Topic Maps Reference Model, 13250-5

Patrick Durusau
patrick@durusau.net

Steve Newcomb
srn@coolheads.com

Robert Barta
rho@bigpond.net.au

Contents

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 rate at which subjects and their representatives appear on the digital landscape continues to increase on a daily basis. Information about subjects, that is topics of conversation, is held in an ever increasing variety of data formats and information systems. Access to that information is governed by diverse paradigms, such as relational and object-oriented models, the XML infoset, and also semantic networks.

Concrete deployments often suffer from incompatible ontologies, taxonomies, vocabularies, and schemas that act as barriers to integration of information about a subject. The Tower of Babel story (Genesis 11:1-9) is one explanation for the diversity of human languages and concept worlds. People identify their subjects in diverse ways, each of which is meaningful to a particular community.

EXAMPLE 1 It has been rumored that within the insurance industry alone, there are at least seventy-two distinct and incompatible ways to identify an insurance policy.

This Standard embraces that diversity of identification as enriching the information known about a subject. It enables the mapping of diverse identifications to a single representative for that subject. The underlying principle of this Standard is that any subject, however identified, remains the same subject. Those subjects are represented by abstractions within a universe of discourse. A *subject map* is defined here as a set of such representatives (called *subject proxies*) and the rules (called *constraints*) for interpreting them.

What distinguishes this Standard from other efforts at mediating between identifications of subjects is that it provides for the disclosure of information about those representatives and the rules for determining when two or more such representatives represent the same subject.

Many approaches to data integration, such as standard ontologies, (and, at a higher level of abstraction, languages for expressing ontologies) rely upon the inarguable presumption that, if all data used the same universal naming system, then data mapping would be facilitated. Unfortunately, the condition posited by that premise has never occurred. Regardless of whether one accepts the Tower of Babel story's explanation, it is fair to observe that no single representation of subjects has ever prevailed in any field of human endeavor.

There are alternatives to the subject maps model for integrating diverse identifications of subjects. One alternative is identification imperialism that reduces the nuanced complexity of human affairs to a uniform grayness that fails to capture anyone's identification of a subject. Another is to continue to reside in information silos, while hoping that relevant information does not exist beyond the wall of a given silo.

New subjects and diverse representatives for those subjects are emerging every day. Whether the field is scientific research, current events, business, politics or others, there is no point at which all interested parties can stop to agree on how to refer to all the subjects of interest to their community, much less usefully communicate that mapping to other communities. Life is not static and any static means of subject identification will inevitably fail.

Subject maps offer a way to change opaque representatives into transparent subject proxies, which can be meaningfully mapped together on the basis of their properties. A common language is not needed for communication across groups, languages, or concept worlds, just subject maps.

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 for 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 Subjects

A subject is anything that can be a topic of conversation. This Standard makes no distinction between subjects that are thought by some to be true, by others to be false and by still others irrelevant. It does not matter whether the subjects are considered to be 'real,' 'imaginary,' or to be denizens of the 'real world', or an information system, or to have some other ontological status in the view of any particular observer.

Since subjects are discussed and information is recorded about them only through representatives, this Standard is applicable to all such cases. Just as subjects are without limitation, so are their representatives. The representative for a subject could be simply a word without more. The same subject may have many different representatives, which is the usual case. That commonplace fact gives rise to the Babel-esque problem addressed by this Standard.

3 Subject Proxies and Maps

Subjects are represented by subject proxies (*proxies*).

NOTE 1 Subjects such as books, cars, love and hate and relationships between subjects can all be represented using proxies.

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 set of *labels*, $\mathcal{L}$. Each label from this set corresponds to exactly one proxy and vice-versa; apart from that, these labels have no other significance. The second set postulated here is $\mathcal{V}$, a 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 set of all such properties is denoted as $\mathcal{P}$.

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

Keys in properties are always labels for proxies.

The value of a property may include, among other things, one or more labels of other subject proxies.

A *proxy* is a finite set of properties, $\{p_1, \ldots, p_n\}$, with $p_i \in \mathcal{P}$.

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

The set of all proxies is the set of all subsets of $\mathcal{P}$: $\mathcal{X} = 2^{\mathcal{P}}$

NOTE 2 Subject proxies are composed of properties, each being a statement about the proxy's subject. Properties can provide a basis for mapping multiple representatives of the same subject to each other.

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, then *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, syntax or data model.

4 Ontological Commitments

This Standard deliberately leaves undefined the methods whereby subject proxies are derived or created. As a consequence, no specific mechanism of subject identification is inherent in or mandated by this Standard, nor does this Standard define any subject proxies.

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

Two types of relationships, *isa* (instance of) and *sub* (subclass of), are defined by this Standard. Both are binary predicates over proxies, so that two given proxies are either in one of the above types of relationships, or not. The predicates are always interpreted *relative* to a given map m and can be used for inferencing:

a) Two proxies c, c' can be in a *subclass-superclass* relationship. In such a case, the same relationship can be stated either c *is a subclass of* c' or c' *is a superclass of* c, or more formally:

$$\mathrm{sub}_m \subseteq m \times m \tag{1}$$

sub_m is 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: $x \operatorname{sub}_m x$ 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. In such a case, the same relationship can be stated either a *is an instance of* c or c *is the type of* a, or more formally:

$$\operatorname{isa}_m \subseteq m \times m \tag{2}$$

The *isa* relationship is non-reflexive, i.e. $x \operatorname{isa}_m x$ 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 $x \operatorname{isa}_m c$ and $c \operatorname{sub}_m c'$, then $x \operatorname{isa}_m c'$ is true by inference.

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 `subclass` and `superclass`.

5 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:

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

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

$$x\!\uparrow_m = \{k \mid \exists y \in m \colon \langle k, x\rangle \in y\} \tag{4}$$

c) Postfix operators $\rightarrow k$ and $\rightarrow_m k^*$ to retrieve the *local values for a particular key* k:

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

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

$$x\!\rightarrow_m\! k^* = \{v \mid \exists \langle k', v\rangle \in x \colon k' \operatorname{sub}_m k\} \tag{6}$$

d) Postfix operators $\leftarrow_m k$ and $\leftarrow_m k^*$ which navigates to all proxies in the given map which use a given value v together with a certain key:

$$v\!\leftarrow_m\! k = \{x \in m \mid \exists \langle k, v\rangle \in x\} \tag{7}$$

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

$$v\!\leftarrow_m\! k^* = \{x \in m \mid \exists \langle k', v\rangle \in x \colon k' \operatorname{sub}_m k\} \tag{8}$$

It is straightforward to generalize all these navigation operators from individual proxies to multisets of proxies (or values). 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 denote as $\mathcal{P}_{\mathcal{M}}$.

NOTE 1 $\mathcal{P}_{\mathcal{M}}$ only serves as minimal baseline for functionality to be provided by conforming implementions. It can be also used as basis for a formal semantics for high-level query and constraint languages. Annex A describes one.

6 Constraints

Maps are defined by sets of constraints (see 8). A given map m either satisfies a constraint c or not.

NOTE 1 Constraints may conditionally or unconditionally require the addition of proxies to maps, properties to proxies, and/or values to properties. Constraints may also prohibit the existence (or addition) 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 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* if the application of the constraint results in a non-empty map.

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 declaratively, in any manner that is sufficiently expressive. It may also be defined in terms of the definitions provided by this Standard. Annex A demonstrates one way of defining $\otimes$.

The operator $\otimes$ is then used as basis to define a *satisfaction relation* $\models \subseteq \mathcal{M} \times \mathcal{C}$:

$$m \models c \qquad \Longleftrightarrow \qquad m \otimes c \neq \emptyset \tag{9}$$

7 Merging

Generic merging of maps only combines two (or more) proxy sets. Application-specific merging includes a second aspect. Applications may define constraints that include criteria for detecting when, in any given map, two or more proxies should be deemed to represent the same subject. When multiple proxies represent the same subject, they are replaced by a single proxy that includes all of their properties.

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.

Any merging criteria, as well as, for example, any processes whereby multiple properties are merged into single properties, are disclosed as constraints.

The result of applying such a constraint is a technically a *view* of the map.

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 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 $\bowtie$ operator is commutative and associative:

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

Additionally, $\bowtie$ is idempotent, as proxies merged with themselves equal themselves:

$$x \bowtie x = x$$

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:

$$m|_{\bowtie} = \{\bowtie c \mid c \in m/\bowtie\} \tag{10}$$

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 $m|_{\bowtie}{}^{*}$.

8 Map Legends

The legend of a subject map plays a role similar to the legend on more familiar city or road maps. The *legend* of a subject map is the 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.

A subject map legend is expressed as constraints on the properties of subject proxies (including their values), equivalence classes for subject proxies, merging rules for equivalence classes, and any other constraint found useful by the legend author. This Standard places no restrictions on the constraints that can be specified by a subject map legend.

A *map legend* (*legend*) $L = \{c_1, \ldots, c_n\}$ is finite set of constraints, all from a given constraint language $\mathcal{C}$.

A legend defines a map m such that $m \models c_i$ for all $c_i \in L$.

NOTE Subject maps are defined by their legends and have no existence in the absence of such legends.

NOTE 1 Depending on the definition of the constraint language, constraints can be combined (using logic operators like AND and OR. In such cases, legends can be used in whole or in part with other legends.

NOTE 2 Like all other representatives of subjects, constraints in legends and their components can themselves be represented by proxies. This Standard does not provide any semantics for such representations.

9 Conformance

An implementation conforms to this Standard if:

a) It presents all information to a user (human or otherwise) isomorphous to the subject map structure in Clauses 3;

b) It implements merging by providing a operator $\oplus$ according to Clause 6;

c) It implements access methods equivalent to those in Clause 5;

d) Those methods must honor the predicates from Clause 4;

e) It implements a constraint language by providing an operator $\otimes$ according to Clause 6;

f) It implements legends according to Clause 8.

Annex A
(normative)
τ Path Language

The primitive path language $\mathcal{P}_\mathcal{M}$ (clause 5) 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 $\otimes$.

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:

$$t = \prod_{i=1}^{n} \langle v_i\rangle = \langle v_1\rangle\langle v_2\rangle \ldots \langle v_n\rangle \tag{11}$$

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

$$s = \sum_{i=1}^{m} t_i = [t_1, \ldots, t_m] \tag{12}$$

if the sequence is *unordered* and $\vec{s}$ for *ordered sequences.* Otherwise, sequences behave like 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:

$$\sum s_i + \sum t_j = [s_1, \ldots, s_m, t_1, \ldots, t_n] \tag{13}$$

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:

$$(s)\left(\sum_{j=1}^{m}\langle v_{1\mathrm{j}}, v_{2\mathrm{j}}, \ldots, v_{\mathrm{lj}}\rangle\right) = \left(s\sum_{j=1}^{m}\langle v_{1\mathrm{j}}\rangle\right)\sum_{j=1}^{m}\langle v_{2\mathrm{j}}, \ldots, v_{\mathrm{lj}}\rangle \tag{14}$$

$$\sum_{i=1}^{n} t_i \sum_{j=1}^{m}\langle v_j\rangle = \sum_{i,j=1}^{nm}(t_i\langle v_j\rangle) \tag{15}$$

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, a order can be postulated on the keys and the values sorted according to it. Conversely, a value tuple can be converted into a proxy, provided that the tuple of keys is available:

$$\langle v_1, \ldots, v_n\rangle¥\langle k_1, \ldots, k_n\rangle = \{\langle k_1, v_1\rangle, \ldots, \langle k_n, \ldots, v_n\rangle\} \tag{16}$$

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. A sequence of proxies can then be interpreted as a map.

A.2 Path Expressions

Starting with a tuple sequence s, a path expression p, can be applied to it, which returns another tuple sequence. This operation is symbolized as $s\otimes_m p$. This operation is in the context of map m but if that is implicit, then the index may be dropped.

The set of path expressions $\mathcal{T}_\mathcal{M}$ is 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) For two path expressions p and q the *concatenation* $p \circ q$ is in $\mathcal{T}_\mathcal{M}$:

$$s \otimes (p \circ q) = (s \otimes p) \otimes q \tag{17}$$

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

c) For two path expressions p and q the *alternation* $p\|q$ is in $\mathcal{T}_\mathcal{M}$. The result of both path expressions are required to be added to the sequence s:

$$s \otimes (p\|q) = (s \otimes p) + (s \otimes q) \tag{18}$$

d) Every value from $\mathcal{V}$ is in $\mathcal{T}_{\mathcal{M}}$.

   If a constant value is applied to a sequence, then the sequence itself is discarded and a new sequence with a singleton tuple is create in which the value is used.

e) The *projection* postfix π_i is in $\mathcal{T}_{\mathcal{M}}$ for any positive integer i. The *tuple projection* $\langle p_1, \ldots, p_n \rangle$ with p_i being a path expression is also in $\mathcal{T}_{\mathcal{M}}$.

f) Given path expressions $p_1, p_2, \ldots, p_n$ and a function $f{:}\mathcal{V}^n \mapsto \mathcal{V}$ then $f(p_1, p_2, \ldots, p_n)$ is in $\mathcal{T}_{\mathcal{M}}$.

g) All navigational operators $\uparrow$, $\downarrow$, $\leftarrow k$, and $\rightarrow k$ are in $\mathcal{T}_{\mathcal{M}}$.

h) The *positive predicate* postfix [p = q] and the *negative predicate* postfix [p != q] are both in $\mathcal{T}_{\mathcal{M}}$ for two path expressions p and q. The special cases [p] and [!p] are included.

i) The postfix *uniq* is in $\mathcal{T}_{\mathcal{M}}$.

   When the postfix *uniq* is applied to a tuple sequence, it computes a tuple sequence in which every tuple sequence of the orginal sequence occurs exactly once. Iff the sequence was ordered, the result will be ordered as well.

j) Given an order $\leq$ on tuples, then $sort_{\leq}$ is in $\mathcal{T}_{\mathcal{M}}$.

   Based on a given ordering $\leq$ on proxies a sequence $s = \sum t_i$ can be ordered: $\vec{s} = \sum t_{i'}$, contains the same proxies and satisfies the constraint: $i' \leq j' \iff t_{i'} \leq t_{j'}$.

The binary operator $\otimes$ is used to symbolize the application of a path expression p to a map m. The operator is not commutative nor is it associative. The empty postfix ε is its identity element.

A.2.1 Filtering

Specific tuples can be filtered out from tuple sequences using predicates. Given a tuple sequence s and two path expressions p and q, applying the *positive predicate postfix* [p = q] to s is defined as

$$s \otimes [\ \mathrm{p} = \mathrm{q}\ ] = [t \in s \mid (t \otimes \mathrm{p}) \cap (t \otimes \mathrm{q}) \neq \emptyset] \tag{19}$$

Any existing ordering in s will be maintained. If p and q are identical, then [p = p] can be abbreviated with [p].

NOTE 1 The result of the positive predicate prefix is that sub-sequence of s for which elements the evaluation of p and q gives at least one common result.

NOTE 2 This implements an *exists semantics* as $s \otimes [\ \mathrm{p} = \mathrm{p}\ ]$ is reducable to $[t \in s \mid t \otimes \mathrm{p} \neq \emptyset]$. Only those tuples of s will be part of the result tuple sequence if there exists at least one result when p is applied to that tuple.

Negation in predicate postfixes implements *forall semantics*. Given a tuple sequence s and two path expressions p and q, the *negative predicate postfix* is defined as

$$s \otimes [\ \mathrm{p}\ \text{!=}\ \mathrm{q}\ ] = [t \in s \mid (t \otimes \mathrm{p}) \cap (t \otimes \mathrm{q}) = \emptyset] \tag{20}$$

Any ordering in s is preserved. If p and q are identical, then [p != p] can be abbreviated with [! p]. In this case the result tuple sequence becomes $[t \in s \mid t \otimes \mathrm{p} = \emptyset]$.

NOTE 3 A particular tuple will only then be part of the result tuple sequence if p applied to it will not render a single value, i.e. *all* evaluations will return no result.

NOTE 4 Logic conjunction and disjunction of predicate postfixes are implicit in the formalism. Logical *and* is provided by concatenating two predicate postfixes as the result of the first postfix will be further tested for the second predicate. The logical *or* between predicate postfixes is given by alternating them.

A.2.2 Functions

N-ary functions can be applied to tuple sequences. Given a function $f{:}\mathcal{V}^n \mapsto \mathcal{V}$, it can be interpreted as one which takes a value tuple of length n and renders one value. To apply a function to a complete tuple sequence, it is applied to every individual tuple. The individual results are then organized into a sequence of singleton values:

$$f(\sum t_i) = \sum \langle f(t_i) \rangle \tag{21}$$

A.2.3 Projection

The *projection postfix* is used to extract a certain column j from a tuple sequence:

$$\sum_{i=1}^{n} \langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}} \rangle \otimes \pi_j = \sum_{i=1}^{n} \langle v_{\mathrm{ji}} \rangle \tag{22}$$

NOTE 1 *Projection* plays a similar role to *like* in query languages like SQL, except that an index is used for selection instead of a pattern being applied against a character string.

Tuple projection is used to organize values into new tuple sequences. For a single tuple it evaluates all the path expressions and builds the cartesian product of all partial result sequences:

$$t \otimes \langle p_1 \ldots, p_n \rangle = \prod_{i=1}^{n} t \otimes p_{\mathrm{i}} \tag{23}$$

A.2.4 Navigation

Navigation postfixes are applied to a tuple sequence by applying it to every proxy tuple:

$$\left(\sum_{i=1}^{n} \langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}} \rangle \right) \otimes \leftarrow k \;=\; \sum_{i=1}^{n} \prod_{j=1}^{l} \langle v_{\mathrm{ji}} {\leftarrow_m} k^* \rangle \tag{24}$$

$$\left(\sum_{i=1}^{n} \langle v_{1\mathrm{i}}, v_{2\mathrm{i}}, \ldots, v_{\mathrm{li}} \rangle \right) \otimes \rightarrow k \;=\; \sum_{i=1}^{n} \prod_{j=1}^{l} \langle v_{\mathrm{ji}} {\rightarrow_m} k^* \rangle \tag{25}$$

This evaluation depends on a context map m.

NOTE 1 This process iterates over each tuple and computes an intermediate result for this one tuple. The intermediate result is achieved by applying the navigation to each value in the current tuple. That results in a set of values, that is implicitly converted into a tuple sequence. All these tuple sequences are multiplied, giving one intermediate result. All these intermediate results are then combined into the result.

A similar approach is used for finding keys:

$$\left(\sum_{i=1}^{n}\langle v_{\mathrm{1i}}, v_{\mathrm{2i}}, \ldots, v_{\mathrm{li}}\rangle\right) \otimes \downarrow \;=\; \sum_{i=1}^{n}\prod_{j=1}^{l}\langle v_{\mathrm{ji}} \downarrow\rangle \tag{26}$$

$$\left(\sum_{i=1}^{n}\langle v_{\mathrm{1i}}, v_{\mathrm{2i}}, \ldots, v_{\mathrm{li}}\rangle\right) \otimes \uparrow \;=\; \sum_{i=1}^{n}\prod_{j=1}^{l}\langle v_{\mathrm{ji}}\uparrow_{m}\rangle \tag{27}$$

Annex B
(informative)
Topic Maps Data Model (ISO 13250-2) Mapping

NOTE **Eds:** The text of the Topic Maps Data Model was not final as of this mapping.

NOTE **Eds:** Other mappings are possible. This mapping is subject to future revision.

This annex maps the constructs of the Topic Maps Data Model (TMDM) to this Standard.[1)]

As used herein, paired braces {} signal the enclosure of all the properties of a proxy. Paired parentheses () enclose the key/value pair, written in that order, that composes a property. The appearance of paired braces {} within paired parentheses () signal a value that is a set of values.

B.1 Subject Maps

The components of a *topic map* are: *topic map*, *topic*, *topic name*, *variant name*, *occurrence*, *association*, and *association role.* As will be seen in Table A.1, these constructs are 'finite set[s] of properties, $\{p_1, \ldots, p_n\}$,' therefore they are subject proxies as defined by this Standard.

A *subject map* is defined in this Standard as: 'a set of subject proxies.' (ISO/IEC 13250-5, 2 Subject Proxies and Subject Maps)

Since a *topic map* is a set of constructs, which are subject proxies as defined by this Standard, a topic map construct is a subject map as defined by this Standard.

B.2 Common Models

Models of matters common to several or all topic map constructs are given here to avoid unnecessary repetition.

B.2.1 Datatypes

Three datatypes, *string*, *IRI*, and *XML* are identified with locators in the TMDM. Despite their use in topic map constructs, *set* and *null* have no locators provided.

The general model for datatypes is to create a proxy to represent the key in question:

$\{(key, locator) \ldots)\}$

and then model that key with a proxy as an instance of *datatype*:

$\{(instance, key), (type, datatype)\}$

B.2.2 Properties

The key/value pairs defined by the TMDM match:

[1)]This mapping has benefited from prior efforts to create a mapping from the TMDM to this Standard, including: *TMDM mapping to TMRM* by Lars Marius Garshol, N0639; *A TMDM disclosure using* τ^+ by Robert Barta and Lar Neuer, and, the *ISO SC34 WG3 Reference Model Workshop*, N0533. Its conclusions, however, are the sole responsibility of the editors.

> A property is a tuple that consist of the key k and the value v in the following expression: $\langle k, v \rangle \in (\mathcal{X} \times V)$.

and therefore are properties as defined by the TMRM.

The forty-eight (48) key-value pairs as defined by the TMDM are summarized at table A.2. Every key is used as a label for a proxy without further explanation in the following modeling.[2)]

The TMDM does not prohibit the presence of additional properties in any of the subject proxies it defines or constraints on those properties, which may lead to merging behavior that is not defined by the TMDM.

B.2.3 Properties of Proxies (items)

The TMDM defines properties that are required to appear with each of the seven topic map constructs listed in Table A.1. Those properties and their required values are constraints imposed upon the proxy equivalents of those constructs by the TMDM.

B.2.4 Scope

Scope is available only for *topic name*, *variant name*, *occurrence*, and *association.*

Scope is modeled in all four cases with the key 'scope' inside the item to be scoped as more closely resembling the TMDM:[3)]

$\{(scope, item.scope), \ldots\}$

B.2.5 Type

Type occurs only on *topic name*, *occurrence*, *association*, and *association role.*

All types are modeled with dedicated proxies:

$\{(instance, item), (type, item.type)\}$

B.3 Topic Map

A topic map is composed of a set of constructs that can be represented as follows:

$$topic\ map = \left\{ \begin{array}{l} (topics, \{topics\}) \\ (topic\ names, \{topic\ names\}) \\ (variant\ names, \{variant\ names\}) \\ (occurrences, \{occurrences\}) \\ (associations, \{associations\}) \\ (association\ role, \{association\ role\}) \end{array} \right\}$$

A *topic map* is itself a set of key, value pairs and so is by definition also a proxy as defined by

2) This Standard does not compel instantiation of these proxies nor requires such instantiated proxies to have particular properties. The values of those keys are constrained by the defintions given in the TMDM.

3) Readers should carefully distinguish between 'scope' and '[scope]' in Table A.2. The brakets were retained as an convenient means to resolve the ambiguity between the terms.

this Standard. However, as will be seen under *Topic Map Item*, a *topic map* cannot appear in an *association role* in an *association* unless it is represented by a *topic*. [4)]

The representation of a *topic map* by a *topic* so that it can appear in an *association role* in an *association* is to enable additional statements being made about the *topic map* as a subject.

B.4 Topic Map Item

The *topic map item* has properties that can be modeled as:

$$topic\ map\ item = \left\{ \begin{array}{l} (topics, \{topics\}) \\ (associations, \{associations\}) \\ (reifier, \{topic\ item\ or\ null\}) \\ (item\ identifiers, \{locators\}) \end{array} \right\}$$

The *topic map item* has no defined subject sameness test nor merging defined by the TMDM.

B.5 Topic Item

The *topic item* has properties[5)] that can be modeled as:

$$topic = \left\{ \begin{array}{l} (topic\ names, \{topic\ name\ items\}) \\ (occurrences, \{occurrence\ items\}) \\ (roles\ played, \{association\ role\ items : computed\ value\}) \\ (subject\ identifiers, \{locators\}) \\ (subject\ locators, \{locators\}) \\ (reified, \{information\ item\ or\ null :\ computed\ value\}) \\ (item\ identifiers, \{locators\}) \\ (parent, \{information\ item : computed\ value\}) \end{array} \right\}$$

B.5.1 Constraints: subject identifiers, subject locators

The TMDM constrains the *subject identifiers* property to be locators that point at *subject indicators*, that is resources that indicate the subject represented by the *topic*. The *subject locators* property is constrained to be locators that point at information resources that are the subject being represented by the topic.

Since different observers will have different opinions about the distinction made by the TMDM with regard to locators, this constraint is not enforceable by any algorithmic means. Nor is any resolution deterministic. However, the TMRM does not require constraints to be enforceable algorithmically nor deterministic, merely that they be disclosed.

Subject identifiers and *subject locators* share a common model in the TMRM:

$r = \{(topic, t), (reifier, l)\}$
$\{(instance, r), (type, subjectidentifier/subjectlocator\}$

4) This constraint holds for all topic map constructs other than *topics*.

5) *Topic items* can have *type*, *instance*, *supertype*, and *subtype* properties represented by means of an *association*. It is not clear if the *scope* property is excluded by that listing, as there is no stated constraint on the properties that can be a part of a topic item beyond those defined by the TMDM.

Where *subject indentifier* appears in the model for subjects that are not 'information resources' and *subject locators* appear for subjects that are information resources.

B.5.2 Constraints: roles played, reified, parent

The TMDM constrains the values of *roles played*, *reified*, *parent* to be computed from the [player] property of *association role items* where the topic appears, the locators in the [item identifiers] property of a topic map construct whose [reifier] property contains a locator to that topic, and the *topic map* containing the topic, respectively.

B.5.3 Constraint: Topic Identity

All topic items are subject to the following constraint:

$$\left\{\begin{array}{l} subject\ identifiers \\ subject\ locators \\ item\ identifiers \end{array}\right\} \neq \emptyset$$

B.5.4 Equivalence Class Definition

The equivalence class definition is used by the $\bowtie$ operator to determine if two or more topic items form a class of equivalent topic items subject to merging. Topic items **A** and **B** are equivalent if any of the following conditions are meet:

a) $A.subject\ identifier\ values \cap B.subject\ identifier\ values \neq \emptyset$

b) $A.item\ identifier\ values \cap B.item\ identifier\ values \neq \emptyset$

c) $A.subject\ locator\ values \cap B.subject\ locator\ values \neq \emptyset$

d) $A.subject\ identifier\ values \cap B.item\ identifier\ values \neq \emptyset$

e) $A.reified \cap B.reified \neq \emptyset$

B.5.5 Merging Topic Items in an Equivalence Class

The $\bowtie$ operator is triggered by *topic items* meeting the definition set forth in the equivalence class **and** where the *parent* properties of both *topic items* have the same *topic map item.*

Constraint: It is an error if the *topic items* to be merged have different non-null values in their respective *reified* properties. [6)]

The merging procedure is as follows:

a) Create a new topic item C.

b) Replace all appearances of topic items A and B in the *topics*, *scope*, *type*, *player*, and *reifier* properties of other topic map constructs with C.

c) $C.topicNames.values = A.topicNames.values \cup B.topicNames.values$

[6)] It is not clear whether this is a merging error or a general error in the absence of merging.

d) $C.occurrences.values = A.occurrence.values \cup B.occurrences.values$

e) $C.subjectIdentifiers.values = A.subjectIdentifiers.values \cup B.subjectIdentifiers.values$

f) $C.subjectLocators.values = A.subjectLocators.values \cup B.subjectLocators.values$

g) $C.itemIdentifiers.values = A.itemIdentifiers.values \cup B.itemIdentifiers.values$

The merging of the properties of the two topic items results in a new proxy whose properties are the union of property values of the topic items that are merged.[7)]

B.6 Topic Name Item

The *topic name item* has properties that can be modeled as:

$$\textit{topic name item} = \left\{ \begin{array}{l} (value, \{string\}) \\ (type, \{topic\ item\}) \\ (scope, \{topic\ items\}) \\ (variants, \{variant\ name\ items\}) \\ (reifier, \{topic\ item\ or\ null\}) \\ (item\ identifiers, \{locators\}) \\ (parent, \{information\ item:\ computed\ value\}) \end{array} \right\}$$

B.6.1 Equivalence Class Definition

The equivalence class definition is used by the $\bowtie$ operator to determine if two or more topic name items form a class of equivalent topic name items subject to merging. Topic name items **A** and **B** are equivalent if all of the following conditions are meet:

a) $A.value = B.value$

b) $A.type.value = B.type.value$

c) $\{A.scope.value\} = \{B.scope.value\}$

d) $A.parent = B.parent$

B.6.2 Merging Topic Name Items in an Equivalence Class

The $\bowtie$ operator is triggered by *topic name items* meeting the definition set forth in the equivalence class.

The merging procedure is as follows:

a) Create a new topic name item C.

b) $C.value = A.value$

c) $C.type.value = A.type.value$

[7)]While not stated explicitly in the TMDM, this model presumes that the computed values for the properties *roles played*, *reified*, and *parent* are calculated for C after merging.

d) $C.scope.value = A.scope.value$

e) $C.variants.value = A.variants.values \cup B.variants.values$

f) $C.reifier.value = A.reifier.value\ if\ A.reifier.value \neq null$
$C.reifier.value = (B.reifier.value)\ if\ B.reifier.value \neq null$
$if A.reifier.value \neq null\ and\ B.reifier.value \neq null,\ then$
$C.reifier.value = New.reifier.value\ from\ A.reifier \bowtie B.reifier$ [8)]

g) $C.itemIdentifiers.values = A.itemIdentifiers.values \cup B.itemIdentifiers.values$

h) $parent.topicName.value = C$

The merging of the properties of the two topic name items results in a new proxy whose properties are the union of property values of the topic name items that were merged.

B.7 Variant Item

The *variant item* has properties that can be modeled as:

$$topic\ name\ item = \left\{ \begin{array}{l} (value, \{string\}) \\ (datatype, \{locator\}) \\ (scope, \{topic\ items\ \ \neq \emptyset) \\ (reifier, \{topic\ item\ or\ null\}) \\ (item\ identifiers, \{locators\}) \\ (parent, \{information\ item:\ \ computed\ value\}) \end{array} \right\}$$

B.7.1 Constraint on Variant Item Scope

B.7.2 Equivalence Class Definition

The equivalence class definition is used by the $\bowtie$ operator to determine if two or more variant items form a class of equivalent variant items subject to merging. Variant items **A** and **B** are equivalent if all of the following conditions are meet:

a) $A.value = B.value$

b) $A.datatype.value = B.datatype.value$

c) $\{A.scope.value\} = \{B.scope.value\}$

d) $A.parent = B.parent$

B.7.3 Merging Variant Items in an Equivalence Class

The $\bowtie$ operator is triggered by *variant items* meeting the definition set forth in the equivalence class.

[8)]Note that the merging compelled here and similar provisions elsewhere will fail due to conflict with TMDM 6.2, which prohibits merging of topic items with different non-null values in their [reified] properties. There is no updating of that property required after merging the items in question, which means by definition the topic items are going to have different values, assuming the items are reified, in their [reified] properties.

The merging procedure is as follows:

a) Create a new variant item C.

b) $C.value = A.value$

c) $C.datatype.value = A.datatype.value$

d) $C.scope.value = A.scope.value$

e) $C.reifier.value = A.reifier.value\ if\ A.reifier.value \neq null$
$C.reifier.value = (B.reifier.value)\ if\ B.reifier.value \neq null$
$if A.reifier.value \neq null\ and\ B.reifier.value \neq null,\ then$
$C.reifier.value = New.reifier.value\ from\ A.reifier \bowtie B.reifier$

f) $C.itemIdentifiers.values = A.itemIdentifiers.values \cup B.itemIdentifiers.values$

g) $parent.variants.value = C$

The merging of the properties of the two variant items results in a new proxy whose properties are the union of property values of the variant items that were merged.

B.8 Occurrence Item

The *occurrence item* has properties that can be modeled as:

$$occurrence\ item = \left\{ \begin{array}{l} (value, \{string\}) \\ (datatype, \{locator\}) \\ (scope, \{topic\ items\}) \\ (type, \{topic\ item\}) \\ (reifier, \{topic\ item\ or\ null\}) \\ (item\ identifiers, \{locators\}) \\ (parent, \{information\ item:\ computed\ value\}) \end{array} \right\}$$

B.8.1 Equivalence Class Definition

The equivalence class definition is used by the $\bowtie$ operator to determine if two or more occurrence items form a class of equivalent occurrence items subject to merging. Occurrence items **A** and **B** are equivalent if all of the following conditions are meet:

a) $A.value = B.value$

b) $A.datatype.value = B.datatype.value$

c) $\{A.scope.value\} = \{B.scope.value\}$

d) $A.type.value = B.type.value$

e) $A.parent.value = B.parent.value$

B.8.2 Merging Occurrence Items in an Equivalence Class

The $\bowtie$ operator is triggered by *occurrence items* meeting the definition set forth in the equivalence class.

The merging procedure is as follows:

a) Create a new occurrence item C.

b) $C.value = A.value$

c) $C.datatype.value = A.datatype.value$

d) $C.scope.value = A.scope.value$

e) $C.type.value = A.type.value$

f) $C.reifier.value = A.reifier.value\ if\ A.reifier.value \neq null$
$C.reifier.value = (B.reifier.value)\ if\ B.reifier.value \neq null$
$if A.reifier.value \neq null\ and\ B.reifier.value \neq null,\ then$
$C.reifier.value = New.reifier.value\ from\ A.reifier \bowtie B.reifier$

g) $C.itemIdentifiers.values = A.itemIdentifiers.values \cup B.itemIdentifiers.values$

h) $parent.occurrences.value = C$

The merging of the properties of the two occurrence items results in a new proxy whose properties are the union of property values of the occurrence items that were merged.

B.9 Association Item

The *association item* has properties that can be modeled as:

$$association\ item = \left\{ \begin{array}{l} (type, \{topic\ item\}) \\ (scope, \{topic\ items\}) \\ (roles, \{association\ role\ items,\ \neq \emptyset\}) \\ (reifier, \{topic\ item\ or\ null\}) \\ (item\ identifiers, \{locators\}) \\ (parent, \{information\ item:\ computed\ value\}) \end{array} \right\}$$

B.9.1 Equivalence Class Definition

The equivalence class definition is used by the $\bowtie$ operator to determine if two or more association items form a class of equivalent association items subject to merging. Association items **A** and **B** are equivalent if all of the following conditions are meet:

a) $\{A.scope\} = \{B.scope\}$

b) $A.type = B.type$

c) $\{A.roles.value\} = \{B.roles.value\}$

B.9.2 Merging Association Items in an Equivalence Class

The $\bowtie$ operator is triggered by *association items* meeting the definition set forth in the equivalence class.

The merging procedure is as follows:

a) Create a new association item C.

b) $C.type.value = A.type.value$

c) $C.scope.value = A.scope.value$

d) $C.roles.value = A.roles.value$

e) $C.reifier.value = A.reifier.value\ if\ A.reifier.value \neq null$
$C.reifier.value = (B.reifier.value)\ if\ B.reifier.value \neq null$
$ifA.reifier.value \neq null\ and\ B.reifier.value \neq null,\ then$
$C.reifier.value = New.reifier.value\ from\ A.reifier \bowtie B.reifier$

f) $C.itemIdentifiers.values = A.itemIdentifiers.values \cup B.itemIdentifiers.values$

g) $parent.association.value = C$

The merging of the properties of the two association items results in a new proxy whose properties are the union of property values of the association items that were merged.

B.10 Association Role Item

The *association role item* has properties that can be modeled as:

$$association\ role\ item = \left\{ \begin{array}{l} (player, \{topic\ item\}) \\ (type, \{topic\ item\}) \\ (reifier, \{topic\ item\ or\ null\}) \\ (item\ identifiers, \{locators\}) \\ (parent, \{information\ item:\ computed\ value\}) \end{array} \right\}$$

B.10.1 Constraint on Role Player

The TMDM imposes constrait that only a *topic item* can be used as the value for the *role player* property. This means that *topic map*, *topic name*, *variant*, *occurrence*, *association* and *association role* items must be represented by a topic proxy in order to be a role player in an association.

That relationship is modeled by:

$\{(topic, reifier), (item, reified)\}$

B.10.2 Equivalence Class Definition

The equivalence class definition is used by the $\bowtie$ operator to determine if two or more association role items form a class of equivalent association role items subject to merging. Association role items **A** and **B** are equivalent if all of the following conditions are meet:

a) $A.type.value = B.type.value$

b) $A.player.value = B.player.value$

c) $A.parent = B.parent$

B.10.3 Merging Association Role Items in an Equivalence Class

The $\bowtie$ operator is triggered by *associaton role items* meeting the definition set forth in the equivalence class.

The merging procedure is as follows:

a) Create a new association role item C.

b) $C.player.value = A.player.value$

c) $C.type.value = A.type.value$

d) $C.itemIdentifiers.values = \{A.itemIdentifiers.values\} \cup \{B.itemIdentifiers.values\}$

e) $C.reifier.value = A.reifier.value\ if\ A.reifier.value \neq null$
$C.reifier.value = B.reifier.value\ if\ B.reifier.value \neq null$
$if A.reifier.value \neq null\ and\ B.reifier.value \neq null,\ then$
$C.reifier.value = New.reifier.value\ from\ A.reifier \bowtie B.reifier$

f) $parent.roles.value = C$

The merging of the properties of the two association role items results in a new proxy whose properties are the union of property values of the association role items that were merged.

Property	topic map	topic	topic name	variant	occurrence	association	association role
associations	X						
datatype			X	X			
item identifiers	X	X	X	X	X	X	X
occurrences	X	X					
parent		X	X	X	X	X	X
player							X
reifier	X		X	X	X	X	X
roles						X	
roles played		X					
scope			X	X	X	X	
subject identifiers		X					
subject locators		X					
topic names		X					
topics	X						
type			X		X	X	X
value			X	X	X		
variants			X				

Table B.1 – Topic Map Data Model Constructs with Properties

Table B.2 – Topic Map Data Model Properties

Key	Value (by constraint or fixed value)
association	http://psi.topicmaps.org/iso13250/glossary/association
association role	http://psi.topicmaps.org/iso13250/glossary/association-role
association role type	http://psi.topicmaps.org/iso13250/glossary/association-role-type
association type	http://psi.topicmaps.org/iso13250/glossary/association-type
associations	A set of association items
datatype	A locator
information resource	http://psi.topicmaps.org/iso13250/glossary/information-resource
IRI	http://www.w3.org/2001/XMLSchema#anyURI
item identifier	http://psi.topicmaps.org/iso13250/glossary/item-identifier
item identifiers	A set of locators
locator	http://psi.topicmaps.org/iso13250/glossary/locator
merging	http://psi.topicmaps.org/iso13250/glossary/merging
null	Property has no value.
occurrence	http://psi.topicmaps.org/iso13250/glossary/occurrence
occurrence type	http://psi.topicmaps.org/iso13250/glossary/occurrence-type
occurrences	A set of occurrence items
parent	An information item
player	A topic item
reification	http://psi.topicmaps.org/iso13250/glossary/reification
reifier	A topic item, or null
roles	A non-empty set of association role items
roles played	A set of association role items
scope	http://psi.topicmaps.org/iso13250/glossary/scope
[**scope**]	A set of topic items
set	Collections of elements with no common elements
statement	http://psi.topicmaps.org/iso13250/glossary/statement
string	http://www.w3.org/2001/XMLSchema#string
subject	http://psi.topicmaps.org/iso13250/glossary/subject
subject identifier	http://psi.topicmaps.org/iso13250/glossary/subject-identifier
subject identifiers	A set of locators.
subject indicator	http://psi.topicmaps.org/iso13250/glossary/subject-indicator
subject locator	http://psi.topicmaps.org/iso13250/glossary/subject-locator
subject locators	A set of locators
topic	http://psi.topicmaps.org/iso13250/glossary/topic
topic map	http://psi.topicmaps.org/iso13250/glossary/topic-map
topic map construct	http://psi.topicmaps.org/iso13250/glossary/topic-map-construct
Topic Maps	http://psi.topicmaps.org/iso13250/glossary/Topic-Maps
topic name	http://psi.topicmaps.org/iso13250/glossary/topic-name
topic names	A set of topic name items
topic name type	http://psi.topicmaps.org/iso13250/glossary/topic-name-type
topic type	http://psi.topicmaps.org/iso13250/glossary/topic-type
topics	A set of topic items
type	A topic item
unconstrained scope	http://psi.topicmaps.org/iso13250/glossary/unconstrained-scope
value	A string
variant name	http://psi.topicmaps.org/iso13250/glossary/variant-name
variants	A set of variant name items.
XML	http://www.w3.org/2001/XMLSchema#anyType