

# TMCL-Grammar Introduction

TMCL-Grammar is a proposal for a “grammar” part of TMCL-Schema

- TMCL-Grammar - why ?
- Generate and constrain paradigm
- Type definition
- Association definition
- Extending TMCL-Grammar with additional constraints
- Summary

## TMCL-Grammar - Why?

- TMCL-Schema should be based on something more basic
- Current TMCL-Model covers “rule” part of TMCL-Schema
- TMCL-Grammar is an experimental grammar part of a basic model
- TMCL-Grammar defines what is “reasonable to ask” about topics or associations
- TMCL-Schema can be represented as TMCL-Grammar + TMCL-Model

## Generate and constrain

- Grammar definitions describe what is "reasonable to ask" about topics of specific types or associations. They play a role of a “generator”.
- Grammar definitions consist of pattern expressions.
- Grammar definitions are "minimum" constraints.
- At this level there is nothing about cardinality, types of role players or other constraints.
- Other constraints can be added "later".
- Grammar pattern is different from logical “exists” quantifier (TM constructs which match pattern can be prohibited with additional constraints)
- We have well defined concept of merging grammar-based definitions.

## Topic Type Definition

Topic Type Definition lists what kind of TM constructs is reasonable to expect for topics of type we are defining

Example:

```
define type composer
{
  subjectIndicator,
  sourceLocator,
  baseName,
  //it is possible to use type variable for scope and "|" operator - (fr | en)
  baseName@%language%
  in::born,
  in::died,
  oc::homePage,
  //alt defines named alternative, merging procedure uses alt names
  alt::bornInCityOrVillage {a::born-in(who,city) | a::born-in(who,village)},
  a::died-in(who,where),
  a::wrote-opera(who,opera)
}
```

## Topic Type Definition: Note 1

Any two Topic Type Definitions for the same type can be merged by combining definitions and deleting possible redundancy.

Example:

```
define type composer
```

```
{
```

```
  subjectIndicator,sourceLocator,baseName,
```

```
  in::born,in::died
```

```
}
```

```
define type composer
```

```
{
```

```
  subjectIndicator,sourceLocator,baseName,
```

```
  a::born-in(who,where), a::died-in(who,where),a::wrote-opera(who,opera)
```

```
}
```

will be merged into

```
define type composer
```

```
{
```

```
  subjectIndicator,sourceLocator,baseName,in::born,in::died,
```

```
  a::born-in(who,where), a::died-in(who,where),a::wrote-opera(who,opera)
```

```
}
```

## Topic Type Definition: Note 2

Occurrences, Names and Associations can have attached scope expressions.

Example:

```
define type document
{
  subjectLocator,
  subjectIndicator,
  sourceLocator,
  baseName,
  baseName@%language%
}
```

Scope expression can include:

- topic (example: en),
- topic alternatives (example: en | fr),
- type topic (example: %language%)

## Topic Type Definition: Note 3

Merging of Topic Type Definitions respects scope expressions in patterns.

Example:

```
define type person {baseName}  
define type person {baseName, baseName@%language}  
define type person {baseName@(en | fr)}
```

will be merged into

```
define type person  
{  
  baseName,  
  baseName@%language,  
  baseName@(en | fr)  
}
```

## Topic Type Definition: Note 4

It is possible to have associations with different "signatures" in the same topic type definition.

Example:

//We do not use 'named alt' construct here because both assertions are possible at the same time

```
define type opera
{
  a::first-performed-at(opera,theater),
  a::first-performed-at(opera,city)
}
```

## Topic Type Definition: Note 5

Type definitions can be propagated through subtype-supertype association

Example:

If we have assertion:

```
subtype-supertype(person : supertype, composer : subtype)
```

and definitions:

```
define type person
{
  subjectIndicator,sourceLocator,baseName,in::born,in::died,oc::homePage,
  a::born-in(who,city),a::born-in(who,village),a::died-in(who,where)
}
define type composer {a::wrote-opera(who,opera)}
```

Then merging (and sub-super-type inheritance) will produce definition:

```
define type composer
{
  subjectIndicator,sourceLocator,baseName,in::born,in::died,oc::homePage,
  a::born-in(who,city),a::born-in(who,village),a::died-in(who,where),
  a::wrote-opera(who,opera)
}
```

# Association Definitions

Association definition lists alternative signatures for an association.

Example 1 (two alternative signatures):

```
define association born-in
{
  (who,city) |
  (who,village)
}
```

Example 2 (only one possible signature):

```
define association died-in
{
  (who,where)
}
```

## Combining TMCL-Grammar with additional constraints

Additional constraints can be added to patterns defined by TMCL-Grammar.

Example:

```
define type composer
{
  //firstly, we define pattern
  a::wrote-opera(who,opera)
  {
    //and we add constraints for constructs which match this pattern
    cardMin=0;
    cardMax=unbounded;
    otherRole opera
    {
      cardMin=1;
      cardMax=1;
      allPlayersFrom=opera
    }
  }
}
```

## Translating TMCL-Grammar definitions into TMCL-Rule

TMCL-Grammar definitions can be converted into TMCL-Rule expressions for validation purposes.

Example:

```
define type person
{
  born-in(who,where)
}
```

can be represented by a TMCL-Rule as:

```
rule "person-1"
{
  forEvery {$X}
  where {$X : person}
  expect
  {
    born-in($X : who, _ : where)
  }
}
```

## Interpretation of 'Expect' construct

When TMCL-Rule engine evaluates 'expect' construct it tags all TM constructs which satisfy pattern as 'expected'

The same TM construct can be 'expected' by multiple patterns

When all rules are applied TMCL-Rule engine can identify “what is expected and what is not” in topic map by checking 'expected' tag.

## Summary

- TMCL-Grammar is a pattern-based experimental language for defining basic (“what is reasonable to ask”) constraints.
- TMCL-Grammar can be combined with additional constraints to support TMCL-Schema