-7c23</ PartNumber >
</rdf:description>

```

In the last example, each object is treated as a separate resource and an interpreter can easily transform an `<rdf:description>` element into a “whole” object and the `<rdf:li>` elements into pointers to parts. A part is also a “whole” object that can be decomposed into parts and so on and so forth.

Xlink

The xlink framework provides a basis for resource linkage. When an element includes certain xlink attributes and follow some structure rules, this element is then converted into a link. From our context, an xlink construct which is quite interesting is the `xlink:extended` element type. I called it an element type since we have, at the time of this writing, no names for the xlink architectural inheritance. An element can inherit some characteristics from xlink elements by a simple inclusion of one or several xlink attributes. An element inherits an xlink element type by including an `xlink:type` attribute set to the appropriate value.

An `xlink:extended` element type is a one to many kind of link and to be so, it should contain locator elements. Locators should be direct children of an `xlink:extended` element.

An interesting secondary advantage to use xlink elements is that an xlink interpreter can provide an alternative interpretation to our elements. Thus, an element can be interpreted in different ways. The xlink element also provide standard browsing behaviors like, for instance, if the document is to be seen in a new window, has to replace the current document or to simply get the linked document to be inserted in the current one.

To implement the whole-parts pattern, our element need only to inherit from the xlink element types.

Semantic links

Semantic links are a merge of the rdf and xlink worlds. We retained the concept of frame as introduced by the RDF framework and the concept of linkage as introduced by the xlink framework. The following example illustrate this new species.

```

<topic id="WebNews" xlink:type="extended" xlink:title="news">
  <description>news about the web technologies</description>
  <format>HTML</format>
  <item xlink:role="resource" xlink:type="locator"
        xlink:href=http://www.zdnet.com/>
  <item xlink:role="resource" xlink:type="locator"
        xlink:href=http://www.cnet.com/>
  <item xlink:role="resource" xlink:type="locator"
        xlink:href=http://www.cmp.net/>
  <item xlink:role="resource" resource xlink:type="locator"
        xlink:href="http://www.talva.com/xml">
    <author>Didier PH Martin</author>
    <date> March 12 2000</date>
    <language>English</language>
    <version>1.0</version>
  </resource>
  <item xlink:role="resource" xlink:type="locator"

```

```

xlink:href="#internet_news"/>
<item xlink:role="topic_association" xlink:type="locator"
xlink:href="#Internet_technologies"/>
</topic>

```

The previous example is the result of a request to a news feed server. A query about the “WebNews” topic is made to the news feed server. The news feed server returns a semantic link that could be interpreted as well by an xlink interpreter as it can also be interpreted by our interpreter. Thus, this structure offers multiple interpretations.

The <topic> element includes an xlink:type attribute which is set to the value “extended”, this last attribute/value pair transforms the topic element into an <xlink extended> element. An xlink interpreter expects that this element contains one or more <locator> elements as children. An xlink extended type is thus a collection of pointers to resources. Therefore, our topic is a link pointing to a collection of resources. The <item> element is transformed into a locator type by including the xlink:type attribute set to the value “locator”. An xlink interpreter would interpret each locator as a reference pointing to a resource.

The <topic> element also includes a set of properties defined as we would do with the RDF framework. For instance, the “description” and “format” properties are included as we would do with an <rdf:description> element.

One of the <item> elements is also treated as a frame that includes the “author”, “date”, “language” and “version” properties. Again, we used the same convention as RDF where properties are declared as sub elements. Thus, this <item> element is, in addition to be a locator, also an RDF frame which contains a set of properties.

The semantic link structure, when locators include properties, lead us to change our basic structure. This is because the locators cannot be isolated as they would be with RDF. In the xlink framework, the locators should be immediate children of an xlink extended element type. So, in order to be able to model both kind of structures and to be able to support constructs like we encountered with the RDF example and the last one, we will slightly modify the basic whole-parts information set. Modify it in such ways that a pointer node can contain properties. Thus the modified structure looks like the structure below.

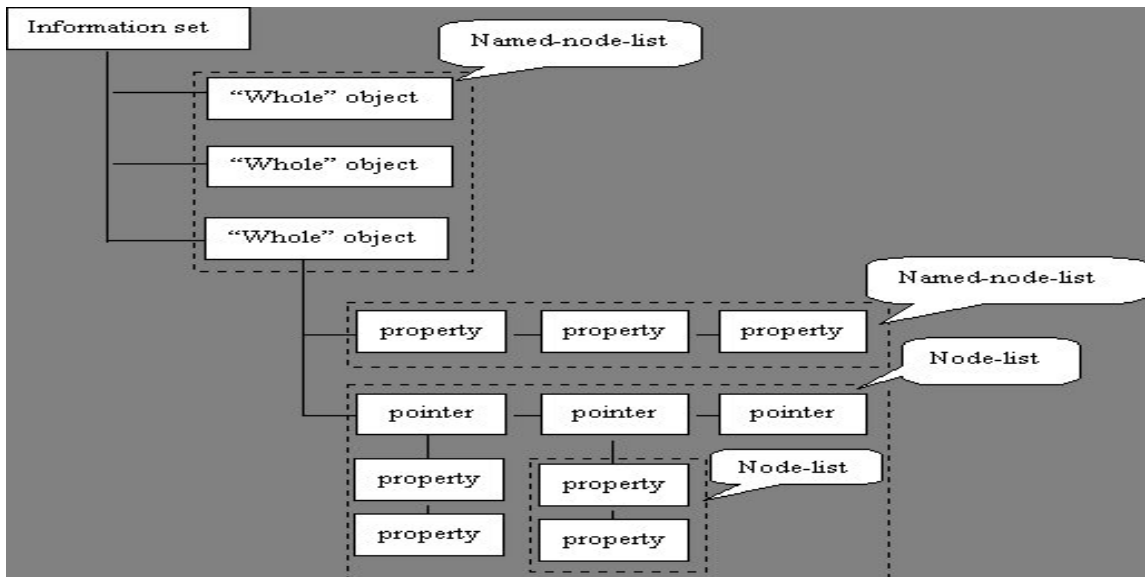


Figure 6. Whole-parts information set structure modified to integrate both topic maps and RDF frames. Semantic links information set

With this new modified information set, a pointer may include properties about the referred part. This is useful in some cases when the part itself is an external resource that is not accessible to the interpreter. Or simply that this information is not available.

One of the <item> element is a locator pointing to an other topic contained in the same file as the “WebNews” topic. This is how we create hierarchies. Thus, a topic resource collection can point to an other topic collection located in the same file but also to an external document identified by a URI. A topic hierarchy can be composed of <topic> elements contained in the same file or contained in external files.

Xlink roles can be assigned to each xlink element. This is particularly useful when particular roles are assigned to locator elements. In our last example, we used the role “resource” to express the idea of a reference to a resource. The last <item> element has a different role. It is assigned to the “topic_association” role. This concept is imported from the topic map universe where a topic can be associated to an other one to form a network of topics. So here, we assigned the “topic_association” role to one of the pointers to underline a special relationship between the topic and one of its parts. Xlink allows us to create families of parts by assigning different role to locators.

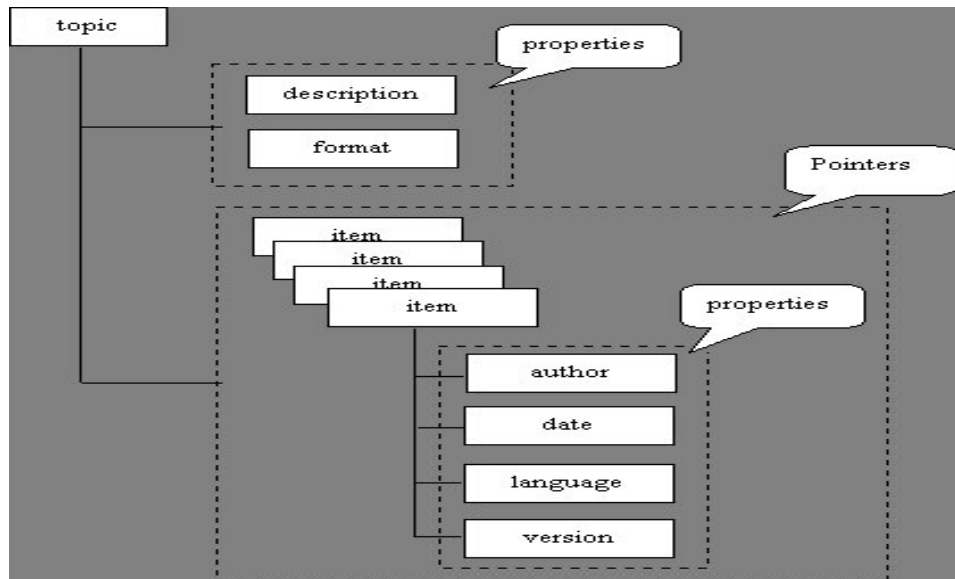


Figure 7. Semantic links information set example

Conclusion

This whole experiment with two XML frameworks helped us understand how each one can be related to the other. For instance, an RDF document can be interpreted and transformed into a whole-parts information set, then from this latter, an xlink document can be created and vice versa. By separating the underlying model from the different serialization formats we were able to translate from a format to another.

This is a work in progress and I am continuing this experiment in order to discover other common grounds.

Bibliography

[RDF]	<u>http://www.w3.org/TR/REC-rdf-syntax</u>
[Xlink]	<u>http://www.w3.org/TR/xlink/</u>
[Facetted classification]	Vickery, B.C., Facetted Classification. New Brunswick, Rutgers University Press, 1965.
[GROVE]	<u>http://www.oasis-open.org/cover/topics.html#groves</u>
[Frame]	Rich & Knight, Artificial Intelligence, McGraw-hill, 1993

Author

Didier PH Martin

President
Talva Corp.
Postal Address:
320 Mongeau
J3N 1E2 St-Basile
Quebec
Canada
Telephon: 450-653-8170
Fax: 775-418-6258
E-mail: martind@netfolder.com
Web: www.netfolder.com

Didier PH Martin - Didier PH Martin is Talva corp. CEO. Talva Corp. develops XML tools since the last two years.