TMQL Issues

Robert Barta rho@bond.edu.au

This live document contains all current issues with the current TMQL specification. Please feel free to feedback directly to the author or to the TMQL mailing list.

$Id: issues.xml,v 1.1 2007/02/23 03:00:16 rho Exp $

TMQL Issue: predefined data types

Impact on Language: very high

Background

At the moment, TMQL has a rather arbitrary choice of primitive data types: integer, URIs, decimals, datetimes, most of them cloned from the XML schema data types

```
http://www.w3.org/TR/xmlschema-2/datatypes.html#typesystem
```

The question is which of these should be in the final standard.

Structured Discussion

```
? should all types from XSD be understood?
  + very rich collection, strong big-vendor support
  + constants can be written 'naturally' (3 instead of "3"^xsd:integer)
=> also all XPath 2.0 functions and operators have to be supported
  - very expensive to produce for small developers
```

TMQL Issue: TM paradigmatic functions, predicates, templates, ontologies

Impact on Language: very high

Background

TMQL uses the TM paradigm not hypocritically in that it offers querying of data according to the TM paradigm; TMQL also tries to exploit the paradigm for language features which are traditionally not seen as topic-mappish.

In this sense language constructs such as functions, predicates, but also namespace prefixes are interpreted as subjects, represented in a map by topics of certain predefined TMQL types, such as tmql:function.

A function is a systematic functional dependency over particular sets of values. The age of a person is functionally dependent on (a) the person's birthdate and (b) the current time. Similarily, a predicate is a constraint on a constellation of particular values.

Accordingly, both are expressing additional knowledge about a problem domain. As the modelling of an application domain is usually done via an ontology definition language, one can argue that functions and predicates should NOT be part of TMQL, which is meants as a data access language.

Structured Discussion

```
? Should Functions and Predicates be included into TMQL
  - both are ontological information, should go into an ontology language
    - yes, but TMCL will not offer them, neither does CTM, ...
      + they are EXTREMELY useful to organize the query, uhm knowledge
  - SPARQL does not have function declarations or predicate declarations
```

TMQL Issue: Functions and Predicates as first-class topics

Impact on Language: very high

Background

If it is accepted that TMQL should contain features to define local functions and predicates (which are specialized functions), the question is then how this is integrated conceptually and syntactically into TMQL.

One option is to use a convential syntax, something like

```
function nr_employees (organisation: $o)
return
    fn:length ( $o <- employer )
```

'function' and 'return' would become TMQL keywords connecting together the function name with the function body. The function body - by its nature - will always be a TMQL expression.

The parameter profile of the function - here after the function name inside a () pair - would specify which variables are to be treated as constants, i.e. those where the caller will provide values for at invocation time. Additionally, a type (here organisation) can provide additional information to the TMQL processor which it may (or may not) use.

Technically it would be sufficient to write

```
function nr_employees ($o)
return
    fn:length ( $o <- employer )
```

or even

```
function nr_employees
return
    fn:length ( $o <- employer )
```

if a rule is introduced that all unbound variables (those not explicitly quantified with FOR, SOME or EVERY) become a parameter. That rule is implicit already as it would be redundant to have a parameter list and a list of unbound variables. It would then be an error if the two lists would be different.

A similar structure and convention can be introduced for predicates

```
predicate is_NGO
where
   not $o <- part -> whole == // government
```

That checks whether an organisation is part of anything which itself is an instance of a government (the = is always interpreted existentially in TMQL!).

If it is also accepted that functions and predicates are actually ontological information then also a more TMish syntax can be chosen, especially since the only task is to bind an expression to a name:

```
nr_employees isa tmql:function
tmql:return : {
   fn:length ( $o <- employer)
}
```

Hereby CTM can be used (AsTMa= here only for demonstration). Otherwise, no special syntax is necessary. The {} bracket pair is used throughout TMQL already to wrap query expressions. The only procurement are two predefined TMQL concepts (tmql:function and tmql:return).

There is, though, an additional requirement for CTM to allow occurrence values to be wrapped inside {} pairs. By itself this would imply that a CTM parser has to parse TMQL expressions. To avoid this burden and to keep CTM parsers independent from TMQL expressions, also the following can be allowed in CTM:

```
nr_employees isa tmql:function
tmql:return : {{
     .... whatever, even with use of brackets {}
}}
```

The opening {{ (any number of brackets is possible as prolog) must only have one matching set }}. This is simple and fast to implement.

For predicates this scheme runs similar:

```
is_NGO isa tmql:predicate
tmql:where :  not $o <- part -> whole == // government
```

As the syntax of the boolean expression in the WHERE clause is quite restricted (one cannot generate content with it), there is would be no need for any terminators. But again, this would imply that an (embedded) CTM parser would have to understand the syntax of boolean expressions. To keep that agnostic, the same {} bracketing can be used; and as above, this is can be kept optional if the end-of-line is terminating:

```
is_NGO isa tmql:predicate
tmql:where : {{
    not $o <- part -> whole = // government
}}
```

Structured Discussion

```
? should functions, predicates and templates be first-class topics?
- many people will find this very unconventional, are used to different syntax
  - "un-conventional" is a marketing argument, not a technical argument
    - "it is just a fu**ing syntax"
      + maybe, but the question is: use yet another new one or re-use an existir
  + it is a TMish view on existing concepts
    + directly reflects a TM-view on functions, no further explanation to a user
    + also predefined functions and predicates can be presented that way
    - it can also be done with conventional syntax,
      - yes, but then more explanation is necessary how particular syntactic elem
        have to be mapped onto a TMish view of them
+ reduces syntax for TMQL
  - yes, but it puts some burden onto CTM
    - which is minimal ({{...}}...)
```

TMQL Issue: Refactorization of Standard−Parts

Impact on Language: high

Background

TMQL contains a number of sections which can be argued to actually belong into other TM standard documents. They have only be added to make TMQL complete.

```
- Atoms and Identifiers -> CTM
- Navigation            -> TMDM
- Environmental Clause  -> CTM
- Predicates            -> TMCL/CTM
- Ontologies            -> TMCL/CTM
- Predef Types & Functions -> CTM
```

If these would be factored out, then the TMQL specification would be reduced by roughly 25% .

Structured Discussion

```
? Refactor TMQL standards part into TMDM, CTM and TMCL?
  + cleaner specification landscape
  - TMDM is put in stone already
  + CTM and TMCL are close to finalization, do it now or never
```

TMQL Issue: Error compatibility

Impact on Language: high

Background

At the moment, the TMQL standard does not detail the errors which can occur during the static/dynamic analysis; and it also does not give the errorenous situations a name.

From a language perspective this is not necessary, but OTOH, it impedes compatibility between TMQL processors as one application has to expect potentially different sets of exceptions.

Structured Discussion

```
? Should the TMQL standard name all error situations
  + higher compatibility between TMQL implementation
  - less freedom for implementors
```

TMQL Issue: XML generation, XML subset

Impact on Language: high

Background

TMQL allows to generate XML content.

```
return
   <terrorists>{
      for $person in // person
      where
         soundex ($person / name , "isimi-bin-lidin")
      return
         <evil-evildoer>{$person / name}</evil-evildoer>
   }</terrorists>
```

In that, XML content is _embedded_ into the TMQL expression, this is NOT just a text template which is expanded. The advantage is that processors can included this in the optimization process.

The XML allowed here at the moment does not reflect XML in all its beauty. So, for instance, TMQL does not support CDATA, processing instructions or XML namespaces. The reasoning being that all this can be much more effectively handled with XSLT.

Structured Discussion

```
=> Issue: TMQL native XML support
 ? TMQL should support 100% of XML
   - quite expensive to implement
   - missing pieces can be added with an XSLT processor much more effective
   + completeness is a beautiful thing
```

TMQL Issue: occurrence type implicit subclassing

Impact on Language: high

Background

Comment: NOT A TMQL, but a CTM issue

When someone writes in CTM (modulo syntax details)

```
tbl isa person
! name: Tim Berners Lee
! nickname: TBL
homepage: http://www.bigtim.com
```

then what it implicitely means is that nickname is a subclass of a name and homepage is a subclass of an occurrence.

TMDM never spells that out explicitly, because it is irrelevant there. Only in serialization syntaxes like XTM or CTM this has to be defined.

Structured Discussion

TMQL Issue: Should Functions and Predicates be included into TMQL

Impact on Language: high

Background

A function is a systematic functional dependency over particular sets of values. The age of a person is functionally dependent on (a) the person's birthdate and (b) the current time. Similarily, a predicate is a constraint on a constellation of particular values.

Accordingly, both are expressing additional knowledge about a problem domain. As the modelling of an application domain is usually done via an ontology definition language, one can argue that functions and predicates should NOT be part of TMQL, which is meants as a data access language.

Structured Discussion

```
? Should Functions and Predicates be included into TMQL
  - both are ontological information, should go into an ontology language
    - yes, but TMCL will not offer them, neither does CTM, ...
      + they are EXTREMELY useful to organize the query, uhm knowledge
  - SPARQL does not have function declarations or predicate declarations
```

TMQL Issue: Removal of EVERY clause

Impact on Language: medium

Background

In the Leipzig meeting it was discussed that the EVERY clause is only a syntactic variation of the SOME clause. Every EVERY can be transformed into an equivalent form using SOME:

```
      every $p in // person
      satisfies $p / born

   ==> not
            some $p in // person
            satisfies  not $p / born
```

And hence, EVERY can be removed.

Following this argumentation much of the language can be removed. Also SOME is just a variation of FLWR:

```
   some $p in // person
   satifies $p / born

   ==>

   for $p in // person
   where
       $p / born
   return
       $p
```

And FLWR expressions can be transformed into path expression, etc.

Structured Discussion

```
? Removal of EVERY clause
  + minimal smaller language
  - users have to twist their brains to get the NOT right
```

TMQL Issue: Is everything a 'thing'

Impact on Language: medium

Background

TMDM defines the concept subject, but it say nowhere that all things in the map (topics and associations, for instance) are 'subjects'. And it also does not say that all classes are subclasses of 'subject'.

Now, what should a TMQL processor deliver here:

```
select $thing
where
   $thing isa tm:subject
```

Or this would get you all things in the map

```
// tm:subject
```

and if * is just a shorthand for tm:subject then also

```
  is-employed-at (* : tbl, organisation: $o)
```

works as expected.

Structured Discussion

```
? Should all map items be an instance of tm:subject implicitly
  + tm:subject is a great placeholder
  - TMDM does not say it (or does it?)

? Should all topics be a subclass of tm:subject implicitly
  + tm:subject is a great placeholder
  - TMDM does not say it (or does it?)
```

TMQL Issue: Quantified Quantifiers

Impact on Language: medium

Background

At the moment quantifiers in TMQL are only of a EVERY or SOME nature, there is no way to say "give me all hands with at least 5 fingers".

It is believed, that such a feature is required by TMCL and since TMCL will define its modelling patterns with TMQL, there is some issue here.

So the idea would be to allow

```
select $person
where
   at least 5 $finger in $person <- * -> finger
   satisfies
        $finger / status == "functional"
```

or

```
select $person
where
   exists at most 2 $finger in $person <- * -> finger
```

This functionality can be mapped into the existing TMQL, though, although this may be cumbersome to do it manually. The first would be transformed into

```
 # there is one finger, satisfying this
 # and another finger, satisfying this
 # and another
 #
 # and the fifth one
```

The other would be transformed into the logic equivalent "the person has not at least 2 fingers satisfying".

Structured Discussion

```
? Should TMQL get quantifiers AT MOST integer, AT LEAST integer?
    + can be mapped into canonical TMQL by the processor
    - can be expensive to compute
    + TMCL would profit from that
```

TMQL Issue: Intuitive Semantics of ’false’

Impact on Language: medium

Background

'true' and 'false' have two meanings. One is that they are just values of a data type boolean, in the same way as 3, 3.14 or "sunshine" are values of other types.

So in this sense

```
select $person
where
   exists 3.14
```

will return all persons as there always exists the constant 3.14. But what about

```
select $person
where
   exists false
```

Also 'false' exists always, but most users would think that the above is equivalent with

```
select $person
where
    false
```

which would - in fact - return nothing.

So there is a constant 'false' outside boolean expression and that behaves like a value. And then there is 'false' inside boolean.

Structured Discussion

```
? introduce a constant NULL which renders always an empty sequence.
  ? is this related to xsd:AnyType?
  - NULL is nothing else than ()
  + would cleanly separate the constant 'false' from the 'untrue' NULL
```

TMQL Issue: Allow variables as role type

Impact on Language: medium

Background

At the moment, TMQL allows only constant for roles when navigating to or from an association:

```
....
where
   $person <- employee -> employer == big-bad-corp
```

It would be possible to generalize this and allow PEs and with them variables as roles

```
   $person <- $whatever_role [ ^ is-employed-at ]
```

As path expressions and association predicates are closely connected, this also means that variables for roles are possible there too:

```
is-employed-at ($whatever_role : $person, ...)
```

Structured Discussion

```
? should full path expressions be allowed for role (types)
  - are there use cases for this? Not in the 'official' list
  + should not be too expensive to implement
  + no computational complexity added
  - makes it impossible to use common index structures, may be slow
```

TMQL Issue: Functions process sequences

Impact on Language: medium

Background

Functions in TMQL are - as usual - invoked with parameters, such as in

```
if math:sqrt ($person / shoesize) < 10
then
    "big square foot"
```

Parameter are usually computed and - because of the nature of topic maps - the number of results may vary. What if a person aber does not have a shoesize at all, or 100eds of them?

One way to deal with that is to generalize all functions being able to compute with whole sequences. If a person has shoesizes 2, 3, and 4, then

```
math:sqr ( $person / shoesize )
```

will return a sequence 4, 9, 16.

That is of course cheap to implement. If a function expects more parameters, say 3, what then?

```
terrorism:arrest-them-all ($person / name, $country / name, $pretext / name)
```

The tuple expression may generate tuple sequences with varying length, depending on how many names exist for the things above.

Structured Discussion

```
? functions operate always on tuple sequences
  + straight-forward semantics
    + formal semantics already defines functions so
  + easy to implement and also fast in implementation as calls can be optimized awa
  - slightly unsual semantics for normal engineers, they are not used to sequence p
    - Python, Ruby, Perl and Haskell all have list comprehension, so what?
```

TMQL Issue: Elimination of the concept ’characteristics’

Impact on Language: medium

Background

TMQL is using the term 'characteristics' as subsumption for topic names and topic occurrences. This is in deviation to the earlier interpretation which also included topic roles. TMDM does not use the term 'characteristics' anymore.

The reason for re-introducing it in TMQL was that that way names and occurrences could be treated via one syntactic structure. It would equally be possible to break names and occurrences in two and introduced identical navigation mechanism for this.

Structured Discussion

```
? Should the concept of characteristics be broken up into 'names' and 'occurrences'
  + alignment with TMDM in this respect
  - pretty useless duplication of navigation, it is the same
```

TMQL Issue: Datatype Awareness

Impact on Language: medium

Background

At the moment, TMQL treats atomic data values as exactly this: atomic. This implies that the following is NOT possible:

```
select $person / name
where
   $person / shoesize >> classes == xsd:float
```

[find all persons which have as shoesize a FLOAT value]

Structured Discussion

```
? Make TMQL data type aware?
  + makes it easy for these kind of queries which navigate to the type of a value
  - it contradicts TMDM which stays silent on that matter
```

TMQL Issue: NULL value

Impact on Language: medium

Background

Sometimes one wants to indicate to the application, that there is NO particular value for something:

```
select $p / shoesize || null
 where $p isa person
```

Structured Discussion

```
? should there be a predefined constant 'null'?
  + allows to express 'undefined' value
=> is this identical to xsd:anyType?
```

TMQL Issue: intermediate types

Impact on Language: low

Background

Some approaches for a TM query language have introduced language features to extract all or a particular part of the types of a topic, usually via a distance operand:

```
types ( cat , 3 )  # get the type, its super types, and theirs
```

```
types ( cat, * ) # get all of them
```

TMQL does not support this, mainly because using distances in the type structure in a query might be brittle as the type structure might be refined later.

Structured Discussion

```
? Add language feature (or function) for intermediate types
  + some people will use it
```

```
  - may lead to brittle queries
    - we are not here to educate the world
```

TMQL Issue: Add inverse reification shorthand

Impact on Language: low

Background

TMQL allows to follow the 'reification' axis in both directions. It leads from a topic to the item it reifies or the other way 'round. For the forward direction there is already a shortcut ~~> defined.

When in CTM a reification is declared, then there also a symbol has to be used, one candidate being <~~. The question is now whether TMQL should have this as another shortcut:

```
<~~    ==>    << reifier
```

Structured Discussion

```
? Add <~~    ==>    << reifier as short cut
  + little cost
  + alignment with CTM
  - little used in TMQL itself
```

TMQL Issue: item identifiers for items

Impact on Language: low

Background

At the moment, TMQL does not allow to retrieve item identifiers within a query. It is still possible to write

```
select $person
where
   $person isa person
```

and that will produce a sequence of items; whether these are passed as items or item identifiers back to the calling application, TMQL does not say. Also when items are used in FLWR expressions, sometimes their id is used, depending on the context where they are embedded.

What does not work (yet) is to say

```
select $person >> id
```

mainly because it is unclear whether there is a use case for it.

Inside a query item identifiers could be used for comparing

```
where
   $person >> id == 'whatever'
```

but this can be expressed less baroque as

```
where

   $person == whatever
```

What could be interesting is to learn about the item identifiers of associations and characteristics

```
// is-employed-at >> id
```

or to create a string from the item identifier

```
select "this:is-all-weird#" + $person >> id
```

Structured Discussion

```
? Should a new navigation to retrieve item identifiers be added
  + does not desturb current navigation structure, returns a string
  - is there a use case for it?
```

TMQL Issue: RegExp operator in language

Impact on Language: low

Background

Sometime it would be convenient to have a Perl-like regexp operator inside the language, such as

```
select $person
 where $person is person
     & $person / name =~ /^Tim/i
```

Structured Discussion

```
? should TMQL have a binary regexp operator
  ? what would be the syntax
  - can be done with functions as well
  ? what regexp exactly (XPath, fn:matches)?
  + better readability
```

TMQL Issue: alias 'except' for --

Impact on Language: low

Background

The operator '--' subtracts two sequence, such as in

```
// person    --    // evildoing-evildoers
```

Should this also be allowed (all things about the person expect its shoesize):

```
$p / *    except $p / shoesize
```

Structured Discussion

```
? Should TMQL alias '--' to except ?
  + human-friendly syntax
  - very little addition
```