ISO/IEC JTC 1/SC 34

Date: 2005-2-12

ISO/IEC CD 19756

ISO/IEC JTC 1/SC 34/WG 3

Secretariat: SCC

Topic Maps Constraint Language

Document type: International Standard
Document subtype:
Document stage:
Document language: E

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

ISO/IEC CD 19756 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 34, Document Description and Processing Languages.

Introduction

This International Standard defines a means to express constraints on topic maps conforming to the Topic Map Data Model [TMDM].

Topic Maps Constraint Language

1 Scope

This International Standard is designed to allow users to constrain any aspect of the topic map data model. TMCL adopts TMQL [TMQL] as a means to express both the topic map constructs to be constrained and topic map structures that must exist in order for the constraint to be met.

This International Standard defines TMCL-Schema and TMCL-Rule. TMCL-Schema provides a type based model of constraints. TMCL-Rule provides a generlised model of constraint based on TMQL.

This International Standard defines a formal language by providing a syntax to form constraints. The document also defines an informal and a formal semantics for every syntactic form, including rules for the reporting of error conditions.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE Each of the following documents has a unique identifier that is used to cite the document in the text. The unique identifier consists of the part of the reference up to the first comma.

Unicode, *The Unicode Standard, Version 3.0*, The Unicode Consortium, Reading, Massachusetts, USA, Addison-Wesley Developer's Press, 2000, ISBN 0-201-61633-5

TMDM, *ISO 13250-2 Topic Maps — Data Model*, ISO, 2005, available at <http://www.isotopicmaps.org/sam/sam-model/>

TMQL, *ISO Topic Maps Query Language Working Draft*, ISO, 2005, available at <http://www.isotopicmaps.org/tmql/>

XML 1.0, *Extensible Markup Language (XML) 1.0*, W3C, Third Edition, W3C Recommendation, 04 February 2004, available at <http://www.w3.org/TR/REC-xml/>

RFC2396, *RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax*, The Internet Society, 1998, available at <http://www.ietf.org/rfc/rfc2396.txt>

3 Notation and Conventions

3.1 Notation and Syntax

The syntax of TMQL is defined using the EBNF formalism defined in [XML 1.0].

3.2 Informal and Formal Semantics

4 TMCL

TMCL defines TMCL-Schema and TMCL-Rule. TMCL-Schema provides a type based model of constraints.

TMCL-Rule provides a generlised model of constraint based on TMQL.

Both TMCL-Rule and TMCL-Schema define sets of constraints. These constraints consist of terms that identify parts of the Topic Map to be constrained and terms that define the predicate that must hold for the Topic Map to be considered to be consistent.

TMCL-Schema and TMCL-Rule use different sub-languages for describing constraints. TMCL-Schema defines grammar like structures for describing constraints while TMCL-Rule defines constraints in a form of "if ... then ..." rules. Both sub-languages can be combined in schema expressions using specified syntax.

4.1 Validation Semantics

TMCL-Rule and TMCL-Schema are used to constrain instances of the Topic Map Data Model. If the topic map is valid in respect to the constraints being tested then validation is said to have succeeded. More formally it can be said that :

```
Given:
  TopicMap: tm1
  Schema  : sc1
Then:
  Validate(tm1, sc1) => (true, notifyItem*)  | (false, conflictItem+, notifyItem*)
```

The Validate function is defined as follows:
Evaluates the 'selector' part of the constraint. This results in one or more bindings of topic map information items to variables.

For each value of each bound variable in the selector that also occurs in the constrainer expression, the constrainer expression is evaluated.

If the constrainer expression returns an empty result set, i.e., no matches were found, then the topic map data model instance does not meet the constraint.

The process is repeated for each constraint in the schema. The topic map is valid with respect to the schema if all constraints are valid.

4.2 TMCL-Schema

TMCL_Schema defines a language for constraining classes of topics and associations.

TMCL-Schema expressions effectively combine shared context for multiple constraints and allow compact representations of these constraints. However, from logical perspective each TMCL-Schema is equivalent to a set of constraints which can be validated independently

4.2.1 Topic Map Schema

Used to group together a collection of constraints.

```
TopicMapSchema:
  SchemaID              # ID which identifies this schema
  Name?                 # defines schema Name
  Include *             # URI -  list of schemas to include
  TopicSchema *         # TopicSchema
  AssociationSchema *   # AssociationSchema
```

4.2.2 Topic Identification

Topic Identification is used to identify exactly 1 topic.

```
TopicIdentification:
  srcLocators *       # URI
  subjectIndicator *  # URI
  subjectAddress *    # URI
```

4.2.3 Set of topic identification constructs

```
TopicSet:
  topicIdentification * # TopicIdentification
```

4.2.4 Scope pattern

```
ScopePattern:
  simpleTopicExpression *   # TopicIdentificaiton
  orTopicExpression *       # OrTopicExpression
  typeTopicExpression *     # TopicIdentificaiton
```

4.2.5 Or Topic Expression

```
OrTopicExpression:
  topicIdentification *     # TopicIdentificaiton
```

4.2.6 Topic Schema

```
TopicSchema:
  schemaID                  # ID which identifies this partial type description
  type                      # TopicIdentification
  subjectAddressSchema *        # SubjectAddressSchema
  subjectIndicatorSchema *      # SubjectIndicatorSchema
  baseNameSchema *              # BaseNameSchema
  internalOccurrenceSchema *    # InternalOccurrenceSchem
  externalOccurrenceSchema *    # ExternalOccurrenceSchema
  playRoleSchema *              # PlayRoleSchema
  oneOfSchema?                  # OneOfSchema
```

4.2.7 Subject Indicator Schema

Constrains the cardinality and shape of subject indicator locator.

```
SubjectIndicatorSchema:
  cardMin          # Integer?
  cardMax          # Integer?
  match            # Regular Expression*
```

4.2.8 Subject Address Schema

Constrains the cardinality and shape of subject address locator.

```
SubjectAddressSchema:
  cardMin          # Integer?
  cardMax          # Integer?
  match            # Regular Expression?
```

4.2.9 Base Name Schema

Constrains topic names.

```
BaseNameSchema:
  type             # TopicIdentification
  scope            # ScopePattern?
  cardMin          # Integer?
  cardMax          # Integer?
  dataType         # xsd and custom xml schemas?
  oneOf            # String*
  match            # Regular Expression*
  variantSchema    # VariantSchema*
```

4.2.10 Variant Schema

Constrains on variants.

```
VariantSchema:
  scope            # ScopePattern?
  dataType         # xsd and custom xml schemas?
  cardMin          # Integer?
  cardMax          # Integer?
  oneOf            # String*
  match            # Regular Expression*
```

Note: dataType for variants is part of a pattern, not a constraint

4.2.11 Internal Occurrence Schema

Constrains internal occurrences.

```
InternalOccurrenceSchema:
  type             # TopicIdentification
  scope            # ScopePattern?
  cardMin          # Integer?
  cardMax          # Integer?
  dataType         # xsd and custom schemas?
  oneOf            # String*
  match            # Regular Expression *
  minExclusive     # String?
  minInclusive     # String?
  maxExclusive     # String?
  maxInclusive     # String?
```

4.2.12 External Occurrence Schema

Constrains external occurrences.

```
ExternalOccurrenceSchema:
  type            # TopicIdentification
  scope           # ScopePattern?
  cardMin         # Integer?
  cardMax         # Integer?
  oneOf           # URI*
  match           # Regular Expression *
```

4.2.13 Role Schema

Constraints the nature of roles on associations of specific types.

```
RoleSchema:
  roleType        # TopicIdentification
  cardMin         # Integer
  cardMax         # Integer
  allPlayersFrom    # TopicSet  // list of Types
  oneOf           # TopicSet  // list of topics
```

4.2.14 Other Player Schema

Constraints the nature of other players of the association

```
OtherPlayerSchema:
  cardMin         # Integer
  cardMax         # Integer
  allPlayersFrom    # TopicSet  // list of Types
  oneOf           # TopicSet  // list of topics
```

4.2.15 Plays Role Schema

Constraints the nature of participation in associations.

```
PlayRoleSchema:
  associationType   # TopicIdentification
  roleType        # TopicIdentification
  otherRoleType     # TopicSet    //association type, roleType and otherRoleTypes define signature
  scope           # ScopePattern
  cardMin         # Integer
  cardMax         # Integer
  otherRoles      # RoleSchema*
  otherPlayers      # OtherPlayerSchema*
```

4.2.16 One Of Schema

One of is used to defined a controlled vocabulary.

```
OneOfSchema:
  oneOf           # TopicSet
```

4.2.17 Association Schema

Constrains classes of association.

```
AssociationSchema:
  schemaID              # ID which identifies this partial association type description
  type                  # TopicIdentification
  associationSignature  # AssociationSignatureSchema+
```

[db] Note: It is possible to have different signatures with associations of the same type. It allows to use 'generalized' associations

4.2.18 Association Signature Schema

Constrains specific association signature

```
AssociationSignatureSchema:
  signature             # TopicSet
  roleSchema            # RoleSchema+
```

4.2.19 TMCL-Schema decriptions of warnings and notifications

Each constraint has "warning" or "notification" equivalent. Warnings typically define more detailed constraints.

```
cardMin          -> cardMinNotify
cardMax          -> cardMaxNotify
dataType         -> dataTypeNotify
oneOf            -> oneOfNotify
match            -> matchNotify
minExclusive     -> minExclusiveNotify
minInclusive     -> minInclusiveNotify
maxExclusive     -> maxExclusiveNotify
maxInclusive     -> maxInclusiveNotify
allPlayersFrom   -> allPlayersFromNotify
oneOf            -> oneOfNotify
```

If Notify constraint is violated then SchemaNotifyItem is generated during validation process

4.2.20 Syntax for TMCL-Schema

Relax NG Schema

TODO: allow mixing TMCL-rules

```
grammar{
   start = element-TopicMapSchema

   element-TopicMapSchema = element TopicMapSchema{
       element-SchemaID,
       element-Name?,
       element-Includes?,
       element-TopicSchemas?,
       element-AssociationSchemas?,
   }

   element-SchemaID = element SchemaID {text}
   element-Name= element Name {text}
```

```
 element-Includes = element Includes {element-Include+}
 element-TopicSchemas = element TopicSchemas {element-TopicSchema+}
 element-AaasociationSchemas = element AssociationSchemas {element-AssociationSchema+}

 element-Include = element Include {text}

 element-TopicSchema = element TopicSchema{
     element-SchemaID,
     element-Type,
     element-SubjectAddressSchemas ?,
     element-SubjectIndicatorSchemas ?,
     element-BaseNameSchemas ?,
     element-InternalOccurrenceSchemas ?,
     element-ExternalOccurrenceSchemas ?
     element-PlayRoleSchemas ?
     element-OneOfSchema ?
 }

 element-Type = element Type {TopicIdentification-Content}

 TopicIdentification-Content =
    element srcLocators {element srcLocator {text}+}?,
    element subjectIndicators {element subjectIndicator {text}+}?,
    element subjectAddresses {element subjectAddress {text}+}?

element-SubjectAddressSchemas = element SubjectAddressSchemas {element-SubjectAddressSchema+}

element-SubjectAddressSchema = {
   element-CardMin ?,
   element-CardMax ?,
   element-Match ?
   element-CardMinNotify ?,
   element-CardMaxNotify ?,
   element-MatchNotify ?
}

element-SubjectIndicatorSchemas = element SubjectIndicatorSchema {element-SubjectIndicatorSchema+}

element-SubjectIndicatorSchema = {
   element-CardMin?,
   element-CardMax?,
   element-Match?,
   element-CardMinNotify?,
   element-CardMaxNotify?,
   element-MatchNotify?
}

element-BaseNameSchemas = element BaseNames {element-BaseNameSchema+}

element-BaseNameSchema = element BaseNameSchema{
   element-Type,
   element-ScopePattern ?,
   element-CardMin?,
   element-CardMax?,
   element-DataType?,
   element-OneOf?,
   element-Match?,
   element-VariantSchemas?,
   element-CardMinNotify?,
   element-CardMaxNotify?,
   element-DataTypeNotify?,
   element-OneOfNotify?,
   element-MatchNotify?
}

element-VariantSchemas = element Variants {element-Variant+}
```

```
element-Variant = {
    element-ScopePattern ?,
    element-CardMin?,
    element-CardMax?,
    element-DataType?,
    element-OneOf?,
    element-Match?,
    element-VariantSchemas?,
    element-CardMinNotify?,
    element-CardMaxNotify?,
    element-DataTypeNotify?,
    element-OneOfNotify?,
    element-MatchNotify?
}

element-InternalOccurrenceSchemas = element InternalOccurrenceSchemas{element-InternalOccurrenceSchema+}

element-InternalOccurrenceSchema = {
    element-Type,
    element-ScopePattern?,
    element-CardMin?,
    element-CardMax?,
    element-DataType?,
    element-OneOf?,
    element-Match?,
    element-MinExclusive?,
    element-MinInclusive?,
    element-MaxExclusive?,
    element-MaxInclusive?,
    element-CardMinNotify?,
    element-CardMaxNotify?,
    element-DataTypeNotify?,
    element-OneOfNotify?,
    element-MatchNotify?,
    element-MinExclusiveNotify?,
    element-MinInclusiveNotify?,
    element-MaxExclusiveNotify?,
    element-MaxInclusiveNotify?
}

element-ExternalOccurrenceSchemas = element ExternalOccurrenceSchemas{element-ExternalOccurrenceSchema+}

element-ExternalOccurrenceSchema = {
    element-Type,
    element-ScopePattern?,
    element-CardMin?,
    element-CardMax?,
    element-DataType?,
    element-OneOf?,
    element-Match?,
    element-CardMinNotify?,
    element-CardMaxNotify?,
    element-DataTypeNotify?,
    element-OneOfNotify?,
    element-MatchNotify?
}

element-PlayRoleSchemas = element PlayRoleSchemas { element-PlayRoleSchema+ }

element-PlayRoleSchema ={
    element-AssociationType,
    element-RoleType,
    element-otherRoleTypes?,
    element-ScopePattern?,
    element-cardMin?,
```

```
    element-cardMax?,
    element-otherRoles?,
    element-otherPlayers?
    element-CardMinNotify?,
    element-CardMaxNotify?,
}

element-AssociationType = element AssociationType {TopicIdentification-Content}

element-RoleType = element RoleType {TopicIdentification-Content}

element-otherRoleTypes = element OtherRoleTypes {element-RoleType+}

element-otherRoles = element OtherRoleSchemas {element-RoleSchema+}

element-otherPlayers = element OtherPlayerSchemas {element-PlayerSchema+}

element-RoleSchema = element RoleSchema{
    element-RoleType,
    element-cardMin?,
    element-cardMax?,
    element-AllPlayersFrom?,
    element-OneOfTopics?
    element-cardMinNotify?,
    element-cardMaxNotify?,
    element-AllPlayersFromNotify?,
    element-OneOfTopicsNotify?
}

element-PLayerSchema = element PlayerSchema{
    element-cardMin?,
    element-cardMax?,
    element-AllPlayersFrom?,
    element-OneOfTopics?
    element-cardMinNotify?,
    element-cardMaxNotify?,
    element-AllPlayersFromNotify?,
    element-OneOfTopicsNotify?
}

element-AllPlayersFrom = element AllPlayersFrom {element Type{TopicIdentification-Content}+}

element-OneOfTopics = element OneOfTopics {element Topic{TopicIdentification-Content}+}

element-AllPlayersFromNotify = element AllPlayersFromNotify {element Type{TopicIdentification-Content}+}

element-OneOfTopicsNotify = element OneOfTopicsNotify {element Topic{TopicIdentification-Content}+}

element-OneOfSchema = element OneOfSchema {element-OneOfTopics}


element-CardMin = element CardMin{text}

element-CardMax = element CardMax{text}

element-Match = element Match{ element Pattern{text}+}

element-MinExclusive = element MinExclusive{text}

element-MinInclusive = element MinInclusive{text}

element-MaxExclusive = element MaxExclusive{text}

element-MaxInclusive = element MaxInclusive{text}

element-CardMinNotify = element CardMinNotify{text}
```

```
element-CardMaxNotify = element CardMaxNotify{text}

element-MatchNotify = element MatchNotify{ element Pattern{text}+}


element-AssociationSchemas = element AssociationSchemas {element-AssociationSchema+}

element-AssociationSchema = element AssociationSchema{
   element-SchemaID,
   element-Type,
   element-associationSignatures?
}

element-associationSignatures = element AssociationSignatures{element-AssociationSignature}

element-associationSignature = element AssociationSignature{
   element-Signature,
   element-Roles?
}

 element-Roles = element RoleSchemas {element-RoleSchema+}

 element-Signature = element Signature{element-RoleType+}

 element-ScopePattern = element ScopePattern{
     element-SimpleTopicExpression?,
     element-OrTopicExpression?,
     element-TypeTopicExpression?
 }

 element-SimpleTopicExpression = element Topics { element Topic {TopicIdentification-Content}+}

 element-SimpleTopicExpression = element OrTopicsSchema {
     element OrTopics{ element Topic {TopicIdentification-Content}+}+
 }

 element-TypeTopicExpression = element Types { element Type {TopicIdentification-Content}+}

}
```

4.2.21 TMCL-Schema Introspection

TMCL-Schema defines set of predicates which allows schema introspection. These predicates are compbined into TMQL module "TMCL".

PSI for TMCL module: http://www.isotopicmaps.org/tmcl/tmcl.html#TMCL

```
tmcl:TopicMapSchema($X)
tmcl:SchemaID($Schema, $ID)
tmcl:SchemaName($Schema,$Name)
tmcl:Include($Schema,$IncludedSchema)
tmcl:TopicSchema($TMSchema,$TopicSchema)
tmcl:AssociationSchema($TMSchema,$AssociationSchema)
tmcl:SubjectAddressSchema($TopicSchema,$SubjectAddressSchema)
tmcl:SubjectIndicatorSchemas($TopicSchema,$SubjectIndicatorSchema)
tmcl:BaseNameSchema($TopicSchema,$BaseNameSchema)
tmcl:InternalOccurrenceSchema($TopicSchema,$InternalOccurrenceSchema)
tmcl:ExternalOccurrenceSchema($TopicSchema,$ExternalOccurrenceSchema)
tmcl:PlayRoleSchema($TopicSchema,$PlayRoleSchema)
tmcl:OneOfSchema($TopicSchema,$OneOfSchema)
tmcl:Type($Schema,$Topic)
tmcl:CardMin($Schema,$CardMin)
```

tmcl:CardMax($Schema,$CardMax)
tmcl:Match($Schema,$Match)
tmcl:CardMinNotify($Schema,$CardMinNotify)
tmcl:CardMaxNotify($Schema,$CardMaxNotify)
tmcl:MatchNotify($Schema,$MatchNotify)
tmcl:MinExclusive($Schema,$MinExclusive)
tmcl:MinInclusive($Schema,$MinInclusive)
tmcl:MinInclusive($Schema,$MinInclusive)
tmcl:MaxInclusive($Schema,$MaxInclusive)
tmcl:MinExclusiveNotify($Schema,$MinExclusiveNotify)
tmcl:MinInclusiveNotify($Schema,$MinInclusiveNotify)
tmcl:MinInclusiveNotify($Schema,$MinInclusiveNotify)
tmcl:MaxInclusiveNotify($Schema,$MaxInclusiveNotify)
tmcl:DataType($Schema,$DataType)
tmcl:OneOfValue($Schema,$OneOfvalue),
tmcl:DataTypeNotify($Schema,$DataTypeNotify),
tmcl:OneOfvalueNotify($Schema,$OneOfvalueNotify),
tmcl:VariantSchemas($BaseNameSchema,$VariantSchema)
tmcl:ScopePattern($Schema,$ScopePatternt)
tmcl:SimpleTopicExpression($Schema,$Topic)
tmcl:OrTopicExpression($Schema,$Topic)
tmcl:TypeTopicExpression($Schema,$Topic)
tmcl:AllPlayersFrom($Schema,$Topic)
tmcl:oneOfTopic($Schema,$Topic)
tmcl:OtherRoleType($Schema,$Topic)
tmcl:OtherRole($Schema,$RoleSchema)
tmcl:OtherPlayer($Schema,$PlayerSchema)
tmcl:AssociationSignature($Schema,$AssociationSignature)
tmcl:Signature($Schema,$Topic)
tmcl:RoleSchema($Schema,$RoleSchema)

4.2.22 Interpretation of TMCL-Schema

Each TMCL-Schema constraint type is defined through logical expressions. These definitions allow to produce corresponding set of constraints for any given TMCL-Schema document. Definition of each possible TMCL-Schema constraint type is provided below.

SubjectAddressSchemaCardMin

TopicSchema:
  type =$T
  subjectAddressSchema:
    cardMin =$N

Every $Ex SuchAs instanceOf($Ex,$T) Satisfies
  ExistsAtLeast $N $Y SuchAs subjectAddress($Ex,$Y)

SubjectAddressSchemaCardMax

TopicSchema:
  type =$T
  subjectAddressSchema:
    cardMax =$N

Every $Ex SuchAs instanceOf($Ex,$T) Satisfies
  ExistsAtMost $N $Y SuchAs subjectAddress($Ex,$Y)

SubjectAddressSchemaMatch

```
TopicSchema:
  type                  =$T
  subjectAddressSchema:
     match              =$P

Every $Ex SuchAs instanceOf($Ex,$T) Satisfies
   Every $Y SuchAs subjectAddress($Ex,$Y) Satisfies
      match($Y,$P)
```

SubjectIndicatorSchemaCardMin

```
TopicSchema:
  type                  =$T
  subjectIndicatorSchema:
     cardMin            =$N

Every $Ex SuchAs instanceOf($Ex,$T) Satisfies
   ExistsAtLeast $N $Y SuchAs subjectIndicator($Ex,$Y)
```

SubjectIndicatorSchemaCardMax

```
TopicSchema:
  type                  =$T
  subjectAddressSchema:
     cardMax            =$N

Every $Ex SuchAs instanceOf($Ex,$T) Satisfies
   ExistsAtMost $N $Y SuchAs subjectIndicator($Ex,$Y)
```

SubjectIndicatorSchemaMatch

```
TopicSchema:
  type                  =$Type
  subjectIndicatorSchema:
     match              =$RegExprSet

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $S SuchAs subjectIndicator($Inst,$S) Satisfies
      Every $Y SuchAs member($RegExprSet,$Y)
        match($Inst,$Y)
```

BaseNameCardMin

```
TopicSchema:
  type              =$Type
    baseNameSchema:
      type          =$NameType
      scope         =$ScopePattern
      cardMin       =$N
```

```
Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  ExistsAtLeast $N $BN SuchAs baseName($Inst,$NameType,$BN)@$Scope and
              matchScope($Scope,$ScopePattern)
```

BaseNameCardMax

```
TopicSchema:
  type              =$Type
    baseNameSchema:
      type          =$NameType
      scope         =$ScopePattern
      cardMax       =$N

  Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
    ExistsAtMost $N $BN SuchAs baseName($Inst,$NameType,$BN)@$Scope and
              matchScope($Scope,$ScopePattern)
```

BaseNameDataType

```
TopicSchema:
  Type              =$Type
    BaseNameSchema:
      type          =$NameType
      scope         =$ScopePattern
      datatype      =$DataType

  Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
    Every $BN  SuchAs baseName($Inst,$NameType,$BN)@$Scope and
              matchScope($Scope,$ScopePattern) Satisfies
      matchDataType($BN,$DataType)
```

BaseNameOneOf

```
TopicSchema:
  Type              =$Type
    BaseNameSchema:
      type          =$NameType
      scope         =$ScopePattern
      oneOf         =$StrSet

  Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
    Every $BN  SuchAs baseName($Inst,$NameType,$BN)@$Scope and
              matchScope($Scope,$ScopePattern) Satisfies
      member($BN,$StrSet)
```

BaseNameMatch

```
TopicSchema:
  Type              =$Type
    BaseNameSchema:
      type          =$NameType
```

```
        scope          =$ScopePattern
        match          =$RegExpSet

    Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
      Every $BN  SuchAs baseName($Inst,$NameType,$BN)@$Scope and
                matchScope($Scope,$ScopePattern) Satisfies
        Every $RegExp SuchAs member($RegExp, $RegExpSet) Satisfies
            match($BN,$RegExp)
```

BaseNameVariantCardMin

```
    TopicSchema:
      Type              =$Type
        BaseNameSchema:
          type          =$NameType
          scope         =$ScopePattern
          variantSchema:
            scope       =$VarScopePattern
            dataType    =$VarDataType
            cardMin     =$N


    Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
      Every $BN  SuchAs baseName($Inst,$NameType,$BN)@$Scope and
                matchScope($Scope,$ScopePattern)  Satisfies
        ExistsAtLeast $N,$Variant SuchAs variant($BN,$Variant)@VarScope and
                matchScope($VarScope,$VarScopePattern) and
                variantDataType($Variant,$VarDataType)
```

BaseNameVariantCardMax

```
    TopicSchema:
      Type              =$Type
        BaseNameSchema:
          type          =$NameType
          scope         =$ScopePattern
          variantSchema:
            scope       =$VarScopePattern
            dataType    =$VarDataType
            cardMax     =$N


    Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
      Every $BN  SuchAs baseName($Inst,$NameType,$BN)@$Scope and
                matchScope($Scope,$ScopePattern)  Satisfies
        ExistsAtMost $N,$Variant SuchAs variant($BN,$Variant)@VarScope and
                matchScope($VarScope,$VarScopePattern) and
                variantDataType($Variant,$VarDataType)
```

BaseNameVariantOneOf

```
    TopicSchema:
      Type              =$Type
        BaseNameSchema:
          type          =$NameType
```

```
        scope          =$ScopePattern
        variantSchema:
          scope        =$VarScopePattern
          dataType     =$VarDataType
          oneOf        =$StrSet


      Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
        Every $BN  SuchAs baseName($Inst,$NameType,$BN)@$Scope and
                  matchScope($Scope,$ScopePattern)  Satisfies
          Every $Variant,$VariantVal SuchAs variant($BN,$Variant)@VarScope and
                  matchScope($VarScope,$VarScopePattern) and
                  variantDataType($Variant,$VarDataType) and
                  variantValue($Variant,$VariantVal) Satisfies
          member($VariantVal,$StrSet)
```

BaseNameVariantMatch

```
      TopicSchema:
        Type               =$Type
          BaseNameSchema:
            type           =$NameType
            scope          =$ScopePattern
            variantSchema:
              scope        =$VarScopePattern
              dataType     =$VarDataType
              match        =$RegExpSet


      Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
        Every $BN  SuchAs baseName($Inst,$NameType,$BN)@$Scope and
                  matchScope($Scope,$ScopePattern)  Satisfies
          Every $Variant,$VariantVal SuchAs variant($BN,$Variant)@VarScope and
                  matchScope($VarScope,$VarScopePattern) and
                  variantDataType($Variant,$VarDataType) and
                  variantValue($Variant,$VariantVal) Satisfies
            Every $RegExp SuchAs member($RegExp, $RegExpSet) Satisfies
              match($BN,$VariantVal)
```

InternalOccurrenceCardMin

```
      TopicSchema:
        Type               =$Type
        internalOccurrenceSchema:
          type             =$OcType
          scope            =$ScopePattern
          cardMin          =$N


      Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
        ExistsAtLeast $N $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
          matchScope($Scope,$ScopePattern)
```

InternalOccurrenceCardMax

```
      TopicSchema:
```

```
 Type                  =$Type
 internalOccurrenceSchema:
    type               =$OcType
    scope              =$ScopePattern
    cardMax             =$N

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  ExistsAtMost $N $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
                        matchScope($Scope,$ScopePattern)
```

InternalOccurrenceDatatype

```
TopicSchema:
 Type                  =$Type
 internalOccurrenceSchema:
    type               =$OcType
    scope              =$ScopePattern
    datatype            =$Datatype


Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  Every $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
                matchScope($Scope,$ScopePattern) Satisfies
     matchDataType($OcVal,$DataType)
```

InternalOccurrenceOneOf

```
TopicSchema:
 Type                  =$Type
 internalOccurrenceSchema:
    type               =$OcType
    scope              =$ScopePattern
    oneOf               =$StrSet


Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  Every $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
                matchScope($Scope,$ScopePattern) Satisfies
     member($OcVal,$StrSet)
```

InternalOccurrenceMatch

```
TopicSchema:
 Type                  =$Type
 internalOccurrenceSchema:
    type               =$OcType
    scope              =$ScopePattern
    match               =$PatternSet


Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  Every $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
                matchScope($Scope,$ScopePattern) Satisfies
     Every $Pattern SuchAs member($Pattern,$PatternSet) Satisfies
        match($OcVal,$Pattern)
```

InternalOccurrenceMinExclusive

```
TopicSchema:
 Type                  =$Type
 internalOccurrenceSchema:
    type               =$OcType
    scope              =$ScopePattern
    match              =$PatternSet
    minExclusive       =$MinVal

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  Every $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
                matchScope($Scope,$ScopePattern) Satisfies
    $OcVal > $MinVal
```

InternalOccurrenceMinInclusive

```
TopicSchema:
 Type                  =$Type
 internalOccurrenceSchema:
    type               =$OcType
    scope              =$ScopePattern
    match              =$PatternSet
    minInclusive       =$MinVal

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  Every $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
                matchScope($Scope,$ScopePattern) Satisfies
    $OcVal >= $MinVal
```

InternalOccurrenceMaxExclusive

```
TopicSchema:
 Type                  =$Type
 internalOccurrenceSchema:
    type               =$OcType
    scope              =$ScopePattern
    match              =$PatternSet
    maxExclusive       =$MaxVal

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  Every $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
                matchScope($Scope,$ScopePattern) Satisfies
    $OcVal < $MaxVal
```

InternalOccurrenceMaxInclusive

```
TopicSchema:
 Type                  =$Type
 internalOccurrenceSchema:
    type               =$OcType
    scope              =$ScopePattern
```

```
      match              =$PatternSet
      maxInclusive         =$MaxVal

    Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
      Every $OcVal SuchAs internalOccurrence($Inst,$OcType,$OcVal)@$Scope and
                  matchScope($Scope,$ScopePattern) Satisfies
       $OcVal <= $MaxVal
```

ExternalOccurrenceCardMin

```
    TopicSchema:
     Type                =$Type
     externalOccurrenceSchema:
       type              =$OcType
       scope             =$ScopePattern
       cardMin             =$N


    Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
      ExistsAtLeast $N $OcVal SuchAs occurrence($Inst,$OcType,$OcVal)@$Scope and
        matchScope($Scope,$ScopePattern)
```

ExternalOccurrenceCardMax

```
    TopicSchema:
     Type                =$Type
     externalOccurrenceSchema:
       type              =$OcType
       scope             =$ScopePattern
       cardMax             =$N

    Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
      ExistsAtMost $N $OcVal SuchAs occurrence($Inst,$OcType,$OcVal)@$Scope and
                    matchScope($Scope,$ScopePattern)
```

ExternalOccurrenceOneOf

```
    TopicSchema:
     Type                =$Type
     externalOccurrenceSchema:
       type              =$OcType
       scope             =$ScopePattern
       oneOf              =$StrSet


    Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
      Every $OcVal SuchAs occurrence($Inst,$OcType,$OcVal)@$Scope and
                  matchScope($Scope,$ScopePattern) Satisfies
        member($OcVal,$StrSet)
```

ExternalOccurrenceMatch

```
    TopicSchema:
     Type                =$Type
```

```
externalOccurrenceSchema:
    type              =$OcType
    scope             =$ScopePattern
    match             =$PatternSet


Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
  Every $OcVal SuchAs occurrence($Inst,$OcType,$OcVal)@$Scope and
              matchScope($Scope,$ScopePattern) Satisfies
      Every $Pattern SuchAs member($Pattern,$PatternSet) Satisfies
          match($OcVal,$Pattern)
```

PlayRoleCardMin

```
TopicSchema:
 Type                  =$Type
 PlayRoleSchema:
    associationType         =$AType
    roleType              =$RoleType
    otherRoleType           =$RoleTypeSet
    scope                 =$ScopePattern
    cardMin                =$N

Every $Inst SuchAs instanceOf($Inst, $Type) Satisfies
    AtLeast $N $A SuchAs association($A)@$Scope and
                matchScope($Scope,$ScopePattern) and
                role($A,$RoleType,$Inst) and
                otherRoleTypes($A,$OtherRoleTypes) and
                setEqv($OtherRoleTypes,$RoleTypeSet)

//Check that association has corresponded signature
```

PlayRoleCardMax

```
TopicSchema:
 Type                  =$Type
 PlayRoleSchema:
    associationType         =$AType
    roleType              =$RoleType
    otherRoleType           =$RoleTypeSet
    scope                 =$ScopePattern
    cardMax                =$N

Every $Inst SuchAs instanceOf($Inst, $Type) Satisfies
    AtMost $N $A SuchAs association($A)@$Scope and
                matchScope($Scope,$ScopePattern) and
                role($A,$RoleType,$Inst) and
                otherRoleTypes($A,$OtherRoleTypes) and
                setEqv($OtherRoleTypes,$RoleTypeSet)

//Check that association has corresponded signature
```

OtherRoleCardMin

```
TopicSchema:
 Type                =$Type
 PlayRoleSchema:
```

```
    roleType            =$RoleType
    otherRoleType          =$RoleTypeSet
    scope               =$ScopePattern
    otherRoles:
      roleType          =$ORT
      cardMin           =$N

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $A SuchAs association($A)@$Scope and
            matchScope($Scope,$ScopePattern) and
            role($A,$RoleType,$Inst) and
            otherRoleTypes($A,$OtherRoleTypes) and
            setEqv($OtherRoleTypes,$RoleTypeSet)
   Satisfies
      AtLeast $N $Player SuchAs role($A,$ORT,$Player)
```

OtherRoleCardMax

```
TopicSchema:
  Type                  =$Type
  PlayRoleSchema:
    roleType            =$RoleType
    otherRoleType          =$RoleTypeSet
    scope               =$ScopePattern
    otherRoles:
      roleType          =$ORT
      cardMax            =$N

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $A SuchAs association($A)@$Scope and
            matchScope($Scope,$ScopePattern) and
            role($A,$RoleType,$Inst) and
            otherRoleTypes($A,$OtherRoleTypes) and
            setEqv($OtherRoleTypes,$RoleTypeSet)
   Satisfies
      AtLeast $N $Player SuchAs role($A,$ORT,$Player)
```

OtherRoleAllPlayersFrom

```
TopicSchema:
  Type                  =$Type
  PlayRoleSchema:
    roleType            =$RoleType
    otherRoleType          =$RoleTypeSet
    scope               =$ScopePattern
    otherRoles:
      roleType          =$ORT
      allPlayersFrom       =$TypeSet

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $A SuchAs association($A)@$Scope and
            matchScope($Scope,$ScopePattern) and
            role($A,$RoleType,$Inst) and
            otherRoleTypes($A,$OtherRoleTypes) and
            setEqv($OtherRoleTypes,$RoleTypeSet)
   Satisfies
      Every $PlayerType SuchAs member($PlyerType,$TypeSet) Satisfies
         Every $Player SuchAs role($A,$ORT,$Player) Satisfies
            instanceOf($Player,$PlayerType)
```

OtherRoleOneOf

```
TopicSchema:
  Type                =$Type
  PlayRoleSchema:
     roleType          =$RoleType
     otherRoleType        =$RoleTypeSet
     scope             =$ScopePattern
     otherRoles:
       roleType        =$ORT
       oneOf           =$OneOfSet

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $A SuchAs association($A)@$Scope and
            matchScope($Scope,$ScopePattern) and
            role($A,$RoleType,$Inst) and
            otherRoleTypes($A,$OtherRoleTypes) and
            setEqv($OtherRoleTypes,$RoleTypeSet)
   Satisfies
      Every $PlayerType SuchAs member($PlyerType,$TypeSet) Satisfies
         Every $Player SuchAs role($A,$ORT,$Player) Satisfies
            member($Player,$OneOfSet)
```

OtherPlayerCardMin

```
TopicSchema:
  Type                =$Type
  PlayRoleSchema:
     roleType          =$RoleType
     otherRoleType        =$RoleTypeSet     //can be empty
     scope             =$ScopePattern
     otherPlayers:
       cardMin         =$N

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $A SuchAs association($A)@$Scope and
            matchScope($Scope,$ScopePattern) and
            role($A,$RoleType,$Inst) and
            otherRoleTypes($A,$OtherRoleTypes) and
            setEqv($OtherRoleTypes,$RoleTypeSet)
   Satisfies
      AtLeast $N $Player SuchAs role($A,$RoleType,$Player) and
                  $Player /= $Inst
```

OtherPlayerCardMax

```
TopicSchema:
  Type                =$Type
  PlayRoleSchema:
     roleType          =$RoleType
     otherRoleType        =$RoleTypeSet     //can be empty
     scope             =$ScopePattern
     otherPlayers:
       cardMax          =$N

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $A SuchAs association($A)@$Scope and
            matchScope($Scope,$ScopePattern) and
            role($A,$RoleType,$Inst) and
            otherRoleTypes($A,$OtherRoleTypes) and
```

```
            setEqv($OtherRoleTypes,$RoleTypeSet)
    Satisfies
        AtMost $N $Player SuchAs role($A,$RoleType,$Player) and
                        $Player /= $Inst
```

OtherPlayerAllPlayersFrom

```
TopicSchema:
  Type                  =$Type
  PlayRoleSchema:
     roleType             =$RoleType
     otherRoleType          =$RoleTypeSet     //can be empty
     scope                =$ScopePattern
     otherPlayers:
       allPlayersFrom       =$TopicSet

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $A SuchAs association($A)@$Scope and
            matchScope($Scope,$ScopePattern) and
            role($A,$RoleType,$Inst) and
            otherRoleTypes($A,$OtherRoleTypes) and
            setEqv($OtherRoleTypes,$RoleTypeSet)
   Satisfies
      Every $Player SuchAs role($A,$RoleType,$Player) and $Player /= $Inst
      Satisfies
         Every $PlayerType SuchAs member($PlayerType, $TopicSet)
         Satisfies instanceOf($Player,$PlayerType)
```

OtherPlayerAllPlayersFrom

```
TopicSchema:
  Type                  =$Type
  PlayRoleSchema:
     roleType             =$RoleType
     otherRoleType          =$RoleTypeSet     //can be empty
     scope                =$ScopePattern
     otherPlayers:
       oneOf              =$TopicSet

Every $Inst SuchAs instanceOf($Inst,$Type) Satisfies
   Every $A SuchAs association($A)@$Scope and
            matchScope($Scope,$ScopePattern) and
            role($A,$RoleType,$Inst) and
            otherRoleTypes($A,$OtherRoleTypes) and
            setEqv($OtherRoleTypes,$RoleTypeSet)
   Satisfies
      Every $Player SuchAs role($A,$RoleType,$Player) and $Player /= $Inst
      Satisfies
         member($Player, $TopicSet)
```

AssociationRolePlayerCardMin

```
AssociationSchema:
  type                =$Type
  associationSignature:
     signature        =$TopicSet
     roleSchema:
```

```
        roleType        =$RoleType
        cardMin         =$N

    Every $A SuchAs association($A) and
                signature($A,$RoleTypes) and
                setEqv($RoleTypes,$TopicSet)
    Satisfies
        AtLeast $N $Player SuchAs role($A,$RoleType,$Player)
```

AssociationRolePlayerCardMax

```
    AssociationSchema:
      type              =$Type
      associationSignature:
         signature      =$TopicSet
         roleSchema:
           roleType     =$RoleType
           cardMax      =$N

    Every $A SuchAs association($A) and
                signature($A,$RoleTypes) and
                setEqv($RoleTypes,$TopicSet)
    Satisfies
        AtMost $N $Player SuchAs role($A,$RoleType,$Player)
```

AssociationRolePlayerAllPlayersFrom

```
    AssociationSchema:
      type              =$Type
      associationSignature:
         signature      =$TopicSet
         roleSchema:
           roleType     =$RoleType
           allPlayersFrom    =$PlayerTypes

    Every $A SuchAs association($A) and
                signature($A,$RoleTypes) and
                setEqv($RoleTypes,$TopicSet)
    Satisfies
       Every $Player SuchAs role($A,$RoleType,$Player) Satisfies
           Every $PlayerType SuchAs member($PlayerType,$PlayerTypes) Satisfies
              intsanceOf($Player,$PlayerType)
```

AssociationRolePlayerOneOf

```
    AssociationSchema:
      type              =$Type
      associationSignature:
         signature      =$TopicSet
         roleSchema:
           roleType     =$RoleType
           oneOf        =$OneOfSet

    Every $A SuchAs association($A) and
                signature($A,$RoleTypes) and
                setEqv($RoleTypes,$TopicSet)
    Satisfies
       Every $Player SuchAs role($A,$RoleType,$Player) Satisfies
```

member($Player,$OneOfSet)

4.3 TMCL-Rule

TMCL-Rule allows to declare set of assertions about topic maps. It is a rule-based language which leverages TMQL constructs for specifying conditions and assertions.

TMCL-Rule is close to ISO/IEC 19757-3 (Document Schema Definition Languages (DSDL)- Part 3: Rule-based validation Schematron). Schematron allows do define validation rules for XML documents. TMCL-Rule leverages experience from other rule-based languages and allows specifying constraints based on TMDM.

4.3.1 TopicMapSchema

The TopicMapSchema collects together a set of rules that can be used to validate a topic map. There is exactly one schema information item in each information set. TopicMapSchema can include at the same time topic and association schemas.

```
TopicMapSchema:
  SchemaID          # defines schema ID
  Name?             # defines schema Name
  Includes *        # URI -  list of schemas to include
  RuleItem*         # set of rules
  DiagnosticItem*   # provides more specific details for assertions and reports
```

4.3.2 RuleItem

The RuleItem defines set of assertions about topic map. The RuleItem consists of optional context item, optional let items and one or more assertion or/and report items.

```
RuleItem:
  ID          #defines rule ID
  Name?       #defines rule Name
  ContextItem?  #locates topic map data model information items to be constrained.
  LetItem*      #introduces local variables which can be used in assertions and report items
  AssertItem*   #if test is negative AssertItem generates ConflictItem
  ReportItem*   #if test is positive ReportItem generates NotifyItem
```

4.3.3 ContextItem

The ContextItem is used to locate topic map data model information items to be constrained. It allows to express assertions in a form of "forevery X,Y... where P(X,Y...) satisfies Q(X,Y,...) Variables defined in ContextItem can be used in LetItems, AssertItems and ReportItems The ContextItem is optional element. If rule does not have ContextItem then assertions are evaluated in the context of full topic map.

```
ContextItem:
  ForEvery+     #list of variables
  Where         #TMQL predicate expression with free varaibles from ForEvery list
```

4.3.4 LetItem

The LetItem introduces local variable which can be used in AssertItem and ReportItem

```
LetItem:
  Variable      #variable which receives value
  Where         #TMQL predicate expression which generates value
```

4.3.5 AssertItem

If rule has ContextItem then AssertItem is an assertion about topic map information items located by the ContextItem. In this case assertion can use variables defined in ContextItem. If rule does not have ContextItem assertions are evaluated in the context of full topic map. If test is negative AssertItem generates ConflictItem

AssertItem:
  Test #TMQL expression which can include variables from ContextItem and LetItems and returns true or false
  Message #string which can include variables(and simple path expressions) from ContextItem and LetItems
  Diagnostics #list of DiagnosticItem IDs, is used for detailed notification

Note 1: Rules without ContextItem allow to express constraints defined on full topic map

Example 1: Topic map must have more than 20 topics of "musician" type.

Example 2: Topic map must have a topic for composer who was born in Milan.

Note 2: If constraint can be formulated in a form of "forevery X,Y... where P(X,Y...) satisfies Q(X,Y,...)" preferable form of a rule includes explicit ContextItem.

4.3.6 ReportItem

If rule has ContextItem then ReportItem is an assertion about topic map information items located by the ContextItem. In this case assertion can use variables defined in ContextItem. If rule does not have ContextItem report assertions are evaluated in the context of full topic map. If test is positive ReportItem generates NotifyItem

ReportItem:
  Test #TMQL expression which can include variables from ContextItem and LetItems and returns true or false
  Message #string which can include variables(and simple path expressions) from ContextItem and LetItems
  Diagnostics #list of DiagnosticItem IDs (with 0 or more parameters) is used for detailed notification

4.3.7 DiagnosticItem

DiagnosticItem provides more specific details for assertion and report notifications. DiagnosticItem can include variables and simple path expressions. Variables receive values when diagnostic item is called during rule evaluation. DiagnosticItem can also provide some recommendations for conflict resolution.

DiagnosticItem:
  Parameter* #list of variables
  Message #string scoped by language which can include variables and simple path expressions

4.3.8 ConflictItem

ConflictItem :- RuleConflictItem | SchemaConflictItem

4.3.9 RuleConflictItem

RuleConflictItem:
  RuleID # reference to rule which generates conflict
  TestMessage # string representing Test item from assertion
  ContextBinding* # defines binding for variables from ContextItem
  Message # string
  DiagosticMessage* # string

4.3.10 SchemaConflictItem

```
SchemaConflictItem:
  ConflictTypeID       # reference to constraint type which generates conflict
  TopicSchemaID*       # reference to topic schemas which contain this constraint
  AssociationSchemaID* # reference to association schemas which contain this constraint
  TestMessage          # string representing logical interpretation of constraint
  ContextBinding*      # defines binding for variables from constraint type definition
  Message              # string
  DiagosticMessage*    # string
```

4.3.11 NotifyItem

NotifyItem :- RuleNotifyItem | SchemaNotifyItem

4.3.12 RuleNotifyItem

```
RuleNotifyItem:
  RuleID           # reference to rule which generates report
  TestMessage      # string representing Test item from assertion
  ContextBinding*  # defines binding for variables from ContextItem
  Message          # string
  DiagosticMessage* # string
```

4.4 SchemaNotifyItem

```
SchemaNotifyItem:
  ConflictTypeID       # reference to constraint type which generates notification
  TopicSchemaID*       # reference to topic schemas which contain this constraint
  AssociationSchemaID* # reference to association schemas which contain this constraint
  TestMessage          # string representing logical interpretation of constraint
  ContextBinding*      # defines binding for variables from constraint type definition
  Message              # string
  DiagosticMessage*    # string
```

4.5 Syntax for TMCL-Rule

To be done....

4.6 Combining TMCL-Rule and TMCL-Schema

TMCL-Rule and TMCL-Schema expressions can be combined in the same TMCL schema. It is also possible to insert rules inside of type descriptions. In this case rules have simplified syntax.

TODO: describe simplified syntax for embedding rules into type description In this case there is an implicit context - type

4.7 Topic Map Representation of Constraints

To be done...

4.8 Topic Map Schema References

TMCL enables topic map authors to specify a schema to which the topic map is conformant. This is achieved by reifying the topic map with a topic and assigning an occurrence to that topic of type TMCLSchemaReference.

The following PSI is used to denote the occurrence type for schema references.

http://www.isotopicmaps.org/tmcl/#TMCLSchemaReference

This is used to type a occurrence on a topic that reifies the topic map in order to reference the schema for this topic map instance. The value of the occurrence must reference a valid TMCL XML representation or a topic of type Schema.

4.9 Schema Composition

Schema composition is the ability to take two or more schemas and compose them into a single schema. Given that a schema consists of a set of constraints, schema composition merely takes all the constraints from all schemas being composed and returns a single schema that consists of all constraints. Applications are free to identify and remove redundant constraints and generate conflicts should any constraints be contradictory.

More formally:

Given
  Schema : s1, s2
  Constraint : c1, c2, c3, c4, c5
  s1 := {c1, c2, c3}
  s2 := {c4, c5}
That
  Compose(s1, s2) => s3
  s3 := {c1, c2, c3, c4, c5}

Bibliography

[1] *TMCL Requirements*, ISO, 2004

[2] *TMCL Use Cases*, ISO, 2004

[3] *TMQL Requirements*, ISO, 2003