

# TMCL and OWL

Lars Marius Garshol

Bouvet, Oslo, Norway

larsga@bouvet.no

**Abstract.** This paper compares the Topic Maps schema language TMCL with the corresponding RDF technologies RDFS/OWL, and describes the first method for bidirectional conversion between TMCL and RDFS/OWL, based on an existing RDF-to-TM mapping for instance data. The conversion from TMCL creates OWL Lite ontologies where possible, and OWL DL ontologies where not.

## 1 Introduction

Today, it is possible to convert instance data from RDF to Topic Maps, and vice versa [Garshol03a], and it is even possible to use the same vocabulary in both technologies. However, it has not been possible to take a vocabulary description in RDFS or OWL and convert this into a TMCL schema [TMCL]. Nor has conversion in the opposite direction been possible.

The result is that anyone wishing to use a vocabulary defined in one technology with the other is forced to translate the schema (or ontology description) manually, in order to be able to use tools such as schema-driven editors, validators, reasoners, and so on. Such work is tedious, error-prone, and also requires users to know both technologies quite intimately.

With a reliable conversion method implemented in tools, migration is dramatically simplified for users, who no longer need to learn three schema languages (RDFS, OWL, and TMCL), being able instead to simply use the editor of their choice. The schema conversion problem has so far been unsolved, despite some early work on OWL and Topic Maps interoperability, described in section 6.

This paper presents a bidirectional conversion method, which it claims effectively solves the schema conversion problem. The conversion method is

based on existing RDF-to-TM and TM-to-RDF mappings, in such a way that valid instance data, once converted, will also validate according to the converted schema. This ensures that the instance and schema conversion methods work well together.

### 1.1 The RTM mapping

RTM is a conversion method from RDF to Topic Maps [Garshol03b]. It is based on the observation that resources<sup>1</sup> in RDF correspond to topics in Topic Maps, while statements correspond to names, occurrences, or associations. The RTM mapping vocabulary, which is an RDF vocabulary for describing the mapping from RDF to Topic Maps of a particular RDF vocabulary, is needed because RDF statements do not contain sufficient information to determine which of the three Topic Maps constructs they should be mapped to.

The basic workings of the mapping can be summarized as follows:

- Resources become topics, and their URIs become subject identifiers.
- Statements become names, occurrences, or associations, and which is determined by a mapping attached to the RDF property.
- Association roles are fixed for properties mapped to associations, by specifying one role type for the subject of the statement, and one for the object.

Below is a simple example mapping for three properties from Dublin Core, expressed in n3[Berners-Lee05], just to show the basic approach used:

```
dc:title rtm:maps-to rtm:basename .
dc:date rtm:maps-to rtm:occurrence .
dc:creator rtm:maps-to rtm:association .
dc:creator rtm:subject-role resource .
dc:creator rtm:object-role value .
```

Given this, the following RDF graph (in n3):

```
<#tmcl-owl> dc:title "TMCL and OWL";
           dc:date "2008-09-15";
           dc:creator <#lmg> .
```

would be converted to the following topic map (in CTM):

---

<sup>1</sup> The RDF terminology does not match that of Topic Maps entirely here, as resources and nodes are conflated in RDF, and so an exact match with the subject/topic distinction in Topic Maps is not possible.

```
tmcl-owl - dc:title "TMCL and OWL";  
          dc:date: "2008-09-15" .  
dc:creator(resource : tmcl-owl; value : lmg)
```

## 2 Comparing TMCL with RDFS/OWL

TMCL is the standard used in the Topic Maps family of technologies to describe the proper use of a vocabulary. On the RDF side, there are two corresponding standards: RDFS and OWL. The relationship between these is that OWL is an extension of RDFS, providing additional facilities.

The structure of TMCL, RDFS, and OWL is mostly similar, in that all are vocabularies allowing the host technology to describe itself. That is, TMCL is a Topic Maps vocabulary, just as RDFS and OWL are RDF vocabularies. TMCL goes a bit further, however, and also allows constraints to be defined in a Schematron-like way, using TMQL expressions.

The vocabulary elements of these three languages can be roughly divided into three groups:

- Constraints. These are rules describing the structure of instance data.
- Documentation. This is information meant for human readers, such as names, descriptions, version information, and so on.
- Reasoning rules, which is information whose only use is to allow further information to be inferred from existing instance data.

RDFS and OWL both have documentation vocabulary elements, while TMCL does not, which means that such elements must be converted by the RTM mapping as is, in the hope that Topic Maps software will display it to humans. Similarly, pure reasoning statements can be converted as is, in the hope that some software will make use of it, although this can by no means be guaranteed.

The conversion rules generally attempt to express all RDFS and OWL constraints by means of the TMCL core constraints. This is not possible in all cases, but the remaining elements can be expressed using TMQL constraints. TMQL constraints allow the validation behaviour to be captured, but the ability of software to introspect the constraints and use the information for other purposes (such as schema-driven editing) is lost.

## 2.1 Validation versus reasoning

The general purpose served by these languages is the same: to allow users to describe the proper use of their vocabularies. However, the approach taken by TMCL is very different from that taken in RDFS and OWL. Generally, one could say that while TMCL has validation semantics, RDFS and OWL have inference semantics.

To give an example: it is possible to say, in both TMCL and RDFS, that `dc:creators` must be persons. However, the effect of encountering something which is a `dc:creator`, but not a person is different. In TMCL this is a validation error; it is assumed that the data is wrong (either the topic is lacking a type, or the topic should not appear in this association). In RDFS this is not an error per se; instead, it is treated as a case of missing data, and the reasoner assumes that the resource is a person after all.

OWL heavily emphasizes the use of logical reasoning, to the extent of having a direct mapping into Description Logic, a class of formal (that is, mathematical) logical languages. Reasoning on OWL datasets is thus underpinned by description logic, which guarantees that reasoning will be efficient. Strictly speaking, OWL is divided into three subsets: Lite, DL, and Full, where only the former two can be mapped to description logic. (OWL Full is known to be undecidable.)

It should be emphasized that this is a difference of purpose more than an essential difference. It is possible to use TMCL for reasoning, even if no reasoning semantics are given in the standard. Similarly, it is possible to do some validation with RDFS and OWL through negative statements like making classes disjoint. Alternatively, one could devise one's own validation semantics for RDFS/OWL schemas.

## 2.2 Consequences for the conversion

That TMCL has validation semantics, while RDFS/OWL have inference semantics complicates the creation of conversions between them, since in general creating schemas that are treated the same way in the two technologies is impossible, given that in one they are used for validation and in another for inference. The conversion method presented in this paper ignores this issue, and instead aims to convert structural information to the nearest possible equivalent in the target technology.

To stay with the example above: if an RDFS schema says that `dc:creators` must be `persons` that particular statement is easily reproduced in TMCL. That TMCL

will treat this information differently from how RDFS will treat it is left as an issue for the user to handle.

An alternative approach might have been to define a reasoning semantics for a TMCL extended with the necessary OWL elements, which would have allowed RDFS/OWL ontologies to be converted into structures with the exact same meaning on the Topic Maps side. However, the general requirement of Topic Maps users is for validation much more than for reasoning, and so this approach was not taken.

### 3 RDFS to TMCL conversion

Any attempt at conversion has to begin with RDFS, which describes the basic structure of ontologies. The core of RDFS is essentially three vocabulary elements: `rdfs:Class`, `rdfs:range`, and `rdfs:domain`. `rdfs:Class` is the class of all classes, and thus equivalent to `tmcl:topicitype`. That is, instances of `rdfs:Class` are classes, like `person`, `organization`, `country` etc.

Which properties an instance of a class can have are defined by `rdfs:domain`, which relates an RDF property to the classes which can be the subjects of statements with this property. Similarly, for these properties `rdfs:range` gives the classes which can be the values (or objects) of statements with these properties. If the property values are literals, the class `rdfs:Literal` (or a subclass of it) is used.

Conversion of `rdfs:Class` is straightforward: instances of this class become instances of `tmcl:topicitype`. This leaves the constraints on properties, which must be converted using a special algorithm, as the mappings here are more complex. The algorithm mapping rules are given below, using a simple RDF triple filter syntax in the GIVEN section and CTM notation in the OUTPUT section.

```
GIVEN x rdfs:domain y &
      x rtm:maps-to rtm:basename
OUTPUT
x isa tmcl:nametype .
?c isa tmcl:topicname-constraint .
tmcl:applies-to(tmcl:constraint-role : ?c,
               tmcl:topicitype-role : y)
tmcl:applies-to(tmcl:constraint-role : ?c,
               tmcl:nametype-role : x)
```

This mapping converts properties which map to topic names into a name type in TMCL, and for each `rdfs:domain` statement creates a corresponding constraint

attaching the property to the given class. It is an error for the range to be something other than `rdfs:Literal` or `xsd:string`.

```
GIVEN x rdfs:domain y &
      x rtm:maps-to rtm:occurrence
OUTPUT
x isa tmcl:occurrencetype .
?c isa tmcl:topicoccurrence-constraint .
tmcl:applies-to(tmcl:constraint-role : ?c, tmcl:topicity-
role : y)
tmcl:applies-to(tmcl:constraint-role : ?c,
                tmcl:occurrencetype-role : x)
```

This mapping is the same as the previous one, except that it deals with occurrence types.

```
GIVEN x rdfs:range z &
      x rtm:maps-to rtm:occurrence
OUTPUT
?c isa tmcl:occurrencedatatype-constraint
tmcl:datatype: z .
tmcl:applies-to(?c : tmcl:constraint-role,
                x : tmcl:occurrencetype-role)
```

This mapping turns range statements for occurrence types into datatype constraints. (This assumes that the range really *is* a datatype. If it is not, conversion software should treat this as an error.)

```
GIVEN x rdfs:domain y &
      x rtm:maps-to rtm:association &
      x rtm:subject-role s
OUTPUT
x isa tmcl:associationtype .
s isa tmcl:roletype .

?c isa tmcl:associationrole-constraint
tmcl:card-min: 1
tmcl:card-max: 1 .
tmcl:applies-to(tmcl:constraint-role : ?c,
                tmcl:assoctype-role : x)
tmcl:applies-to(tmcl:constraint-role : ?c,
                tmcl:roletype-role : s)

?c2 isa tmcl:roleplayer-constraint
tmcl:applies-to(tmcl:constraint-role : ?c2,
                tmcl:assoctype-role : x)
tmcl:applies-to(tmcl:constraint-role : ?c2,
                tmcl:assoctype-role : s)
tmcl:applies-to(tmcl:constraint-role : ?c2,
                tmcl:assoctype-role : y)
```

This mapping handles domain statements for association types, marking the association type and subject role type as the correct kind of type, then creating the constraints to connect the role type to the association type and to the topic type that the domain is converted to.

```
GIVEN x rdfs:range z &
      x rtm:maps-to rtm:association &
      x rtm:object-role o
OUTPUT
x isa tmcl:associationtype .
o isa tmcl:roletype .

?c isa tmcl:associationrole-constraint
  tmcl:card-min: 1
  tmcl:card-max: 1 .
tmcl:applies-to(tmcl:constraint-role : ?c,
                tmcl:asstype-role : x)
tmcl:applies-to(tmcl:constraint-role : ?c,
                tmcl:roletype-role : o)

?c2 isa tmcl:roleplayer-constraint
tmcl:applies-to(tmcl:constraint-role : ?c2,
                tmcl:asstype-role : x)
tmcl:applies-to(tmcl:constraint-role : ?c2,
                tmcl:asstype-role : o)
tmcl:applies-to(tmcl:constraint-role : ?c2,
                tmcl:asstype-role : z)
```

This mapping is the same as the previous one, except that it handles the range. Cardinalities are not specified with RDFS, but can be inferred from OWL, as in section 4.2. Symmetric association types will also not be converted correctly here, as this conversion assumes that the subject and object roles are different. Section 4.4 shows how to convert such properties correctly.

The rest of the RDFS vocabulary can be handled by the RTM mapping, as follows:

- `rdfs:subClassOf` is mapped to TMDM superclass-subclass associations.
- `rdfs:subPropertyOf` is mapped to TMDM superclass-subclass associations.
- `rdfs:label` is mapped to a base name with the default type.
- `rdfs:comment` is mapped to an occurrence of type `rdfs:comment`.
- `rdfs:seeAlso` is mapped to an occurrence of type `rdfs:seeAlso`.
- `rdfs:isDefinedBy` is mapped to an association of type `tmcl:definedBySchema`<sup>2</sup>.

<sup>2</sup> This association type has disappeared in the latest TMCL draft, but may return.

## 4 OWL to TMCL conversion

While RDF Schema is a relatively straightforward schema language describing constraints on vocabularies, OWL is a rather more complicated language which does not translate so easily into vocabulary constraints, since the emphasis is much more on supporting inferencing. The abstract syntax is also much more involved, which greatly complicates conversion.

### 4.1 Class and property relationships

A central part of OWL is the ability to define classes in terms of their relationships with other classes. For example, one can state that one class has no instances in common with another, or that a class is the intersection of two other classes, and so on.

`owl:disjointWith` is easily convertible, since TMCL has exactly the same capability:

```
GIVEN x owl:disjointWith y
OUTPUT
  ?c isa tmcl:exclusive-instance .
  tmcl:applies-to(tmcl:constraint-role : ?c,
                  tmcl:topic-type-role : x)
  tmcl:applies-to(tmcl:constraint-role : ?c,
                  tmcl:topic-type-role : y)
```

`owl:complementOf` states that the instances of a class is exactly those topics which are not instances of the other class. This cannot be expressed in TMCL. However, it is merely a stronger form of `owl:disjointWith`, in the sense that the statement `A owl:disjointWith B` divides the universe of instances into three disjoint sets: instances of A, instances of B, and those which are instances of neither.

The statement, `A owl:complementOf B`, however, says that all instances which are not instances of B are instances of A, and vice versa. This strengthens the previous statement by saying that the last set (those which are instances of neither) is empty.

In TMCL, this can be expressed with the same `tmcl:exclusive-instance` constraint, and an additional TMQL constraint which makes it an error for instances not to be an instance of either of the two classes.

`owl:intersectionOf` states that the instances of a class is exactly the intersection of the extensions of a set of other classes. That is, every instance of

the class is an instance of all the other classes. This can be expressed with a TMQL constraint.

`owl:unionOf` states that the instances of a class is exactly the union of the extensions of a set of other classes. That is, every instance of the class is an instance of at least one of the other classes. This can be expressed with a TMQL constraint.

OWL has two properties for stating that two classes or properties have the same set of instances (extension), but that their meanings (intentions) are different. This can be expressed indirectly in Topic Maps by creating a subtyping loop:

```
GIVEN c1 owl:equivalentClass c2
OUTPUT
  tmdm:superclass-subclass(tmdm:superclass : c1,
                           tmdm:subclass : c2)
  tmdm:superclass-subclass(tmdm:superclass : c2,
                           tmdm:subclass : c1)

GIVEN p1 owl:equivalentProperty p2
OUTPUT
  tmdm:superclass-subclass(tmdm:superclass : p1,
                           tmdm:subclass : p2)
  tmdm:superclass-subclass(tmdm:superclass : p2,
                           tmdm:subclass : p1)
```

## 4.2 Cardinalities

Some constraints in OWL ontologies must be specified in a rather unusual way. First, `owl:Restriction` is used to define a nameless class which specifies the restriction on a particular property (indicated with `owl:onProperty`). Second, the class you wish to constrain is related to the nameless class, usually by making the constrained class a subclass of the restriction. This makes the constraint also apply to the desired class. In OWL, (some kinds of) cardinalities and the tightened restrictions on allowed property values must be specified in this way.

Cardinalities can be expressed with `owl:minCardinality` and `owl:maxCardinality`, or with `owl:cardinality`, which is a shorthand for setting both to the same value. However, it's also possible to set cardinality on the property itself without using a restriction, simply by making the property an instance of `owl:FunctionalProperty`, which states that it is inherent to the property itself that it can have no more than one value per resource.

So, finding the minimum cardinality of property `p` on class `c` becomes rather involved, but with

```
GIVEN c rdfs:subClassOf r,
      r rdf:type owl:Restriction,
      r owl:onProperty p,
      r owl:minCardinality m
```

we can assume that the minimum cardinality is  $m$ . If this yields nothing, replace `owl:minCardinality` with `owl:cardinality` and try again. If the result is nothing, repeat for all superclasses and choose the largest value. If nothing is found, assume 0.

For maximum cardinality the procedure is the same, except that if the property is an instance of `owl:FunctionalProperty` the maximum is 1. If nothing is found, assume infinity.

The cardinalities produced by this process can be inserted in the constraints produced by the RDFS-to-TMCL conversion algorithm defined earlier to make it more accurate.

### 4.3 Value restrictions

Another use of restrictions in OWL is to specify constraints on the possible values of a property. The three properties which can be used for this purpose are described below.

`owl:allValuesFrom` states that all values of a given property in a certain class must be of a certain class. This is effectively what `rdfs:range` does, but `owl:allValuesFrom` is used to narrow the range of a property on a specific class.

`owl:someValuesFrom` states that some values of a given property on a certain class must be of a certain class. That is, at least one value must be of that class. This is like the previous property, except that values of other classes are also allowed.

`owl:hasValue` states that instances of a class must have a specific value for a specific property. Other values are allowed, but this one specific instance must be among the values.

None of these three properties can be expressed directly in TMCL; however, they can be expressed with TMQL constraints.

### 4.4 Input to mapping

Some OWL vocabulary elements are, strangely, not applicable in Topic Maps, but can be used to infer the possible settings of the RTM mapping. This is due to

the structural differences between RDF and Topic Maps, which means that some schema information that is relevant in RDF quite simply does not apply in Topic Maps.

The `owl:inverseOf` property says that one property `p2` is the inverse of another property `p1`. From a Topic Maps point of view this means that both `p1` and `p2` must be association types, and further that they must be the *same* association type, but with the role types reversed. (That is, the subject role of `p1` is the object role of `p2`, and vice versa.) This is easily expressed using the RTM mapping.

The `owl:DatatypeProperty` is a class of properties whose values are literals, and implies that the property must map to a name or an occurrence in Topic Maps. Similarly, `owl:ObjectProperty` instances must have resource values, and must map to an occurrence or an association in Topic Maps.

`owl:SymmetricProperty` is a property class which implies that the property is symmetric, meaning that `x p y` implies `y p x`. This means that the property must be an association type in Topic Maps, and that the subject and object roles must be the same. It also means that the conversion of the association type must be a little different from that given for RDFS:

```
GIVEN: x rdfs:domain y &
       x rdfs:range z &
       x rtm:maps-to rtm:association &
       x rtm:subject-role r &
       x rdf:type owl:SymmetricProperty

OUTPUT:
x isa associationtype .
r isa roletype .

?c isa associationrole-constraint
  card-min: 2
  card-max: 2 .
applies-to(constraint-role : ?c, associationtype-role : x)
applies-to(constraint-role : ?c, roletype-role : r)
```

Associations expressed in this way will automatically be treated as symmetric by Topic Maps software, while in RDF this requires reasoning. However, the OWL ontology expresses the symmetric nature of the relationship much more directly than the TMCL schema does. It might be, therefore, that the `owl:SymmetricProperty` class should be converted to Topic Maps, to preserve this information. Or, perhaps, that TMCL should be extended to allow this to be expressed in TMCL.

#### 4.5 Documentary information

The documentary parts of OWL can be handled straightforwardly by the RTM mapping, as these are structurally simple and are not required to match any particular structure on the Topic Maps side.

The vocabulary elements in question are:

- `owl:DeprecatedClass` is the class of deprecated classes and can be converted as is.
- `owl:DeprecatedProperty` is the class of deprecated properties and can be converted as is.
- `owl:versionInfo` becomes an occurrence of the same type, containing the same version information.
- `owl:priorVersion` becomes an association of the same type in Topic Maps, relating the schema to another schema.
- `owl:backwardCompatibleWith` becomes an association of the same type in Topic Maps, relating the schema to another schema.
- `owl:incompatibleWith` becomes an association of the same type in Topic Maps, relating the schema to another schema.
- `owl:Schema` corresponded to `tmcl:Schema` in earlier TMCL drafts, but this has disappeared in the latest draft. It's not yet clear whether it might return.

#### 4.6 Various elements

OWL contains three classes which there is no particular need to convert:

- `owl:Class` is the same as `rdfs:Class`.
- `owl:Thing` is the same as `rdfs:Resource`.
- `owl:Nothing` is the empty class. As such it can be converted directly into Topic Maps. The semantics can be reproduced with a TMQL constraint, though it is difficult to see what purpose this could possibly serve.

`owl:sameAs` is used to say that two instances, properties, or classes are the same, but without merging them. There is no way to do this in Topic Maps, but in Topic Maps it is possible to merge topics without losing identifiers, and so the two topics can simply be merged. This does lose the distinction between the two topics, thus losing information, and so an alternative would be to convert the statement to an association of type `owl:sameAs`. However, as this association type is not supported by, for example, TMQL engines, the former approach is preferable.

`owl:InverseFunctionalProperty` is the class of all properties for which every value must be unique. That is, two different resources cannot have the same value for an inverse functional property. In the current TMCL draft this is only expressible for occurrence types, as follows:

```
GIVEN p rdf:type owl:InverseFunctionalProperty &
      p rtm:maps-to rtm:occurrence &
      p rdfs:domain c
OUTPUT
?c isa tmcl:uniqueoccurrence-constraint .
tmcl:applies-to(tmcl:constraint-role : ?c,
               tmcl:topic-type-role : c)
tmcl:applies-to(tmcl:constraint-role : ?c,
               tmcl:occurrence-type-role : p)
```

For names and associations the constraint must be expressed with TMQL constraints.

`owl:differentFrom` and `owl:AllDifferent` is like `owl:disjointWith`, except it states that individuals are different. This is expressible with TMQL constraints.

`owl:oneOf` is used to state that the instance set of a class is closed. It can also be combined with `owl:DataRange` to define a datatype as a set of values. This is not possible in TMCL, but is easily replicated with a TMQL constraint.

#### 4.7 Non-convertible elements

The following OWL vocabulary elements cannot be converted:

- `owl:TransitiveProperty` is only used for reasoning and so has no equivalent in TMCL. However, it is a simple topic type, and so can be converted as is, to be used by systems which understand it.
- `owl:imports` is an import mechanism, and as such best handled by doing the import before conversion to TMCL.
- `owl:OntologyProperty` is a class of properties which simply must relate an `owl:Ontology` to another. There is no real need to convert this, as the same information will be expressed in RDFS.
- `owl:AnnotationProperty` is a class of properties for which no constraints (“property axioms”) are allowed in OWL DL. It is up to the user to decide which properties are annotation properties, and which are not. This ensures that ontologies remain within what can be expressed in description logic. It carries no special semantics, and so there is no need to convert it to Topic Maps.

## 5 TMCL to RDFS/OWL conversion

The TMCL to RDFS/OWL conversion relies on the TMR vocabulary [Garshol03c] for describing mappings from Topic Maps to RDF, and produces an RDFS/OWL schema for the RDF models that result from the conversion. The conversion is designed to produce ontologies that conform to OWL Lite where possible, and OWL DL where it is not. The conversion will produce OWL Lite ontologies as long as the TMCL schema has no abstract classes and no exclusive instance constraints. It will always produce OWL DL ontologies, as long as all the `tmcl:*type` topic types are disjoint, and as long as topics do not appear both as types and as instances.

No attempt is made to convert TMQL constraints, even though in many cases it may be that the TMQL constraints might have been expressible with OWL, especially if they have been converted from RDFS/OWL in the first place. TMQL constraints are very difficult to introspect, and so this was considered out of scope.

### 5.1 The TMR mapping

The TMR mapping vocabulary essentially provides two pieces of information for the conversion. One is what RDF property type to convert names of the default type to for a given topic type. The other is which of the roles in a binary association to turn into the subject of the resulting RDF statement, and which to turn into the object.

A TMR mapping of the Dublin Core example in 1.1 is shown below in CTM:

```
tmr:name-property(tmr:type : dcc:resource,
                 tmr:property : dc:title)
tmr:preferred-role(tmr:association-type : dc:creator,
                  tmr:role-type : dcc:resource)
```

### 5.2 The conversion

The actual conversion is rather complicated, and so no attempt is made to define it formally in this paper. However, the general gist of it can be given rather quickly.

The TMR conversion is used for elements not explicitly mentioned her, and so the names of all topics which become properties and classes turn into

`rdfs:labels`. Similarly, documentary information, such as comments etc are converted using the usual TMR conversion.

Every `tmcl:topicType` becomes an `owl:Class`. Subclassing associations are turned into `rdfs:subClassOf` statements.

Every name type becomes an `owl:DatatypeProperty`. This includes both those defined as name types, and those which replace the default name type in the TMR mapping. If the name type applies to exactly one topic type, that topic type is set as the `rdfs:domain` of the property. Otherwise, no domain is specified. The `rdfs:range` is always set to `rdfs:Literal`.

Occurrence types are treated the same way, except that there is no mapping of types with TMR. `rdfs:range` is set to the specified data type, if any, and otherwise defaults to literal.

Association types are rather more difficult. If they are n-ary (for n larger than 2) associations become resources, and the schema must be converted accordingly. If they have only a single role type, with an association role constraint stating that cardinality of the role type is exactly 2, then the association type is symmetric. It then becomes an instance of `owl:SymmetricProperty` and domain and range assume the same value.

For binary associations, the conversion is easier. They are specified as `owl:ObjectProperty`. One role type is either specified as the preferred role type, or one is chosen at random. If only one topic type can play this role type, that becomes the domain. Handling of the other role type is the same, except that the topic type becomes the range.

The cardinalities of names, occurrences, and associations on various topic types can be expressed easily with `owl:Restrictions`.

`tmcl:exclusive-instance` constraints can be mapped directly into `owl:disjointWith` statements.

### 5.3 Abstract classes

The abstract topic type constraint states that a particular topic type cannot have instances which are direct instances of that type, but that they must instead be instances of a subclass of it. This is difficult to express in OWL, since OWL does not distinguish between statements which are made directly and those which are inferred.

However, a kind of solution is possible<sup>3</sup>, by stating that the abstract class is `owl:equivalentClass` to the `owl:unionOf` its subclasses. This would make it impossible for a resource to be an instance of a the class without being an instance of its subclasses.

#### 5.4 Non-convertible elements

Some parts of TMCL cannot be converted because no corresponding constraints exist in RDFS and OWL. These are:

- Constraints on subject identifiers and subject locators.
- Regular expression constraints on string values.
- Scope constraints.
- TMQL constraints (as mentioned above).

Unique occurrence constraints are not convertible, because in TMCL these state that occurrence values are unique within a particular topic type. Since this allows the same value to occur with a different topic type, it is not possible to state that the occurrence type is an `owl:InverseFunctionalProperty`. One could make this assertion if the occurrence type is legal for only one topic type, but this is clearly not safe. It may be that TMCL will change on this point.

## 6 Related work

Some work has already been published on Topic Maps/OWL interoperability, but unfortunately these efforts have had a different focus.

[Garshol03a] focuses on interchange of instance data, but had some early notes on how parts of OWL could be translated into Topic Maps. These were just notes, however, and as RDFS was not covered, the work was not sufficient to allow schema conversion in any real sense.

[Cregan05] mapped the TMDM to an RDF vocabulary, representing every item type as an RDF class, and every property as an RDF property. The resulting vocabulary was then described with OWL. This is what [pepper06] describes as an “object mapping” and therefore deficient as a way of mapping instance data. OWL was only used to describe the RDF vocabulary, and so there was really no mapping from Topic Maps to OWL or vice versa.

---

<sup>3</sup> I am indebted to David Norheim for this solution.

[Vatant04] proposed describing Topic Maps ontologies in OWL, using a small extra Topic Maps vocabulary. Strictly speaking, this was not work on Topic Maps/OWL interoperability, but more a suggestion for how to create Topic Maps schemas at a time when TMCL was not available. [Vatant03] is a precursor to this work, and provides an earlier view of the same ideas.

The OMG's Ontology Definition Metamodel specification [ODM] contains a mapping from Topic Maps to OWL, which really is a mapping to RDF and OWL. The mapping essentially infers an ontology from the structure of the topic map. So every topic which has at least one instance, or participates in a superclass-subclass association, becomes an `owl:Class`, for example. As such, it does provide as much interoperability on the schema level as was possible before the introduction of TMCL. However, since it was published before TMCL, TMCL schemas are not covered at all.

## 7 Conclusion and further work

The conversion method presented here has been prototypically implemented in Jython on top of the Ontopia Knowledge Suite and Jena. Several existing ontologies have been converted in both directions, allowing the conversion method to be verified, and results have been satisfactory on RDFS/OWL to TMCL conversions.

In the opposite direction it has been found in many cases that name types, occurrence types, and association types can often apply to more than one topic type. In these cases the domain and range cannot be expressed with RDFS, and so part of the schema's structure is lost. This can be corrected in RDFS tools by manually introducing new common superclasses, but giving these correct URIs and names is not possible automatically.

Generally, what remains is to implement the conversion methods in the OKS and to gain more experience with them to see how well they work in practice. Ideally, the Ontopoly Topic Maps editor should start using TMCL instead of the current Ontopoly-specific schema language, and support for these conversion methods should be added to Ontopoly.

TMCL is also not yet finished, and so the conversion method presented here must be updated as future changes occur. It may also be that some of the concerns expressed in this paper will motivate changes to TMCL.

## References

- [Cregan05] Cregan, A.: *Building Topic Maps in OWL-DL*; Proceedings of Extreme Markup Languages 2005;  
<http://www.idealliance.org/papers/extreme/proceedings/html/2005/Cregan01/EML2005Cregan01.html>
- [Garshol03a] Garshol, L. M.: *Living with Topic Maps and RDF*. Proceedings of XML Europe 2003, 5-8 May 2003, organized by IDEAlliance, London, UK.  
<http://www.ontopia.net/topicmaps/materials/tmrdf.html>
- [Garshol03b] Garshol, L. M.: *The RTM RDF to topic maps mapping – Definition and introduction*. Ontopia technical report, 2003-12-28.  
<http://www.ontopia.net/topicmaps/materials/rdf2tm.html>
- [Garshol03c] Garshol, L. M.: *TMR: A TM-to-RDF mapping*. Ontopia published subject documentation, 2003-12-17. <http://psi.ontopia.net/tm2rdf/>
- [Berners-Lee05] *Primer: Getting into RDF & Semantic Web using N3*; T. Berners-Lee; World Wide Web Consortium; 2005-08-16; <http://www.w3.org/2000/10/swap/Primer>
- [ODM] *Ontology Definition Metamodel*; Object Management Group; OMG Adopted Specification; November 2007; OMG Document Number: ptc/2007-09-09;  
<http://www.omg.org/docs/ptc/07-09-09.pdf>
- [Pepper06] Pepper, S.; *A Survey of RDF/Topic Maps Interoperability Proposals*; W3C Working Group Note 10 February 2006;  
<http://www.w3.org/TR/rdfm-survey/>
- [TMCL] *ISO 19756 – Information technology – Topic Maps – Constraint Language*; ISO Committee Draft; 2008-08-07;  
<http://www.isotopicmaps.org/tmcl/tmcl.html>
- [Vatant04] Vatant, B.; *Ontology-driven topic maps*; Proceedings of XML Europe 2004;  
<http://www.idealliance.org/europe/04/call/xmlpapers/03-03-03.91/.03-03-03.html>
- [Vatant03] Vatant, B.; *OWL and Topic Map Pudding*; Mondeca white paper; 2003-08-01;  
<http://web.archive.org/web/20061113154655/http://www.mondeca.com/owl/owltm.htm>