Topic Maps Reference Model, 13250-5

Patrick Durusau
patrick@durusau.net

Steve Newcomb
srn@coolheads.com

Robert Barta
rho@bigpond.net.au

Contents

Page

Figures

Tables

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

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

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75% of the member bodies casting a vote.

Introduction

The Topic Maps family of standards is designed to facilitate the gathering of all the information about a subject at a single location. The information about a subject includes its relationships to other subjects; such relationships may also be treated as subjects (*subject-centric*).

ISO/IEC 13250-2 (TMDM, Topic Maps Data Model) provides a foundation for syntaxes and notations, such as those defined in *ISO/IEC 13250-3* Topic Maps XML Syntax and *ISO/IEC 13250-4* Topic Maps Canonicalization. Of necessity, the TMDM makes ontological commitments in terms of how particular subjects are identified (topics, associations, occurrences), what properties are required, the tests to be used to determine whether two or more proxies represent the same subject, and other matters.

This Standard defines TMRM (Topic Maps Reference Model), which is more abstract and has fewer ontological commitments. Its purpose is to serve as a minimal, conceptual foundation for subject-centric data models such as the TMDM, and to supply ontologically neutral terminology for disclosing these. It defines what is required to enable the mapping of different subject-centric data models together to meet the overall goal of the Topic Maps standards, that each subject has a single location for all the information about it.

TMRM also provides a formal foundation for related Topic Maps standards such as *ISO/IEC 18048* Topic Maps Query Lanuage (TMQL) and *ISO/IEC 19756* Topic Maps Constraint Language (TMCL).

Topic Maps — Reference Model — ISO 13250-5

1 Scope

The following are within the scope of this part of ISO 13250:

— a formal model for subject maps;

— minimal access functionality and information retrieval from subject maps;

— a constraint framework governing the interpretation of subject maps.

The following are outside the scope of this part of ISO 13250:

— a particular formalism to constrain subject maps.

2 Normative References

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.

ISO/IEC 13250-2, *Topic Maps–Data Model*

ISO/IEC 13250-3, *Topic Maps–XML Syntax*

ISO/IEC 13250-4, *Topic Maps–Canonicalization*

ISO/IEC 18048, *Topic Maps Query Language*

ISO/IEC 19756, *Topic Maps Constraint Language*

3 Subjects

A subject is defined in the Topic Maps family of standards as something which '[...] can be anything whatsoever, regardless of whether it exists or has any other specific characteristics, about which anything whatsoever may be asserted by any means whatsoever' (*ISO/IEC 13250-2* 5.3.1). According to the TMRM, there is only one representative for subjects: subject proxies (*proxies*).

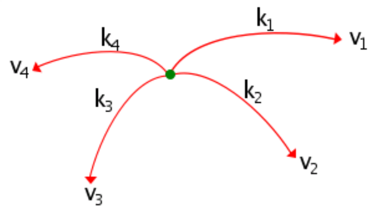

Figure 1 – Proxy Structure

4 Subject Proxies and Maps

Proxies consist of properties. These are key/value pairs which—in turn—may contain references to other proxies. This recursive relationship is defined via two postulated sets. One is the finite set of (proxy) *labels*, $\mathcal{L}$. The second set postulated here is $\mathcal{V}$, a finite set of values. It contains values (such as numbers, strings, etc.), and all the labels in $\mathcal{L}$.

A *property* is the pair $\langle k, v\rangle \in \mathcal{L} \times \mathcal{V}$. The first component of this pair is called the *key*, the other the *value* of the property. The (consequently finite) set of all such properties is denoted as $\mathcal{P}$. Accordingly, keys in properties are always labels, the values in properties may be labels or any other value from the value set $\mathcal{V}$.

EXAMPLE 1 Given the label `shoesize` and the integer 43, then $\langle \texttt{shoesize}, \texttt{43}\rangle$ is a property.

A *proxy* is a finite set of properties, $\{p_1, \ldots, p_n\}$, with $p_i \in \mathcal{P}$ (see Fig. 1). The set of all proxies $\mathcal{X}$ is the set of all subsets of $\mathcal{P}$, $\mathcal{X} = 2^{\mathcal{P}}$. The connection between proxies and their labels is modeled with a partial, injective function $\tilde{}: \mathcal{X} \mapsto \mathcal{V}$. It returns the label for a given proxy x whereby two different proxies never share the same label. The function is extended to values in that $\tilde{v} = v$.

EXAMPLE 2 A particular person may be represented by the following proxy:
$\{\langle \texttt{shoesize}, \texttt{43}\rangle, \langle \texttt{beardcolor}, \texttt{white}\rangle, \langle \texttt{beardlength}, \texttt{verylong}\rangle\}$

NOTE 1 Subject proxies are composed of properties, each (isolated or in combination with other properties) being a statement about the proxy's subject. A proxy is defined by the totality of its properties. Properties can provide a basis for mapping multiple representatives of the same subject to each other.

NOTE 2 One proxy may contain several properties which all share the same key but have different values; or share the same value, but have differing keys.

A *subject map* (*map*) is a finite set of *proxies*. The set of all such maps is denoted as $\mathcal{M}$. As maps are simply sets of proxies, *generic merging of maps* is achieved via set union, $m \cup m'$.

NOTE 3 The model of subject maps described herein assumes no particular implementation strategy.

5 Ontological Commitments

This Standard deliberately leaves undefined the methods whereby subject proxies are derived or created. No specific mechanism of subject identification is inherent in or mandated by this Standard, nor does it predefine any subject proxies.

NOTE 1 Any subject proxy design choices would be specific to a particular application domain and would exclude equally valid alternatives that might be appropriate or necessary in the contexts of various requirements.

Two types of relationships, *sub* (subclass of) and *isa* (instance of), are defined. These predicates are always interpreted *relative* to a given map m:

a) Two proxies c, c' can be in a *subclass-superclass* relationship, $\text{sub}_m \subseteq m \times m$. In such a case, the same relationship can be stated either c *is a subclass of* c' or c' *is a superclass of* c.

   sub_m is supposed to be reflexive and transitive. *Reflexive* implies that any proxy is a subclass of itself, regardless whether the proxy is used as a class in the map or not: $x \text{ sub}_m\ x$ for all $x \in m$.

   Transitive implies that if a proxy c is a subclass of another, c', and that subclasses c'', then c is also a subclass of c'', i.e. if $c\ sub_m\ c'$ and $c'\ sub_m\ c''$ then also $c\ sub_m\ c''$ must be true.

   NOTE 2 Circular subclass relationships may exist in a map.

b) Two proxies a, c can be in an *isa* relationship, $\text{isa}_m \subseteq m \times m$. In such a case, the same relationship can be stated either a *is an instance of* c or c *is the type of* a.

   The *isa* relationship is supposed to be non-reflexive, i.e. $x \text{ isa}_m\ x$ for no $x \in m$, so that no proxy can be an instance of itself. Additionally, whenever a proxy a is an instance of another c, then a is an instance of any superclass of c: if $x \text{ isa}_m\ c$ and $c \text{ sub}_m\ c'$, then $x \text{ isa}_m\ c'$ is true.

NOTE 3 This Standard does not mandate any particular way of representing such relationships inside a map. One option is to model such a relationship simply with a property using a certain key (say `type`). An alternative way is to provide a proxy for each such relationship. Such relationship proxies could, for example, have properties whose keys are `instance` and `class`, or respectively `subclass` and `superclass`.

6 Navigation

Given a map m and particular proxies $x, y \in m$ in it, the following *primitive navigation operators* are defined:

a) A postfix operator $\downarrow$ to return the multiset of *all local keys* of a given proxy:

$$x\!\downarrow\, = \{k \mid \exists v\colon \langle k, v\rangle \in x\} \tag{1}$$

b) A postfix operator $\uparrow_m$ to retrieve the multiset of *remote keys* of a proxy inside m. These are those where the given proxy (more precisely its label) *is* the value in another proxy:

$$x\!\uparrow_m\, = \{k \mid \exists y \in m\colon \langle k, \tilde{x}\rangle \in y\} \tag{2}$$

c) A postfix operator $\rightarrow k$ to retrieve the multiset of *local values for a particular key* $k \in \mathcal{L}$:

$$x \rightarrow k = \{v \mid \exists \langle k, v\rangle \in x\} \tag{3}$$

   Using the predicate sub_m, the operator can be generalized to honor subclasses of the key k:

$$x \!\rightarrow_m\! k^* = \{v \mid \exists \langle k', v\rangle \in x\colon k' \text{ sub}_m\ k\} \tag{4}$$

If all values (regardless of the key) should be retrieved, the notation $\rightarrow *$ can be used.

d) A postfix operator $\leftarrow_m k$ which navigates to all proxies in the given map which use a given value v together with a certain key $k \in \mathcal{L}$:

$$v{\leftarrow_m}k = \{x \in m \mid \exists \langle k, \tilde{v} \rangle \in x\} \tag{5}$$

Using the predicate sub_m, the operator can be generalized to honor subclasses of the key k:

$$v{\leftarrow_m}k^* = \{x \in m \mid \exists \langle k', \tilde{v} \rangle \in x \colon k' \ \mathrm{sub}_m \ k\} \tag{6}$$

If all proxies (regardless of the key) should be retrieved, the notation $\leftarrow_m *$ can be used.

NOTE 1 All operators return multisets, i.e. one and the same element may occur any number of times.

It is straightforward to generalize all these navigation operators from individual proxies (and values) to multisets of them. As a consequence the result of one postfix can be used as startpoint for another postfix, enabling the building of *postfix chains*. This primitive path language is denoted as $\mathcal{P}_\mathcal{M}$.

NOTE 2 $\mathcal{P}_\mathcal{M}$ only serves as a minimal baseline for functionality to be provided by conforming implementations. It can be also used as the basis for a formal semantics for higher-level query and constraint languages. Annex A describes one.

7 Constraints

Subject maps are structures which are used to encode assertional knowledge. To interpret a map, be it for modelling, retrieval or modification, some background information about the map may be necessary. That information is provided in form of *constraints*, so that a given map m either satisfies a constraint, or not. A constraint language is a formalism which allows the expression of such constraints.

NOTE 1 Constraints may conditionally or unconditionally require the existence of certain proxies in maps, the existence of properties in proxies, and/or values in properties. Constraints may also prohibit the existence of any of the foregoing.

EXAMPLE 1 A constraint language may allow the expression of constraints such as *all instances of the concept* `person` *must have at least one* `shoesize` property or *any* `shoesize` *property must have an integer value between 10 and 50.*

NOTE 2 The precise ways in which constraints may be expressed are not constrained by this Standard. Different constraint languages will differ in expressitivity and, consequently, in computational complexity.

This Standard imposes two requirements on any constraint language $\mathcal{C}$:

a) $\mathcal{C}$ must define the *application of a constraint to a map* in the form of a binary operator $\otimes : \mathcal{M} \times \mathcal{C} \mapsto \mathcal{M}$. A particular map m is said to *satisfy a constraint*, $m \models c$, if the application of the constraint results in a non-empty map:

$$m \models c \qquad \Longleftrightarrow \qquad m \otimes c \neq \emptyset \tag{7}$$

The operator $\otimes$ is used to define the *satisfaction relation* $\models \subseteq \mathcal{M} \times \mathcal{C}$ between a map and a constraint.

b) $\mathcal{C}$ must define a *merging operator* $\oplus : \mathcal{M} \times \mathcal{M} \mapsto \mathcal{M}$ as binary operator between two maps. It must be commutative, associative and idempotent.

NOTE 3 The provision of $\oplus$ and $\otimes$ may be done in any manner that is sufficiently expressive. Annex A demonstrates one way of defining $\otimes$.

8 Merging

Generic merging of maps only combines two (or more) proxy sets. Application-specific merging includes a second aspect. A mechanism has to be found to state whether—in a given map—two proxies are *regarded to be about the same subject.* Then all such equivalent proxies have to be actually merged.

NOTE 1 How *subject sameness* is determined and how the actual *proxy merging* is effectively done is not constrained by this Standard. Such a process may be defined as having inputs that consist only of the proxies to be merged. Alternatively, the inputs may also include other information that may appear either inside the map or elsewhere in the merging process's environment.

NOTE 2 Given the appropriate expressitivity of the used constraint language, such equivalence and the consequent merging process can be described with a constraint.

Merging is modeled with a partial function $\bowtie: \mathcal{X} \times \mathcal{X} \times \mathcal{E} \mapsto \mathcal{X}$. It takes two proxies and an—otherwise unconstrained—environment $\mathcal{E}$ as parameters and produces a new proxy. In the case that the environment has no influence in this process, $\bowtie$ is an infix operator $\mathcal{X} \times \mathcal{X} \mapsto \mathcal{X}$ between proxies.

NOTE 3 The reason for including $\mathcal{E}$ as a term in the definition of merging is to account for the fact that merging criteria may be defined as being dependent on conditions external to the maps. When environmental differences affect the results of merging, a single interchangeable subject map may be realized as different subject maps in different environments. Such differences may interfere with information interchange and create confusion, or aid such interchange and mitigate confusion, or both.

The fact that $\bowtie$ is *partial* means that it may be applicable only to some pairs of proxies but not to others. Those where the result is defined are supposed to be merged.

The operator $\bowtie$ must be commutative and associative:

$$\begin{aligned} x \bowtie x' &= x' \bowtie x \\ (x \bowtie x') \bowtie x'' &= x \bowtie (x' \bowtie x'') \end{aligned}$$

Additionally, $\bowtie$ must be idempotent, as proxies merged with themselves should not result in a different one:

$$x \bowtie x = x$$

The operator $\bowtie$ factors a given proxy set into equivalence classes: two proxies x, y from a given map $m \in \mathcal{M}$ are then in the same class if $x \bowtie y$ is defined. The set of equivalence classes is written $m/\bowtie$. Every such class can be merged into a single proxy by combining all its members by applying $\bowtie$. Given a set of proxies $c = \{x_1, \ldots, x_n\}$, the merging of members of an equivalence class is defined as: $\bowtie c = x_1 \bowtie x_2 \bowtie \ldots \bowtie x_n$.

Given a map m and a particular function $\bowtie$, the *merged view of the map* $m|_{\bowtie}$ is defined:

$$m|_{\bowtie} = \{\bowtie c \mid c \in m/\bowtie\} \tag{8}$$

One such merging step may result in proxies being created which again may be mergeable. The process can be repeated until a *fully merged map* is computed, so that $m|_{\bowtie} = m$. Fully merged maps are symbolized as $m|_{\bowtie}{}^{*}$.

9 Map Legends

A *map legend* (*legend*) $G = \{c_1, \ldots, c_n\}$ is a finite set of constraints, all from a given constraint language $\mathcal{C}$. A legend G is said to *govern* a map m if $m \models c_i$ for all $c_i \in G$.

NOTE 1 The legend of a subject map plays a role similar to the legend on more familiar city or road maps. The legend associated to a subject map is one key to its interpretation. Just as the legends of geographic maps describe and define the symbols that appear in them, their scaling rules, etc., subject map legends explain the symbols that appear in them (such as property keys) and other interpretation rules.

NOTE 2 There is no explicit connection between maps and legends, other than whether the map is governed by a legend or not. Any subject map can be simultaneously viewed with multiple legends written for different purposes or users. Such views may be quite different from each other. Maps themselves exist without any legend describing the rules how they may be used.

10 Conformance

An implementation conforms to this Standard if:

a) Its information structures are homomorphous to the subject map structure in Clause 4;

b) It implements merging by providing an operator $\oplus$ according to Clause 7;

c) It implements access methods equivalent to those in Clause 6;

d) Those methods must honor the predicates from Clause 5;

e) It implements a constraint language by providing an operator $\otimes$ according to Clause 7;

f) It implements legends according to Clause 9.

Annex A
(normative)
Path Language

The primitive path language $\mathcal{P}_{\mathcal{M}}$ (clause 6) can be used to define a more expressive language $\mathcal{T}_{\mathcal{M}}$ which provides not only navigation, but also filtering, sorting and general function application.

In the following, tuple sequences are first defined as the results of applying a $\mathcal{T}_{\mathcal{M}}$ path expression to a map (or any multiset of proxies and values). Then, the path expression language itself is compiled, defining the semantics for the application operator $\otimes$.

A.1 Tuples and Tuple Sequences

Path expressions are patterns to be identified in a map. In order to provide a model for both queries and constraints, the results of path expressions are modeled as tuples of values and organized into tuple sequences.

A single tuple with values from a value set $\mathcal{V}$ is denoted as $\langle v_1, v_2, \ldots, v_n\rangle$. Tuples are identical if all their values in corresponding positions are identical.

Tuples can be concatenated simply by collating their values: $\langle u_1, \ldots, u_m\rangle \cdot \langle v_1, v_2, \ldots, v_n\rangle = \langle u_1, \ldots, u_m, v_1, \ldots, v_n\rangle$. This enables the representation of tuples as the products of singleton tuples:

$$t = \prod_{i=1}^{n} \langle v_i\rangle = \langle v_1\rangle \cdot \langle v_2\rangle \cdot \ldots \cdot \langle v_n\rangle \tag{9}$$

The concatenation symbol $\cdot$ will be omitted from now on.

When tuples are organized into sequences, a single sequence is written:

$$s = \sum_{i=1}^{m} t_i = [t_1, \ldots, t_m] \tag{10}$$

if the sequence is *unordered* and $\vec{s}$ for *ordered sequences*. Otherwise, sequences behave like multisets, i.e. operations such as union $\cup$, intersection $\cap$ and subtraction $\setminus$ are those of multisets.

Sets of values can be interpreted in such that every value builds exactly one tuple. For a given set of values $\{v_1, \ldots, v_n\}$ the tuple sequence $\sum_{i=1}^{n}\langle v_i\rangle$ can be built. This conversion is denoted as $\langle\{v_1, \ldots, v_n\}\rangle$. Under this interpretation, a map $m = \{x_1, \ldots, x_n\}$ can be represented as the tuple sequence $[\langle x_1\rangle, \ldots, \langle x_n\rangle]$. Conversely, a tuple sequence can be interpreted as a map when the tuples it contains are singleton proxies.

Tuple sequences can be concatenated:

$$\sum_{i=1}^{m} s_i + \sum_{j=1}^{n} t_j = [s_1, \ldots, s_m, t_1, \ldots, t_n] \tag{11}$$

by interleaving the tuples of the second operand with those of the first. If both operand sequences are ordered, the result is ordered. Otherwise, the result is unordered. The indices may be omitted if the range is obvious.

Tuple sequences can also be combined by *multiplying* (joining) them. The product of two tuple sequences is defined recursively:

$$(s)\left(\sum_{j=1}^{m}\langle v_{1\mathrm{j}}, v_{2\mathrm{j}}, \ldots, v_{\mathrm{lj}}\rangle\right) = \left(s\sum_{j=1}^{m}\langle v_{1\mathrm{j}}\rangle\right)\sum_{j=1}^{m}\langle v_{2\mathrm{j}}, \ldots, v_{\mathrm{lj}}\rangle \tag{12}$$

$$\sum_{i=1}^{n} t_i \sum_{j=1}^{m}\langle v_j\rangle = \sum_{i,j=1}^{nm}(t_i\langle v_j\rangle) \tag{13}$$

NOTE 1 Every tuple of the left hand operand sequence is concatenated with every other one of the right-hand operand. The first value of each tuple of the second operand is removed and combined with every tuple of the first operand. This is repeated until the second operand does not have tuples with any values.

The resulting sequence is unordered.

Tuples and proxies are closely related. All the values can be taken out of a proxy and arranged into a value tuple. If order is important, an order can be postulated on the keys and the values are sorted according to it. Conversely, a value tuple can be converted into a proxy, provided that the tuple of keys is available:

$$\langle v_1, \ldots, v_n\rangle \yen \langle k_1, \ldots, k_n\rangle = \{\langle k_1, v_1\rangle, \ldots, \langle k_n, v_n\rangle\} \tag{14}$$

This operation can be generalized to tuple sequences by repeating the process for every tuple of the sequence. Every tuple sequence can be interpreted as a sequence of proxies once a set of keys has been chosen. This sequence of proxies can then be interpreted as a map.

A.2 Path Expressions

A particular path expression can be interpreted as an *expression of interest*, i.e. as a pattern to be identified in a map. Given a tuple sequence s, a path expression p can be applied to it, with the expectation of another tuple sequence in return. This application can be symbolized as $s \otimes_m p$. This operation is to be understood in the context of a map m. If that is implicit, the index can be dropped.

The set of path expressions $\mathcal{T}_\mathcal{M}$ is characterized as the smallest set satisfying the following conditions:

a) The *empty* postfix ε is in $\mathcal{T}_\mathcal{M}$. When it is applied to a sequence, the same sequence is returned.

b) Every value from $\mathcal{V}$ (and consequently every proxy label) is in $\mathcal{T}_\mathcal{M}$. If such a value is applied to a sequence, then the sequence itself is discarded. Instead a new sequence with a singleton tuple is created in which the value is used.

c) The navigation operators $\uparrow$, $\downarrow$, $\leftarrow k$ and $\rightarrow k$ are in $\mathcal{T}_\mathcal{M}$. When applied to a tuple sequence these operators are applied to every tuple:

$$\left(\sum_{i=1}^{n}\langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}}\rangle\right) \otimes \leftarrow k = \sum_{i=1}^{n}\prod_{j=1}^{l}\langle v_{\mathrm{ji}} \leftarrow_m k^*\rangle \tag{15}$$

$$\left(\sum_{i=1}^{n}\langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}}\rangle\right) \otimes \rightarrow k = \sum_{i=1}^{n}\prod_{j=1}^{l}\langle v_{\mathrm{ji}} \rightarrow_m k^*\rangle \tag{16}$$

These operators simply iterate over each tuple and compute an intermediate result for each tuple. This intermediate result is achieved by applying the navigation to each value in the current tuple. As one such application results in a multiset of values, that is converted into a singelton tuple sequence. All these singleton tuple sequences are multiplied and all these intermediate results are then concatenated into the overall result.

An analogous approach is used for finding keys:

$$\left(\sum_{i=1}^{n}\langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}}\rangle\right) \otimes \downarrow = \sum_{i=1}^{n}\prod_{j=1}^{l}\langle v_{\mathrm{ji}} \downarrow\rangle \tag{17}$$

$$\left(\sum_{i=1}^{n}\langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}}\rangle\right) \otimes \uparrow = \sum_{i=1}^{n}\prod_{j=1}^{l}\langle v_{\mathrm{ji}} \uparrow_m\rangle \tag{18}$$

d) Given path expressions $p_1, \ldots, p_n$ and a function $f : \mathcal{V}^n \mapsto \mathcal{V}$ then also $f(p_1, \ldots, p_n)$ is in $\mathcal{T}_\mathcal{M}$. When an n-ary function $f : \mathcal{V}^n \mapsto \mathcal{V}$ is applied to a tuple sequence, it is interpreted as one which takes a value tuple of length n and renders one value from $\mathcal{V}$. To apply it to a tuple sequence, it is applied to every individual tuple and the singleton results are organized back into a sequence:

$$f(\sum t_i) = \sum\langle f(t_i)\rangle \tag{19}$$

e) The *projection* postfix π_i is in $\mathcal{T}_\mathcal{M}$ for any positive integer i. It can be used to extract a certain column from a given tuple sequence:

$$\sum_{i=1}^{n}\langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}}\rangle \otimes \pi_j = \sum_{i=1}^{n}\langle v_{\mathrm{ji}}\rangle \tag{20}$$

Projection here plays a similar role like in query languages like SQL, except that an index is used for selection instead of names.

To organize values freely into a new tuple sequence, the *tuple projection* $\langle p_1, \ldots, p_n\rangle$ is used with p_i being path expressions. For a single tuple it evaluates all the path expressions and builds the product of all partial result sequences:

$$t \otimes \langle p_1, \ldots, p_n\rangle = \prod_{i=1}^{n} t \otimes p_i \tag{21}$$

When applied to a tuple sequence, all applications to its tuples are concatenated. As a special case, the *empty projection* $\langle\rangle$ always returns an empty tuple sequence.

f) The *conditional* $p?q{:}r$ is in $\mathcal{T}_{\mathcal{M}}$ for path expressions p, q and r. When it is applied to a tuple sequence, every tuple is tested whether it produces a result when p is applied to it. If that is the case, the *then* branch is used, i.e. the tuple will be subjected to q and these results will be added to the overall result. Otherwise, the tuple will get r applied:

$$\left(\sum_{i=1}^{n} t_i\right) \otimes (p?q{:}r) = \sum_{i=1}^{n} ([t_i \otimes q \mid t_i \otimes p \neq \emptyset] \cup [t_i \otimes r \mid t_i \otimes p = \emptyset]) \tag{22}$$

g) For two path expressions p and q the *alternation* $p + q$, the *reduction* $p - q$, and the *comparison* $p = q$ are in $\mathcal{T}_{\mathcal{M}}$:

$$s \otimes (p + q) = (s \otimes p) \cup (s \otimes q) \tag{23}$$

$$s \otimes (p - q) = (s \otimes p) \setminus (s \otimes q) \tag{24}$$

$$s \otimes (p = q) = (s \otimes p) \cap (s \otimes q) \tag{25}$$

h) For two path expressions p and q the *concatenation* $p \circ q$ is in $\mathcal{T}_{\mathcal{M}}$. It is defined that

$$s \otimes (p \circ q) = (s \otimes p) \otimes q \tag{26}$$

If—from the context—it is clear that two path expressions are to be concatenated, the infix is omitted.

Annex B
(normative)
TMRM mapping of Topic Maps Data Model (ISO 13250-2)

B.1 Introduction

This annex describes a mapping of the Topic Maps Data Model in terms of the astract model of the TMRM.

The mapping from TMDM to TMRM consists of three parts:

— Transformation from TMDM to TMRM

— Formal semantics

— Constraints

B.1.1 General principles

Two general principles have been followed in the design of this mapping:

— The properties of a proxy should be as exact a representation of its identity as possible. Information about the subject which does not contribute to its identity should be external to the proxy representing the subject. (A related principle is that proxies should not contain artificial identifiers whose only purpose is to make proxies distinguishable.)

— The proxies representing different things in the TMDM model should be as uniform in structure as possible, in order to make the representation easier to understand and follow.

The entire mapping follows from these two principles.

B.1.2 Overview

The following is a summary overview of the mapping that is developed in more detail within. It uses normal set and tuple syntax to represent proxies and properties. It also uses `+` to represent required and repeatable properties, `*` for optional and repeatable properties, and `?` for optional properties.

The TMRM representation of TMDM primarily consists of proxies representing either topics or statements. (Statements in TMDM are associations, occurrences, topic names, and variants.)

B.1.2.1 Statements

The proxies for statements have the following form:

```
{(type,      ____),
 (scope,     ____),
 (roletype1, ____),
 (roletype2, ____),
 ...}
```

NOTE Topic names, occurrences, and variants don't have role types in TMDM, but in this representation they have been extended so that they do.

The statements for occurrences take the following form:

```
{(type,     ____),
 (scope,    ____),
 (subject,  ____),
 (resource, ____)}
```

The value of `subject` is here the topic the occurrence is attached to, and `resource` is the actual occurrence.

The statements for topic names take the following form:

```
{(type,    ____),
 (scope,   ____),
 (subject, ____),
 (value,   ____)}
```

NOTE The `subject` is the topic, and `value` is the actual name.

The statements for variant names take the following form:

```
{(type,    variant),
 (scope,   ____),
 (subject, ____),
 (value,   ____)}
```

NOTE The `subject` is the topic name the variant is attached to. The type is fixed because variant names do not have any type in the TMDM, and this statement must be identified as being a variant.

B.1.2.2 Topics

The topic proxies have the following form:

```
{(itemid,  _____)*,
 (subjid,  _____)*,
 (subjloc, _____)*,
 (reified, _____)}
```

NOTE Topic proxies only contain the identity of the topic, and nothing else.

B.1.2.3 Scope and Topic Map Proxies

Proxies are defined for scope and topic maps. The scope proxies are the values of the scope properties in the statements, and they have the following form:

```
{(theme, ____)*}
```

The topic map proxy is the proxy that represents the topic map item in TMDM. There is only one such proxy and it represents the topic map when a topic reifies the topic map. It takes the following form:

`{(type, topicmap)}`

NOTE There can be only one topic map proxy in any given topic map.

B.2 The mapping

The mapping of the TMDM to the TMRM is expressed in the form of preconditions and post-conditions. Individual parts of the TMRM model are mapped in the following way:

a) If i is an information item, then *i.property* refers to the value of the property of i whose name is `[property]`

b) The $m(i)$ function takes a TMDM information item as input and produces the corresponding proxy as output. The function is recursive.

This function has not been fully formalized.

B.2.0.4 Predefined proxies

The TMRM representation of the TMDM requires a number of predefined proxies in order to work. *First define subject_identifier on top of the empty proxy, then define all the others on top of that. Note that this means using the empty proxy for two different things.*

B.2.0.5 Topic items

If t is a topic item the resulting TMRM model must contain a proxy t' such that:

$$\begin{aligned} t' = \; & \{(\texttt{itemid}, u) | u \in t.item\_identifier\} \cup \\ & \{(\texttt{subjid}, u) | u \in t.subject\_identifier\} \cup \\ & \{(\texttt{subjloc}, u) | u \in t.subject\_locator\} \end{aligned}$$

B.2.0.6 Topic name items

If n is a topic name item the resulting TMRM model must contain a proxy n' such that:

$$\begin{aligned} n' = \; & \{(\texttt{type}, m(n.type)), \\ & (\texttt{scope}, scope(n.scope)), \\ & (\texttt{subject}, m(n.parent)), \\ & (\texttt{value}, n.value)\} \end{aligned}$$

B.2.0.7 Datatype translation

This part needs to be written.

B.2.0.8 Variant name items

If v is a variant name item the resulting TMRM model must contain a proxy v' such that:

$$\begin{aligned} v' = \; & \{(\texttt{type}, \texttt{variant}, \\ & (\texttt{scope}, scope(v.scope)), \\ & (\texttt{subject}, m(v.parent)), \\ & (\texttt{value}, n.value)\} \end{aligned}$$

B.2.0.9 Occurrence items

If o is an occurrence item the resulting TMRM model must contain a proxy o' such that:

$$\begin{aligned} o' = \ & \{(\texttt{type}, m(o.type), \\ & (\texttt{scope}, scope(o.scope)), \\ & (\texttt{subject}, m(o.parent)), \\ & (\texttt{resource}, o.value)\} \end{aligned}$$

B.2.0.10 Association items

If a is an association item the resulting TMRM model must contain a proxy a' such that:

$$\begin{aligned} a' = \ & \{(\texttt{type}, m(a.type), \\ & (\texttt{scope}, scope(a.scope))\} \cup \\ & \{(t,p) | \exists r \in a.roles : r.type = t \wedge r.player = p\} \end{aligned}$$

B.2.0.11 The topic map item

The topic map item is mapped to the following proxy m:

$$m = \{(\texttt{type}, \texttt{topicmap})\}$$

B.2.0.12 Scope

The function $scope(s)$ takes a set of topics and produces the proxy representing that set of topics. The function is defined as follows:

$$scope(s) = \{(\texttt{theme}, t) | t \in s\}$$

B.3 Formal semantics

To be written.

B.4 Constraints

To be written.

Annex C
(informative)
TMRM Notation

<table>
<tr><th>Symbol</th><th>Used For</th><th>Email Equivalent</th></tr>
<tr><td>k^*</td><td>All subclasses (key or value)</td><td>k*</td></tr>
<tr><td>$m|_{\bowtie}$</td><td>Set of equivalence classes</td><td>m||><|</td></tr>
<tr><td>$m|_{\bowtie}^{*}$</td><td>Fully merged subject map</td><td>m||><|*</td></tr>
<tr><td>$\downarrow$</td><td>All keys in a proxy</td><td>\</td></tr>
<tr><td>$\uparrow$</td><td>Proxy as a key</td><td>/</td></tr>
<tr><td>$\rightarrow$</td><td>All values for a key</td><td>-></td></tr>
<tr><td>$\leftarrow$</td><td>Proxies with given value for a key</td><td><-</td></tr>
<tr><td>$\mapsto$</td><td>Results in</td><td>|-></td></tr>
<tr><td>$\models$</td><td>Satisfaction relationship</td><td>|=</td></tr>
<tr><td>$\bowtie$</td><td>Merging function</td><td>|><|</td></tr>
<tr><td>$\otimes$</td><td>Constraint operator</td><td>(x)</td></tr>
<tr><td>$\oplus$</td><td>Merging operator</td><td>(+)</td></tr>
</table>