

ISO/IEC JTC 1/SC 34 N 1324

DATE: 2009-11-19

ISO/IEC JTC 1/SC 34
Document Description and Processing Languages
Secretariat: [Japan \(JISC\)](#)

DOC. TYPE	Meeting Report						
TITLE	TMQL path language proposal [WG 3]						
SOURCE	WG 3 Convenor [Mr. Patrick DURUSAU]						
PROJECT	JTC 1.34.18048						
STATUS	Notes from the WG 3 Leipzig November 2009 Meeting on FCD 18048, Information technology -- Topic Maps -- Query Language (TMQL). This document is circulated to the SC 34 members for information.						
ACTION ID	FYI						
DUE DATE							
DISTRIBUTION	P, O and L Members of ISO/IEC JTC 1/SC 34 ; ISO/IEC JTC 1 Secretariat; ISO/IEC ITTF						
ACCESS LEVEL	Open						
ISSUE NO.	105						
FILE	<table border="1"><thead><tr><th>NAME</th><th>SIZE (KB)</th><th>PAGES</th></tr></thead><tbody><tr><td></td><td></td><td>44</td></tr></tbody></table>	NAME	SIZE (KB)	PAGES			44
NAME	SIZE (KB)	PAGES					
		44					

Secretariat ISO/IEC JTC 1/SC 34 - IPSJ/ITSCJ (Information Processing Society of Japan/Information Technology Standards Commission of Japan)* Room 308-3, Kikai-Shinko-Kaikan Bldg., 3-5-8, Shiba-Koen, Minato-ku, Tokyo 105-0011 Japan *Standard Organization Accredited by JISC
Telephone: +81-3-3431-2808 ; Facsimile: +81-3-3431-2808; E-mail: kimura@itscj.ipsj.or.jp



TMQL path language proposal

Leipzig, November 2009

Feedback from committee appears
in blue, like this.



The proposal

- This is a proposal for a replacement to the current TMQL draft path language
- It's based on the proposal posted on LMG's blog, but modified
 - <http://www.garshol.priv.no/blog/205.html>
- This is an informal presentation of the language in order to
 - judge what the community thinks of the proposal, and
 - get feedback on alternative design choices in the language

Generally OK. However, still many open questions.



Topic references (issue 1362)

- We use the same syntax as CTM for this
 - ids are item identifiers
 - qnames are subject identifiers
 - same way to define prefixes
 - etc etc
- *Maybe* subject identifier and locator references as in CTM
- Not providing any way to refer to topics by name

Basics

- foo # a topic reference
- ... / axis::type [filter] # a navigation step
- ... / axis:: [filter] # same, without type filtering
- ... / type # shorthand navigation step

- lmg / email # all my email addresses
- lmg / occurrence::email # the same
- lmg / email @ private # email addresses in “private” scope
- lmg / email [. / scope:: = private] # fully expanded version



More examples requested in meeting

- Emails in unconstrained scope
 - `Img / email [not(. / scope::)]`
- Emails with exactly private
 - `Img / email @private [length(. / scope::) == 1]`
- Emails with private and past
 - `Img / email @private @past`
- Email with private or work
 - `Img / email [. / scope:: == private OR . / scope:: == work]`

Some more examples

- Path expressions starting with “/” start from the topic map item
 - / person # all person topics
 - / default::person # full expansion
 - / employed-by # all employed-by associations
 - / person / email # all email addresses of all persons
 - / person [. / email] # all persons which have an email occ.
 - / topic:: / email # all email addresses



Alternative syntax

- Inge Henriksen points out that technically, only the first slash is needed
 - the first slash is needed so you know whether to start from the topic map or not
- The result would be
 - / person email
 - lmg / employee association::employed-by employer *
- instead of
 - / person / email
 - lmg / employee / association::employed-by / employer / *

We like the slashes

Filters

- The [filter] contains a boolean expression
 - simple path expression: true if it produces at least one value
 - / person [. / email] really means / person [exists(. / email)]
 - comparison expression: <, >, <=, >=, !=, =
 - AND, OR, NOT
- NOT Parentheses are optional with not
 - / person [not . / email] # one possible syntax (not is operator)
 - / person [not(. / email)] # another (not is function)
- ./
 - lets us distinguish topic references from path navigation steps
 - / person [. / start-date = . / end-date]
 - / person [. / employed-by(employee -> employer) = tmlab]
 - / person [not(email)] # true if email topic exists
 - / person [not(. / email)] # true if person has email



Alternative

- / person [employed-by(employee -> employer) = item-id(tmlab)]
 - in filters, any topic reference, say (“foo”) is interpreted as “. / foo”, that is, a navigation step
 - / event [start-date = end-date]
 - item-id may be a function, maybe not?
 - item-id(tmlab) -> / default:: [item-identifier = full-uri(tmlab)]

Nobody likes the . /

Unfortunately, it's hard to get rid of
This slide suggests a way to lose it
The editors will work on this



Alternative #2 (proposed in meeting)

- / person [employed-by(employee -> employer) = / iid::tmlab]
 - unfortunately, this does not match the model of the current language
 - to make it work, we would have to
 - introduce a new iid axis on the topic map item, and
 - change how type filtering works for this particular axis



Navigating from topic map to topic by id

- \$topicmap / topic:: [item-identifier:: = whatever]
 - this is equivalent to “whatever”

The axis syntax

- We are not 100% satisfied with the axis::type notation
- Problems with it
 - lmg / occurrence::email # fine
 - lmg / occurrence:: # all occurrences of any type
 - lmg / subject-identifier:: # not as pretty, perhaps?
 - \$c / occurrence::tmcl:card-min # lots of colons there...
- Alternative #1
 - lmg / occurrence::email
 - lmg / occurrence # ambiguous, unfortunately
- Alternative #2
 - lmg / occurrence(email) # looks like function call. problem?
 - lmg / occurrence() # of any type
 - lmg / subject-identifier()
 - \$c / occurrence(tmcl:card-min) # no colon collision any more

The axes

- topic
 - default
 - name
 - occurrence
 - role
 - subject-identifier
 - subject-locator
 - item-identifier
 - reified
 - type
 - instance
 - supertype
 - subtype
- association
 - role
 - type
 - scope
 - reifier
 - item-identifier
- role
 - default
 - type
 - player
 - association
- name
 - type
 - value
 - ...

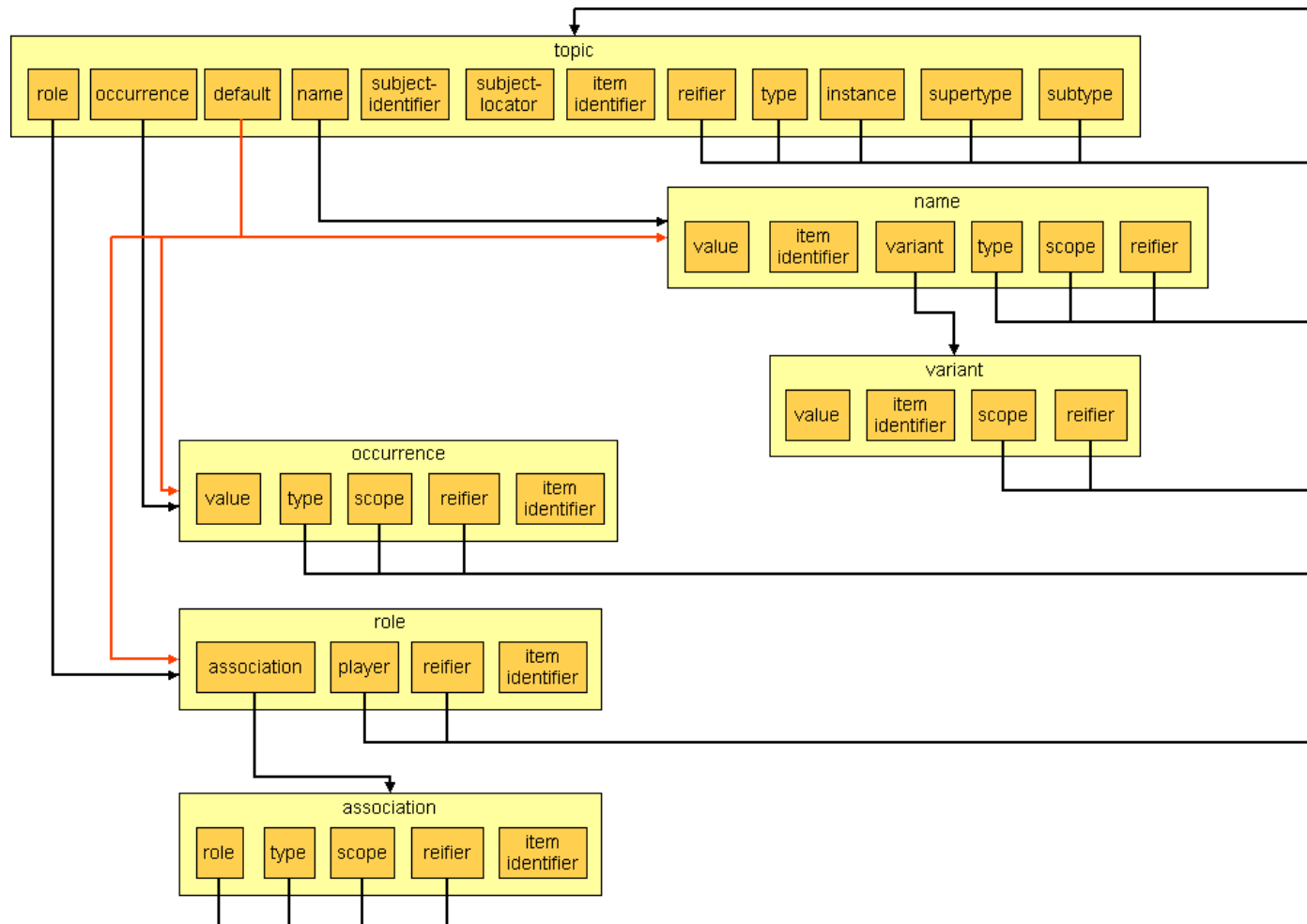


Proposal for change to default axis

- Always make “default” equal union of all other axes
 - this applies to all information item types equally
- More limited proposal: only do this for role items
 - rationale: association types, player types, and types of role types will never intersect
 - unfortunately: slide 17 (Association traversal (2)) shows that this is not actually true...

No clear conclusion on this

The axes



Association traversal

Dmitry does not like that the association:: is required here.
The difficulty is that we're not sure how to remove it.

- Somewhat cumbersome
 - lmg / employee / association::employed-by / employer / tm:subject
- Built-in operator for traversal
 - lmg / employed-by(employee -> employer)
 - handles symmetric associations automatically
- Supports additional constraints
 - lmg / employed-by(employee -> employer) @ past
 - lmg / employed-by(employee -> employer) [. / reifier:: / start-date < "2000-01-01"]
 - lmg / employed-by(employee -> employer)

Dmitry thinks it's confusing that the context node in association operators is not the item produced by the navigation step, but actually the association item. He suggests that we should use .. instead. (Or perhaps to have another filter syntax in this case.)

How to filter the players that are produced by the association traversal operator? For example, how to produce only employers based in oslo?



Association traversal (2)

- `Img` # the 'Img' topic
- `Img / employee` # all roles of type "employee" (of `Img`)
- `Img / employee / association::employed-by`
 - # all associations of type `employed-by` (via `employee` role)
- `Img / employee / association::employed-by / employer`
 - # all roles of type "employer" (from the `employed-by` associations)
- `Img / employee / association::employed-by / employer/ tm:subject`
 - # all players of the `employer` roles
 - # (with modified default axis): all players, all role types, all association types

Association traversal filtering

- `Img / employed-by(employee [. = role] -> [. = assoc] employer [. = role])`
- `Img / employed-by [. = assoc] (employee [. = role] -> employer [. = role])`
- `Img / employed-by [. = assoc] (employee -> employer)`

- Another suggestion is:
 - let the operator produce association roles (instead of the players)
 - user then has to produce players explicitly
 - filter on the entire operator would use `. = role`
 - with the modification that `scope:: then` really applies to the association

Constraints on roles

- Supported via filters
 - \$company / represented-by(represented -> representative) [. / at = tmra]
- However, one could also reuse association predicate syntax:
 - \$company / represented-by(represented -> representative, at : tmra)
 - benefits
 - makes association traversal look like association predicates (familiar)
 - shorter
 - disadvantages
 - makes association traversal look like association predicates (which they are not)
 - bigger language



Other ideas for predicate integration

- / killed-by(-> victim, perperator: tosca, method: stabbing)
 - path expression with topic map as input
 - produces player of victim role
- killed-by(perperator -> , ...)
- Supports having the filters in different positions in the operator



Should path expressions be usable alone?

- That is, should the path expressions be considered a complete sub-language that can be implemented and used on its own, without the rest of TMQL?
- Alternatives:
 - path expressions only exist within SELECT/FLWR/...
 - path expressions work differently in SELECT/FLWR from when used on their own
 - path expressions can be used alone, and always follow the same rules

Variables in predicates

- If we reuse the predicate syntax, what about
 - \$company / represented-by(represented -> representative, at : \$event)
 - \$event is unbound
 - \$company / \$foo # \$company and \$foo is unbound
 - lmg / email
- Possible answers
 - this is confusing, so we shouldn't use this syntax
 - path expressions can't bind variable values, so this is a runtime error
 - path expressions *can* bind variable values, so this is OK
- Note that the last answer raises a bigger issue
 - should path expressions return just a set, or a more complex result?

Committee is undecided on this point.
Need more complete explanation
of how variable binding would work.
Look at how to solve use cases without
variable binding.

Proposal for variable binding

- Proposal as it stands does not support binding context items to variable values
 - for example, `lmg / $foo` binds `$foo` to all types of characteristics (not to the characteristics themselves)
- Proposed solution
 - `lmg / tmdm:subject as $foo #` binds `$foo` to the actual items
 - the “as <variable>” can appear anywhere a filter can appear



Allowing any type

- Original proposal had *
 - \$person / occurrence::* # all occurrences of person
- We now use a blank
 - \$person / occurrence:: # same
- tmdm:subject can always be used
 - \$person / occurrence::tmdm:subject # same

Boolean and set operators

- Booleans
 - / person or dog # not allowed
 - / topic:: [. / type = person or . / type = dog] # allowed
 - / (person or dog) # not allowed today
- Set operators
 - (/ person UNION / dog) / email # would have worked
 - however, we don't allow set operators in the path language
 - these are left for SELECT/FLWR statements

We want set operators: union,
intersect, except (set difference).



Allow general TMQL in path expressions?

- The current proposal does not do this
- The previous draft did

Taxonomy

- Given
 - `Img isa person . person ako creature . creature ako subject .`
 - `/ creature # returns Img`
- The problem is querying for the types
 - `Img / type::` # returns person, creature, subject
 - `person / supertypes::` # returns creature, subject
- How to get only what is said explicitly in the topic map?
 - `Img / tm:type-instance(tm:instance -> tm:type)` # do it explicitly
 - `Img / direct-type::` # add extra axis
 - # requires direct-type, direct-instance, direct-supertype & direct-subtype
- A more general proposal
 - `Img / type(0)` # `Img`
 - `Img / type()` # `person`
 - `Img / type(1)` # `person`
 - `Img / type(2)` # type of person (which is nothing)
 - `Img / type(1) / supertype(1)` # `creature`
 - `Img / type(1) / supertype(0..*)` # `person, creature, subject`



Another taxonometry proposal

- `Img / type:: // supertype::`
 - the “//” means apply next navigation step to get transitive closure
 - ie: apply the step again and again until it doesn't produce new values
- This turns the situation around
 - `Img / type::` now produces only explicit types
 - `Img / type:: // supertype::` produces implicit types as well



Taxonomy (2)

- Rani says:
 - “I don't like that ‘type’ refer both to types and to supertypes. This seems not clean to me.”
- Proposed solution
 - `Img / parent-type()` # person, creature, subject
 - `Img / type()` # person

Functions

Tentatively approved

- `concat(str, str)`
- `starts-with(str, str)`
- `ends-with(str, str)`
- `contains(str, str)`
- `substring-before(str, str)`
- `substring-after(str, str)`
- `substring(str, int, int?)`
- `string-length(str)`
- `normalize-space(str)`
- `translate(str, str, str)`
- `find(str, str)`
- `ceiling(float)`
- `floor(float)`
- `round(float)`
- `count(resultset)`
- `matches-regexp(str, str)`
 - uses XML Schema regexps
 - boolean result
- `extract-regexp(str, str)`
 - returns the first substring which matched the regexp
- `topicmap()`
 - returns the topic map set in the query context as *the* topic map



Extensibility of functions

- TMQL allows the environment to provide additional functions
 - probably their names should be qualified (as in qnames) somehow
 - maybe one has to bind a prefix and refer to them that way
- However, there is no way to define them within TMQL itself

Approved



Functions and sets

- func(pathexpr) # passes result set to function
 - count(/ person) # counts person topics in TM
 - string-length(/ person / email / value) # gets length of *one* email
- Not possible to do make a list of the lengths of all emails
 - in the path language, that is!
 - in SELECT statements this *is* possible

Type conversion

- Implicit in comparisons
 - / person [. / email = “larsga@bouvet.no”] # occurrence converted
- Implicit in function calls
 - / person [string-length(. / email) > 20] # occurrence converted
- Explicit elsewhere
 - string(lmg) # converts ‘lmg’ topic to string
- Note that if we do this we can drop the value axis
 - or should we keep it for completeness?

Generally this is OK.

Xuân suggests that we keep the value axis and that we have a length axis on strings.

Slicing

- In the existing TMQL draft slicing is supported in two ways
 - / foo / bar [1] # 1-based indexing (which XPath uses)
 - / foo / bar [1 .. *]
 - select ... offset 50 limit 25
- Inge Henriksen suggests we should preserve this feature
- Benjamin Bock suggests we allow negative numbers (like in Python slicing)
- Xuân suggests we should have 0-based counting

Generally this is OK.



Tuple constructors

- In the existing TMQL draft path expressions can produce tabular results, as follows:
 - / person (. / name, . / email)
- This would produce a set of (name, email) pairs
- This has been deliberately omitted in the current proposal



Axes & functions

- They could be merged into a single concept
 - editors will consider this



Operators

- Editors need to consider which ones to include
- Should probably be expressed as functions



= or == for comparison?

- Make our minds up!



Execution model

- Will results be sequences, set, multisets...?



Language support for which datatypes?

- Meaning
 - literals, operators, comparators, ...
- Editors need to decide
 - probably same as CTM and TMCL



Comparison of values

- We need to define how this works
 - particularly with sets and datatyped values
- (Also, what is the type of the value produced by the value:: axis)



Ordering in path language?

- Don't know yet whether we will have it



Next steps

- Editors to do their homework
 - output is a document rather like the Toma description document
 - http://ontopia.googlecode.com/svn/trunk/sandbox/toma/doc/TW_UM_TOMA_110.pdf
- Later on
 - editors probably produce an implementation & spec in parallel