TM/XML – Topic Maps fragments in XML

Lars Marius Garshol[1] and Dmitry Bogachev[2]

[1] Ontopia AS, Oslo, Norway,
larsga@ontopia.net, http://www.ontopia.net
[2] Omega Consulting,
dmitryv@cogeco.ca

Abstract. This paper describes TM/XML, an XML syntax for Topic Maps that is very close to the natural, or colloquial, XML representation of the information in the topic map. It can be used to process Topic Maps data with XML tools, and integrate non-Topic Maps systems with Topic Maps systems.
Further, TM-Views, a mechanism for describing what to include when extracting a fragment from a topic map, is described. TM-Views improves the usability of TM/XML in the described use cases, but can also be used independently of TM/XML.

1 Introduction

Topic Maps is often described as ideal for information integration, because of the clear conceptual model and built-in support for merging. However, one of the key challenges for those who wish to build Topic Maps-based applications is making the Topic Maps application communicate with other applications, which are nearly invariably not Topic Maps-based. Solutions can of course be developed, but generally require labour- and knowledge-intensive custom programming against the Topic Maps engine API [TMAPI].

XML is today the lingua franca for information interchange, and so this paper attempts to simplify such integrations by making it easier to move data between XML and Topic Maps representations. The rationale is that any pre-existing applications will most likely be able to export and import some form of XML, or at least find XML data easier to work with.

Admittedly, an XML representation of Topic Maps already exists. The XTM syntax for Topic Maps is defined in the Topic Maps ISO standard [ISO13250-3] and is the de facto standard for interchange of Topic Maps today. This syntax is, however, difficult to process with existing XML tools [Robie01] and requires an in-depth understanding of Topic Maps to both produce and interpret.

This paper therefore proposes a more natural XML representation of Topic Maps, called TM/XML, which aims to make it easier to integrate Topic Maps into XML-capable environments. The TM/XML syntax is a private proposal, to be implemented by anyone who is interested, and not an official standard.

Most scenarios where TM/XML is useful (described in the following use cases section) involve integration with remote systems, which also requires a web service interface for accessing the Topic Maps system. This is provided by TMRAP [Garshol05], and TM/XML is designed to be used together with TMRAP. In TMRAP operations clients can request to receive the retrieved fragment in a specific Topic Maps syntax, and TM/XML is one of the possible syntaxes.

The web service interface operates on fragments of topic maps, which is what TM/XML is used to represent, and this requires an ability to describe the boundaries of the fragments. This paper also describes a syntax, called TM-Views (see section 4), for defining views of topic maps that also determine the fragment boundaries.

1.1 Applications of the syntax

The uses of the TM/XML syntax are theoretically the same as those for XTM, but in practice TM/XML is intended and optimized for a particular set of uses, which are described below. Simply stated, TM/XML is meant for use when Topic Maps are processed with normal XML tools rather than with Topic Maps-aware tools.

Implementing presentation using XML tools In most cases, a Topic Maps application contains a presentation layer developed using a Topic Maps-aware tool. However, in some cases, this may not be practical, for example because:

- The Topic Maps application is part of a larger application or infrastructure where all presentation is implemented using XML tools, for example XSLT. The choice to use XML tools for all processing effectively precludes the use of Topic Maps-aware tools.
- The presentation layer is deployed in some application server or framework for which no Topic Maps support is available, either because it does not exist, or because the Topic Maps software used by the organization runs on a different platform.
- The presentation layer is developed by a team which has no skill in Topic Maps, and where it is not considered cost-effective to train them on Topic Maps and related technology necessary to implement presentation of Topic Maps content. The presentation team is most likely already familiar with XML technologies, however.

In all of these cases, extracting fragments from the topic map and passing them to the presentation layer in XML form and processing them with standard XML tools is the preferred solution. TM/XML greatly simplifies this, as shown below.

Building knowledge hubs One very interesting use case is creating "knowledge hubs" where information from many sources is brought together and integrated. In this scenario, information providers invoke operations on a Topic

Maps server to add information coming from outside sources, and the server then integrates this information into the existing topic map. Information consumers invoke other operations to retrieve fragments of information which are extracted from the topic map.

A very important variation on this scenario is using the knowledge hub to create a common view of some information domain across many different portals. (This has often been referred to as "portal integration".) In this scenario the knowledge hub (often itself part of a portal) contains the topic map of the information domain, whereas the client portals use information from this central topic map as part of their own presentation. This enables common subjects to be presented in a larger context than that possessed by an individual portal.

A key question in this scenario is how the client portals should present the Topic Maps fragments they retrieve from the hub. In many cases, purchasing separate Topic Maps software licenses and training the developer teams for each portal is out of the question for economic reasons. However, since the Topic Maps fragments are in any case being transmitted in XML syntax, a natural alternative is to let the client portals use ordinary XML tools to present the fragments.

Again, TM/XML supports this use case by enabling information to be inserted (going XML-to-TM) and retrieved (going TM-to-XML) using a natural XML representation that is easily processable with existing XML tools and also easily understood by developers who have little understanding of Topic Maps. This greatly simplifies creation of the knowledge hubs.

When to use XTM It is not the intention of the authors that TM/XML should replace XTM as the standard XML syntax for Topic Maps. Instead, the intention is that TM/XML should be used in use cases like those described in this section, whereas XTM should continue to be used for interchange of topic maps between Topic Maps-aware tools. In the latter case, the difficulties of extracting domain-specific information from XTM (learning curve, complexity of processing, etc) are irrelevant, as the processing is done by Topic Maps-aware software. That software will provide Topic Maps-specific means of processing the content, such as TMQL and TMAPI, which are much simpler than working directly with XTM.

1.2 Design principles

The goals for TM/XML were to produce an XML syntax for Topic Maps that

- is easy to learn, and does not require in-depth knowledge of Topic Maps,
- represents information close to the way it would naturally be represented in XML by someone not having Topic Maps in mind, and
- is compact and easy to process with common XML tools, like XSLT.

Essentially, the TM/XML syntax is a mapping between two data models: that of Topic Maps, and that of XML. The two main ways to approach this, in the terminology of [RDFTM], are object mappings and semantic mappings. XTM effectively represents an object mapping, in that the constructs of the Topic Maps data model are represented directly using XML constructs[3].

For TM/XML we have chosen a semantic mapping, as such mappings generally generally score higher on naturalness and compactness[RDFTM], which again leads to ease of learning and processing.

Choosing a semantic mapping effectively means that instead of getting a generic domain-independent representation of the topic map in XML (like XTM), in which all topic maps use the same XML vocabulary, we get a domain-specific representation. In other words, XTM is generic, while TM/XML adapts itself to the domain vocabulary of the domain.

2 Introduction to the syntax

The TM/XML syntax for Topic Maps is inspired by the RDF/XML syntax for RDF [RDF/XML]. RDF/XML is the standardized XML syntax for interchange of RDF content, like XTM for Topic Maps, but unlike XTM it represents a semantic mapping from RDF to XML. The general principles of RDF/XML are thus rather similar to those of TM/XML, but due to the nature of Topic Maps, the details are rather different.

The status of the syntax at the moment is that it exists as a fully-developed and specified private proposal. A prototype implementation in Jython based on the Ontopia Topic Maps Engine exists, as does an XSLT stylesheet converting TM/XML into XTM. Productized implementations are likely to follow shortly.

Note that TM/XML is not defined as a serialization of an entire topic map, but as a serialization of a set of topics. This is because the use cases described above all involve the use of TM/XML with fragments. Serializing an entire topic map is no harder than making the set of topics to be serialized the set of all topics in the topic map.

2.1 How it works

The general principle for the syntax is that each topic is represented by an XML element whose type is derived from the topic's type. The characteristics of the topic are represented as child elements of the topic element, again with element types derived from the type of the characteristic. This gives a simple, compact, and natural XML representation for Topic Maps data. Untyped constructs (like variant names) have built-in elements defined as part of TM/XML.

Another principle is that where possible element type names are formed from the subject identifiers of typing topics using namespaces. Where no subject identifiers are available, simple IDs are used.

Below is a simple topic map in LTM syntax:

[3] Of course, in reality the mapping went from XTM to TMDM, rather than vice versa, as XTM predates TMDM.

```
#TOPICMAP ~tm
#PREFIX dc @"http://purl.org/dc/elements/1.1/"

[tm : topicmap = "TM/XML example topic map"]
{tm, dc:description, [[This topic map is a simple example of the use
of TM/XML.]]}

[lmg : person = "Lars Marius Garshol"; "garshol, lars marius"]
{lmg, homepage, "http://www.garshol.priv.no"}

created-by(tm : work, lmg : creator)
presentation(lmg : presenter, tmxml : presented, tmra05 : event)
```

In TM/XML, this topic map would be represented as follows:

```
<topicmap xmlns:iso="http://psi.topicmaps.com/iso13250/"
          xmlns:tm="http://psi.ontopia.net/xml/tm-xml/"
          xmlns:core="http://www.topicmaps.org/xtm/1.0/core.xtm#"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          reifier="tmtopic">

  <topicmap id="tmtopic">
    <iso:topic-name>
      <tm:value>TM/XML example topic map</tm:value>
    </iso:topic-name>
    <dc:description>This topic map is a simple example of the use of
    TM/XML.</dc:description>
  </topicmap>

  <person id="lmg">
    <iso:topic-name>
      <tm:value>Lars Marius Garshol</tm:value>
      <tm:variant scope="core:sort">garshol, lars marius</tm:variant>
    </iso:topic-name>
    <homepage datatype="http://www.w3.org/2001/XMLSchema#anyURI"
       >http://www.garshol.priv.no</homepage>

    <created-by role="creator" topicref="tmtopic" otherrole="work"/>

    <presentation role="presenter">
      <presented topicref="tmxml"/>
      <event topicref="tmra05"/>
    </presentation>
  </person>
</topicmap>
```

The reifier attribute on the topicmap element refers to the ID of the topic reifying the topic map. This is the opposite of the XTM representation, where the reifying topic would refer to the topic map. (See the XTM version in 2.2.)

The second topicmap element represents a topic of type topicmap (the one that reifies the topic map). The iso:topic-name element name appears because this is the PSI of the default topic name type in TMDM [ISO13250-2], given to topic names which have no type (like those in the LTM fragment). The tm:value element is introduced as a wrapper element for the topic name value in order to ensure that topic names and occurrences can be distinguished, and that variant names can be accomodated together with the topic name value without difficulty.

The tm:variant element is used to represent variants, and the scope attribute contains the scope of the variant. The same attribute can be used throughout TM/XML to represent scope.

The dc:description and homepage elements represent occurrences. The element type name dc:description is a QName, so this refers to the occurrence type by PSI, where the PSI is the concatenation of the namespace URI and the local name.

The created-by and presentation elements both represent associations. The presentation element has one sub-element for each role in the association not played by the parent topic.

2.2 Comparison with XTM

That TM/XML provides a simpler, and more easily understandable, representation of the topic map information than the equivalent XTM representation should be self-evident. However, for those wanting evidence, the following shows the same topic map in XTM. (We describe this as being “the same” topic map, as the XTM and TM/XML representations would produce identical TMDM instances.)

```
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
          xmlns:xlink="http://www.w3.org/1999/xlink"
          id="id2588719">
  <topic id="tmtopic">
    <instanceOf>
      <topicRef xlink:href="#topicmap"/>
    </instanceOf>
    <subjectIdentity>
      <subjectIndicatorRef xlink:href="#id2588719"/>
    </subjectIdentity>
    <baseName>
      <baseNameString>TM/XML example topic map</baseNameString>
    </baseName>
    <occurrence>
      <instanceOf>
        <subjectIndicatorRef
        xlink:href="http://purl.org/dc/elements/1.1/description"/>
    </instanceOf>
    <resourceData>This topic map is a simple example of the use of
  TM/XML.</resourceData>
  </occurrence>
</topic>
<topic id="lmg">
  <instanceOf>
    <topicRef xlink:href="#person"/>
  </instanceOf>
  <baseName>
    <baseNameString>Lars Marius Garshol</baseNameString>
    <variant>
      <parameters>
        <subjectIndicatorRef
           xlink:href="http://www.topicmaps.org/xtm/1.0/core.xtm#sort"/>
      </parameters>
      <variantName>
        <resourceData>garshol, lars marius</resourceData>
      </variantName>
    </variant>
  </baseName>
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#homepage"/>
    </instanceOf>
    <resourceRef xlink:href="http://www.garshol.priv.no"/>
  </occurrence>
</topic>
<association>
  <instanceOf>
    <topicRef xlink:href="#created-by"/>
  </instanceOf>
  <member>
    <roleSpec>
      <topicRef xlink:href="#creator"/>
    </roleSpec>
    <topicRef xlink:href="#lmg"/>
  </member>
  <member>
    <roleSpec>
      <topicRef xlink:href="#work"/>
    </roleSpec>
    <topicRef xlink:href="#tmtopic"/>
  </member>
  </association>
  <association>
    <instanceOf>
      <topicRef xlink:href="#presentation"/>
    </instanceOf>
    <member>
      <roleSpec>
        <topicRef xlink:href="#presenter"/>
      </roleSpec>
      <topicRef xlink:href="#lmg"/>
    </member>
    <member>
      <roleSpec>
        <topicRef xlink:href="#presented"/>
      </roleSpec>
      <topicRef xlink:href="#tmxml"/>
    </member>
    <member>
      <roleSpec>
        <topicRef xlink:href="#event"/>
      </roleSpec>
      <topicRef xlink:href="#tmra05"/>
    </member>
  </association>
</topicMap>
```

The complexity of the syntax affects the ease of processing with standard XML tools quite dramatically, and more than may be immediately obvious. For example, the XPath to find all creators of a work is shown below. (The `$work` variable contains the ID of the work.)

```
//xtm:association
  [xtm:member[xtm:roleSpec / xtm:topicRef / @xlink:href = '#work']
             [xtm:topicRef / @xlink:href = concat('#', $work)]]
  [xtm:instanceOf / xtm:topicRef / @xlink:href = '#created-by']
  / xtm:member[xtm:roleSpec / xtm:topicRef / @xlink:href = '#creator]
  / xtm:topicRef / @xlink:href
```

This query actually only finds the IDs of the creators, rather than the creators themselves. It also makes the simplifying assumption that no `subjectIndicatorRef` and `resourceRef` elements are used to refer to topics. In reality, this assumption is often wrong.

For comparison, a query that returns the elements representing the creators (as opposed to just the IDs as above) in the same situation on TM/XML would look as follows:

```
//person [created-by/@topicref = $work]
```

XPath is only one way among many to process XML, but we here assume that implementing the same operation using other XML tools would be equally complex.

3 Formal definition

This section defines the syntax and its processing more precisely. All TM/XML documents will be valid according to the RELAX-NG [ISO19757-2] schema shown below[4].

```
default namespace = "http://psi.ontopia.net/xml/tm-xml/"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

start = topicmap

topicmap = element * { reifier?, topic+ }
reifier = attribute reifier { text }

# some form of identifier is required
topic = element * { ((id, identifier*, locator*) |
                     (id?, ((identifier+, locator*) |
                            (identifier*, locator+)))),
                    topicname*, occurrence*, association* }
id = attribute id { xsd:ID }
identifier = element identifier { xsd:anyURI }
locator = element locator { xsd:anyURI }

topicname = element * { reifier?, scope?, value, variant* }
scope = attribute scope { text }
value = element value { text }
variant = element variant { scope, reifier?, datatype?, text }
occurrence = element * { reifier?, scope?, datatype?, text }
datatype = attribute datatype { xsd:anyURI }

association = unary | binary | nary
unary = element * { reifier?, scope?, role }
role = attribute role { text }
binary = element * { reifier?, scope?, role, otherrole, topicref }
otherrole = attribute otherrole { text }
nary = element * { reifier?, scope?, role, assocrole, assocrole+ }
assocrole = element * { topicref }
topicref = attribute topicref { text }
```

[4] RELAX-NG was used because DTDs can not describe a vocabulary where element types are defined by their signature instead of by their names, nor can W3C XML Schema.

3.1 The serialization process

The input to the serialization process is an element type name, and a set of topics. The element type name is used for the root element, since there is nothing in the set of topics that could tell us what root element type name would be suitable, and since the element type name is in any case not meaningful.

To produce the output, perform the steps below in order for each topic in the input set, and wrap the entire output in an element of the type given as input to the process. If the topic map is reified, add a `reifier` attribute containing a topic reference to the reifying topic (procedure below).

All specific elements mentioned in the steps below belong to the `http://psi.ontopia.net/xml/tm-xml/` namespace. The conventional namespace prefix for this namespace is `tm`, although any prefix may be used.

1. Produce the element type name from the type of the topic (following the procedure described below). If it has no type, use `topic`. If it has more than one type, pick one arbitrarily. (The remaining types will be captured as associations.)
2. If the topic has no subject identifier or subject locator, produce a unique ID for the topic.
3. Output the start tag for the element with the element type name produced in step 1, and, if an ID was produced in step 2, an `id` attribute with that ID as the value.
4. For each subject identifier of the topic, output the identifier in an `identifier` element.
5. For each subject locator of the topic, output the locator in a `locator` element.
6. For each topic name of the topic, produce an element name (procedure below) from the type of the topic name. If the name's scope is non-empty, add a `scope` attribute, containing whitespace-separated topic references (as defined below). Add `reifier` as for the topic map. Then do the following:
   – Output a child element `value` containing the string value of the name.
   – For every variant of the topic name, output a `variant` element, with the `reifier` and `scope` attributes produced the same way as for the topic name. The string value of the variant is output as the content of the element, and if the datatype is not string, the datatype URI is output in the `datatype` attribute.
7. For each occurrence of the topic, produce an element type name (procedure below) from the occurrence type topic. Add scope, reifier, and datatype as above. The string value of the occurrence becomes the element content.
8. For each association role of the topic, except that in the type-instance association used to produce the element type name for the current topic, produce a new element as described below.
   Produce the element type name of the new element from the association type. Add an attribute named `role` containing a topic reference to the type of the role played by the current topic. The `scope` and `reifier` attributes

are added as above (with the scope and reifier of the association the role is part of).

   (a) If the association has only one role, output an empty element with the attributes and name described above.
   (b) If the association has two roles, add a `topicref` attribute with a reference to the topic playing the other role, and a `otherrole` attribute with a reference to the type of the other role. Then output the element with the name and attributes as given above, and no content.
   (c) If the association has more than two roles, output the element with the name and attributes as given above. Then, for each other role in the association output an empty child element with an element type name produced from the role type. Give each element a `topicref` attribute with a reference to the topic playing the role.

9. Output the end tag for the topic element.

No particular requirements are placed on whitespace, except that whitespace must not be added to the contents of elements with text content. All text content must of course be properly escaped to ensure that the resulting XML is well-formed.

3.2 Producing element type names

To make an element type name from a topic follow the procedure below:

– If the topic has a subject identifier, use that. If it has more than one, pick one at random. Divide the URI in two at the first '#' or '/' character from the end. The first part becomes the namespace URI, the second part the local name. The choice of namespace prefix is undefined.
– Failing that, use the last part of the topic's item identifier (if there is one; if there is more than one again pick randomly) after the first '/' or '#' from the end. If the result is not unique, add a numeric suffix (starting with 1) to ensure uniqueness.
– Failing that, auto-generate an element type name from the topic's name (again taking care not to create duplicates).
– It is an error if no element type name can be assigned.

3.3 Producing topic references

Given a topic, a reference to it is produced according to the procedure below:

1. If the topic has a subject identifier, use it. If it has more than one, select one at random. Create a qualified name as for element type names.
2. Failing this, produce a unique ID. How this is done is left to the implementation to determine.

3.4 Deserialization

Producing a topic map from a TM/XML instance is simply the reverse of the serialization process described above. The document element can be ignored (except its `reifier` attribute, if present). Its child elements all create topics. The child elements of topics can be tracked in order, and the various kinds of elements are all structurally different, in such a way that each can be correctly mapped to the corresponding Topic Maps construct.

4 Filtering fragments with views

Topic Maps servers can be integrated with external systems by providing basic operations such as "get-topic", "get-topic-list", "update-topic", "add-topic" and "delete-topic" [Garshol05]. In simple cases these operations can use a predefined set of rules to determine what kind of information about a topic can be retrieved from or submitted to a topic map server.

For example, a typical rule for the "get-topic" operation could be:

1. Retrieve all information about main topic including identifiers, names, occurrences, and associations.
2. For all referenced topics retrieve identifiers and names only.

External systems can use a sequence of "get-topic" operations if additional information about referenced topics is required. If the topic map server is restricted by these basic rules then support of interesting use cases would require implementing quite "chatty" sessions between external systems and a topic map server. Client applications would also receive a lot of information about topics which this client is not interested in. Another problem occurs with updates: different external systems can be responsible for updates of different slices of information about topics. A requirement to submit complete information about topic can complicate the communication protocol between external systems and a topic map server.

This section introduces TM-Views – a mechanism for defining flexible filtering rules which can be used in combination with TM/XML for serialization/deserialization of topic map fragments.

4.1 TM-Views – what is it?

The main goal of TM-Views is to provide the ability to specify which pieces of information about a topic of interest and related topics to include in fragments during communication between external systems and a topic map server.

TM-Views includes an XML vocabulary for defining views and a procedural component which implements filtering rules and integration with TM/XML serialization/de-serialization.

Views enable users to specify which specific topic map constructs should be selected from a topic map for a topic in question. Views also help manage traversal of required associations.

Views consist of a set of patterns. Each pattern selects some constructs from a topic (identifiers, names, occurrences, or associations). Using TM/XML these constructs are mapped to a natural XML syntax.

Views allow patterns to be specified not only for the starting topic but also for topics referenced by associations, thus allowing "path-based" filtering rules. For example, we can specify that if the main topic is a person then a topic for a company-employer and topic for location of this company should be included in the fragment.

Generally speaking, some topics can be referenced several times by different paths during association traversing. Information from different paths is combined for each referenced topic in one topic element.

The example below demonstrates how a view can be defined.

4.2 Example of view definition

```
<view xmlns="http://psi.ontopia.net/xml/tm-views/"
      id="person_view" name="Person view">

  <topic type="person">
    <identifier type="subjectIdentifier" />

     <basename type="*">
       <except>
         <basename type="nickname" />
       </except>
     </basename>

     <occurrence type="homepage" />

     <association type="employed-by" role="employee"
                  otherrole="employer">
       <topic type="company">
         <identifier type="*"/>
         <basename type="*"/>
         <occurrence type="homepage"/>
         <association type="located-at" role="object"
                      otherrole="location"
                      playertype="city town location"/>
       </topic>
     </association>

     <association type="knows-person" role="person"
                  otherrole="person" playertype="person"/>
  </topic>
</view>
```

This view defines fragments for a main topic of type “person”. The `type` attribute on `topic` and the `playertype` attribute on `association` provide “hints” for choosing the type during TM/XML serialization.

If a topic is an instance of the specified type then this type is used as the basis for a mapping to an XML element. If there are several type hints, then they are validated in some order. If there is no valid type hint, then the general TM/XML rule is used.

View definitions allow patterns to be specified based on the types of names, occurrences, identifiers, and associations. Wild card “*” and exceptions are supported to simplify definition of filtering rules.

4.3 View definition language

In this section we define the syntax for the view definition language using RELAX-NG schema.

```
default namespace = "http://psi.ontopia.net/xml/tm-views/"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

start = view

view = element view {
    attribute id { xsd:ID },
    attribute name {text}?,
    topic*
}

topic = element topic {
        attribute type {text},
        identifier-pattern*,
        topicname-pattern*,
        occurrence-pattern*,
        association-pattern*
}

identifier-pattern =  element identifier{
       attribute type{text}
}


topicname-pattern =  element basename{
       attribute type{text},
       element except{
            element basename{attribute type{text}}+
       }?
}
```

```
occurrence-pattern =  element occurrence{
       attribute type{text},
       element except{
            element occurrence{attribute type{text}}+
       }?
}

association-pattern =  unary-pattern | binary-pattern | nary-pattern

unary-pattern= element association{
       attribute type{text},
       attribute role{text},
       element except{
            element association{
                    attribute type{text},
                    attribute role{text}
            }+
       }?
}

binary-pattern = element association{
       attribute type{text},
       attribute role{text},
       attribute otherrole{text},
       attribute playertype{text}?,
       topic*,
       element except{
            element association{
                    attribute type{text},
                    attribute role{text},
                    attribute otherrole{text}
            }+
       }?
}

nary-pattern= element association{
       attribute type{text},
       attribute role{text},
       other-role-pattern,
       other-role-pattern+,
       element except{
            element association{
                    attribute type{text},
                    attribute role{text},
                    attribute otherroles{text}
            }+
        }?
}

other-role-pattern=element otherrole{
     attribute type{text},
     attribute playertype{text}?,
     topic*
}
```

4.4 Applying a view to a topic map

The example below demonstrates the result of applying the previously defined view to an example topic map. (Some details in the example have been replaced with ... for readability.)

```
<topicmap view="person_view" ...>
   <person id="lmg" top_topic="true">
     <iso:topic-name>
       <tm:value>Lars Marius Garshol</tm:value>
     </iso:topic-name>
     <homepage datatype="...">http://www.garshol.priv.no</homepage>
     <employed-by topicref="ontopia" role="employee" otherrole="employer"/>
     <knows-person topicref="dmitrybv" role="person" otherrole="person"/>
   </person>

   <person id="dmitrybv">
     <iso:topic-name><tm:value>Dmitry Bogachev</tm:value></iso:topic-name>
   </person>

   <company id="ontopia">
     <iso:topic-name><tm:value>Ontopia</tm:value></iso:topic-name>
     <homepage datatype="...">http://www.ontopia.net</homepage>
     <located-at topicref="city_Oslo" role="object" otherrole="location"/>
   </company>

   <city id="city_Oslo">
      <iso:topic-name><tm:value>Oslo</tm:value></iso:topic-name>
   </city>
</topicmap>
```

The XML elements at the top level of this fragment represent topics. Sub-elements are used to encode identifiers, names, occurrences and associations. The only pieces of information requested by the view definition provided above are presented in this fragment (combined with the default filtering rules).

4.5 The fragmentation process

In this section we define the fragmentation process. The input to the fragmentation process is a set of "top topics" and a view definition.

To produce the output:

1. "normalize" the view definition by replacing each "playertype" attribute with a "topic" sub-element which has a "type" attribute equal to the value of the "playertype" attribute; add "any identifier" and "any name" patterns to created "topic" sub-element;
2. let Topic Description List denote an empty list of topic descriptions;
3. let Topic View List denote a list of topic view definitions ("topic" sub-elements inside of a "view" element);
4. for each topic in the "top topics" set apply Create a Topic Description procedure described below with the "view-list" argument binded to the Topic View List and the "top-topic" argument bound to the "true" value;
5. for each topic description in the Topic Description List produce a "topic wrapper" element with an "id" attribute based on the TM/XML serialization rules;
6. for each construct inside of the topic description produce its XML representation based on TM/XML serialization rules;
7. create a "fragment wrapper" element based on TM/XML serialization rules and insert the results of the serialization of the topic descriptions inside of the "fragment wrapper" element;

"Create a Topic Description" procedure:

1. if there is no topic description for a given topic in the Topic Description List then create an empty topic description;
2. if the "top-topic" argument is "true" then mark the topic description as "top_topic";
3. for a given topic and a given list of topic view definitions find a first topic view definition which matches the topic based on the "type" attribute and types of the topic;
4. if a matching topic view definition is found then add matching topic type to the topic description;
5. if a matching topic view definition is found then use it to add identifier, name, occurrence and association descriptions based on the procedures described below;
6. if there is no matching topic view definition or the topic view list is empty then add all identifiers and names from the given topic to the topic description;

"Create an Identifier Description" procedure:

1. for each identifier related to a given topic find a first identifier pattern which matches the identifier based on the "type" attribute;

2. if there is a matching pattern then include a description of the identifier into a topic description;

"Create a Topic Name Description" procedure:

1. for each topic name related to a given topic find a first name pattern which matches the topic name based on the "type" attribute;
2. if there is a matching pattern then include a description of the topic name into a topic description;
3. if the topic name has a scope, for each topic in the scope apply "Create a Topic Description" procedure with an empty topic view list;
4. if the topic name is reified by a topic, apply "Create a Topic Description" procedure for this topic with an empty topic view list;

"Create an Occurrence Description" procedure:

1. for each occurrence related to a given topic find a first occurrence pattern which matches the occurrence based on the "type" attribute;
2. if there is a matching pattern then include a description of the occurrence into a topic description;
3. if the occurrence has a scope, for each topic in the scope apply "Create a Topic Description" procedure with an empty topic view list;
4. if occurrence is reified by a topic, apply "Create a Topic Description" procedure for this topic with an empty topic view list;

"Create an Association Description" procedure:

1. for each association related to a given topic find a first association pattern which matches the association based on "type", "role", and "otherrole" attributes;
2. if there is a matching pattern then include a description of the association into a topic description;
3. for each other role player of the association create a list of topic view definitions based on "topic" sub-elements of the "association" element (can be empty)
4. apply recursively "Create a Topic Description" procedure for the role player and the topic view definition list;
5. if association has a scope, for each topic in the scope apply "Create a Topic Description" procedure with an empty topic view list;
6. if association is reified by a topic, apply "Create a Topic Description" procedure with an empty topic view list;x

4.6 TM-Views and updates

TM-Views support remote editing use cases when some information can be extracted from a topic map as a fragment, transformed into an XML resource using domain-specific vocabulary, modified by users or other systems as an XML resource, and pushed back to a topic map as a modified topic map fragment. Views effectively describe boundaries of the information that can be transferred from and into a topic map. When modified fragment is pushed back, only topic map constructs defined by these boundaries need to be changed.

5 Related work

The Topic Maps syntax in the original Topic Maps standard was similar to XTM 1.0, but unlike XTM 1.0 it was defined as an SGML architecture. This meant that the architectural forms facility in HyTime could be used to map domain syntaxes to the SGML architecture declaratively. The actual mapping would be performed by the SGML/XML parser, or an associated architectural forms processor. The workings of this were actually similar to those of TM/XML, with element type names of element types mapped to topics becoming the topic type, etc. However, architectural forms is more or less dead today, and also requires either modifying the DTD or the instance document, and so is less attractive.

Another obviously related syntax is RDF/XML, which is the standard interchange syntax for RDF[RDF/XML]. It is very similar to TM/XML, but is for RDF only, and is not directly applicable to Topic Maps.

Meaning Definition Language (MDL)[Worden01] provided a means to annotate an XML vocabulary to describe its mapping into an object model, which can then be connected to either UML or RDF Schema. The language relies heavily on XPath for describing the cases in which individual mappings apply. The language has clear similarities with TM/XML, but is not an XML data syntax and more of a method for mapping near-arbitrary XML into an object model.

Other syntaxes have been defined for both RDF and Topic Maps, but all of them have been object mappings (like XTM) instead of semantic mappings like RDF/XML and TM/XML. Some work has also been done on mapping XML to RDF [Miller2004], but this relied on manually writing XSLT stylesheets for each mapping. An earlier work by one of the authors showed how to use XPath to create complex mappings from near-arbitrary XML to RDF using XPath, which could then be converted from RDF to Topic Maps[Pepper02].

The fragmentation process described in this paper is close to other approaches for defining Topic Maps fragments, for example TMShare [Ahmed01] and XTM Fragment Interchange [Garshol02]. The main difference with the solution proposed in this paper is the ability to filter topics being included in fragments and the ability to selectively traverse associations.

6 Conclusion

This paper presents a syntax for Topic Maps, called TM/XML, that can represent all of Topic Maps without loss of information[5], formulated in terms of the information domain. TM/XML is far easier to read for human beings, and also to process with XML tools such as XSLT, than is XTM.

The TM/XML syntax is already integrated with the TMRAP protocol, and can thus meet our use cases. However, for improved usability, more work on updating fragments with new information that only partially replaces existing information is necessary. Further, in order to make it easier for those receiving TM/XML fragments to process these (whether for presentation or other

[5] Or nearly so. Reification of association roles is deliberately not supported.

purposes), conversion from the Topic Maps schema to an XML schema for the received fragments would be desirable.

It is not the intention of the authors that TM/XML become a standardized Topic Maps syntax. If it should be widely adopted, and standardization proposed by others, the authors would not oppose this, but for the time being standardization seems premature.

References

[Ahmed01] K. Ahmed; TMShare - Topic Map Fragment Exchange In a Peer-To-Peer Application; XML Europe 2003, London, England http://www.idealliance.org/papers/dx_xmle03/papers/02-03-03/02-03-03.html

[Garshol05] L. M. Garshol; TMRAP – Topic Maps Remote Access Protocol; forthcoming, to be published in proceedings of TMRA'05.

[Garshol02] L. M. Garshol; XTM Fragment Interchange 0.1 Ontopia Technical Report 2002-09-23; http://www.ontopia.net/topicmaps/materials/xtm-fragments.html

[ISO13250-2] ISO 13250-3: Topic Maps – Data Model; International Organization for Standardization; Geneva. http://www.isotopicmaps.org/sam/sam-model/

[ISO13250-3] ISO 13250-3: Topic Maps – XML Syntax; International Organization for Standardization; Geneva. http://www.isotopicmaps.org/sam/sam-xtm/

[ISO19757-2] ISO 19757-2: Document Schema Definition Languages (DSDL) – Part 2: Regular-grammar- based validation – RELAX NG; International Organization for Standardization; Geneva. http://www.y12.doe.gov/sgml/sc34/document/0362_files/relaxng-is.pdf

[Miller2004] E. Miller, C. M. Sperberg-McQueen; On mapping from colloquial XML to RDF using XSLT; Extreme Markup 2004. http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Sperberg-McQueen01/ EML2004Sperberg-McQueen01.html

[Pepper02] S. Pepper, L. M. Garshol; The XML Papers: Lessons on Applying Topic Maps; XML USA 2002, Baltimore, USA. http://www.ontopia.net/topicmaps/materials/xmlconf.html

[RDFTM] S. Pepper, F. Vitali, L. M. Garshol, V. Presutti; A Survey of RDF/Topic Maps Interoperability Proposals; W3C Note; World Wide Web Consortium, 2005-03-29; http://www.w3.org/TR/rdftm-survey/

[RDF/XML] D. Beckett; RDF/XML Syntax Specification (Revised); W3C Recommendation, World Wide Web Consortium, 10 February 2004; http://www.w3.org/TR/rdf-syntax-grammar/

[Robie01] J. Robie, et al; The Syntactic Web; Markup Languages: Theory & Practice 3.4 (2002): 411-440. http://www.w3.org/XML/2002/08/robie.syntacticweb.html

[TMAPI] TMAPI. http://www.tmapi.org

[Worden01] R. Worden; Meaning Definition Language; in *Professional XML Meta Data*; Wrox Press, 2001.