AsTMa* TM Engineering

Robert Barta
ρτ information systems
rho@bigpond.net.au

Apologies/Disclaimers

- NOT a proposal for a standard
  - too many open issues
- set of ideas which may make sense
  - or not, choose yourself
- slides produced with PowerPoint
  - violates all BCP of knowledge engineering
- Larry Wall:
  - all language designers must be arrogant

TM Engineering

- this is NOT reasoning
- this is NOT philosophy about “things”
- this is NOT “social meaning”
- tasks
  - authoring, maintaining
  - constraining, filtering
  - retrieval, query

AsTMa* Language Family

- common formal foundation
  - maplets, tau-algebra
    - expressions: (M1 + M2) * C * Q
  - reasoning ABOUT the language possible
    - optimization
  - relationship to outside semantics
    - RDF, XBRL, ...
- common notation and concepts
  - natural migration path for TM users

AsTMa* Pyramid

- AsTMa?
  - query, ontology transformation
- AsTMa!
  - constraints, ontology definition, filter
- AsTMa=
  - factual maps, optimized for human authoring
- AsTMa+
  - update, map differential

AsTMa= authoring

```
# this is a topic
astma-equal is-a tm-authoring-language
bn: AsTMa=
oc (homepage): http://astma.it.bond.edu.au/
in (comment): not for XML lovers

# this is an association
(is-part-of)
whole: astma-family
part: astma-equal
```

AsTMa= Features

- define topics, associations
- scoping of bn, oc, in, and associations
  - only one scope
- typing of oc, in, associations
  - bn to be added
- reification of associations [topics ?]
- no merging facility
  - tau-algebra

AsTMa! Example

```
forall [ $a
         bn : /AsTMa/i ]
  => exist ]
            (is-emplyed)
            emp : $a
            employer: bond ]


ts [ $a
              oc (homepage) @ rho : xxx
             ]
     and
     exists [ (is-part-of)
```

AsTMa! Objectives

- constraint language
  - validation: M * C -> {true, false}
  - filtering: M * C = M'’
- ontology definition language
  - vocabulary (AsTMa=)
  - taxonomy, type system (AsTMa=)
  - (app-specific) constraints (AsTMa!)

AsTMa! Objectives (cont'd)

- Ontology management
  - OR: O1 + O2
    - are two ontologies compatible?
    - is there a map which can conform?
  - AND: O1 * O2
    - does a particular map conform to both?
  - transformation: O * Q
    - ontology ITSELF is transformed (futuristic)

AsTMa! Formal Semantics

- good to scare students
- allow to define precisely when a map matches a constraint:
  - $C \models M$
- maps built from ‘maplets’
  - maplet is association
  - topics are simply characteristic tuples
    - (type, scope, value)

AsTMa? Rationale (Phases)

- define a collection
  - SQL: FROM table, XQuery: LET ...
- iterate over all components
  - SQL: all rows, XQuery: all nodes
- filter out wanted ones
  - SQL: WHERE bool-expr, XQuery: eval XPath
- generate content from matches
  - SQL: SELECT ..., XQuery: ...{$x}...

AsTMa? Rationale (Pipelining)

- pipeline
  - output of SQL is table
  - output of XQuery is XML
- consequences
  - output can be queried again
  - subqueries
  - optimizations
- input structure must be output structure

AsTMa? Design Goals

- retrieval language for applications
  - classical query (open iterator, iterate, close)
  - embedding into XML application servers
- stand-alone ontology transformation
  - ala XSLT for XML
  - map M1 conforms to C1: C1 |= M1,
  - transform with query Q: M1 * Q = M2
  - C2 |= M2, i.e. mediate between ontologies C1, C2

AsTMa? Influences

- SQL (tuple select, WHERE)
- XQuery (LET, functions, RETURN, ...)
- Prolog/Datalog (matching, backtrack, ...)
- Perl & Friends
- and pretty much everything good from Lars

AsTMa? Example

- find all operas, return them one by one

```
IN http://whereever/opera.xtm
WHERE
   exists $t [ * (opera)
               oc (homepgae) : $h ]
RETURN
   ($t, $h)
```

AsTMa? Example

- same but using defaults

```
IN http://whereever/opera.xtm
WHERE
   exists [ * (opera) ]
```

AsTMa? Example

- same but now from a TM backend

```
IN tm://server1/opera
WHERE
   exists [ * (opera) ]
```

AsTMa? Example

- same but using a variable first

```
LET $m := tm://server1/opera
IN  $m WHERE
    exists [ * (opera) ]
```

AsTMa? Example

- enriching a map with another map

```
LET $m := tm:opera
IN  $m + tm:blues # merging maps
...
```

- enriching with ontological knowledge

```
LET $m := tm:opera
LET $o := tm:music
IN  $m + $o
...
```

AsTMa? Example

- why query at all?

```
LET $m := ...
LET $o := ...
RETURN
        $m + $o
```

- for the impatient:

```
RETURN ... + ...
```

AsTMa? Example

- generate XML code

```
<tosca>
   IN tm:opera
   WHERE
      exists $t [ tosca ]
   RETURN
      FOR $b IN $t/baseName
          <name>{$b}</name>
</tosca>
```

AsTMa? Example

- generate XTM code

```
<topicMap>
      IN ....
      WHERE ....
      RETURN
            {as_xtm($t)}
</topicMap>
```

AsTMa? Functions

- define a function

```
function enrich ($o : ontology) {
   RETURN tm:opera + $o
}
```

- invoke a function

```
IN enrich(http://ontologies.org/music.atm)
...
```

AsTMa? Sorting

- sorting using RETURN

```
IN tm:opera
WHERE
  exists $t [ * (opera) ]
RETURN # tuple mode
  ({$t/baseName[scope = “#pravi”]})
SORT BY
  $t/baseName[scope = “#sort”]
```

AsTMa? Example

- sorting in FOR

```
IN tm:opera
WHERE
   exists $t [ * (opera) ]
RETURN # tuple mode again
   FOR $o IN $t/topic
     ({$t/baseName[scope =“#pravi”]})
   SORT BY
      $o/baseName[scope = “#sort”]
```

AsTMa? Example (Negation)

- all operas which have no composer

```
IN tm:opera
WHERE
   exists [ $o (opera) ]
   AND
   not exists [ (is-composer-of)
                 opus: $o ]
```

AsTMa? Example (Subquery)

- compute the statistics

```
LET $m := tm:opera
LET $o := IN $m WHERE exists [ * (opera) ]+
LET $s := $o * STATS
RETURN $s
```

- for the impatient Perl hacker

```
RETURN (IN tm:opera WHERE exists [ * (opera) ]+)
       *
       STATS
```

AsTMa? Summary

- LET assignments, tau expressions
- WHERE filters
  - AsTMa! constraint, matches to a submap,
  - bind variables
- RETURN
  - constructor generates content, uses variables
  - XML, raw topicMap, tuples
- FUNCTIONS

AsTMa? Controlled Extensions

- for vendors to distinguish their products
- specialized map operators
  - STATS, transitive hulls, other graph operations
- specialized ontologies
  - cartridges for application domains (biology)
- specialized inference rules (transitivity, ...)
- specialized reasoning ??

AsTMa? and TagLibs

```
<astma:query>
  <operas>
    <astma:in>tm:opera</astma:in>
    <astma:where>.....</astma:where>
    <astma:return>...
       <opera>{$b}</opera>
    </astma:return>
  </operas>
</astma:query>
```

AsTMa? Language Bindings

```
my $tm = new XTM (url  => ‘tm:opera’);
my $q  = new XTM (text =>
     ‘WHERE
        exists [ * (opera) ]
      RETURN’);
my $it = $q->execute ($tm);
while (my $s = $it->fetch) {
    print $xtmp->(‘baseName...’, $s);
} # using XTMPath to access a component
```

AsTMa? Language Bindings

```
my $q  = new XTM (text =>
     'WHERE
        exists [ * (opera)
                 bn: $bn
                 oc: $oc ]
      RETURN ($bn, $oc)');
my $it = $q->execute ($tm);
while (my ($b, $o) = $it->fetch) {
    print $b, $o;
}
```

But I want XML, XML, XML

- encode tau expressions in XML
- encode AsTMa! as XML
  - XAsTMa
- use X(TM)Path as (shorthand) notation

Architecture (dedicated)

- how to integrate lower-order entropy data?
  - SQL, XML
- how to integrate RDF?
- virtual maps
  - bringing all into TM space
  - query there
  - query will be transparently split
- NO knowledge of this in the query

Architecture (generic)

```
IN tm:opera + rdbms:mysql:fans + xml:db:books
WHERE
    exists [ sin: skdsfdkjs ]
    AND
    exists [ (fans) - fan: $fid - opera: $o ]
    AND
    exists [ (is-xpath)
             xpath: books/book[title="$b"]/isbn
             value: $isbn
RETURN
   ($o, $b, $fid, $isbn)
```

AsTMa* Status

- AsTMa=: stable, implemented
- AsTMa!: stable, proof of concept
  - transformation into Prolog
  - no experiences yet (read: not student-proof)
- AsTMa?
  - vaporware yet, NOTHING implemented
  - estimates: only over my dead body!
- AsTMa+: speculation only

AsTMa* Formal Considerations

- tau algebra is a mathematical model
  - defines the operations * and +
  - on maps, constraints and queries (and updates)
- expressiveness?
  - based on AsTMa!
  - can be implemented in Prolog
  - but looks like a much simpler logic (DL?)
    - anyway HUGE theories are available

AsTMa* Conclusions

- no need for a transformation language
- no need for artificial predicates

- everything is treated TMish
- queries and constraints can be factorized
- integration with other databases feasible
- same notation throughout

One notation to write them all,
One to retrieve them,
One notation to constrain them all
And with its semantics bind them
In the Land of London where the
Fogs do lie.