Topic Maps run from XML and is coming back with Flowers

FLWR-Style in the Topic Maps Query Language using a TMAPI-based implementation

Benjamin Bock
Topic Maps Lab, University of Leipzig
<bock@informatik.uni-leipzig.de>

Sven Krosse
Topic Maps Lab, University of Leipzig
<krosse@informatik.uni-leipzig.de>

Lutz Maicher
Topic Maps Lab, University of Leipzig
<maicher@informatik.uni-leipzig.de>

Abstract

In its history, Topic Maps developed from a syntax-based standard to a pure data model without any syntax defined within its core data model. The syntaxes defined by the ISO for the exchange of Topic Maps are conforming to the generic data model, one of them, XTM, being based on XML. The usage of XTM without a Topic Maps engine is cumbersome because of the generalized schema and the merging rules. For example, extracting useful information from XTM using XSLT requires to query for the typing topics, which is a new subquery just for selecting the right subject whereas it was the entity name in a domain specific XML format. Querying the properties, called Names and Occurrences in Topic Maps, requires additional subqueries because their types and scopes are again Topics and not simple XML entity- and attribute names. The Topic Maps Query Language which is the latest draft in the ISO standardization presented here allows formulating queries against a Topic Maps store in a concise way and outputting the result in various representations. Our implementation TMQL4J uses any TMAPI-compatible store to operate on and allows optimized queries and outputting domain-specific XML. This is demonstrated by generating an ATOM feed for the subject identity record service subj3ct.com.

Keywords: Topic Maps, XML, Topic Maps Query Language, TMAPI, XTM, XSLT, FLWR

1. Introduction

Topic Maps was developed as an SGML and HyTime-based standard [1] a decade ago. With the advancements and popularity of XML, the 2001 version was completely transitioned to XML [2]. The XML version of Topic Maps disclosed the paradigm to a wider audience and enabled several implementations. In the continuing progress of both, development and standardization, the stakeholders and editors came to the conclusion that the data model should be independent of a notation or syntactic representation [3]. While XML may be a good way to *exchange* topic maps, it is not always ideal to *handle* them as XML internally. Especially the merging of constructs - one of the core functionalities of the integration model ([4], Section 6) is described significantly easier using the specialized data model instead of a notation-based model. Consequently, the Topic Maps community focussed on the development of a data model and using XML Topic Maps just as one of several exchange formats.

Due to the way XML is intended to be used from a Topic Maps standards perspective, the XML schemata defined in [2] and [5] are not designed to be extended or modified ([2]: Introduction). This major drawback eliminates the possibility to exploit one of the core features of the *Extensible* Markup Language: Extensibility. Another way is to be found.

First we are going to describe the Topic Maps Data Model and how it is currently accessed using APIs. Then we will illuminate different ways of representing and querying a topic map in different containers, document-based and in relational databases. After that we will compare several possibilities to query a topic map. Finally we will show an application of the here-proposed and recommended way to query a Topic Map for XML data, creating a feed for the Web 3.0 identity service subj3ct.com. We conclude with a self-assessment and outlook.

2. The Topic Maps Data Model

The Topic Maps Data Model [4] is defined in terms of XML Infoset [7]. It is a generic model to encode knowledge and connect encoded knowledge to relevant information resources. This means it is not dedicated to a specific usage or model but generally usable for any particular or general application. A *topic map* consists solely of *topics* and role-based *associations* between topics. Topics represent subjects of discourse which may be anything whatsoever, regardless of whether it exists or has any other specific characteristics, about which anything whatsoever may be asserted by any means whatsoever ([4], section 3.14). Topics consist of their characteristics, i.e. *names* and *occurrences*. Names in turn may consist of several *variants* thereof. Associations consist of *roles*, each role referring to a player which is one topic. Figure 1 shows the hierarchical aspect of this meta-model, the class names being representative for a set of instances thereof.

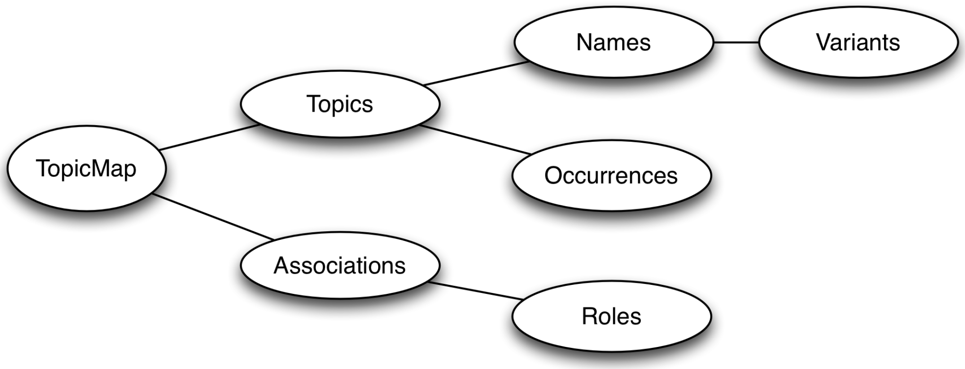

Figure 1. Hierarchical aspect of the TMDM

Names, occurrences, associations, and roles are typed. Types are again topics. Names, variants, occurrences, and associations are referred to as *statements*, while roles are not statements on their own but only within their specific association. The validity of a statement may be constrained by a *scope*. A scope is a set of topics and may be empty. The empty scope is called the *unconstrained scope*. Figure 2 shows the seven entities of the meta-model, i.e. topic map, topic, name, variant, occurrence, association, and role, in their inheritance hierarchy. The inheritance hierarchy shown also introduces another abstract construct: *reifiable*. In the class model of TMDM, all constructs except topics have a restricted set of properties. The concept of reification allows to make additional assertions about a construct by creating a topic representing this construct. All assertions made about this topic are in fact assertions about the reified construct. Obviously a topic is not reifiable because all assertions can be made directly to it.

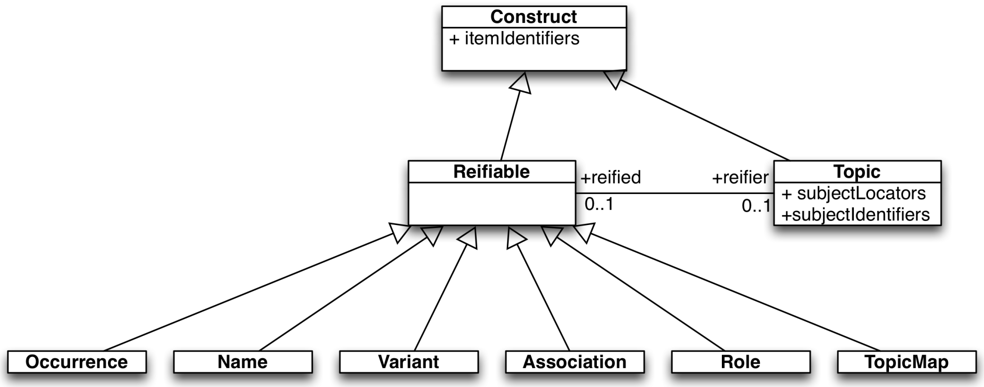

Figure 2. Class diagramm showing inheritance in the TMDM

A topic may have any number of types, instances, supertypes, or subtypes; each of these is again a topic. Additionally, some of the topics are used as type or in the scope of a construct. These topics constitute the *ontology*. A topic map usually contains a model of the real world. For practical reasons, the ontology for a specific application should be further restricted. This task is done using a topic map schema, defined using the Topic Maps Constraint Language [8]. A topic map schema defined in TMCL is again a topic map.

All constructs (including topics) can uniquely be adressed using item identifiers. An item identifier is a *locator*, i.e. a string conforming to the IRI notation defined in RFC 3987. Additionally, topics are addressable using subject identifiers and subject locators. Subject identifiers point to subject indicators in an attempt to unambiguously identify the subject represented by a topic to a human being ([4], section 3.16). Thus subject identifiers are indirect, symbolic representations of a topic. A subject locator refers to the information resources that *is* the subject of a topic. The subject *identifier* http://www.topicmapslab.de/ represents a working group at the University of Leipzig while the subject *locator* http://www.topicmapslab.de/ represents the website of this working group. These two different locators refer to two different subjects and therefor point to two different topics. However, these topics would most likely be associated somehow to each other in a topic map.

A topic can be refered to by any number of identifiers (greater zero). Any topics sharing a common item identifier or subject identifier must be merged by the Topic Maps engine. Item identifiers and subject identifiers share the same notion of *indirectly* adressing a subject, consequently they're treated equally and combined (i.e. given two topics, one with an item identifier `x` and one with a subject identifier `x`, these two are merged). Any two topics sharing a common subject locator must also be merged because they describe the same resources. When merging two topics, these two are replaced by a new one having the union of all their properties (with duplicates removed). The concept of merging facilitates having one single construct representing a real world concept, thus eliminating the need for joins etc.

2.1. TMAPI

The Topic Maps Application Programming interface [9] is a community-driven effort to create a minimal but universal API for programmatically accessing and manipulating data conforming to the TMDM. TMAPI consists of a number of interfaces defined in Java. Each Topic Maps construct is represented by an interface. Additionally there are interfaces for several indexes, exceptions, and locators. A Topic Maps engine is a library implementing these interfaces. The defined goal of TMAPI is to do for Topic Maps what SAX and DOM did for XML: providing a single common API which all developers can code to and which means that their applications can be moved from one underlying platform to another with minimum fuss ([9], section "Why Bother?").

The original TMAPI 1.0 was designed to represent an in-memory representation of (an earlier version of) XTM. During its development, TMDM was in an early stage. This led to the decision to fix TMAPI to an intermediate state between XTM 1.0 and XTM 2.0, the latter being the biunique counterpart to TMDM. This discrepancy is being fixed with TMAPI 2.0 which is aligned to TMDM. However, it has insufficient support for the supertype-subtype relationship and allows the implementation some freedom of interpretation regarding the TMDM's requirement to represent the type-instance relationship between topics as queryable associations. This may require additional handling for some queries and hinders exchange of implementations. On the other hand, this relief simplifies the implementation of engines and is therefor an understandable approach.

TMAPI being defined using the Java-language inspired interfaces for other programming languages like PHPTMAPI for PHP. Similar APIs are being used in Ruby, Python, C#, and C++. Some of the implementing engines offer enhancements, the one for Ruby is described below. When refering to TMAPI in the following, any of the before-mentioned interfaces on the abstraction level of Java TMAPI 2.0 and their implementations are included.

2.2. Ontopia Interfaces and Tolog

For a long time Ontopia was the strongest commercial Topic Maps engine vendor and implementation. It is implemented in Java and comes with a huge set of tools and extending packages. The recent open sourcing of Ontopia brought a popularity boost to the engine and therefor its interfaces and query language *tolog*. Ontopia's interfaces are similar to TMAPI's but they additionally feature transactions, concurrency management etc. There is no need to look at Ontopia's interfaces in detail as they are easily mapped to TMAPI, even though they were not inspired by TMAPI in the first place.

Tolog (written in all small letters originally) [10] is a logic-based query language inspired by Prolog's Datalog and SQL. It was Ontopia's (the company's) approach to the a Topic Maps Query Language and since serves as a temporary solution as long as the official TMQL standard is not completed [TOLOG, chapter 6]. Ibidem has been shown that tolog queries can be implemented efficiently on top of Topic Maps engines using a relational database as backend.

While the tolog language's implementations can execute quite sophisticated queries, it lacks on flexibility regarding the query results. It can only return topics and the other contructs but not any other representation like custom XML. This drawback excludes the otherwise useful language for our purpose: returning custom XML from a Topic Maps query.

2.3. Ruby Topic Maps' Approach to TMAPI

Ruby Topic Maps [11] is a Topic Maps engine written in the Ruby programming language. From a language style perspective, Ruby has several advantages compared to Java and therefor enabled several tweaks in the API. Besides language-specific enhancements, other features where implemented to minimize the effort needed to implement an application on top of RTM's API. Finally, the functional features of Ruby offer powerful functions on sets, simplifying the further processing of the query results.

As mentioned before, type and scope of a construct is always a topic resp. a set of topics. Hence the typing or scoping topic(s) must be queried before they can be used in another query. Here, query refers to a method call, not to a complex query with multiple processing steps. In TMAPI this requires to

1. create a locator representing the string of the IRI: `topicMap.createLocator("some:IRI");`
2. query the topic using the locator either as subject identifier, subject locator, or item identifier: `topicMap.getTopicBySubjectIdentifier(theLocator);`
3. use the topic in the query.

Wherever the TMAPI interfaces expect a topic, it is possible to use a string, a locator, or a topic in RTM. This eliminates the first two of the three steps above but creates the necessity to specify how the IRI string is to be interpreted. RTM allows two syntaxes for that, one inspired from the Compact Topic Maps notation [12]: Prefixing item identifiers with "^" and subject locators with "=", while subject identifiers are used without prefix. The other is inspired from the JSON Topic Maps notation [13], using the prefixes "ii:", "sl:", and "si:" with the obvious meanings. Another feature to point out here is the (switchable) inference of supertypes resp. subtypes when querying.

While RTM's API provides several benefits over TMAPI, its disadvantages are the restriction to use Ruby, only indirectly supported output formats using programming, and the lack of integrated query optimization facilities.

3. Storing a Topic Map and Accessing the Store

3.1. XTM and other document-based formats

The XML Topic Maps Serialization Format, Version 2.0 [5] is defined using RELAX NG [6] and an exact counterpart of the TMDM. It features some shortcuts to allow smaller files like an `instanceOf`-Element instead of a type-instance association. Furthermore it uses different XML elements for the value of an occurrence resp. variant to allow a simpler schema while being precise about the content. Names, occurrences, and variants share the value property in TMDM. The datatype of a name is imlicitly `xsd:string`. The datatype of occurrences and variants can be anything, including `xsd:anyURI` where the value is persisted in a `resourceRef` element instead of a `resourceData` element. A XTM file does not need to be completely merged. It is allowed that there exist two topic definitions in an XTM file which are semantically identical but not yet brought together. Under certain conditions this complicates queries against an XTM file. This restriction is ignored in the example of querying an XTM2 file using XQuery presented later.

For the same reasons XQuery is not the right tool for this job, neither XSLT solves this problem. The disadvantage lies in the non-matching layers of modeling: XQuery and XSLT are optimized to operate on domain-specific XML documents where the schema of a XTM document is generic from a users perspective. Of course the XTM schema is domain-specific from an XML-perspective, but it just serves as a container format in the case of XTM.

The above-mentioned Compact Topic Maps notation is, besides XTM, the only other ISO-standardized format for Topic Maps. It is optimized for users to write down a topic map directly in a file. This file format is not supported by any query languages directly and, compared to XML, hard to parse. Another common format is the also above-mentioned JTM. Neiter CTM nor JTM is free of the drawbacks explained for XTM.

3.2. Relational Databases and NOSQL Approaches

There is no standarized schema in which a topic map should be represented with a relational database. However, several imlementations, including Ontopia and RTM, are able to use relational databases to persist their data. A topic map represented in one of the schemas of the two mentioned implementations is persisted fully merged. It is theoretically possible to query a topic map using SQL directly from the database. This assumes detailed knowledge of the specific database schema, of additional index tables etc. and is not recommended by the authors.

A trend in these days is called NOSQL. NOSQL is loosely summing up all non-relational data stores. There are no citable academic publications for NOSQL yet, so we have to refer to the homonymous entry in the English Wikipedia. NOSQL focusses on horizontal scalability while not necessarily requiring the ACID guarantees provided by relational databases. To short cut this divagation: All query languages created for other data models suffer the mismatch between their particular way of modeling data and the Topic Maps way. The consequent and obvious solution is to create a query language which fits the TMDM and allows advanced processing to fulfill the user's expectations.

4. Querying a Topic Map

4.1. Querying an XTM File Using XQuery

It was already argued not to use XQuery [14] to query an XTM file. Nevertheless we will show an XQuery example to directly compare it with the TMQL approach. The Italian Opera topic map is a commonly used example in the Topic Map community. We're assuming a completely merged topic map so we don't have to care about duplicates. Take the following example for a query of all cities mentioned in this topic map:

Example 1. XQuery cities from the Italian Opera topic map by id

```
declare default element namespace "http://www.topicmaps.org/xtm/";
.//topic[./instanceOf/topicRef/@href="#city"]
```

In this query, a city is refered to by its fragment identifier `#city` which points to a topic element within the same file. This topic element contains additional information like a subject identifier (`http://psi.ontopedia.net/City`), names in different languages (each language designated by a scoping topic in the name elements) and so on. To combine and integrate information from several sources - one of the core competences of Topic Maps - a published subject identifier (PSI) should be used.

Example 2. XQuery cities from the Italian Opera topic map by PSI

```
declare default element namespace "http://www.topicmaps.org/xtm/";
.//topic[./instanceOf/topicRef/@href=concat(
    "#",
    //topic[./subjectIdentifier/@href=
        "http://psi.ontopedia.net/City"]/@id
)]
```

If we now (simplifying but wrongly) assume that for each result `/count(./name)` equals 1, we can append `/name/value/string(.)` to get all city names. Still, we did not query for city names constrained using a given scope. We did not take into account that instanceOf can be equally modeled as association and so on. We stop here and move on to TMQL which fits the given purpose much more natural.

4.2. TMQL Path and SELECT Queries

It was shown that within an XML environment, using Topic Maps in form of generic XTM documents may be cumbersome. It is highly desirable to create documents which adhere to a given schema or may be easily transformed to one using XSLT. One approach to this was the creation of TM/XML which provides a domain-specific representation of a topic map. The schema of TM/XML is dependent of the topic map's schema, not the desired output schema. We present a draft of the Topic Maps Query Language [15] standard which features a query style which is heavily inspired by XQuery FLWOR and allows outputting domain-specific XML documents queried from a Topic Map. This draft uses a path style as its fundamental processing model. The path style is also exposed to the user and can be used, especially for queries for a result with a uniform structure. A third query style is the SELECT style which was inspired by SQL. Within a TMQL processor, both FLWR style and SELECT style can easily be converted to Path style. The expressiveness of these three styles is identical. The only differences are that path expressions cannot have explicit variables and that FLWR is the only style able to output complex content [BA06].

The expressions in a path style navigate along predefined axes. The separator used is >>. The resulting values can be filtered using boolean conditions or used as new starting points for further navigation ([15], section 6.6.1). The following TMQL path query extracts all cities from the Italian Opera topic map.

Example 3. TMQL path querying cities from the Italian Opera topic map

```
http://psi.ontopedia.net/City >> instances
```

The alignment of the query languate to the data model obviously simplifies the structure of the query significantly. Every navigation item is a set of Topic Maps constructs, locators or simple values (e.g. string). All merging is provided by the engine. All inferencing of subtypes etc is done automatically. If there was a subtype of City, e.g. Capital, the list of capitals would be included in the list of cities. There is a shortcut for the instances axis shown in the following snippet:

Example 4. TMQL path querying cities from the Italian Opera topic map using shortcut

```
// http://psi.ontopedia.net/City
```

Except for another single character to signify the traversal of the instances axis this cannot be any shorter. There are shortcuts for many axes but their terminal symbols are still disputed. We look forward to a normative standardization of the query language as an ISO standard.

4.3. The TMQL FLWR Query Style

The FLWR style is a reason to look at a slightly more sophisticated query and using XML output. The following code snipped shows a TMQL FLWR query to get all operas and their composers. As we use the namespace `http://psi.ontopedia.net/` multiple times it is useful to define a prefix `o` for this. The query result is to be wrapped within a root element, named `root` in the example. In the following a naïve approach to the processing model is explained using the example: The set of all instances of opera is iterated and assigned to the variable `$OPERA` variable. The set in the variable is then iterated over to find all instances matching the pattern in the WHERE-clause. In each iteration, the variable `$COMPOSER` is assigned to the matching counterpart in the association. The boolean expression in the WHERE-clause matches an association typed by `o:composed_by` which contains two roles, one of type `o:Composer` and one of type `o:Work`. For each matching association an XML template is filled with the elements given in the templates and the subqueries in curly braces evaluated.

Example 5. TMQL FLWR querying operas and their composers

```
%prefix o http://psi.ontopedia.net/
RETURN
<root>
{
  FOR $OPERA IN // o:Opera
  WHERE o:composed_by( o:Composer : $COMPOSER , o:Work : $OPERA )
  RETURN
  <composed_by>
    <opera>
      { $OPERA / tm:name }
    </opera>
    <composer>
      { $COMPOSER / tm:name }
    </composer>
  </composed_by>
}
</root>
```

5. Application: A subj3ct.com feed

5.1. About subj3ct.com

Subj3ct describes itself as infrastructure technology for Web 3.0 applications. The criterion for being "Web 3.0" is the focus on subjects and semantics rather than documents and links. The goal of Subj3ct is to provide technology and services to enable applications to define and exchange subject definitions [17]. Its goal is not to collect any data but only the information relevant to uniquely identify subjects. Subj3ct provides a web based search interface which provides links to resources about the subjects as well as a REST API for identifier and full text search.

The Subj3ct data is updated using ATOM and SKOS feeds. ATOM is an XML-based format and SKOS is transferred in RDF/XML format which is, as the name suggests, also XML-based. To update subj3ct.com from a Topic Maps-based application, the most obvious way is to use a TMQL FLWR query which is suggested in this paper. The ATOM variant will be discussed here.

Because everybody can publish everything about any subject, some way of weighting relevant sources is useful. Subj3ct assigns trust scores to identifiers. "The Trust Score provides a rough measure of how likely it is that the creator of an identifier agrees with any statement being made about the identifier" ([17], chapter 4.1). This does not mean a trust in the creator of an identifier or the individual who creates statements about that identifier. Neither does it qualify the content which may be returned when requesting any URLs contained in those statments.

5.2. Updating subj3ct.com via ATOM

The following figure shows an excerpt of the ATOM feed provided by the topicmapslab.de community portal. Besides the standard ATOM elements used in the header part, the entries are the interesting part. Additional to the classic atom entries, a subj3ct.com ATOM feed includes links with special `rel` attributes. The attribute values `SubjectIdentifier` and `SubjectEquivalence` are used to designate the subject identifiers. We assume that the different names were chosen to distinguish between those primarily used in the source providing the feed and external ones.

In the ideal case, the subject identifier points to a subject indicator (which is a resource describing the real world concept, as said before). Practically often the representation is at another address which is thought to be provided using `link` elements with a `rel="SubjectRepresentation"`. As the topicmapslab.de portal is a subject-centric portal, all these links match in the example.

Example 6. Excerpt of the ATOM-based topicmapslab.de people update feed.

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Subject Identifier for People at Topic Maps Lab</title>
  <id>http://www.topicmapslab.de/people?locale=en</id>
  <updated>2010-01-22T11:45:14Z</updated>
  <author>
    <name>Topic Maps Lab</name>
    <uri>http://www.topicmapslab.de/</uri>
  </author>
  <entry>
    <title>Benjamin Bock</title>
    <id>http://www.topicmapslab.de/people/Benjamin_Bock</id>
    <updated>2010-01-18T18:12:23Z</updated>
    <summary>Is involved in &quot;Ruby Topic Maps&quot;
and &quot;Topic Maps Lab Community Portal&quot;</summary>
    <link href="http://www.topicmapslab.de/people/Benjamin_Bock"
      rel="SubjectIdentifier"/>
    <link href="http://www.topicmapslab.de/people/Benjamin_Bock"
      rel="SubjectRepresentation"/>
    <!-- snip -->
    <link href="http://bock.be/njamin" rel="SubjectEquivalence"/>
    <link href="http://psi.ontopedia.net/Benjamin_Bock"
      rel="SubjectEquivalence"/>
  </entry>
  <entry>
    <title>Sven Krosse</title>
    <id>http://www.topicmapslab.de/people/Sven_Krosse</id>
    <!-- snip -->
  </entry>
  <entry>
    <title>Lutz Maicher</title>
    <id>http://www.topicmapslab.de/people/Lutz_Maicher</id>
    <!-- snip -->
  </entry>
</feed>
```

5.3. Using TMQL FLWR to Create a Feed for subj3ct.com

The following code snipped shows a single TMQL FLWR query to create the complete ATOM feed for subj3ct.com in one step. In this example the header is returned in one step. For simplicity the current time is chosen for the `updated` element. Of course, this could also be determined by the modification time of the topic which was modified last. In the first `FOR` part of the query all people are queried. The `WHERE` clause constraints the list to only those person-topics which have been marked as published using the corresponding occurrence. For each person an entry is returned which is then filled using subqueries for all properties requested. Please notice the use of built-in functions for matching regular expressions.

Example 7. A TMQL FLWR query to create a subj3ct.com ATOM feed

```
%prefix tml http://www.topicmapslab.de/
%prefix t http://www.topicmapslab.de/types/
RETURN
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Subject Identifier for People at Topic Maps Lab</title>
  <id>http://www.topicmapslab.de/people?locale=en</id>
  <updated>{ fn:current-dateTime }</updated>
  <author>
    <name>Topic Maps Lab</name>
    <uri>http://www.topicmapslab.de/</uri>
  </author>
{
  FOR $person IN // tml:people
  WHERE $person / t:published
  RETURN
  <entry>
    <title>{ $person / t:firstname } { $ person / t:lastname}</title>
    <id>{ $person >> indicators [0] }</id>
    <updated>{ $person / t:updated_at }</updated>
    <summary>{ $person / t:summary }</summary>
    {
      FOR $si in $person >> indicators
      WHERE fn:regexp( $si >> atomify,
          "http://www.topicmapslab.de/people/.*")
      RETURN
      <link href="{$si}" rel="SubjectIdentifier"/>
      <link href="{$si}" rel="SubjectRepresentation"/>
    }
    {
      FOR $si in $person >> indicators
      WHERE not fn:regexp( $si >> atomify,
          "http://www.topicmapslab.de/people/.*")
      RETURN
      <link href="{$si}" rel="SubjectEquivalence"/>
    }
  </entry>
}
</feed>
```

Compared to a previous approach (which collected the information needed using RTMAPI method calls and then outputting a string), the time needed for implementation was reduced; the amount of code is smaller while maintaining a good readability.

6. Conclusion and Outlook

The creation of the ATOM feed query is mostly a work of taking an existing feed and replacing all repeating parts with `FOR-WHERE` expressions and selecting the properties also easy to create. The syntax of TMQL FLWR has many parallels to XQuery and thus can easily be learned. A big advantage of TMQL is the alignment to the data model and the therefor concise and fitting syntax. The ability to output XML eases the integration with other applications significantly.

We implemented TMQL in Java using the before mentioned TMAPI interfaces. The implementation is called TMQL4J and we use it for several academic projects. Our implementation is published as part of the Ontopia open source project [18].

Additionally, Ruby Topic Maps will be enhanced by an additional package rtm-tmql allowing ease of use within the Ruby library.

The query language is still a work in progress and may change before its final standardization in ISO, therefor the implementation will also change. The naïve implementation of the processing model showed significant worse performance compared to the highly optimized tolog engine used in Ontopia. Some obvious optimizations already show massive performance increases. An reasonable next step would be not to implement TMQL on top of TMAPI but to directly access a backend optimized for the engine.

The current draft of the upcoming TMQL standard describes only read access. Several parties are working on drafts supporting modifing access. With a stable query language for reading and modifying Topic Maps, a solid foundation for wide adoption will be laid.

Bibliography

[1] ISO/IEC IS 13250:3004: Information Technology - SGML applications - Topic Maps (Second Edition). International Organisation for Standardization, Geneva, Switzerland, 2003

[2] Members of the TopicMaps.Org Authoring Group: XML Topic Maps (XTM) 1.0. http://topicmaps.org/

[3] ISO/IEC JTC 1/SC34 N0266: Topic map foundational model requirements. International Organisation for Standardization, Geneva, Switzerland, 30 October 2001. http://www1.y12.doe.gov/capabilities/sgml/sc34/document/0266.htm

[4] ISO/IEC IS 13250-2:2006: Information Technology — Document Description and Processing Languages: Topic Maps — Data Model. International Organisation for Standardization, Geneva, Switzerland, 18 June 2006. http://www.isotopicmaps.org/sam/sam-model/2006-06-18/

[5] ISO/IEC IS 13250-2:2006: Information Technology — Document Description and Processing Languages: Topic Maps — XML Syntax. International Organisation for Standardization, Geneva, Switzerland, 2006. http://www.isotopicmaps.org/sam/sam-xtm/2006-06-19/

[6] Clark, James – Cowan, John – MURATA, Makoto: RELAX NG Compact Syntax Tutorial. Working Draft, 26 March 2003. OASIS. http://relaxng.org/compact-tutorial-20030326.html

[7] W3C: XML-Infoset, XML Information Set (Second Edition), W3C Recommendation, 4 February 2004. http://www.w3.org/TR/2004/REC-xml-infoset-20040204

[8] ISO/IEC FCD 19756: Information Technology — Document Description and Processing Languages: Topic Maps Constraint Language. International Organisation for Standardization, Geneva, Switzerland, 15 July 2009. http://www.isotopicmaps.org/tmcl/2009-07-15/

[9] Ahmed, Kal; Garshol, Lars M.; Grønmo, Geir O.; Heuer, Lars; Lischke, Stefan; Moore, Graham: TMAPI 1.0. http://tmapi.org/

[10] Garshol, Lars M.: tolog A topic map query language. Ontopia, 24 August 2007. http://www.ontopia.net/topicmaps/materials/tolog.html

[11] Bleier, Arnim; Bock, Benjamin; Schulze, Uta; Maicher, Lutz: JRuby Topic Maps. in: Linked Topic Maps, LIV Series volume XIX, 12 November 2009.

[12] ISO/IEC WD 13250-6: Information Technology — Document Description and Processing Languages: Topic Maps — Compact Syntax. International Organisation for Standardization, Geneva, Switzerland, 15 May 2008, http://isotopicmaps.org/ctm/ctm.html

[13] Cerny, Robert: JSON Topic Maps 1.0. 23 June 2009. http://www.cerny-online.com/jtm/1.0/

[14] W3C: XQuery 1.0: An XML Query Language 23 January 2007. http://www.w3.org/TR/xquery/

[15] ISO/IEC WD 18048: Information Technology – Document Description and Processing Languages – Topic Maps – Query Language. International Organization for Standardization, Geneva, Switzerland, 2007-07-13 http://www.isotopicmaps.org/tmql/

[16] Barta, Robert: Towards a Formal TMQL Semantics. in: Leveraging the Semantics of Topic Maps, Springer 4 September 2007.

[17] Moore, Graham; Ahmed, Khalil: Subj3ct - A Subject Identity Resolution Service. in: Linked Topic Maps, LIV Series volume XIX, 12 November 2009.

[18] Garshol, Lars M.; Grønmo, Geir O.: Ontopia: Tools for building, maintaining, and deploying Topic Maps-based applications. Ontopia 5.0.0, 8 July 2009, http://code.google.com/p/ontopia/