SPARQL queries to RDFS views of Topic Maps

Silvia Stefanova* and Tore Risch

Department of Information Technology, Uppsala University, Box 337, Uppsala 75105, Sweden
E-mail: Silvia.Stefanova@it.uu.se
E-mail: Tore.Risch@it.uu.se
*Corresponding author

Abstract: Both Topic Maps and RDF are popular semantic web standards designed for machine processing of web documents. Since these representations were originally created for different purposes, they have differences both in their concepts and in their data models. The Topic Map data model can be seen as an ontology for building indices over web documents, whereas RDF is a language to define arbitrary properties of web resources. Ontologies can be defined using the RDF vocabulary description language RDF-Schema (RDFS) language. RDF(S) repositories can be searched using the query language SPARQL. To make Topic Maps exposed to RDF-based tools and searchable with SPARQL, our approach is to map Topic Maps to a view expressed in RDFS. The view can be queried using SPARQL. The problem concerning efficient query processing of SPARQL queries to the rather complex RDF view of Topic Maps has been studied in detail. Our approach has been applied on searching indices to e-government services.

Keywords: topic map; RDFS; RDF-schema; views; query processing; SPARQL.


Biographical notes: Silvia Stefanova is a PhD student at the Department of Informational Technology, Uppsala University, Sweden. She earned her MSc Degree in Electronics in University of Rousse, Bulgaria. Her research interests are in the area of viewing, querying, searching scientific data in terms of semantic web representations. The topic of her PhD project is to investigate how semantic web representations can be utilised for long term preservation, documentation, evolution, and efficient search of scientific data. She has been doing research also on viewing and querying Topic Maps in terms of RDFS.

Tore Risch is Professor of Database Technology at Uppsala University (Sweden) where he leads the Uppsala DataBase Laboratory (UDBL) research group. He was previously Professor at Linköping University, staff member at Hewlett-Packard Laboratories (Palo Alto), and Visiting Scholar from HP in the database group at Stanford University. Prior to joining HP, he worked for Syntelligence Inc. (Sunnyvale) designing a product for large scale knowledge bases. He is author of over 100 technical publications and five US patents. He was General Chair of two international scientific conferences and member of the program committees of over 70 international scientific conferences.

1 Introduction

Both semantic web standards Topic Maps (Pepper, 1999; Pepper and Moore, 2001; Pepper, 2002) and RDFS (Brickley and Guha, 2004) have similarities but they have been originally created for different purposes. Topic Maps started in the 1990s from the idea of managing indices to documents and was published as ISO standard in 2000, whereas RDF (Manola and Miller, 2004) came out from work on the Meta Content Framework (Garshol, 2003) and became a W3C Recommendation in 1999 as a general knowledge representation language.

Despite the fact that both standards are often mentioned as two alternatives for the semantic web (Garshol, 2003, 2001; Ogievetsky, 2001; Pepper et al., 2006a), they differ in their data models, i.e., they treat concepts in different ways. In Topic Maps, each entity, called a topic, is identified by a URI that represents either the subject of the topic itself or refers to resources that indicate the subject of the topic (Garshol, 2001). In RDF, the entity is always identified by the subject’s URI (Manola and Miller, 2004). Classification of entities is managed in Topic Maps by making a topic an instance of another topic, so any topic can classify any other topic. Furthermore, occurrences and associations in Topic Maps can also be instances of other topics. By contrast, classifications in the RDF vocabulary description language RDFS (Brickley and Guha, 2004) are always uniformly specified as instances of RDFS classes. Assigning predefined attributes in Topic Maps is made
by built-in semantics, for instance giving names of the topics and defining occurrences or participation in associations. RDFS provides classes and properties as basic meta-attributes and the user can define domain-specific ontologies in terms of these.

The Topic Map data model provides built-in concepts useful to describe indices of documents or websites, i.e., it can be seen as an ontology describing how to navigate into document collections. For example, Topic Maps are suitable for organising entry point to web portal (Pepper and Garshol, 2003; Garshol, 2002b). By contrast, RDF (Manola and Miller, 2004; Klyne and Carroll, 2004) is a basic schema-free language for associating arbitrary properties with any web resource. RDFS (Brickley and Guha, 2004) is a language defined in terms of RDF to provide a schema for a domain as classes and properties. Compared with Topic Maps, RDFS has a small number of basic primitives that provide extensibility to define any kind of general ontology. RDF documents are searchable by the RDF query language SPARQL (Prud’hommeaux and Seaborne, 2008).

In this work, we view Topic Maps in terms of RDFS and allow SPARQL queries to these views. This enables RDF-based tools to process Topic Maps and SPARQL is used to search indices of websites defined by Topic Maps. This requires the Topic Map data model to be mapped to RDFS.

Our use case is the eGov-Bus Project (http://www.egov-bus.org/web/guest/home) for developing and providing e-services to European citizens and industries. In the eGov-Bus scenario, the information about the e-services, involved countries, citizens, authorities, documents, etc., is indexed by a large Topic Map. It serves as a constantly evolving ontology for the provided e-services. The data needed to complete the e-services is saved in external repositories. Required data is made available on the web as RDFS. For viewing the Topic Map data in terms of an RDFS and querying it by SPARQL, we have developed a system, Topic Map Viewer (TM-Viewer). It is based on defining a general conceptual schema of the Topic Map data model. A declarative RDFS view, the TM view, of Topic Maps is defined in terms of the conceptual schema. SPARQL queries can be specified over the RDFS view.

A functional data model (Risch et al., 2004), which is straightforward to map to RDFS, is used for representing the conceptual schema in terms of functions and types. On the basis of the functional conceptual schema, corresponding RDFS classes are defined for each type, and RDF properties for each function. The conceptual schema can represent any Topic Map database, which enables automatic generation of a generic RDFS view that represents any imported Topic Map.

In TM-Viewer, the TM Importer (Qin, 2007) first parses Topic Map data stored in XTM files, the standard XML representation of Topic Map databases (Pepper and Moore, 2001). The TM importer populates a database in terms of the Topic Map conceptual schema.

Once Topic Maps are imported, the user can issue SPARQL queries to the TM view. The SPARQL queries can search both meta-data representing Topic Map concepts and the content of any imported XTM file. However, the declarative definition of the view is complex and therefore efficient query processing becomes an issue. Query processing of SPARQL queries to the RDF view of Topic Maps has been studied in detail. The query-processing time is shown to be substantially reduced by a small number of rewrite rules, both optimisation and execution performance. Furthermore, the technique of multidirectional (invertible) foreign functions (Litwin and Risch, 1992) for bi-directional encoding of URIs leads to a substantial reduction of query execution times for queries to large XTM files.

2 Architecture

The architecture of the TM-Viewer system is depicted in Figure 1. The core of the system is the TM-Viewer database that represents imported Topic Map data. The Topic Map data model is defined by the Topic Map conceptual schema. The TM view is a system generic RDFS view of Topic Maps. It is generated by the RDF view generator based on mapping rules between basic Topic Map concepts and the corresponding RDFS. The TM Importer parses queried XTM files and populates the TM-Viewer database.

The TM view itself is defined in terms of a TM schema and a TM data view. The TM schema view maps the elements of the Topic Map data model to RDFS concepts, whereas the TM data view represents imported Topic Map data objects.

Figure 1 TM-viewer architecture

A SPARQL query is specified in terms of the TM view. It is first parsed into a Datalog dialect by the SPARQL parser (Yu, 2007). The query processor rewrites and optimises the generated Datalog query to produce an execution plan, which is interpreted. However, queries over the TM view definitions are often very complex to process, because the TM view definition is complicated with many
disjunctions. Hence, the intermediate expressions become large. Therefore, as in the system in Petrini (2008), the query processor uses rewriting techniques based on compile time evaluation (Jones, 1996) of query fragments, to reduce the size of the query. This substantially improves the query-processing time and is often required for being able to optimise large queries in reasonable time.

3 Topic Map conceptual schema

The mapping between the Topic Map data model and the RDFS data model is based on the conceptual schema of the Topic Map data model illustrated in Figure 2. The conceptual schema is a modification of the definitions in XML Topic Map DTD (2000–2001), Pepper and Moore (2001) and Mugnaini (2000) to enable 1:1 mappings between Topic Map and RDFS representations. A functional data model (Risch et al., 2003) is used for representing the conceptual schema.

In Figure 2, the rectangles represent types and the ovals attributes. The arrows between the types depict functions representing relationships between types. Both attributes and relationships are represented as functions in the functional data model. A double arrow indicates a bag-valued function.

![Topic Map conceptual schema](image)

To enable 1:1 mappings between Topic Map and RDFS concepts, all function and type names in the conceptual schema are unique. We have modified the names of Topic Map elements in the XTM 1.0 standard (Pepper and Moore, 2001) to remove ambiguous names. This allows the RDF view generator to generate automatically unique URIs of RDFS classes and RDF properties in the process of generating the TM schema view from the conceptual schema.

4 Topic Map view in terms of RDF

The notation (subject, property, value) is used to represent RDF triples (Manola and Miller, 2004). Using ObjectLog (Litwin and Risch, 1992), which is a Datalog dialect having disjunctions and object representation, TM-Viewer defines a view of Topic Map data as RDF triples. The meta-model of RDFS (Brickley and Guha, 2004) is used for defining mapping rules between the Topic Map and RDFS concepts. Both triples inferred from the Topic Map conceptual schema definition itself (TM Schema view) and triples inferred from imported XTM data files (TM data view) constitute the TM view.

The TM view definition, TMTriples, over a Topic Map data source src is defined as a disjunction (union) of the TM schema and TM data views, TMSchemaTriples and TMDataTriples, respectively:

$$\text{TMTriples}(src, s, p, v) :\neg\text{TMDataTriples}(src, s, p, v) \text{ OR } \text{TMSchemaTriples}(s, p, v)$$

The TM schema view TMSchemaTriples maps the meta-information in the conceptual schema in Figure 2 into a general RDFS view for the Topic Map data model. The TM data view, TMDataTriples, represents the data imported from XTM files. The variable src holds the URL of the imported Topic Map data, e.g., http://www.isotopicmaps.org/tmql/uc-literature.xtm. The TMSchemaTriples view does not have a source, since it defines general concepts of the Topic Map data model, whereas TMDataTriples represents a Topic Map database in a particular source.

The following namespaces are used in the TM view definitions:

- rdf: is the namespace for RDF, http://www.w3.org/1999/02/22-rdf-syntax-ns/
- rdfs: is the namespace for RDFS, http://www.w3.org/2000/01/rdf-schema
- swatm: is the namespace used to represent the TM view e.g., http://udbl.swatm.net

4.1 TM schema view definition

The mapping rules used in the definitions here were inspired by Garshol’s RDFS (Garshol, 2002a) for Topic Maps. Each type in the conceptual schema (rectangles in Figure 2) is mapped to an RDFS class, whereas its attributes (the ovals in Figure 2) and functional relationships (the arrows in Figure 2) are mapped to RDF properties. The argument of a function specifies the RDFS domain of a property. The result of a function specifies the RDFS range of the property. In case a function represents a relationship, the range is the RDFS class of the destination of the arrow in Figure 2; in case it represents an attribute,
the range is a literal. The corresponding TM schema triples are defined based on these mapping rules.

The TM schema triples are defined as a union between

i

triples representing RDFS classes, **TMClassTriples**

ii

triples representing RDF properties, **TMPropertyTriples**

**TMSchemaTriples** *(s, p, v)*:

**TMClassTriples** *(s, p, v)* OR

**TMPropertyTriples** *(s, p, v)* (2)

The TM schema class triples are defined as:

**TMClassTriples** *(s, p, v)*:

**TypeMap** *(t, s)* AND

*p* = 'rdf:type' AND

*v* = 'rdfs:Class' (3)

The predicate **TypeMap** maps between type *t* in the conceptual schema and the URI representing the corresponding RDF subject *s*:

**TypeMap** *(t, s)* :-

isTopicMapType *(t)* AND

*s* = concat('swatm:', name *(t)*) (4)

The predicate **isTopicMapType** *(t)* is true if type *t* is an entity type in the conceptual schema in Figure 2. The name of the RDFS class *s* is computed by concatenating the namespace 'swatm' with the name of the type *t*. The function **concat**, which concatenates strings, is an example of a foreign function in TM Viewer, which is a function implemented in an external programming language.

An example of a **TMClassTriple** is:

*(swatm:Topic, rdf:type, rdf:Class)*

Here, *swatm:Topic* is the URI corresponding to the **Topic** type.

The Topic Map property triples are defined as:

**TMPropertyTriples** *(s, p, v)*:

**TFunFunction** *(f)* AND

**FunctionPropertyTriple** *(f, s, p, v)* OR

**FunctionDomainTriple** *(f, s, p, v)* OR

**FunctionRangeTriple** *(f, s, p, v)* (5)

The predicate **TFunFunction** *(f)* is true if the function *f* represents a function in the conceptual schema. **FunctionPropertyTriple** *(f, s, p, v)* states that a function *f* is mapped to an RDF property:

**FunctionPropertyTriple** *(f, s, p, v)*:

**functionMap** *(f, s)* :-

*s* = concat('swatm:', name *(f)*) (7)

An example of a triple *(s, p, v)* defined by **FunctionPropertyTriple** is:

*(swatm:subjectIdentity, rdf:type, rdf:Property)*

Here, *swatm:subjectIdentity* is the URI corresponding to the subjectIdentity attribute of the type **Topic** in the conceptual schema.

Analogously, the relationship represented by the function **baseNameOfTopic** is mapped to the triple:

*(swatm:baseNameOfTopic, rdf:type, rdf:Property)*

**FunctionDomainTriple** defines the domain of an RDF property and **FunctionRangeTriple** its range. **FunctionDomainTriple** has the definition:

**FunctionDomainTriple** *(f, s, p, v)*:-

**functionMap** *(f, s)* AND

*p* = 'rdfs:domain' AND

argtype *(f, t)* AND

**TypeMap** *(t, v)* (8)

Definition (8) states that the argument of a function corresponds to the RDFS domain of the RDF property. The built-in predicate **argtype** *(f, t)* maps the argument type *t* to the function *f*.

An example of triples defined by **FunctionDomainTriple** for the above-mentioned two RDF properties are:

*(swatm:subjectIdentity, rdfs:domain, swatm:Topic)*

*(swatm:baseNameOfTopic, rdfs:domain, swatm:Topic)*

The range of an RDF property in TM Viewer is defined as:

**FunctionRangeTriple** *(f, s, p, v)*:-

**functionMap** *(f, s)* AND

*p* = 'rdfs:range' AND

restype *(f, t)* AND

**TypeMap** *(t, v)* OR

**LiteralTypeMap** *(t, v)* (9)

The result type *t* of a function *f* is defined by the built-in predicate **restype** *(f, t)*. A function *f* can represent either an attribute or a relationship. In case it represents a relationship, the range *v* is the URI mapped to the result type *t* of the function. In case it represents an attribute, the range *v* is a literal defined by **LiteralTypeMap**:

**LiteralTypeMap** *(t, v)* :-

isLiteral *(t)* AND

*v* = 'rdfs:Literal' (10)

The predicate **isLiteral** *(t)* is true if *t* is a literal type.

An example of triples defined by **FunctionRangeTriple** for the two earlier defined RDF properties are:

*(swatm:subjectIdentity, rdfs:range, rdf:Literal)*

*(swatm:baseNameOfTopic, rdfs:range, swatm:BaseName)*

This concludes the TM schema view definition in terms of RDFS. Since it is independent of Topic Map data,
it is materialised by the system once and for all. The materialised view contains 91 triples and is listed in the Appendix. Table 1 summarises the contents of the TM schema view. The RDFS URI mapped to a Topic Map attribute or relationship is called a property URI.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI of a Topic Map entity</td>
<td>rdf:type</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>Property URI</td>
<td>rdf:type</td>
<td>Rdf:Property</td>
</tr>
<tr>
<td>Property URI</td>
<td>rdfs:domain</td>
<td>URI of a Topic Map entity</td>
</tr>
<tr>
<td></td>
<td>rdfs:range</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td></td>
<td>URI of a Topic Map entity</td>
<td></td>
</tr>
</tbody>
</table>

### 4.2 TM data view definition

The TM data view represents URIs of Topic Map objects imported from one or several XTM files along with their RDF properties. The importation of the Topic Map objects and the assigning of their properties is made by the TM Importer. The TM data view, TMDATATriples, is defined as a union of three subviews:

i. The class membership view, TMInstanceOf, classifying imported Topic Map objects,

ii. The attribute view, TMAttrView, defining attribute values of imported Topic Map objects

iii. The relationship view, TMRelationshipView, defining relationships between imported Topic Map objects:

\[
\text{TMDATATriples}(\text{src}, s, p, v) :\neg \\
\text{TMInstanceOf}(\text{src}, s, p, v) \text{ OR } (11) \\
\text{TMAttrView}(\text{src}, s, p, v) \text{ OR } (12) \\
\text{TMRelationshipView}(\text{src}, s, p, v) 
\]

The TM Importer translates each Topic Map entity read from the XTM file into a Topic Map object representing an entity in terms of the Topic Map conceptual schema. Depending on what kinds of Topic Map elements are read, imported Topic Map objects of different types according to the conceptual schema are created. The type of the imported object, furthermore, determines the corresponding RDF class in the TM data view.

Unique URIs are associated with each created object by concatenating the URL of the XTM file src with an identifier number, e.g., http://www.isotopicmaps.org/tmq1/uc-literature.xtml#4. To generate unique URIs, the enumeration is separate for each XTM file. This makes it possible to load and query several Topic Maps, originating in different XTM files. The same URIs are generated if it is possible to load and query several Topic Maps, originating in different XTM files. The specific attribute view SAa, one for each attribute, which can be expressed as:

\[
\text{TMAttrView}(\text{src}, s, p, v) :\neg \text{SA}_a(\text{src}, s, p, v) (14) 
\]

TM Viewer generates a specific attribute view SAa for each attribute a as follows:

\[
\text{SA}_a(\text{src}, s, p, v) :\neg \text{TMURI}(\text{src}, s) \text{ AND } (15) \\
p = \text{swatm:a’} \text{ AND } \text{a}(o, v) 
\]

The specific attribute view SAa defines the RDF triples of the attribute a for an XTM source src. The attribute table a(o, v) is populated by the TM Importer and represents the attribute value v for each imported object o. Attribute tables are indexed on both o and v. For example,
the following is the specific attribute view for the baseNameString attribute in the conceptual schema:

\[
SA_{\text{baseNameString}}(src, s, p, v):= \\
\text{TMURI}(o, src, s) \quad \text{AND} \\
p = {\text{'swatm:baseNameString'}} \quad \text{AND} \\
\text{baseNameString}(o, v)
\]  

(16)

When the TM Importer has created \( o \) and added a row to \( idTMO \), it sets the attribute \( a \) (e.g., baseNameString) to map between \( o \) and \( a \). As before the URI corresponding to \( o \) is defined by \( \text{TMURI}(o, src, s) \).

An example of an attribute view triple is:

\[
(\text{http://www.isotopicmaps.org/tmql/uc-literature.xtm#66, swatm:baseNameString, 'John Smith'}),
\]

Here, \( \text{swatm:baseNameString} \) is the URI corresponding to the baseNameString attribute of the entity type BaseName in the conceptual schema, whereas \( \text{http://www.isotopicmaps.org/tmql/uc-literature.xtm#66} \) is an instance of the RDFS class \( \text{swatm:BaseName} \) corresponding to the schema entity type 'BaseName'.

The relationship view \( \text{TMRelationshipView} \) defines the RDF properties for all 17 relationships in the conceptual schema. It is defined as a disjunction of all specific relationship views, \( SR_r \), one for each relationship \( r \) in the conceptual schema:

\[
\text{TMRelationshipView}(src, s, p, v):= \\
\text{ORSR}_r(src, s, p, v)
\]  

(17)

For each relationship \( r \) in the conceptual schema, TM Viewer generates a corresponding definition of a specific relationship view, \( SR_r \):

\[
SR_r(src, s, p, v):= \\
\text{TMURI}(o1, src, s) \quad \text{AND} \\
p = {\text{'swatm:r'}} \quad \text{AND} \\
r(o1, o2) \quad \text{AND} \\
\text{TMURI}(o2, src, v)
\]  

(18)

\( SR_r \) is defined in terms of a relationship table named \( r(o1, o2) \) that represents the specific relationship between two imported Topic Map objects \( o1 \) and \( o2 \). Relationship tables are indexed on both \( o1 \) and \( o2 \). The table is populated by the TM Importer.

For example, \( SR_{\text{scopeOfOccurrence}} \) has the definition:

\[
SR_{\text{scopeOfOccurrence}}(src, s, p, v):= \\
\text{TMURI}(o1, src, s) \quad \text{AND} \\
p = {\text{'swatm:scopeOfOccurrence'}} \quad \text{AND} \\
\text{scopeOfOccurrence}(o1, o2) \quad \text{AND} \\
\text{TMURI}(o2, src, v)
\]

An example of a triple in an \( SR_{\text{scopeOfOccurrence}} \) is:

\[
\]

where:

- \( \text{swatm:scopeOfOccurrence} \) is the URI of the RDF property corresponding to the relationship \( \text{scopeOfOccurrence} \) between the entity types Occurrence and Topic in Figure 2.
- \( \text{http://www.isotopicmaps.org/tmql/uc-literature.xtm#12} \) is the URI of an imported Topic Map object, representing an instance of the RDFS class corresponding to the entity type Occurrence.
- \( \text{http://www.isotopicmaps.org/tmql/uc-literature.xtm#14} \) is the URI of an imported Topic Map object, representing an instance of the RDFS class corresponding to the entity type Topic.

Table 2 summarises the content of the TM data view.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI of an imported Topic Map object</td>
<td>rdf:type</td>
<td>URI of a Topic Map entity</td>
</tr>
<tr>
<td>Property URI</td>
<td>Literal</td>
<td>URI of an imported Topic Map object</td>
</tr>
</tbody>
</table>

5 Queries to RDF views of Topic Map

The TM view can be queried using SPARQL. The XTM file must be accessed by the TM Importer and RDF view generator (Figure 1) before it can be queried, using the command:

```
importTopicMap(Charstring URL);
```

e.g.,

```
importTopicMap("http://user.swatm.net/software/swatm/mondial.xtm");
```

An example of a SPARQL query to this Topic Map is:

**Query Q1:**

```
SELECT ?tm1 
FROM <tmdata:mondial.xtm> 
WHERE 
{ ?tm1 <swatm:baseNameOfTopic> ?tm1. 
  ?tm1 <swatm:baseNameString> 'Nottinghamshire'. 
  ?tm1 <swatm:OccurrenceOfTopic> ?tm1. }
```

The FROM clause in SPARQL specifies the data to search and has to be the URI of an imported Topic Map. Query Q1 searches a large XTM file tmdata:mondial.xtm of
SPARQL queries to RDFS views of Topic Maps

size 10.5 MB, which contains 69,800 imported Topic Map objects. The namespace tmdata: is defined as http://user.swatm.net/software/swatm. If no FROM clause is specified, the system will search the last imported XTM file.

SPARQL selects data based on search conditions specified as conjunctions of query triple patterns \((s, p, v)\). Each triple pattern specifies a search condition matching RDF triples. Query Q1 has three triple patterns: \((?tm0, swatm:baseNameOfTopic, ?tm1)\), \((?tm1, swatm:baseNameString, 'Nottinghamshire')\) and \((?tm0, swatm:occurrenceOfTopic, ?tm2)\).

The result of query Q1 is:

\((\text{"tmdata:mondial.xtm#2671"})\)

\((\text{"tmdata:mondial.xtm#2670"})\).

**Definition 1:** A triple pattern where the property \(p\) is a URI is called a property match.

For example, in query Q1 all triple patterns are property matches.

**Definition 2:** A Topic Map property query is a SPARQL query where all triple patterns are property matches and all the properties are URIs of Topic Map attributes or relationships.

Query Q1 is a Topic Map property query. This is the most common kind of query since it searches properties of imported data.

Query Q2 is an example of a complex Topic Map property query taken from the Topic Map query language use cases (Garshol and Barta, 2003). It searches the file http://www.isotopicmaps.org/tmql/uc-literature.xtm, which is a small XTM file of size 14 KB containing 113 imported Topic Map objects. It retrieves the ‘sort’ names of all authors. Several Topic Map concepts have to be traversed to answer the query.

**Query Q2:**

SELECT DISTINCT ?ntn1
FROM <http://www.isotopicmaps.org/tmql/uc-literature.xtm>
WHERE
?t3 <swatm:idTopic> 'author'.
?t1 <swatm:baseNameOfTopic> ?bn.
?t4 <swatm:idTopic> 'sort ').}

The result of the query is the following tuples:

("Pepper, Steve")
("Newcomb, Steve")
("Rath, Holger")
("Biezunski, Michel")

This is also the expected result given in Garshol and Barta (2003).

Query Q3 is an example of a query searching the TM Schema view, which we call a TM Schema query. Such queries do not search imported XTM-data, only Topic Map concepts. Query Q3 retrieves all the attributes of the class Topic (with URI swatm:Topic), except the property idTopic (with URI swatm:idTopic):

**Query Q3:**

SELECT ?tp
FROM <tmdata:mondial.xtm>
WHERE
{(?tp <rdfs:domain> <swatm:Topic>.
?tp <rdfs:range> <rdfs:Literal>.
FILTER (?tp != <swatm:idTopic>)).}

The result of query Q3 is the following tuples:

("swatm:subjectIdentity")
("swatm:subjectAddress")

Query Q4 searches the large XTM file tmdata:mondial.xtm. It retrieves all the attributes of the topic named ‘Province’:

**Query Q4:**

SELECT ?x
FROM <tmdata:mondial.xtm>
WHERE
{(?tm0 <swatm:baseNameOfTopic> ?tm1.
?tm1 <swatm:baseNameString> 'Province'.
?x <rdfs:range> <rdfs:Literal>).}

The result of the query is:

("swatm:subjectIdentity")
("swatm:idTopic")

**Definition 3:** A property pattern \((s, p, v)\) where \(p\) is a variable is called an unknown property pattern.

In query Q4, the triple pattern \((?tm0, ?x, ?y)\) is an unknown property pattern. A query, such as query Q4, having some unknown property pattern is called an unknown property query.

The two first triple patterns in Q4 search the TM Data view for the topic named ‘Province’, whereas the last triple pattern searches the TM Schema view to restrict the meta-data search to retrieve only the attributes (not the relationships) of the topic. The unknown property pattern in query Q4 ties together selections from the TM schema and the TM data views.

Query Q5 is a complex unknown property query with nine triple patterns, of them seven selecting from the TM data view, one from the TM schema, and one unknown property pattern. It retrieves the relationships (not the attributes) of the topics that are instances of another topic named ‘Malta’, which is included in an association with the topic named ‘Owned’. As in Q4, the unknown property pattern ties together TM schema and TM data view searches.
Query Q5

```sparql
SELECT ?x
FROM <tmdata:mondial.xtm>
WHERE
(?tm0 <swatm:baseNameOfTopic> ?tml.
?tml <swatm:baseNameString> 'Owned'.
?m <swatm:roleSpec> ?tm0.
?m <swatm:topicOfMember> ?tm2.
?tm3 <swatm:baseNameString> 'Malta'.
?tm4 ?x <rdfs:range> <swatm:BaseName> )
```

The result of the query is
```
"swatm:baseNameOfTopic"
```

6 Query processing

The steps of the query processor are illustrated in Figure 3. First, the SPARQL parser translates the SPARQL query into a Datalog query expressed in terms of the TM view (TMTriples).

![Diagram of query processing](image)

For example, query Q1 is translated into the Datalog query in (19). The first argument of TMTriples, the data source src, is defined by the FROM clause in the SPARQL query, or a default URI (the last imported XTM file) if no FROM clause is specified.

**Q1(bnt1):**

TMTriples(tmdata:mondial.xtm, tm0, swatm:baseNameOfTopic, tml) AND
TMTriples(tmdata:mondial.xtm, ass, tml, swatm:baseNameString, 'Nottinghamshire') AND
TMTriples(tmdata:mondial.xtm, ass, tm0, swatm:occurrenceOfTopic, tm2)

In the first rewrite step, the View expander recursively substitutes in the query references to the view TMTriples and all its subviews. Naïve view expansion and normalisation make the expanded query large. To speed up query processing substantially, several query reduction rules are applied during the different Datalog transformation steps.

**Property reduction** iteratively applies rewrite rules that reduce most property matches to simple conjunctions. The normaliser transforms the reduced query to disjunctive normal form. **Disjunct reduction** removes from the normalised predicate those disjuncts that will not contribute to the query result.

The reduced Datalog query is translated into an optimised algebra representation by the cost-based query optimiser. It will choose the cheapest implementation to minimise the total cost of the final execution plan. An important technique shown to improve execution plan scalability is the use of multidirectional foreign functions (authors’ ref), which enables bi-directional mappings between imported objects and URIs. The query executor finally interprets the optimised algebra expression to produce the result of the query.

Next, it will be shown how the rewrite rules impact the performance of query processing for the queries defined in Section 5.

6.1 Property reduction

The property reduction step iteratively applies rules named **equality reduction** and **no-result** to reduce the view expanded query. These rules have the following definitions:

- **Equality reduction rule:** In a conjunction, if there is a predicate \( a = b \) where \( a \) and \( b \) are different constants, then the entire conjunction is reduced to false.
- **No-result rule:** If there is a predicate \( p_1 \) where
  i. the predicate is declared to be evaluated at compile time
  ii. there is no matching tuple for the given arguments in \( p_1 \), then it is reduced to false.

The system combines these rules with the usual Boolean algebra rules.

**Theorem 1:** The equality reduction and no-result rules reduce a property match where \( p \) is a property URI to a single conjunction.

**Proof:** The property match is parsed into a predicate \( TMTriples(src, s, p, v) \) where the source \( src \) is the URI in the FROM clause and the property parameter \( p \) is a property URI. \( TMTriples \) is defined as a union of \( TMSchemaTriples \) and \( TMDataTriples \) (1). The no-result rule probes the materialised TM Schema view \( TMSchemaTriples \) where there will be no triple matching the property URI \( p \) (see Table 1) and therefore the reference to \( TMSchemaTriples \) will be eliminated from the disjunction. \( TMDataTriples \) (11) is defined as a union of the class membership view \( TMInstanceOf \) (12), the attribute view \( TMAttrView \) (14), and the relationship view \( TMRelationshipView \) (17). The attribute and relationship views are in their turn defined as unions of the specific conjunctive subviews in (15) and (18), respectively. The view definitions in (12), (15) and (18) all contain an equality between the property variable \( p \) and a constant URI. After view expansion, \( p \) in these views is substituted...
with the constant property URI in the TMTriples call. The equality reduction rule eliminates those subviews where the equality predicate is false. Since the equality test is different in all the subview definitions, each call to TMTriples is reduced to the single conjunction where p is restricted to be equal to the property URI of the property match. TMinstanceOf is always eliminated, since no attribute or relationship URI can be rdf:type (12).

Theorem 1 implies that conjunctive Topic Map property queries are always reduced to a single conjunctive expression and thus no DNF normalisation will take place. Without this reduction, DNF normalisation can produce a very large predicate. For example, without property reduction, query Q1 after normalisation will contain 741 primitive predicates. With property reduction, this is reduced to 23.

Similar reductions can be made for TM Schema queries such as Q3 that restrict attributes or relationships to retrieve from the TM Schema view:

**Theorem 2**: A property match where the property p is one of rdf:domain or rdf:range is reduced to a conjunction.

**Proof**: The properties rdf:domain and rdf:range cannot be in the TM data view (see Table 2). Therefore, the equality reduction rule reduces to false all equality predicates inside TMDataTriples and thus TMDataTriples is removed in the reduced query. What remains in each reduced triple pattern is a single call to the materialised TMSchemaTriples.

Theorem 2 implies that conjunctive TM Schema queries selecting RDF properties representing Topic Map attributes or relationships based on domains and ranges are reduced to conjunctions. Query Q3 is an example of such a query.

**Theorem 3**: A property match where \( p = \text{rdf:type} \) is reduced to a disjunction with two disjuncts.

**Proof**: Property matches to the property URI rdf:type, i.e., \((s, \langle \text{rdf:type} \rangle, v)\), search both the TM Schema view and the TM data view (see Tables 1 and 2). Such triple patterns will be reduced by the equality reduction rule to a disjunction between TMinstanceOf and the materialised TMSchemaTriples, where TMinstanceOf is reduced to a conjunction (12). The attribute and relationship views are all removed, since rdf:type cannot be a URI of an attribute or a relationship. Therefore, only the reference to the reduced TMinstanceOf and TMSchemaTriples remain in the disjunction.

Theorem 3 implies that property matches where \( p = \text{rdf:type} \) will not be reduced to conjunctions, but the rewrite rules still reduce them very substantially.

**Theorem 4**: Property matches to other URIs than rdf:type will be reduced to either a single conjunction, a single predicate, or false.

**Proof**: If the property \( p \) in a property match is different than rdf:type, it can be one of the following URIs:

i. a Topic Map attribute
ii. a Topic Map relationship
iii. rdfs:range
iv. rdfs:domain
v. some other URI.

In cases (i) and (ii), the property match is reduced to a conjunction (Theorem 1) and so also in cases (iii) and (iv) (Theorem 2). In case (v), the URI is not in the TM Viewer database and there will be not matching property URI, so the equality reduction rule reduces it to false.

The above-mentioned reasoning shows that the property reduction rules substantially reduce the sizes of queries where all triple patterns are property matches. Normally, such triple patterns are reduced to a single conjunction and only in the case when the property identifier is rdf:type they will be reduced to a binary disjunction.

### 6.2 Disjunct Reduction

Unknown property patterns cannot be reduced before normalisation. Assume conjunctive queries such as Q4 and Q5, where there is one unknown property pattern and some property matches. Then, each of the property matches will be reduced to either

i. a conjunction (Theorem 4)
ii. false (Theorem 4)
iii. to a binary disjunction if \( p = \text{rdf:type} \) (Theorem 3).

In case (i), the query has the following shape after view expansion and before normalisation:

\[
C(s, r, v) \text{ AND } U(s, p, v),
\]

\( C \) is a conjunction produced by the property matches and \( U \) is the view expanded unknown property pattern, i.e., the view expansion of TMDataTriples\((s, p, v)\). After view expansion of the disjunctive views TMDataTriples, TMAtrView and TMRelationshipView (see (11), (14), (17)), \( U \) will consist of the following conjunctive subviews:

\[
U(s, p, v):=
\begin{align*}
& \text{ (OR } \text{ SA}(s, p, v) \text{) OR (OR } \text{ SR}(s, p, v) \text{) OR } \\
& \text{ TMinstanceOf}(s, p, v) \text{ OR } \text{TMSchemaTriples}(s, p, v)
\end{align*}
\]

This is a disjunction of 30 conjunctions, since there are 11 attributes \( a \) and 17 relationships \( r \) in the conceptual schema. After the normalisation \( C \) is conjuncted with each of the 30 conjunctive views \( W_i \) in \( U \). Since \( C \) is also a conjunction, the normalised predicate \( C \text{ AND } U \) becomes a disjunction of 30 conjunctions \( C_i \), where \( C_i = W_i \text{ AND } C \). We also know that all views \( W_i \) restrict the property \( p \), i.e., they contain a predicate \( p = \text{URI}_i \) (see (12), (15), (18), Table 1). This equality on \( p \) is substituted into the other predicates in \( C \) with the equality substitution rule:
Equality substitution rule: If an equality \( a = K \) or \( K = a \), where \( a \) is a variable and \( K \) a constant, appears in a conjunctive expression, then the equality can be removed and \( a \) replaced with \( K \) in the entire conjunction.

The equality substitution rule combined with the equality reduction rule will eliminate most conjunctions in the normalised predicate of unknown property queries. The degree of the reduction depends on the query. To illustrate the impact of these rules, we go through the reductions for query Q4. Furthermore, in the evaluation section we measure the impact of these reductions on query optimisation and execution times for our example queries, showing significant performance improvements.

For example, query Q4 has one unknown property pattern (\(?tm0, x, y\)), which is translated to:

\[
\text{TMTriples(tmdata:mondial.xtm, tm0, x, y)} (21)
\]

The schema restricting pattern in Q4, (\(?x <\text{rdfs:range}> <\text{rdfs:Literal}>\)), is translated to:

\[
\text{TMTriples(tmdata:mondial.xtm, x, rdfs:range, rdfs:Literal)} (22)
\]

(22) is further reduced to \(\text{TMSchemaTriples}(x, \text{rdfs:range, rdfs:Literal)}\) (Theorem 2). After the normalisation, (21) and (22) are conjuncted with the reduced Datalog translations of the two triple matches of Q4 (the first two triple patterns in Q4)

\[
\begin{align*}
\text{?tm0 } <\text{swatm:baseNameOfTopic} \text{ ?tm1.} \\
\text{?tm1 } <\text{swatm:baseNameString}> \text{ Province'}
\end{align*}
\]

These two triple matches are reduced to two conjunctions: the view \(S_A\text{baseNameOfTopic} \text{ representing the relationship baseNameOfTopic and } S_R\text{baseNameString} \text{ represents the attribute baseNameString (see proof of Theorem 1). Thus, } C_i \text{ becomes the query fragment:}

\[
\begin{align*}
S_A\text{baseNameOfTopic}(tmdata:mondial.xtm, tm0, \text{swatm:baseNameOfTopic, tm1 AND S_R\text{baseNameString}}(tmdata:mondial.xtm, tm1, \text{swatm:baseNameString, "Province"}) \text{AND (23) TMSchemaTriples}(x, \text{rdfs:range, rdfs:Literal})
\end{align*}
\]

Now, consider one of the 30 views \(W_i\) in the normalised predicate representing the relationship \(\text{occurrenceOfTopic, SA}\text{occurrenceOfTopic}\). After substituting in the definition of \(SA\text{occurrenceOfTopic}\) in (18) the variables in the unknown property pattern (\(?tm0, x, y\)), we get:

\[
\begin{align*}
\text{TMURI}(o1, \text{tmdata:mondial.xtm, tm0}) \text{ AND x = swatm:occurrenceOfTopic AND occurrenceOfTopic}(o1, o2) \text{ AND (24) TMURI}(o2, \text{tmdata:mondial.xtm, y})
\end{align*}
\]

\(C_i\) is the conjunction of \(W_i\) in (24) and \(C\) in (23). The equality substitution rule will replace \(x\) in (24) with \(\text{swatm:occurrenceOfTopic}\) in the entire conjunction \(C_i\). Then, the no-result rule will probe \(\text{TMSchemaTriples}\) and, since \(\text{occurrenceOfTopic}\) is not an attribute (it does match in (22)), it will be reduced to \(\text{false}\) and the entire \(C_i\) will be removed.

Since there is a property restriction to \(p = \text{URI}\) in each \(W_i\) in the normalised predicate, for each \(C\), the equality substitution rule will substitute \(x\) with \(\text{URI}\) in the \(\text{TMSchemaTriples}\) predicate in C (23). Then, the no-result rule will probe for \(\text{URI}\) in \(\text{TMSchemaTriples}\), which will yield \(\text{false}\) when \(x\) does not match. There will be no match for all \(C\), where the restriction on \(p\) is not an attribute. This will produce a normalised predicate with 11 conjunctions, as there are 11 attributes in the conceptual schema.

The reduction will be even stronger in query Q5, since the triple pattern (\(?x, <\text{rdfs:range}>, <\text{swatm:BaseName}>\)) has only one possible match of \(x\) to \(\text{swatm:baseNameOfTopic}\), since only the function \text{baseNameOfTopic} in the conceptual schema is a relationship to the type \text{BaseName}. Since there is only one binding of \(x\), the normalised predicate will be a conjunction.

In general, the reduction rules will reduce the query significantly, given that there are property matches in the query restricting a single unknown property pattern. In our performance measurements shown here, it is verified that the optimisation and execution time are indeed substantially reduced by these reductions.

Queries not having restrictions on the unknown property pattern will not be reduced by the reduction rules. Furthermore, if there is more than one unknown property pattern, the normalised predicate can become very large. In order for the system to handle cases where the reductions are insignificant and the normalised predicate therefore becomes too large, TM Viewer has a limit on how large the normalised predicate is allowed to be and will keep the unnormalised predicate when this limit is exceeded. The limit is currently set to a maximum of 10,000 predicates in the normalised expression.

6.3 Bi-directional encoding of URIs

Consider the first triple pattern of query Q2, (\(?ass, <\text{swatm:instanceOfAssociation}>, ?t2\)). The query processing will reduce this property match to a single conjunction (Theorem 1). This conjunction is defined by the relationship view \(S_{\text{instanceOfAssociation}}\) in terms of \(\text{TMURI}\) (18). After view expanding \(\text{TMURI}\) in (13), \(S_{\text{instanceOfAssociation}}\) looks like:

\[
\begin{align*}
1. & \ S_{\text{instanceOfAssociation}}(\text{tmql:uc-literature.xtm, ass, swatm:instanceOfAssociation, t2}):- \\
2. & \ \text{idTMO}(o1, \text{tmql:uc-literature.xtm, num1}) \text{ AND} \\
3. & \ \text{concat}(\text{tmql:uc-literature.xtm, num1}, \text{ass}) \text{ AND} \\
4. & \ \text{instanceOfAssociation}(o1, o2) \text{ AND (25) } \\
5. & \ \text{idTMO}(o2, \text{tmql:uc-literature.xtm, num2}) \text{ AND} \\
6. & \ \text{concat}(\text{tmql:uc-literature.xtm, num2}) = t2
\end{align*}
\]

The two pairs of clauses \(\text{idTMO}\) and \(\text{concat}\) in (25) are the expanded \(\text{TMURI}\) view (13), which maps the imported Topic Map objects \(o1\) and \(o2\) to their URIs \(\text{ass}\) and \(\text{t2}\),
respectively. The numbers num1 and num2 are the internal numbers for o1 and o2 assigned by the TM Importer per XTM file and saved in the table idTMO. The foreign function concat concatenates the URI of the XTM file source, i.e., tmql:uc-literature with the internal number of an imported Topic Map object to produce the URI representing the conceptual object, e.g., tmql:uc-literature#34.

Now, the second triple pattern in Q2, the property match (?t2, <swatm:idTopic>, 'is-author-of') is reduced to the specific attribute view SAidTopic (Theorem 1) with the following view expanded and reduced definition (15):

\[
SAidTopic(tmql:uc-literature, t2, swatm:idTopic, 'is-author-of') :- \\
\text{idTMO(o4, tmql:uc-literature, num) AND } \\
t2 = \text{concat(tmql:uc-literature, num) AND } \\
\text{idTopic(o4, 'is-author-of')}.
\]

Query Q2 is reordered by the optimizer for efficient execution so that the indexed attribute table idTMO is first accessed to bind o4 to the imported Topic Map object representing the topic with identity 'is-author-of'. Then, the system table idTMO is accessed to get the numeric identifier num for the imported object o4 from the XTM file tmql:uc-literature. After that, string concatenation binds t2 to the corresponding URI tmql:uc-literature.xtm#34. Thus, the second triple pattern in query Q2 provides an efficient way to bind t2 to its corresponding URI. No other triple pattern in query Q2 will bind t2. Given that we have bound t2 to tmql:uc-literature.xtm#34 in the second triple pattern of query Q2, the problem is now how to compute ass in (25) in an efficient way. In (25), t2 occurs only on line 6 as a result of the foreign function concat. If we can compute num2 given that

\[
\text{concat(tmql:uc-literature.xtm, num2) = tmql:uc-literature.xtm#34}
\]

then ass can be efficiently computed by lines 5, 4, 2 and 3 in (25). This equation is automatically solved in TM Viewer since concat is defined as a multidirectional foreign function (authors’ ref), where both concat and its inverses are defined as foreign functions in an external language:

\[
\text{create function concat(Charstring x, Charstring y) -> Charstring r as multidirectional} \\
\text{('bfb' foreign 'concat')} \\
\text{('bfb' foreign 'suffix')} \\
\text{('fbb' foreign 'prefix')}
\]

Multidirectional foreign functions provide a way to define different implementations depending on what arguments or the result of a foreign function is known, i.e., depending on the binding pattern (Garcia-Molina et al., 2002). A binding pattern is a string indicating with b if an argument or result is known and with f if it is to be computed. For example, regular string concatenation is performed when the arguments x and y are bound (bfb).

In our case, the binding pattern is bbb, i.e., x and r are known and y is computed. The implementation computes y as the suffix of r given that the prefix is x. In our example, x = tmql:uc-literature.xtm, r = tmql:uc-literature.xtm#34 and y = num2 is computed as 34.

Without multidirectional foreign functions, one of the calls to idTMO or instanceOfAssociation in (25) would have to be scanned. This would not scale since the tables grow linearly with the size of the imported XTM file. This is verified by our performance measurements shown here.

In general, a multidirectional concat is required for scalable bi-directional encoding of Topic Map URIs.

The effect of using bi-directional encoding of URIs can be seen in Table 3. This effect is very considerable if the Topic Map database is large, i.e., with large XTM files. In such cases, the difference between the execution times with bi- and one-directional encoding is substantial, a factor more than 10^2 for Q5 for example (Table 3).

### 7 Performance measurements

To see the impact on the query-processing techniques in TM Viewer, we measured the performance of executing queries Q1–Q5. We separated the time for query reduction and the time to interpret the final execution plan. We also measured the size of the generated execution plans in terms of number of operators. The measurements were made on a Lenovo T61, Dual core, 2 GHz CPU, 3 GB main memory and Windows XP Professional OS.

The results from the measurements are presented in Table 3. It uses the following notation:

- **QR**: The measurements are made with both property and disjunct reduction enabled.
- **MD**: Multidirectional implementation of concat is enabled.

The equality reduction rule is always enabled, since otherwise only very simple queries can be processed. The **optimisation time** shows the time (in sec) spent in query processing and cost-based optimisation. To enable processing very large queries, a greedy cost-based optimisation method is used (Krishnamurthy et al., 1986). The **execution time** (in sec) shows the time spent in interpreting the final execution plan. The **plan size** shows the number of operators in the final execution time. The column Query Reduction Factor (QRF) measures the importance of query reduction alone by dividing the measures in column Only MD with QR + MD. The column Multidirectional Reduction Factor (MDRF) measures the impact of multidirectional concat alone by dividing Only QR with QR + MD.
### Table 3: Impact of the query reduction rules and the multidirectional concat

<table>
<thead>
<tr>
<th>Queries</th>
<th>Measures</th>
<th>QR + MD</th>
<th>Only MD</th>
<th>QRF</th>
<th>Only QR</th>
<th>MDF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Q1</strong></td>
<td>Optimisation time</td>
<td>0.15</td>
<td>0.26</td>
<td>1.7</td>
<td>0.11</td>
<td>0.73</td>
</tr>
<tr>
<td><strong>TM property query</strong></td>
<td>Execution time</td>
<td>$3 \times 10^{-5}$</td>
<td>$7 \times 10^{-5}$</td>
<td>2.3</td>
<td>0.19</td>
<td>6300</td>
</tr>
<tr>
<td><strong>Q2</strong></td>
<td>Optimisation time</td>
<td>1</td>
<td>180</td>
<td>180</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td><strong>TM property query</strong></td>
<td>Execution time</td>
<td>$4.7 \times 10^{-4}$</td>
<td>2.8</td>
<td>6000</td>
<td>0.03</td>
<td>63</td>
</tr>
<tr>
<td><strong>Q3</strong></td>
<td>Optimisation time</td>
<td>0.007</td>
<td>0.007</td>
<td>1</td>
<td>0.007</td>
<td>1</td>
</tr>
<tr>
<td><strong>TMSchema query</strong></td>
<td>Execution time</td>
<td>$1.5 \times 10^{-5}$</td>
<td>1.5 $\times 10^{-5}$</td>
<td>1</td>
<td>$1.5 \times 10^{-5}$</td>
<td>1</td>
</tr>
<tr>
<td><strong>Q4</strong></td>
<td>Optimisation time</td>
<td>0.6</td>
<td>3.3</td>
<td>5.5</td>
<td>6</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>Unknown property query</strong></td>
<td>Execution time</td>
<td>$2.5 \times 10^{-4}$</td>
<td>0.016</td>
<td>64</td>
<td>$3.1 \times 10^{6}$</td>
<td></td>
</tr>
<tr>
<td><strong>Q5</strong></td>
<td>Optimisation time</td>
<td>1.3</td>
<td>772</td>
<td>590</td>
<td>7</td>
<td>5.4</td>
</tr>
<tr>
<td><strong>Unknown property query</strong></td>
<td>Execution time</td>
<td>0.014</td>
<td>0.0015</td>
<td>$1.1 \times 10^{5}$</td>
<td>145</td>
<td>$10^{5}$</td>
</tr>
<tr>
<td><strong>complex query</strong></td>
<td>Plan size</td>
<td>65</td>
<td>154496</td>
<td>2377</td>
<td>51</td>
<td>78</td>
</tr>
</tbody>
</table>

The Topic Map property query Q1 demonstrates the scalability of TM Viewer when the database grows. It searches a large database (69,800 imported Topic Map objects) while the query itself is small. It shows that the execution time is substantially reduced (over 6300 times) by multidirectional concat, thus showing the importance of multidirectional foreign functions for scalable performance. The query reduction rules provide improvement with a factor 2.3, which can be explained by the smaller size of the reduced query (factor 4.5).

The second Topic Map property query Q2 has a complex definition, while the database is small (113 imported Topic Map objects). It investigates the impact of property reduction for complex queries. As expected, the optimisation time is significantly improved by the property reduction rules (factor 180). The execution time for QR alone is higher than MD alone (factor 93), which can be explained with the much larger execution plan without property reduction (factor 573). Interesting is also that the combined QR + MD is 63 times faster than QR alone. This can also be explained with the much smaller execution plan by QR (factor 0.77).

Query Q3 accesses only Topic Map meta-data, which is limited to a table with 91 rows. Neither QR nor MD has any impact.

Query Q4 is a simple unknown property query over a large database. It illustrates the impact of both property and disjunct reduction rules for unknown property queries. First of all, MD alone improves query execution time with a factor $1.2 \times 10^{6}$, clearly showing that it is important for the execution speed for large databases. QR improves here the speed with a factor 64. The query plan reduction factor is 8.8.

Finally, Q5 is a complex unknown property query over a large database. The reduction in the size of the execution plan by QR is a factor 2377, so the query reduction techniques are more effective here than for Q4. The reason is that the restriction on the unknown property here matches only a single URI whereas in Q4 there are 11 matches. Therefore, the disjunct reduction in Q5 produces a conjunctive predicate, which explains the higher query execution plan reduction factor. The speed improvement by query reduction is here a factor $1.1 \times 10^{5}$.

To conclude, our performance measurements clearly show that TM Viewer’s query reduction rules combined with multidirectional concat significantly improve the query processing time for both Topic Map property queries and unknown property queries. All the queries in the Topic Map query use cases (Garshol and Barta, 2003) can be expressed by Topic Map property queries, for which we proved that our techniques substantially improve query performance. In addition, we have shown that our reduction techniques are also very effective for unknown property queries. Such queries are not covered by the Topic Map query use cases (Garshol and Barta, 2003).

### 8 Related work

The existing approaches for transformation between Topic Map and RDF can be divided into two main groups, called object mappings and semantic mappings (Moore, 2001; Pepper et al., 2006a). Semantic mappings are based on finding equivalences between a Topic Map schema and the corresponding RDFS, whereas object mappings are based on representing the Topic Map data model in terms of the RDFS model (Moore, 2001).

The most extensive proposals for semantic mapping are described in Garshol (2003), Garshol (2002a) and Pepper et al. (2006b). The contribution (Pepper et al., 2006b) is in fact guidelines for interoperability between the
two standards Topic Maps and RDF, based on the semantic approach. It is basically a complete proposal for semantic mapping with some limitations of its use in terms of non-deterministic results and unsupported constructs. The semantic mapping can give good results from a ‘naturalness’ and flexibility point of view (Garshol, 2003; Moore, 2001) but it is not always possible due to lack of semantic equivalences (Ciancarini et al., 2003). It is not of general usefulness either since it requires an application-dependent approach.

On the other hand, the object-mapping approach provides a general mapping from Topic Map to RDF. This has been our motivation to base TM Viewer on an object-mapping framework. Work on object mappings from the Topic Maps data model to RDF was done by Garshol (2002a), Lacher and Decker (2001) and Ogievetsky (2001). The proposal (Garshol, 2002a) is based on an earlier version of the ISO/IEC model of Topic Maps, i.e., TMDM (Garshol and Moore, 2008). The so-called items in TMDM become RDF classes and the properties of the items become RDF properties. Unlike TM-Viewer the proposal of Garshol (2002a) is incomplete because there are no definitions of the RDFS ranges and domains of the properties and no description of how Topic Map data is mapped to RDF. The authors in Lacher and Decker (2001) use the Processing Model for Topic Maps, PMTM4 (Biezunski and Newcomb, 2001), which is a very simple model and is not considered as a complete model for Topic Maps (Pepper et al., 2006a). This disadvantage has been overcome in Ogievetsky (2001) where a Topic Maps model is defined partly in terms of PMTM4 and completed with extra XTM terms. The proposal (Ogievetsky, 2001) is fairly complete but very complicated and the translation from the Topic Map data model to an RDFS is non-reversible (Pepper et al., 2006a). For example, it requires seven statements to represent the information content that would be modelled using one statement in RDF (Pepper et al., 2006a). By contrast, TM-Viewer is based on a canonical and yet simple conceptual schema representation that maps 1 : 1 to both Topic Maps and the corresponding RDFS ontology representation of Topic Maps. The mapping rules from the conceptual schema to RDFS are very straightforward: An RDFS class is defined for each entity type as well as an RDF property for each function along with its range and domain definitions. These rules define the declarative general RDFS-based TM view over any Topic Map data imported to TM Viewer. The TM view can be queried with SPARQL.

In Lacher and Decker (2001) and Ogievetsky (2001), it was shown that a Topic Map transformed to RDF can be queried using F-Logic syntax (Decker et al., 1998) or the RDF query language SquishQL (Miller, 2002). We support querying of the Topic Map view by the standard RDF query language SPARQL (Prud’hommeaux and Seaborne, 2008). We are not aware of any other implementation of general queries over RDF views of Topic Maps. Moreover, we studied in detail the problem concerning the query processing of SPARQL queries to the RDFS views of Topic Maps. We defined a small number of rules important for improving performance of the queries to the RDFS view of Topic Maps. We made theoretical proofs how queries were reduced by the rules and showed by measurements their practical impact on different query types.

9 Summary
A general system for exposing Topic Maps as RDFS was implemented, the TM-Viewer. With TM-Viewer, RDFS views of Topic Maps can be queried using SPARQL. It was shown how to optimise such SPARQL queries for scalable execution. The approach has been used for querying e-government services indexed by a Topic Map. The following results were presented:

- A functional conceptual schema was defined for the Topic Map data model.
- Generic 1 : 1 mappings from the conceptual schema into the RDFS model were defined as an automatically generated declarative RDF view, the TM view. The TM View consists of two parts, the TM schema view and the TM data view. The TM Schema view describes the Topic Map conceptual schema as RDF triples, whereas the TM data view describes data represented by the Topic Map conceptual schema as RDF.
- It was shown how to process SPARQL queries to the TM view. This enables SPARQL queries to any Topic Map XTM file.
- The query processing was based on two kinds of query rewrite rules: property reduction rules and disjunct reduction rules.
- It was proved that the property reduction rules reduce a class of common disjunctive SPARQL queries, Topic Map property queries, to the TM View into conjunctions. Thus, no normalisation is needed and these queries are executed efficiently. This class includes all queries in the standard Topic Map query use cases (Garshol and Barata, 2003).
- Furthermore, the approach also allows to express SPARQL queries that combine Topic Map meta-data with Topic Map contents, unknown property queries. It was shown that processing of unknown property queries was substantially improved by the disjunct reduction rules.
- It was shown that a multidirectional (invertible) concat function enables scalable bi-directional encoding of imported Topic Map URIs. The performance improvement of the multidirectional concat is substantial with large XTM files.
References


Appendix

TM Schema view triples

(\texttt{<swatm:TopicMap>, <rdf:type>, <rdfs:Class>})
(\texttt{<swatm:Topic>, <rdf:type>, <rdfs:Class>})
(\texttt{<swatm:BaseName>, <rdf:type>, <rdfs:Class>})
(\texttt{<swatm:Variant>, <rdf:type>, <rdfs:Class>})
(\texttt{<swatm:Occurrence>, <rdf:type>, <rdfs:Class>})
(\texttt{<swatm:Association>, <rdf:type>, <rdfs:Class>})
(\texttt{<swatm:Member>, <rdf:type>, <rdfs:Class>})
(\texttt{<swatm:associationOfTopicMap>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:idTopicMap>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:mergeMap>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:topicOfTopicMap>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:baseNameOfTopic>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:idTopic>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:instanceOfAssociation>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:instanceOfOccurrence>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:instanceOfTopic>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:occurrenceOfTopic>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:parameters>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:roleSpec>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:scopeOfAssociation>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:scopeOfBaseName>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:scopeOfOccurrence>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:subjectAddress>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:subjectIdentity>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:topicOfMember>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:baseNameString>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:idBaseName>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:variantOfBaseName>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:subvariant>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:variantName>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:data>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:idOccurrence>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:reference>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:idAssociation>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:memberOfAssociation>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:associationOfTopicMap>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:topicMap>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:mergeMap>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:topicOfTopicMap>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:baseNameOfTopic>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:idTopic>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:instanceOfAssociation>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:instanceOfOccurrence>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:instanceOfTopic>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:occurrenceOfTopic>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:parameters>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:roleSpec>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:scopeOfAssociation>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:scopeOfBaseName>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:scopeOfOccurrence>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:subjectAddress>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:subjectIdentity>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:topicOfMember>, <rdf:type>, <rdf:Property>})
(\texttt{<swatm:baseNameString>, <rdf:type>, <rdf:Property>})
(<swatm:idBaseName>, <rdfs:range>, <rdfs:Literal>)
(<swatm:variantOfBaseName>, <rdfs:range>, <swatm:Variant>)
(<swatm:subvariant>, <rdfs:range>, <swatm:Variant>)
(<swatm:variantName>, <rdfs:range>, <rdfs:Literal>)
(<swatm:data>, <rdfs:range>, <rdfs:Literal>)

(<swatm:idOccurrence>, <rdfs:range>, <rdfs:Literal>)
(<swatm:reference>, <rdfs:range>, <rdfs:Literal>)
(<swatm:idAssociation>, <rdfs:range>, <rdfs:Literal>)
(<swatm:memberOfAssociation>, <rdfs:range>, <swatm:Member>)