

Toma – TMQL, TMCL, TMML

Space Applications Services

Rani Pinchuk

Richard Aked

Juan-Jose de Orus

Els Dessin

David De Weerd

Georges Focant

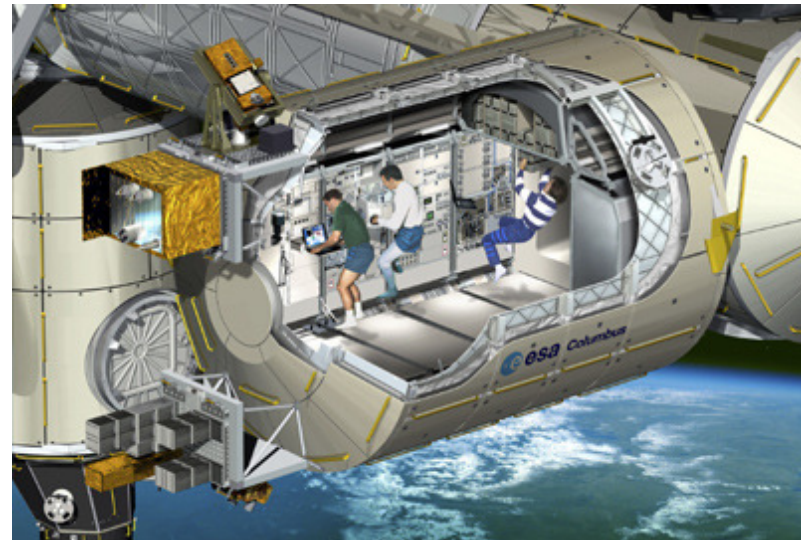
Bernard Fontaine

Introduction

- Why Toma
- Toma in 10 minutes
- Conclusion

Why Toma

- Early version of Toma has been used in a project for ESA – Q/A system for astronauts on board the ISS.
- Toma has been further developed because of the development of **TopiWriter**.



Columbus – Europe's laboratory on the International Space Station (Courtesy of ESA)

Toma Origin

Toma was influenced by:

- SQL
- Ideas from OO notation
- Tolog
- Astma*

Toma Characteristics

- Very powerful path expression syntax
- Similar to SQL → easy to learn
- Not only for querying, but also for manipulating Topic Maps

⇒ Toma is “all-in-one” TM*L:

TMQL, TMCL and TMML

- Toma is implemented: TopiEngine

Toma in 10 Minutes

Topic Literals

Topic Literals:

```
id( 'host-location' )  
bn( 'Processor' )  
var( 'CPU' )  
al( 'CPU' )  
si( 'http://www.topicmap.org/' )
```

- Naked Literals

```
$topic.bn@en # equivalent to $topic.bn@id( 'en' )  
$topic.oc(location) # $topic.oc(id( 'location' ))
```

Toma Variables

- Variable notation: `$var1`
- Variable types
 - topic
 - subjectIdentity
 - baseName
 - variant
 - occurrence
 - association
- Anonymous variables: `$$`

Toma Path Expression

- Chained with dot.
- Input – to the left of the path expression.
- Output – is the result of the expression.
- Result – textual representation of the output.

`$topic.bn.var`

The path expressions:

`id, bn, si, oc, var, tr, sir, rr, sc, player, role,
al, reify, type, instance, sub(), super()`

Associations

```
association_id(association_type) -> role
```

```
  $a (host-location) -> host = $h
```

```
and $a (host-location) -> location = $l
```

Associations: chaining

```
.role1<-association_id(association_type)->role2
```

```
id(`finger`) .$r1<-(connect_to)->$r2  
                .$r3<-(connect_to)->$r4 = $topic
```

```
id(`finger`) .$<-(connect_to)->$  
                .$<-(connect_to)->$ = $topic
```

Square brackets - Range

- **specify one item out of sequence**

```
$topic.bn['central processing unit'];
```

- **select part out of sequence**

```
select $topic.bn[$bn]  
where $topic.id = 'foo' and $bn ~ '^a';
```

- **access intermediate players**

```
select $p1  
where id('foo') . $$ <- ($at1) -> $$ [$p1]  
      . $$ <- ($at2) -> $$ = 'bar';
```

Brackets in Path Expressions

- specify types

<code>.bn (type)</code>	<code>\$topic.bn (abbreviation)</code>
<code>.oc (type)</code>	<code>\$topic.oc (description)</code>
<code>assoc_id (type) ->role</code>	<code>\$a (part-whole) ->part</code>

- group expressions and control precedence

```
($association_class.instance(1)) ->  
  (role_class.instance(1)).bn
```

Scope - @

baseName	<code>.bn@scope</code> <code>.bn (type) @scope</code>
variant	<code>.var@scope</code>
alias	<code>.al@scope</code>
occurrence	<code>.oc@scope</code> <code>.oc (type) @scope</code>
association	<code>assoc_id (assoc_type) @scope->role</code>

@ comes always right after brackets that indicate type

The USE statement

- Specifies Topic Map to be queried or manipulated.

```
USE topic_map_path;
```

```
use ` .db/computers.db' ;
```

```
use file:///usr/local/tm/db/computers.db;
```

- Applicable until another definition is done.

The SELECT statement

```
SELECT  [ ALL | DISTINCT ] navigation_list
        [ WHERE formula ]
        [ { UNION | INTERSECT | EXCEPT} [ ALL ]
          other_select ]
        [ ORDER BY expr1 [ ASC | DESC ]
          [, expr2 [ ASC | DESC ] ... ] ]
        [ LIMIT integer ]
        [ OFFSET integer ];
```

The SELECT statement

- Define projections over values of variables

```
select $topic, $topic.id where $topic.al = 'CPU';
```

```
  $topic      | $topic.id  
-----+-----  
  processor | processor  
(1 row)
```

```
select $topic.bn where $topic.bn = 'lung';
```

```
  $topic.bn  
-----  
  lung  
  long  
(2 rows)
```

The SELECT statement

- Use for translation between baseNames

```
select $topic.bn@dutch where $topic.bn@english = 'lung' ;  
      $topic.bn@dutch
```

```
long  
(1 row)
```

- Introduction of new variable

```
select $topic.bn@$scope, $scope.id where $topic.bn='lung' ;  
      $topic.bn | $scope.id
```

-----+-----

```
lung      | english  
long      | dutch  
(2 rows)
```

The INSERT statement

```
INSERT value INTO simple_path_expr  
[, value INTO simple_path_expr2 [ ...] ];
```

- Simple Path Expression:
 - No variable is allowed.
 - Only literals can follow the @ sign.
 - Only literals can be placed in type brackets.
 - Only right-arrow association expressions.
- simple_path_expr should be resolved to a not yet existing value.
- The concept of springing into existence.

The INSERT statement

```
insert 'http://en.wikipedia.org/wiki/Cpu'  
  into id('cpu').si.rr,  
      'CPU'  
  into id('cpu').(bn@long)['central processing unit'].var@short,  
      'The processor processes all the instructions.'  
  into $topic.oc(description)@textual;
```

creates the topic:

```
<topic id='cpu'>  
  <instanceOf><topicRef xlink:href="#processing_part"/></instanceOf>  
  <subjectIdentity><resourceRef xlink:href="http://en.wikipedia.org/wiki/Cpu"/></subjectIdentity>  
  <baseName>  
    <baseNameString>central processing unit</baseNameString>  
    <scope><topicRef xlink:href="#long"/></scope>  
    <variant>  
      <parameters><topicRef xlink:href="#short"/></parameters>  
      <resourceData>CPU</resourceData>  
    </variant>  
  </baseName>  
  <occurrence>  
    <resourceData>The processor is the device that processes all instructions.</resourceData>  
    <instanceOf><topicRef xlink:href="description"/></instanceOf>  
    <scope><topicRef xlink:href="textual"/></scope>  
  </occurrence>  
</topic>
```

The INSERT statement

- If the statement contains association path expressions, they are assumed to belong to the very same association.

```
insert `adapter`  
  into (provider-provided-receiver) ->provider,  
      `laptop`  
  into (provider-provided-receiver) ->receiver,  
      `electricity220`  
  into (provider-provided-receiver) ->provided;
```

The UPDATE statement

```
UPDATE expression1 = string [ WHERE formula];
```

- Update the values of Topic Map elements
- Only one variable is allowed

```
update id( 'processor' ) .bn[ 'processor' ]  
      = 'Processor' ;
```

The DELETE statement

```
DELETE expression WHERE [formula];
```

- Delete Topic Map elements
- Only one variable is allowed

```
delete $topic.oc(mass) .sc[ 'textual' ]  
  where $topic.type.super(*) = 'device' ;
```

Constraints

```
DEFINE CONSTRAINT identifier
  EACH TOPIC | ASSOCIATION variable
  [ WHERE formula1 ]
  SATISFIES formula2;
```

- Unique name: identifier.
- Cannot be redefined.
- Broken if formula2 is false.

Constraints

```
# each topic must have a base name
define constraint basename_constraint
  each topic $topic
    satisfies exists $topic.bn;
```

```
# each topic of type 'device' must have occurrences of
# type 'mass' and 'description' and it must play the
# role 'host' in a 'host-location' association.
define constraint device_constraint
  each topic $topic
    where $topic.type.id = 'device'
    satisfies exists $topic.oc(mass) and
      exists $topic.oc(description) and
      (host-location)->host = $topic;
```

Toma Functions

String Functions

||

`LOWERCASE(string)`

`UPPERCASE(string)`

`TITLECASE(string)`

`LENGTH(string)`

`SUBSTR(string,
 from,
 [length])`

`TRIM(string,
 [LEADING | TRAILING | BOTH],
 [characters])`

Conversion Functions

`TO_NUM(text)`

Converts text to a number if possible

Example:

“3.4 kg” will be converted to 3.4

`TO_UNIT(text, target_unit)`

Converts between units

Example:

```
select to_unit($topic.oc(mass), 'kg')  
      where $topic.bn = 'parcel';
```

Toma Aggregation Functions

`COUNT (expression)`

`SUM (expression)`

`MAX (expression)`

`MIN (expression)`

`AVG (expression)`

`CONCAT (expression, [string])`

```
select concat(id('mlu').al, ', ');
```

```
# returns: MLU, module lighting unit
```

- NULL values is valuated as zero

Merging

```
use 't/db/columbus-epds.db';  
merge with file:./.db/columbus-msm.db  
mark columbus-msm;
```

```
use columbus-epds;  
merge XTM <<EOF  
  <topicMap id="only-mlu">  
    <topic id="mlu">  
      <instanceOf>  
        <topicRef xlink:href="#device"/>  
      </instanceOf>  
      <baseName>  
        <baseNameString>module lighting  
          unit</baseNameString>  
      </baseName>  
    </topic>  
  </topicMap>  
EOF;
```

The EXPORT statement

```
EXPORT [ TO file_path ] [AS XTM];
```

- Export Topic Maps as a Topic Map representation (such as XTM).
- AS clause allows alternative formats.

Conclusions

- Very powerful path expression syntax
 - Similar to SQL → easy to learn
 - Not only for querying, but also for manipulating Topic Maps
- ⇒ Toma is “all-in-one” TM*L language:
TMQL, TMCL and TMML
- Toma is implemented: TopiEngine

Questions?