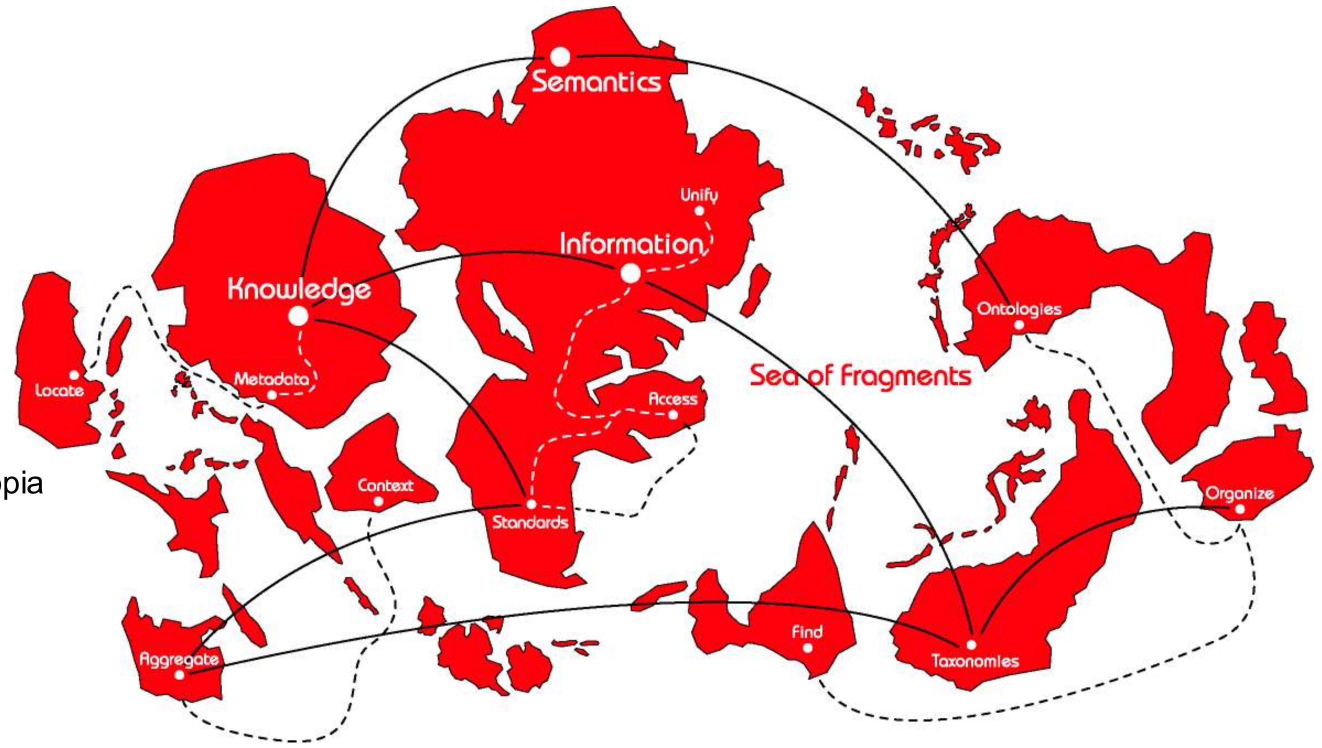
TMQL

Tutorial and discussion



Lars Marius Garshol

Development Manager, Ontopia
<larsga@ontopia.net>

2005-05-21

Agenda

- **Status overview**
- **A little background**
- **A tutorial in today's TMQL**
  - path expressions
  - SELECT expressions
  - FLWR expressions
- **Discussion**
  - national body comments
  - issues suggested by the editors
  - issues raised in the meeting

Current status

History, voting, etc

The state of the TMQL work

- **A requirements document has been published**
- **A set of use cases has been published**
- **Solutions to use cases in four proposal languages collected**
- **Workshop held to evaluate proposals**
  - guidelines for first working draft provided
- **First working draft published 2005-02-18, balloted as CD**
- **Voting results:**
  - Yes: Korea, Netherlands
  - Yes, with comments: Canada, Japan, US, Norway
  - No, with comments: UK (this needs to be discussed)

UK comments

- **UK**
  - "At this stage a wider consultation is required as to how best to integrate TMQL with existing data querying mechanisms. Every feature defined in TMQL should be demonstrated to be implementable before the standard moves to FCD stage. This could be achieved, for example, by having a clearly defined mapping between TMQL and the relational data model, of the type proposed in the TMRQL proposal."
- **This comment led to discussion within the UK national body, which led to a rephrasing of the rationale for the UK vote by Graham Moore, as given on the next slide**

UK comments, take 2

- **Graham Moore**

  "The UK does not approve the current working draft as the basis of producing a CD.

  "As this is a new Query Language and not based on extensive implementation knowledge then the standard must address issues of implementability in respect of the TMDM and existing technologies storage and query technologies.

  "The missing Annexes, D,E,F are critical to the definition and understanding of the language. Without a formal expression of how the TMDM relates to the query constructs it is hard to understand how the language will operate in detail. Further, it is felt that as this is a new language the standard needs to demonstrate its implementablity. Thus the standard should show how the different and combined use of the new language constructs map to an existing query langauge such as Relational Query Language or XML Query Language and the mapping to TMDM. Efforts such as TMRQL could be utilised in this process as it already exposes the TMDM in a Relational Model.

The formal machinery

- **It is the case, as the UK NB states, that the formal machinery for specifying TMQL is missing in the present draft**
- **This is mainly to do with the sheer effort required to produce this**
- **However, the TMRA '05 conference paper proposal by Robert Barta and Lars Heuer contains some of the missing pieces**
  - Tau+
  - TMDM-in-Tau+
- **What's missing now is mostly the operations on Tau+ needed to define TMQL, plus a mapping down to it**
- **The editors hope to have this in the next draft**
- **The question is, does this satisfy the UK national body?**

Robert on formal model

- **Robert (on sc34wg3 list)**
  - "The parts concerning the formal semantics have *not* been written, because partly they depend on [the language] and partly on what happens with TMRM. Doing this sort of thing takes *much* time, so we want to do it *once* only."

Resolution

- **New requirement:**
  - the core TMQL language must be efficiently implementable on SQL and XQuery (ie: there must exist a TM representation in RDBMS/XML that makes this possible)
  - the draft does not need to demonstrate this, it just needs to provide sufficient detail for others to be able to demonstrate it
  - this does not mean that TMQL cannot be more powerful than SQL, only that operations similar to those in SQL must be efficiently implementable in SQL
- **The editors are instructed to produce a new draft containing**
  - a formal semantics for the language
  - an updated syntax according to the decisions in this meeting

Some background

TMQL? What? Why? When?

What is TMQL?

- The purpose of the TMQL work is to create a standard query language for topic maps
- Currently, the topic maps standard only standardizes *data*
- You can move your data between systems, but not your *application*
- Support for TMQL in topic map systems will provide some support for application portability

Why not just an API?

- **APIs can only be used by hard-core developers**
- **APIs are programming-language specific**
  - the failure of the DOM API for XML proves this
- **APIs are hard to use**
  - what would you prefer to use? XSLT or DOM?
- **APIs scale poorly for data access**
  - an API to TMDM must provide fine-grained access to all aspects of the model
  - it must also live with uncontrolled references to stateful data objects
  - an API cannot provide optimizations and alternative access strategies

What should TMQL be able to do?

- **Return data from the topic map based on criteria**
  - single values
  - collections of values
  - values arranged in columns and rows
  - constructed XML values
  - topic map fragments
- **Usages**
  - building presentation logic
  - returning fragments from topic map servers
  - used in creating topic map applications of all sorts
  - updating the topic map (in part 2!)

The purpose of this presentation

- **Firstly, to teach the committee TMQL as it is now**
- **Secondly, to let the committee shape TMQL as it will be**
- **In other words:**
  - let us hear your feedback!
  - if you like something, say it!
  - if you don't like something, say it!
  - if you don't understand something, shout it!
  - and so on :-)

larsbot: I'm presenting my view and my understanding here, not Robert's. I've tried to speak for both of us, but probably don't succeed entirely. Personal asides are written like this

The current TMQL

- **Has three sub-languages**
  - path expressions
  - SELECT expressions
  - FLWR expressions
- **The sub-languages build on each other**
  - path expressions can be used in SELECT expressions
  - path expressions and part of SELECT expressions used in FLWR expressions
- **Result is a language with a small set of core constructs**
  - this means that a lot of the syntactic variation matters much less to an implementation than it may seem
  - however, it has to seem simple to the users, too

TMQL tutorial

Path expressions

Path expressions?

- Path expressions are rather like XPath
- They start from a value, generate new values with "steps", and finally produce a value or collection of values
- Conditions can be used to constrain values generated
- This part is essentially AsTMa? and TMPath reborn

How path expressions work

- **foo**
  - "foo" generates values, maybe one, maybe more
- **foo / bar**
  - the "/ bar" takes a set of values from "foo" and produces a new set of values from it
  - you could think of it as bar(foo), but it's a bit more complex, as you'll see
- **foo / bar / baz**
  - this is just simple chaining (like baz(bar(foo))))
- **foo / bar [quux] / baz**
  - the [quux] is a constraint; it removes values from what bar produces
  - this is where the analogy with functions starts breaking down

larsbot: I'm using the term "constraints" here, but Robert (and XPath) call these "predicates". We also use "predicate" for something else, however. We will return to this at the end.

The simplest possible path expressions

- **opera**
  - returns the topic type "opera" by ID (relies on source locators...)
- **i'http://psi.ontopia.net/music/#opera**
  - returns the topic type "opera" by subject identifier

Finding the name of a topic

- **il-tabarro / bn**
  - finds all base names of the topic il-tabarro: "Il Tabarro", "The Cloak"

Formal model

- **Likely to be based on sequences of tuples (ordered or not)**
- **In foo / bar**
  - "foo" generates one sequence of tuples
  - "/ bar" then operates on this to produce a new sequence
- **The thinking is that this will be the model for the entire language**

Finding the English name

- **il-tabarro / bn [@en]**
  - returns only the base name(s) in the English scope (in this case, "The Cloak")

Finding all operas

- **%_ // opera**
  - %_ must be a variable containing the topic map
  - the "// x" operator filters a collection, producing only instances of x
- **%_ [* opera]**
  - this is actually the same expression with a different syntax
- **%_ // opera / bn**
  - finds all base names of all opera topics
- **%_ // opera / bn [@en]**
  - finds all base names of all opera topics that are in the English scope
  - if an opera has no name in that scope none of its names will make it
  - note that the opera itself is not returned

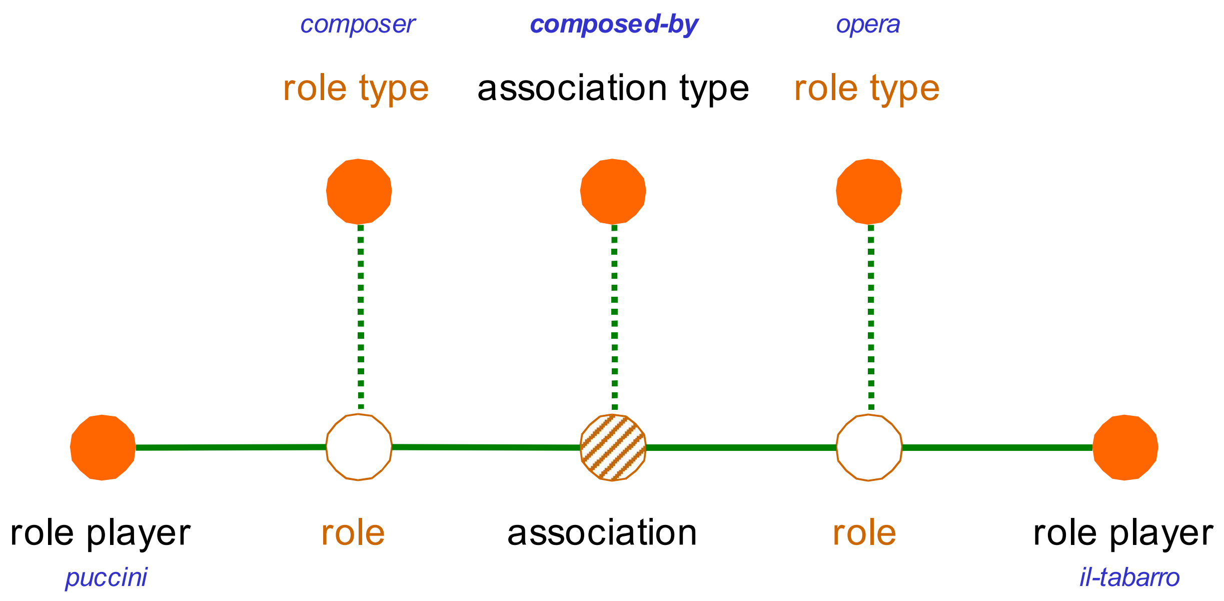
Feedback

- Seems like cannot query reified topic map
- That there is no "axis" for topics/associations seems strange
- Also, that the topics/associations are not produced with a step also seems strange
- F-Logic / Hilog can provide useful input for TMQL (higher-order syntax, and support for reified statements)

Finding the libretto

- **il-tabarro / oc**
  - returns all external occurrences of il-tabarro
  - we want only the librettoes, however
- **il-tabarro / oc [*libretto]**
  - this only gives us the occurrences of type libretto

The structure of associations



Finding the composer (step by step)

- **il-tabarro -> opera**
  - this gives us all associations where il-tabarro plays the role of "opera"
  - the "->" operator is a little special; it goes via the association roles, but produces the associations rather than the roles
  - that's too wide; we want only the associations of type "composed-by"
- **il-tabarro -> opera [* composed-by]**
  - this solves that problem, by adding a predicate filtering by type
  - however, we didn't want the associations, but the topic at the other end
- **il-tabarro -> opera [* composed-by] / composer**
  - the last step finds the roles within the association that are of type "composer", then produces the topics playing those roles
  - **note:** the result of "/ composer" is not the roles of type composer, but the topics that play roles of type composer in the associations produced by the previous step

larsbot: some asymmetry here in that filtering by type is done implicitly by "/ foo" on associations, but not on topics

Feedback

- **The slash in "/ composer" is not symmetric with the "->"**
  - feels like an inconsistency
- **The arrow causes problems for embedding path expressions in XML**
  - the ">" is not allowed in attribute values
- **Would be good if "role1 ... atype ... role2" were collapsible into a single step with a single name**
  - %_ opera / composed-by   # gives composer

More complex combinations

- **%_ // composer [. -> person [* born-in] / place = italy]**
  - finds composers born in italy
  - the [...] after composer is a condition on the composers included in the result
- **%_ // composer [. -> person [* born-in] / place != italy]**
  - composers not born in Italy
- **%_ // composer [! (. -> person [* born-in] / place)]**
  - composers not born anywhere
- **%_ // opera [. / rd [* premiere-date] < 1900]**
  [. / rd [* premiere-date] > 1860]
  [. -> opera [* composed-by] / composer = puccini]
  - operas by Puccini premiered before 1900

More operators

- **tosca / bn ^**
  - this produces the topic reifying the base name
- **tosca / bn @**
  - this produces the topics making up the scope of the base name
  - if the scope is unconstrained, a special topic representing that is returned

larsbot: I don't like the "special topic representing the unconstrained scope"

SELECT expressions

A bit more power

Predicates

- **This sub-language uses predicates for the queries, like tolog**
- **In one sense it's tolog fitted on top of the path expressions**
- **General workings:**
  - predicates generate bindings for *variables*
  - the result of a query is therefore all possible combinations of values for the variables in the the query that make the conditions in the query true
  - you can think of this as a *table of results* with one column per variable
  - how this is achieved is up to the processor
  - however, you can think of it as happening predicate by predicate from start to finish

Finding the composer (in one step)

- **composed-by(il-tabarro : opera, $COMPOSER : composer)**
  - this is a *dynamic association predicate*
  - every association type can be used as a predicate
  - the meaning of this one is: find every topic that plays the role of "composer" in a "composed-by" association where il-tabarro plays the role of "opera"
- **composed-by(il-tabarro, $COMPOSER)**
  - leaves out the role types
  - roles of *any type* can match both arguments
- **Feedback**
  - maybe allow "default direction" for association type to be inferred or declared, so that composed-by($OPERA, $COMPOSER) will not produce wrong combinations where operas occur in $COMPOSER and vice versa
  - UML has something similar in the use of the arrow to specify associations

Finding the operas

- **composed-by($OPERA : opera, puccini : composer)**
  - here we start from the composer (since he is given explicitly) and find all his operas (since the opera is a variable)

Finding all combinations

- **composed-by($OPERA : opera, $COMPOSER : composer)**
  - here we find all the opera/composer combinations (both are variables)

Is it true that...?

- **composed-by(il-tabarro : opera, puccini : composer)**
  - equivalent to asking "is it true that puccini composed il-tabarro?" (no variables)
  - returns a table with no columns and 0/1 rows
  - 0 = false (he didn't)
  - 1 = true (he did)

How dynamic association predicates work

- **a(b : c, d : e) is equivalent to the following in tolog:**
  - a($T1, $T2) :- role-player($R1, $T1), type($R1, c), association-role($A, $R1), association-role($A, $R2), $R1 /= $R2, type($R2, e), role-player($R2, $T2).
  - a(b, e)
- **a(b, d) is equivalent to:**
  - a($T1, $T2) :- role-player($R1, $T1), association-role($A, $R1), association-role($A, $R2), $R1 /= $R2, role-player($R2, $T2).
  - a(b, e)

Finding all operas by the same composer

- **composed-by(il-tabarro : opera, $COMPOSER : composer), composed-by($COMPOSER : composer, $OPERA : opera)**
  - this finds all operas by the composer of "il-tabarro", including "il-tabarro" itself
  - first the composer of "il-tabarro" gets bound into $COMPOSER
  - then the second predicate is evaluated, starting from the composer, finding all operas composed by this composer
- **composed-by(il-tabarro : opera, $COMPOSER : composer), composed-by($COMPOSER : composer, $OPERA : opera), $OPERA != il-tabarro**
  - this query excludes "il-tabarro" by adding an extra condition

Composers who wrote ...

- **SELECT $COMPOSER**
  WHERE
  composed-by($COMPOSER : composer, $OPERA : opera),
  based-on($OPERA : result, $WORK : source),
  written-by($WORK : work, shakespeare : author)
  - the query binds three variables, but the SELECT keeps only $COMPOSER

English names of Puccini operas

- **SELECT $OPERA / bn [@en]**
  WHERE
  composed-by(puccini : composer, $OPERA : opera)
  - here we use a path expression in the SELECT clause in order to get the right base name
- **SELECT $NAME**
  WHERE
  composed-by(puccini : composer, $OPERA : opera),
  $NAME = $OPERA / bn [@en]
  - here the '=' predicate is used to achieve the same thing inside the query itself

larsbot: the = predicate is strictly speaking not necessary in tolog, but has proven itself to be very useful. Not sure if this is the case in TMQL.

Feedback

- **The best thing would be to**
  - let path expressions producing multiple values produce multiple matches/rows
  - and provide a simple operator to say "I want just one value"
  - this is related to aggregate functions (effectively the simplest such function)
- **Concerns about making = operator more than just simple assignment**
- **Could do aggregate functions by**
  - introducing GROUP BY, and
  - allowing nested queries (in SELECT expressions)
- **Scope selection should support both**
  - scope filtering on individual query parts, and
  - the query as a whole

Puccini operas before 1900

- SELECT $opera / bn
  WHERE
  $opera / premiere-date < 1900,
  composed-by ($opera : opera, puccini : composer)
  - here we use path expressions to get hold of the occurrence values
  - this can alternatively be done with a *dynamic occurrence predicate*
- SELECT $opera / bn
  WHERE
  premiere-date($opera, $date), $date < 1900,
  composed-by($opera : opera, puccini : composer)

ISSUE: Dynamic occurrence predicates

- **Robert**
  - "These dynamic occurrence predicates should either be dropped, and/or we allow to define these via rules."
- **LMG**
  - with improved path expressions we don't need these predicates

Finding all operas

- **$OPERA : opera**
  - the ':' is a built-in predicate for topic types
  - in tolog this was "instance-of($OPERA, opera)?"
- **Note that this operator makes use of the supertype-subtype PSI**
- **tosca : $TYPE   # gives all the supertypes as well**

ISSUE: is-a?

- **Robert has suggested the is-a operator as an alternative for ":"**
- **$OPERA is-a opera**
  - equivalent to query on previous slide

Feedback

- **foo:bar : baz:quux**
  - felt to be not very elegant (point in favour of is-a)
- **Predicate for finding directly specified types necessary**
  - actually, this is there
  - using iso for i'http://psi.topicmaps.com/iso13250/ iso:type-instance(tosca, $TYPE)

More complex combinations

- **$C : composer, born-in($C : person, italy : place)**
  - composers born in Italy
- **$C : composer, not(born-in($C : person, italy : place))**
  - composers not born in Italy
- **$C : composer, born-in($C : person, $P : place), $P != italy**
  - composers not born in Italy, take 2
  - difference: person must be born in some $P
  - (ie: we don't allow null bindings)

Alternatives

- **$C : composer, { born-in($C : person, italy : place) |**
  born-in($C : person, sweden : place) }
  - composers born in Italy or Sweden
- **$C : composer, born-in($C : person, $P : place),**
  { $P = italy | $P = sweden }
  - same, formulated differently
- **$C : composer, born-in($C : person, $P : place),**
  in($P, italy, sweden)
  - yet another way to do it
- **$C : composer, born-in($C : person, $P : place),**
  $P in <italy, sweden>
  - here "in" is an operator in the path language instead

larsbot: 'in' predicate suggested by Geir Ove; never made it into official tolog; might be worth considering

Optional clause

- **$C : composer, { homepage($C, $H) }**
  - returns all composers with their home page, if they have one, and null if they don't
- **SELECT $C, $C / oc [* homepage] WHERE $C : composer**
  - doing the same in the select clause

larsbot: we're not sure about these, since path expressions in SELECT do much the same thing

Feedback

- **Using both { ... | ...} and { ... } is not good**
  - the meanings are sufficiently different that the syntax should be different

Alternative syntax

- **SELECT $COMPOSER**
  WHERE composed-by($COMPOSER : composer, $OPERA : opera)
  AND based-on($OPERA : result, $WORK : source)
  AND (written-by($WORK : work, shakespeare : author) OR
  written-by($WORK : work, goethe : author))
  - the difference is that we use AND instead of "," and OR instead of "{ ... | ... }"

larsbot: no decision taken on alternatives here

Feedback

- **Not too enthusiastic about AND and OR**
- **Proposal**
  - keep , as AND
  - use (... | ...) for OR
  - use { ... } for optional
  - not(...) as before

Non-existential queries

- **$CLUSTER : cluster,**
  part-of($MACHINE : part, $CLUSTER : whole),
  is-down($MACHINE)
  - finds all (machine, cluster) combinations where the machine is down
  - however, we want the clusters that are down because all machines in them are down
  - how to do that?

larsbot: all tolog queries are existential in the sense that they ask "are there any $Xs and $Ys so that this is true?" and there is enough with *one* ($X, $Y) for this to work.

This query is non-existential because it is asking "is there any $X where all the $Ys are like this?". The language is geared towards the other kind of query, which makes this type of queries difficult

Non-existential queries (the hard way)

- **$CLUSTER : cluster,**
  not(part-of($MACHINE : part, $CLUSTER : whole),
  not(is-down($MACHINE)))
  - finds every cluster [where there is no machine in the cluster [that is not down]]
  - in other words: clusters where all machines in the cluster are down
  - this may be easier to see if formulated like this: we're asking "is there any $X so that there are no $Ys that are not like this?"
  - this works, but requires substantial mental gymnastics on the part of query authors not deeply versed in logic (ie: 99.9% of all query authors...)

Non-existential queries (the easy way)

- **$CLUSTER : cluster,
  EVERY part-of($MACHINE : part, $CLUSTER : whole)
  SATISFIES is-down($MACHINE)**
  - this finds [clusters [where every machine in the cluster [is down]]]

Feedback

- **TMCL uses syntax that is effectively**
  - EVERY $a IN part-of($a, $c) SATISFIES is-down($a)
- **XQuery has**
  - EVERY $a IN $c -> part-of SATISFIES is-down($a)

Features from tolog

- **SELECT $composer, count($opera) WHERE ...**
  - counts the number of operas per composer
- **SELECT $composer, count($opera) WHERE ... ORDER BY $opera**
  - sorts by the number of operas
- **SELECT ... WHERE ... LIMIT 5 OFFSET 2**
  - same as in SQL

larsbot: we want similar capabilities, but haven't really settled how

Feedback

- Letting ORDER BY $opera order by counted operas is confusing

Inference rules

-
  ```
  inspired-by($COMPOSER, $AUTHOR) :-
     composed-by($COMPOSER : composer, $OPERA : opera),
     based-on($OPERA : result, $WORK : source),
     written-by($WORK : work, $AUTHOR : author).

  inspired-by($COMPOSER, shakespeare)
  ```

ISSUE: Inference rules

- **Robert is skeptical to adding tolog-style inference rules to the language**
  - "Implementations should be allowed to host their own inferencing machinery. Since there are many inferencing methods with *wildly* different expressivity and theoretical performance variations, it would be very pretentious to select one for the lifetime of the standard.
  - "And, as I said before, this can/should be factored out to an ontology description language (TMCL, OWL, EXPRESS, CG, ...)"

Feedback

- **This is not really inferencing, it's just "definition of new predicate"**
  - some people find this acceptable, others prefer "real inferencing"
- **Could have real inferencing that allows topic names, occurrences, and associations to be inferred**
  - this is complex (because of scope, reification and association roles)
  - not clear where it should go (TMCL-Inference, TMQL part 3, nowhere, ...)

FLWR expressions

Even more power

FLWR expressions

- **Directly inspired by XQuery and the way it uses XPath**
- **Uses predicate lists in the same way as SELECT expressions**
- **How it works**
  - (for | let)* where? return
  - in other words, for and let can be repeated in any order
  - where is optional
  - return is not optional

Using just RETURN

- **RETURN opera**
  - gives us the topic "opera"
- **RETURN opera / bn**
  - gives us the base names of the topic "opera"

Using FOR

- **FOR $opera in %m // opera RETURN $opera**
  - gives us all operas
  - path expression produces all operas, each one gets a binding for $opera, and the RETURN is evaluated once for each binding

Using WHERE

- **FOR $topic in %m WHERE $topic : opera RETURN $topic**
  - first produces all topics, then filters away those which are not instances of opera, and finally returns the opera instances
  - ie: same as previous query
- **FOR $opera in %m // opera**
  WHERE composed-by($opera : opera, puccini : composer)
  RETURN $opera
  - returns operas composed by Puccini

Using LET

- **FOR $opera in %m // opera**
  LET $composer := puccini
  WHERE composed-by($opera : opera, $composer : composer)
  RETURN $opera
  - all operas composed by Puccini

larsbot: LET lets us bind temporary values, but the predicate list is good at that anyway. Not clear that LET is actually needed

RETURN is like SELECT (almost)

- **FOR $opera in %m // opera**
  WHERE composed-by($opera : opera, $composer : composer)
  RETURN ($opera, $composer)
  - returns all operas with one composer each
  - WHERE cannot generate new tuples, only remove them
  - it can bind variables, but only to one value

RETURN is like SELECT (2)

- **FOR $opera in %m // opera**
  FOR $composer in %m // person
  WHERE composed-by($opera : opera, $composer : composer)
  RETURN ($opera, $composer)
  - now we get all opera/composer combinations
  - the two FORs generate all possible combinations
  - the WHERE then removes the ones that aren't connected

RETURN can do more than SELECT

```
<rss>
  <title>Recent opera premieres</title>
  <link>http://...</link>

{ for $opera in %m // opera
  let $premiere = $opera / oc [* premiere ]
  order by $premiere desc
  return
    <item>
      <title>{$opera / bn}</title>
      <link>http://...opera.jsp?id={id($opera)}</link>
      <description>Premiere on {$premiere}</description>
    </item>
} </rss>
```

larsbot: the id function is just conjecture at this point. we need to be able to do this somehow, but not sure how yet

RETURN can make topic maps

- **But we're not yet sure how :-)**
  - this functionality will be added, but we've been too busy with other things so far

Reflection/introspection

Discussion

NB comments

Issues from the editors

Issues raised in the meeting

General comment #1

- **US**
  - "Acknowledging changes are needed."
- **The editors are not sure what this means...**

General comment #2

- **Canada**
  - "Suggest that proposed normative Annex F, "Relationship between tau and TMDM" belongs more appropriately as a normative part of ISO/IEC 13250 Part 5 - Reference Model. Annex F could then contain an informative reference to this portion of Part 5."
- **Robert**
  - *"If* Tau+ makes it somewhere into the ISO 13250 series, then TMQL is probably not the right place. The most natural would be TMDM, IMHO, because "TMDM is disclosed to TMRM" and not the other way around."

Resolution

- **TMDM-in-\Tau+ should be a normative annex to part 5**
  - no particular reason to prefer 5 over 2, except that 2 is going to be finalized first

General comment #3

- **Japan**
  - "Nested queries should be supported. Because query-expression and value-expression are disjoint, TMQL cannot formulate nested queries.
  - Nested queries are used in querying to a tm defined by using TMQL. For example, a tm is defined by using FLWR expression e. A query using FLWR expression "For $v IN tm/..." could be "For $v IN e/..." by replaceing tm with its definition e. But this FLWR expression is NOT TMQL because "e/..." is query-expression not value-expression in TMQL."
  - That is: predicate lists should be allowed wherever path expressions are
    - if they really are equivalent it should make no difference
    - closure should let us compose expressions freely

Feedback

- **Conformance levels may be a good idea**
  - TMQL Lite, TMQL Not-so-lite, TMQL Heavy, ...
  - this would lower the bar to implementation
  - path expressions seem a natural candidate for TMQL Lite
  - alternatively, some part of the core could be TMQL Lite, regardless of surface syntax

General comment #4

- **Japan**
  - ""URI" should be changed to "IRI", because an IRI is a sequence of characters from the Universal Character Set (Unicode/ISO 10646) and allows to use non-Latin scripts in it."
- **What this means is effectively that it should be allowed to write non-Latin characters directly in URIs in TMQL queries**

larsbot: I agree with this. Don't know what Robert thinks.

Clause 7: Path expressions

- **Norway**
  - "A new term should be used instead of "predicates" in path expressions in order to avoid collision with "predicates" in SELECT expressions. Possible alternatives are "qualifier" or "filter"."
- **Robert**
  - "I think that path expressions should be called 'predicates', mostly because of the similarity with XPath. Alternative names for the others could be 'association templates'. Not really sure, whether this makes it clearer."

Clause 7: Path expressions

- **Norway**
  - "The path expression language should avoid excessive use of delimiters in order to avoid becoming overly cryptic."
- **At the Norwegian TMUG review meeting the path expression language was generally seen as "having too much syntax"; users seemed to want a simplification of the syntax for the simple cases**

Proposal #1

- **Make type filtering take primacy over axis filtering**
- **That is, not**
  - %_ // person / oc [*birthdate]
- **but**
  - %_ // person / birthdate
- **Also allow**
  - %_ // person / birthdate [* oc]
  - %_ // person / * [* oc]
- **Feedback**
  - filter by occurrence is strange, axis is better
  - %_ // person / occurrence::birthdate
  - %_ // person / birthdate
  - %_ // person / occurrence::*

Proposal #2

- **Add rule that path expressions beginning with / implicitly operate on the topic map**
- **This would reduce**
  - %_ // person / birthdate
- **to**
  - // person / birthdate
- **Still supported:**
  - %opera // composer

Proposal #3

- **Make the type filtering shorthand "/ foo" instead of "// foo"**
- **Result**
  - / person / birthdate
- **Note that**
  - lmg / birthdate
- **is different (means the "birthdate" occurrence of the "lmg" topic)**
- **Feedback**
  - resistance to this, as it seems non-intuitive to see what it means
  - problem is that topic types are entity types and occurrence types are properties, and this syntax conflates the two

Clause 7: Path expressions

- **Norway**
  - "The abbreviations "bn", "oc", and "rd" should be changed so that they look less like topic references."

larsbot: I agree with this. Don't know what Robert thinks.

ISO 13250-6?

- **TMQL is going to need a compact syntax for topic maps**
  - this will be needed in part 2, for the INSERT operation
- **The query syntax for TMQL should be aligned with this compact syntax, however**
- **The compact syntax of TMCL should also be aligned**
- **Therefore, we cannot wait with developing the compact syntax**
  - so, why not make it ISO 13250-6?

Resolution

- **New work item proposal:**
  - ISO 13250-6: Topic Maps – Compact Syntax
  - possible extension: .ctm
- **Volunteer editors wanted**
- **Tasks**
  - collect requirements
  - evaluate existing proposals (LTM, AsTMa=, and extended LTMs)
  - write actual standard
    - EBNF specification of syntax
    - specify mapping to TMDM
    - liaise with editors of TMCL and TMQL to keep syntaxes consistent
- **Consider graphical notation at same time**
  - possibly in part 6, possibly part 7

ISSUE: has-at-least/has-at-most

- Should this be expressible in TMQL?
- TMCL needs it...

ISSUE: Scope operations in predicate lists