TMQL Issues

Robert Barta

<rho@devc.at>

$Id: issues.xml,v 1.4 2007/07/14 09:02:32 rho Exp $

Abstract

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.

Table of Contents

== UNRESOLVED ISSUES ==========

TMQL Issue: No templating, but templating in CTM

Impact on Language: low

Background

After a looooooong discussion it was decided that CTM hosts a feature called templates (href="http://www.semagia.com/tmp/ctm.html#sec-template). The basic idea is that arbitrary topic map constructs can be captured with a template first, without having any impact on the map to be generated. The template can have formal parameters. Later, templates can be invoked, provided with parameters and so inject chunks of content into the generated map.

Technically templates are functions (the 80ies were great), although templates cannot be recursive as there are no conditionals in CTM.

Ontologically speaking, a template is a new terminus introduced. It is a pattern which will occur in the map as often as the template is invoked. So, it can be argued, templates are a poor-man's solution to ontology engineering. And ontology engineering should be done in an ontology engineering language.

So, as long as there is no available ontology definition language for TMs, we will run into these problems, maybe solving them in as many ways.

Structured Discussion

```
? has this anything to do with TMQL?
  - probably not, it is a standards engineering issue
```

TMQL Issue: Reification syntax inconsistent with CTM

Impact on Language: low

Background

CTM uses the symbol ~ to indicate reification of TM items. The tilde does not incorporate a direction, so the convention is that the thing on the left of ~ is reified by a topic (ref) on the right.

Oh no! There is also a topicmap reifier which works completely different, although it is doing the same thing:

```
~ whatever -: "This is a strange planet"
```

introduces a full topic (not only a topic-ref) whatever which reifies the map. There is certainly a good explanation for this inconsistency.

TMQL uses the /reifier/ axis, abridged by ~~> in forward direction. Yes, that's it.

Structured Discussion

```
? align CTM syntax zoo with TMQL cleanliness?
```

== RESOLVED ISSUES ================

TMQL Issue: Removal of EVERY clause

RESOLVED

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
  + minimally smaller language
  - users have to twist their brains to get the NOT right
! Oslo 2007: consensus: rejected

? Removal of SOME clause
  + it is just a convenience
  - the symmetry is something which people expect
! Oslo 2007: vote (1/many) rejected
```

TMQL Issue: Refactorization of Standard-Parts

RESOLVED

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 work.

```
   - Atoms and Identifiers -> CTM
   - Navigation            -> TMDM
   - Environmental Clause  -> CTM
   - Predicates            -> TMCL/CTM
   - Ontologies            -> TMCL/CTM
```

```
   - Predef Types & Functions -> CTM/TMCL
```

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
! Oslo 2007: refine it as below

? Refactor atom handling into CTM
  = which atoms are there, how do they look like
  = would mean that this is defined in CTM
  = would mean that TMQL is just reusing the definitions
! Oslo 2007: accepted by consensus

? Refactor inferencing
  = at the moment inferencing is "predicates", i.e. associations only
    not general (inferring values for occurrences, for example)
  = taxonometric reasoning is implicitly defined by TMDM, should stay in TMQL
  = means: refactor into what?
    = new language for ontology definition (TIMBL, TOWL), or
      - may take time to jump-start
        - but Patrick says, this can be relatively fast
    = squeeze it into TMCL?
      - constraints and ontological info not the same
  - still: it means, that TMQL must ___ALLOW___ to specify onto info inline
  + huge benefit: TMQL is itself inferencing-agnostic and is operating on
    virtual map
    + different people have different needs
! Oslo 2007: accepted by consensus

? Refactoring ontology
   = means how to define that something is a TMish resource (can be anything)
     + yes, this does not affect TMQL at all
       - but there still needs a way to 'access' ontologies (as topics)
         = means a 'prefix' mechanics
         = AND!!!! means that if it is a topic it can be treated as such
   = can go into TIMBL/TMOL
! Oslo 2007: accepted by consensus

? Refactor 'Navigation'
  = Navigation is effectively the expectation a TMQL processor
    has about the map it navigates through.
  ? refactor where to?
    = into TMDM annex?
      - but this may be difficult as TMDM is '60.60'
    = into separate document?
      - does this have a value by itself?
      + maybe only for architects
! Oslo 2007: rejected by vote
  = keep navigation axes semantics inside TMQL
```

TMQL Issue: intermediate types

RESOLVED

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
     - no request or use case appeared
   ! dismissed by editor
```

TMQL Issue: Add inverse reification shorthand

RESOLVED

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
  - CTM reification is currently a huge mess
! dismissed by editor (may be re-raised later)
```

TMQL Issue: Is everything a 'thing'

RESOLVED

Impact on Language: medium

Background

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

The problem is that TMQL should offer users a 'catchall' concept, such that they can express a 'dont care semantics':

```
select $person
 where is-employed-by (* : tbl, organisation: $o) &
       $o isa ....
```

And the TMQL semantics of course also needs access to 'all the things in a map'.

What does this * amount to now? Is it identical to tm:subject? And are there specialized terms for (a) topics, (b) associations, (c) occurrences?

Structured Discussion

```
? Does tm:subject return all items in the map?
  + tm:subject is a great placeholder
  - TMDM does not say it (or does it?)
    - now it does, via the TMDM -> TMRM mapping
! fixed in TMRM appendix B

? How do topics, associations, .... relate then to tm:subject?
! defined in TMRM appendix B
```

TMQL Issue: Quantified Quantifiers

RESOLVED

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
    - a function fn:count (....) >= 10 has the same effect, so why introduce new keywords
      - not true: fn:count has to produce ALL results, count them and then do compare
        = an 'at least' expresses more what a user needs
        + can be used for optimization
! Oslo 2007: vote (7 yes, 1 no)
  = editors will present a proposal in the next draft
```

TMQL Issue: item identifiers for items

RESOLVED

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 baroquely 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 be added to retrieve item identifiers
  + does not disturb current navigation structure, returns a string
  - is there a use case for it?
! dismissed by editor
```

TMQL Issue: Intuitive Semantics of 'false'

RESOLVED

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
! constant 'undef' is added via type 'undefined' to cover these cases
```

TMQL Issue: Allow variables as role type

RESOLVED

Impact on Language: medium

Background

At the moment, TMQL allows only constants 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, ...)
```

This can also be extended to 'variables as assoc type':

```
select $person / name,
where
   $assoc_type ($whatever: $person, ....)

select $person, $foo
where
    $person / $foo
```

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
! Oslo 2007, rejected by consensus

? should variables be allowed for role types?
  - impedes optimization (use of indices)
! Oslo 2007: weak acceptance

? should variables be allowed for assoc types?
  - impedes optimization (use of indices)
! Oslo 2007: weak acceptance
```

TMQL Issue: Functions process sequences

RESOLVED

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 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 process 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.

This procedure would also be consistent with the case that we had several parameters, each potentially generated a sequence of values:

```
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 away
  - slightly unsual semantics for normal engineers, they are not used to sequence processing
    - Python, Ruby, Perl and Haskell all have list comprehension, so what?
! adopted into TMQM specification and formal semantics (by editor)
```

TMQL Issue: Elimination of the concept 'characteristics'

RESOLVED

Impact on Language: medium

Background

TMQL is using the term 'characteristics' as subsumption for topic names and topic occurrences. This is in deviation to an 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
  + 'characteristics' had a different meaning in the past
! Oslo 2007: consensus, better align it with TMDM
```

TMQL Issue: Datatype Awareness

RESOLVED

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 >> types == 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
  - can be done with a function tm:datatype ($person / shoesize)
! dismissed by editor
```

TMQL Issue: Error compatibility

RESOLVED

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 reduces compatibility between TMQL processors as one application has to expect potentially different sets of exceptions.

An example of a transient error would be

```
select $p
from http://www.topicmaps-are-us.com/ ~ ~~>
where
   $p isa person
```

Structured Discussion

```
? Should TMQL specify the mechanism?
  = i.e. exception propagation, or error codes?
  - this does preempt implementation styles
! Oslo 2007: consensus, maybe better not

? Should the TMQL standard name all error situations?
  + higher compatibility between TMQL implementations
  - less freedom for implementors
  - makes the standard take longer
    - ok, maybe a small bit
  - some conditions may be hard to differentiate, depending on the implementation strategy
  - compatibility on the error-level would be API compatibility
! Oslo 2007: rejected by vote (2/many)
  = no list

? Should TMQL name only a top-level?
  = error in analysis, error in evaluation
  = persistent error, transient error (Geir Ove Gronmo)
  + somewhat a middleground
! Oslo 2007: consensus: then only 'good' vs. 'bad'
```

TMQL Issue: XML generation, XML subset

RESOLVED

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.

Other ideas are to use an XSLTish syntax additionally for some flexibility:

```
<tmql:comment>
     sdfsfdsfd
</tmql:comment>

<tmql:element name="whatever">
     <tmql:attribute name="aname">avalue</tmql:attribute>
     content
</tmql:element>
```

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
 ! Oslo 2007: what do we *really* need?
   CDATA                       NO
   PI                          NO
   namespace                   YES
   DOCTYPE                     NO
   comments                    NO
   variable element names      YES
   variable attribute names    YES
```

TMQL Issue: native XML support

RESOLVED

Impact on Language: High

Background

In principle TMQL could do without any XML support. The only thing to be specified is that results should be generated in a fix, defined XML structure.

The downside of this is that SELECT, FLWR and path expressions only can generate tuple sequences and that the XML format only can contain exactly these.

Structured Discussion

```
? TMQL should support 0% of XML
  + TMQL syntax becomes smaller by 6 rules
  + implementation is a bit simpler
  - results cannot be free, deeply nested XML, but only flat
  - one always will need XSLT to beat the results into proper shape
! Oslo 2007: implicit rejection, see issue native-XML-support
```

TMQL Issue: RegExp operator in language

RESOLVED

Impact on Language: low

Background

Sometimes 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
! adopted, but not as =~ symbol but via generic function fn:regexp (or so), (by editor decision)
```

TMQL Issue: predefined data types

RESOLVED

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. Maybe the following (lheuer)?

```
- One of the floating point numbers
- one integer datatype
- xs:string
- xs:anyURI
- xs:boolean
- xs:date
- xs:dateTime ???
- xs:time ???
```

And what about an 'undefined' data type to express that the results is _undefined_?

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

Maybe also aggregation functions?

```
http://www.w3.org/TR/xpath-functions/#aggregate-functions
```

Another alternative is not to define _any_ primitive data type and leave that to implementations. The only thing to define in the standard are minimal requirements each data type has to satisfy:

```
serialisation rules (text to value, and back)
ordering (the meaning of <= operator)
```

Still, in TMQL, the binary operators +, -, *, / and the unary operator - can be left between value expressions. Implementations can the detail how these operators are then be interpreted for particular data types.

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
  - supported data types and supported functions are not strictly connected
! Oslo 2007: rejected by consensus
  => see issue 'refactoring': all predefined datatypes are named by CTM

? should NO type be native?
  + postpones decision to implementation
  - reduces the compatibility between implementations
! Oslo 2007: rejected by consensus

? should there be a predefined constant 'undefined'?
  + allows to express 'undefined' value
! Oslo 2007: weak acceptance
```

TMQL Issue: alias 'except' for --

RESOLVED

Impact on Language: low

Background

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

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

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

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

Structured Discussion

```
? Should TMQL alias '--' to except ?
  + human-friendly syntax
  - very little addition
! dismissed by editor: too little value
```

TMQL Issue: occurrence type implicit subclassing

RESOLVED

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

```
? should an occurrence type be implicitly a subtype of 'occurrence'?
  + this is what people will expect
  = also that a nickname is a special name
! Oslo 2007 resolution: accepted

? where should this fact be standardized?
  = TMDM
    - even an annex is difficult to add
  ! Oslo 2007: rejected by consensus
  = CTM
    - why CTM and not XTM (et.al)?
      + because it would be at least one place to get it right?
```

```
! Oslo 2007: rejected
= put it into TMDM -> TMRM mapping (LarsM)
! Oslo 2007: accepted by consensus
```

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

RESOLVED

Impact on Language: very high

Background

A function is a systematic functional dependency over particular sets of values. The age of a person, for example, 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, and items within a topic map.

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
! Oslo 2007: no, by consensus
  => all this should be hosted inside a TM 'ontology language'
  => still, TMQL must now allow statements made in this "TMOL" language
```

TMQL Issue: Functions and Predicates as first-class topics

RESOLVED

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 == comparison 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 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 occurrence values:

```
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 existing one?
  + it is a TMish view on existing concepts
    + directly reflects a TM-view on functions, no further explanation to a user necessary
    + 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 elements
        have to be mapped onto a TMish view of them
      - yes, but we need syntax for: functions, predicates and 'prefixes'
+ reduces syntax for TMQL
  - yes, but it puts some burden onto CTM
    - which is minimal ({{...}}...)
- dedicated syntax has less clutter
  - every dedicated syntax is _expected_ to have less clutter
  - and a second step to expose functions as topics has to be defined
- parsing gets complicated when functions are subclassed (my_func iko tmql:function)
  - not really, the functionality must be in a TMQL processor already
- parsing gets more complicated when scoping is used
  - maybe, not sure how much of an issue this is
- with dedicated syntax the requirements on the (function and predicate) body
  can be tested better by a parser
  + it is not possible to 'hide' a function using subtypes of tmql:function
  + it is not possible to abuse scope
! Oslo 2007: this question will be refactored into a TMOL language
  => see 'refactorisation' issue
  => how functions and predicates can be declared is one thing
  => how TMQL 'experiences' these functions and predicates is another thing
```