TMQL

Getting started

Agenda for the day (0900-1400)

- **Introduction**
  - goals and requirements
  - status and work remaining
- **Query language presentations**
  - assorted attempts LMG
  - AsTMa? Robert Barta
  - tolog LMG
- **Discussion**
  - find out how to move forward from here

What we want

- **A query language that**
  - simplifies topic map application development
  - removes the need to use an API to extract information
  - can help the adoption of topic maps
  - play a role for topic maps similar to that of SQL in RDBMSs
  - can be used in higher-level technologies

Status of TMQL work right now

- **ISO has**
  - decided to create TMQL as ISO 18048 (multi-part)
  - appointed two editors: yours truly and Hans Holger Rath of DIN
  - created a requirements document (N0249)
  - started work on a use case collection
  - invited proposals for query languages
- **A number of query languages have been proposed**
  - AsTMa? by Robert Barta
  - tolog by Ontopia
  - eTMQL by empolis
  - Ann's LTM-based strawman
  - “let's use XPath or XML Query” by multiple people

What we want to achieve today

- **Decide on the way forward**
  - will we create a use case collection?
  - should we update the requirements document?
  - how do we kick-start the work on the language itself?
- **Decide how to come up with a language proposal**
  - select one of the languages presented today as the starting point?
  - give the editors the task of creating one (or more) new proposals?
  - attendees should evaluate the query languages presented and consider how appropriate they find them

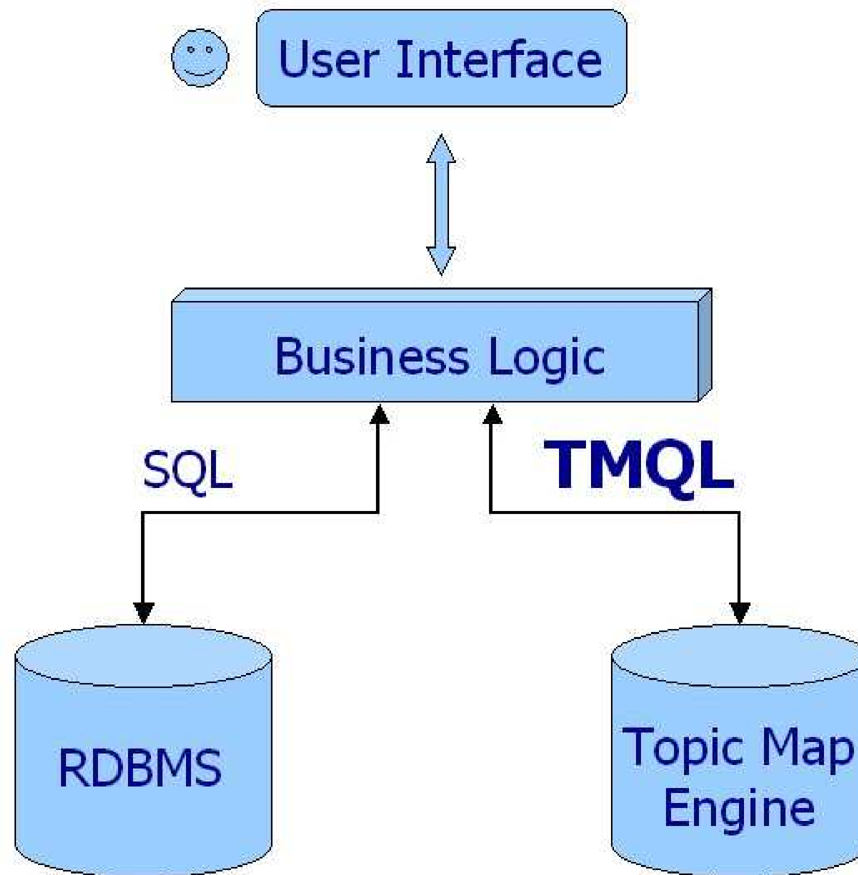
Overview of requirements

- **Syntax must be concise and human-readable**
- **Language must be defined in terms of SAM**
  - thus it can support XTM, HyTM, LTM, and AsTMa= at the same time
- **Language must be independent of usage context**
- **Language must be properly internationalized**
- **Language must be strictly defined**
- **Language must have support for third-party extensions in a controlled way**
- **May support logical inferencing**
- **Should be optimizable and possible to implement efficiently**

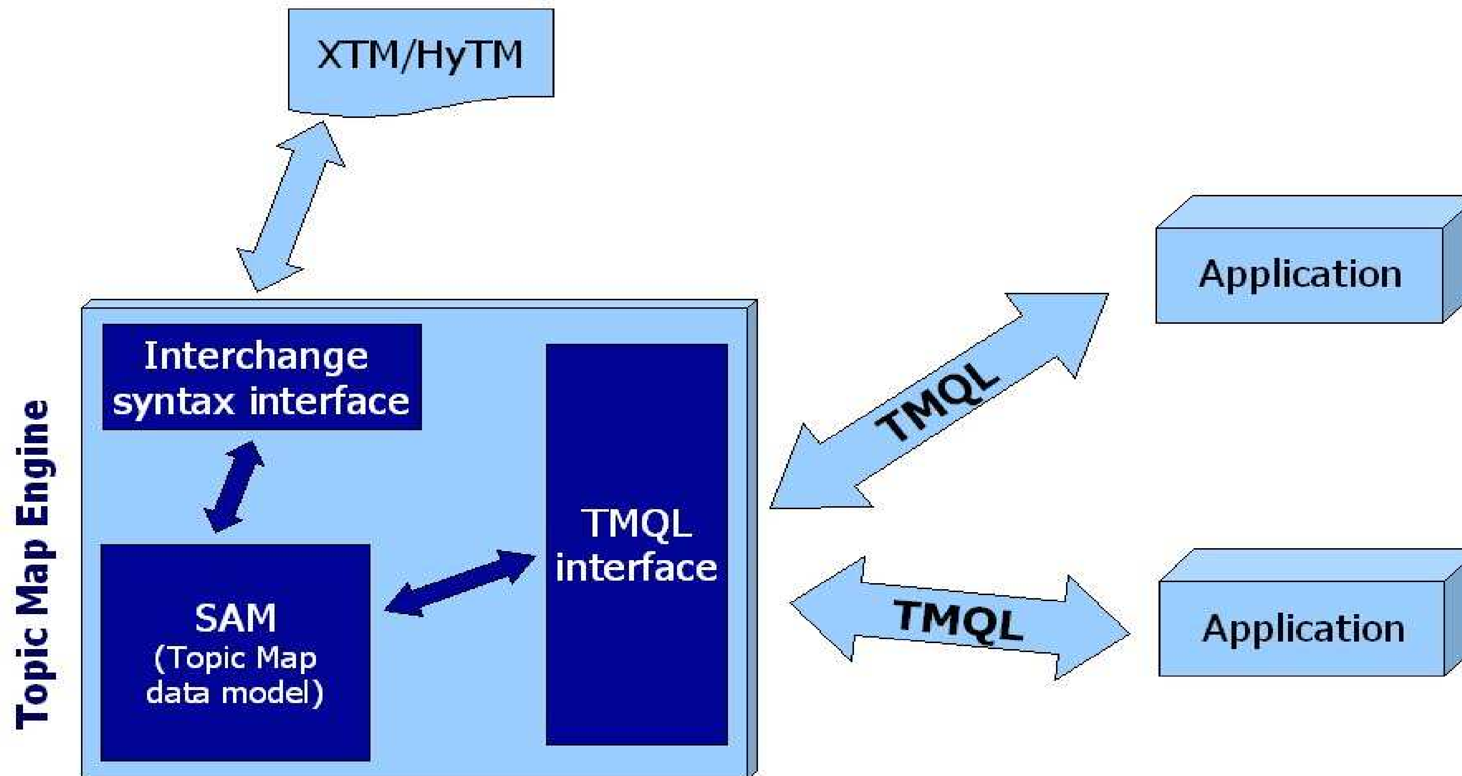
Uses of TMQL

- **In applications, when extracting info from TM**
  - our customers use tolog in web applications, for example
  - to list all students in course, query, then traverse result to output list
- **Also used in auto-generation of topic maps**
  - specifying conditions for special processing and deletion, etc
- **Could be used in topic map access protocol on the net**

TMQL in business logic



Anatomy of TMQL processors



Empolis TMQL

Examples, evaluation

empolis TMQL

- **The first topic map query language**
- **Implemented in their K42 product**
- **Designed to resemble SQL**
- **Uses name searches to address topics**
- **Could query all aspects of topic maps**
- **Status**
  - will not be developed further
  - has been replaced by the eRQL RDF query language
  - their new eKMS product is a “metadata service supporting both RDF and XTM” which will use eRQL
  - note: empolis remains committed to implementing ISO TMQL

Example query

- **Which operas were composed by Germans influenced by Mozart?**
- **More formally**
  - All topics of type "Opera"
  - which were composed by "Persons"
  - which were influenced by "Mozart"
  - and born in "Germany"

empolis TMQL

```
SELECT topic x WHERE
 x instance_of topic named "Opera"
AND
 x in (assoc template_is assoctemp named "composed by") has
 topic person instance_of topic named "Person"
AND
 person in (role named "influenced person") in
 (assoc template_is assoctemp named "influenced by") has
 (role named "influencing person") has topic named "Mozart"
AND
 person in (assoc template_is assoctemp named "born in") has
 topic named "Germany"
```

Holger's evaluation of eTMQL

- **Pros**
  - supports querying of all parts of topic maps, even regexps in names
  - quite a complete set of query constructs
- **Cons**
  - the syntax is “read-only”; hard to write, easy to read
  - lacks sorting and functions on the result set
    - this can of course be done in the programming language
  - insufficient variable handling, e.g.
    - after a variable has been given a value it cannot be further constrained
    - variable pairs in SELECT are not returned as pairs, so information about which x goes with which y is lost

tmfun

An example query language

tmfun

- **My other attempt to create a query language**
  - inspired by the Ontopia Navigator Framework
- **Based around the idea of a kind of TM “algebra”**
- **Functions are applied to sets of objects to produce new sets**
- **mozart**
  - returns a set containing the 'mozart' topic
- **occurrences(mozart)**
  - returns a set containing all occurrences of the 'mozart' topic
- **occurrences(mozart, date-of-birth)**
  - filters the set returned so that only 'date-of-birth' occurrences are returned

Traversing associations

- **Find Mozart's birthplace**
  - player(roles(associations(roles(mozart, person), born-in), place))
- **Clearly, this works**
- **Equally clearly, it's very verbose and not very readable**
- **Possible solutions**
  - special functions for association traversal
    - traverse(mozart, person, born-in, place)
  - special traversal syntax (instead of functions)
    - mozart person born-in place
- **Both of these seem to work, the second perhaps being the easiest to understand**
  - mozart date-of-birth

The Mozart influence

- **opera instances composed-by ...**
  - here we get into trouble
  - we've found the topic we want, but we want to put conditions on it
  - we can't traverse further, because that'd give us Germany or Mozart
  - possible solution: insert [condition] like in XPath
- **opera instances composed-by**
  [influenced influenced-by influencing ... AND
  born-in ...]
  - we can't just insert constants here, since they are not traversal steps
  - special syntax like == operator could be used to do this
- **opera instances composed-by**
  [influenced influenced-by influencing == mozart AND
  born-in == Germany]

Interactions

- **People born same place they died**
  - person instance-of [born-in = died-in]
    - we use '=' (not '==') to indicate traversal on both sides
- **Number of opera premieres per city**
  - city instance-of (premiere-of UNION located-in premiere-of)
    - now we've found all operas by traversing that path, but no counting
  - city instance-of count(premiere-of UNION located-in premiere-of)
    - now we've found the numbers, but we lose the cities...
  - city instance-of
tuple(this, count(premiere-of UNION located-in premiere-of))
    - tuple function produces (x, y) value pairs
- **Unresolved issues with no dependencies**
  - sam issue-in [not(status-of == resolved) AND
not(dependent depends-on prerequisite)]

Conclusion

- The traversal approach *appears* to work
- Quite easy when producing a single set of values
- Not as easy when producing collections of values
- Queries look a little bit strange
- Can *probably* be implemented efficiently

Coming up...

Robert Barta with AsTMa?