Merging Topics in Well-Formed XML Topic Maps

Richard Widhalm and Thomas A. Mueck

University Of Vienna
Institute for Computer Science and Business Informatics
Rathausstr. 19
1010 Vienna, Austria
rwidhalm@gmx.at
mueck@ifs.univie.ac.at

Abstract. Topic Maps are a standardized modelling approach for the semantic annotation and description of WWW resources. They enable an improved search and navigational access on information objects stored in semi-structured information spaces like the WWW. However, the according standards ISO 13250 and XTM (XML Topic Maps) lack formal semantics, several questions concerning e.g. subclassing, inheritance or merging of topics are left open. The proposed TMUML meta model, directly derived from the well known UML meta model, is a meta model for Topic Maps which enables semantic constraints to be formulated in OCL (object constraint language) in order to answer such open questions and overcome possible inconsistencies in Topic Map repositories. We will examine the XTM merging conditions and show, in several examples, how the TMUML meta model enables semantic constraints for Topic Map merging to be formulated in OCL. Finally, we will show how the TM validation process, i.e., checking if a Topic Map is well formed, includes our merging conditions.

1 Introduction

The Topic Maps standard ISO 13250 [7] as well as XTM (XML Topic Maps) [12] provide for the semantic characterization of information objects across the WWW or company-controlled intranet or extranet platforms. They externally describe the underlying information objects (documents, web pages, etc.) without changing them. With Topic Maps, which are syntactically based on XML (or SGML for ISO13250), semantic networks consisting of topics and their relationships can be built, thus enhancing the flexibility of search queries and navigational access on the underlying information objects. They can be used as a flexible, generic index for knowledge bases. In a topic map, every real world subject (equal if a WWW resource or an abstract thing) is represented by a *topic*. Each topic may be an instance of several *topic types* (which are also topics), may have several lexical *names* in different *scopes* (where a scope is the area where the according name for the topic is said to be valid and each scope

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 64–79, 2003.

is described by a set of topics) and may show several *occurrences* in different WWW resources. An occurrence is a link, typed by a topic, to an information object. Topics can be interrelated via n-ary *associations*, where each topic plays a certain *role* which is again expressed by another topic. Associations can be typed via a topic. Further, it is possible to create generalization hierarchies for topic types, association types and occurrence types. Occurrences, names and associations can be placed within a scope. Each topic T has an *identificator*, which can be a WWW resource or another topic, indicating or itself being the subject for which T stands. Furthermore, Topic Maps define two topics T_1 and T_2 as identical, if they are identified by the same resource or if they both exhibit the same name N in the same scope S. In that case, they shall be merged to a single topic bearing all the characteristics of the original topics. In this paper, we will focus on the issue of merging topics. We will show how the TMUML meta model lets one formulate the actual XTM merging conditions with the OCL (*object constraint language*), defined within the UML specification [9]. We will also show specific circumstances under which merging would not be desirable, although the XTM criteria would be met. Therefore, we will inspect the consequences of the merging of two topics of a different kind (like an association type merged with an occurrence type). Consequently, we present suggestions for additional OCL constraints showing how the TMUML meta model can help to overcome such issues. Of course, the presented constraints are only considered as suggestions that may further be extended or only partially applied according to specific needs. The following section briefly introduces the TMUML meta model, succeeded by an overview about related work. Afterwards, we will describe the synonymy relation used for merging in the TMUML meta model. Chapter 5 expresses the XTM merging rules in OCL and adds a further, implicated merging rule. Afterwards, we will inspect situations of merging different kinds of topics (like association types, occurrence types, topic types or topic instances) and show the processing steps within our XTM validation and the functionality of the OCL checker. The conclusion will point out the meaningfulness of merging in largely designed Topic Maps systems and give a prospect on our further work.

2 Topic Maps and the TMUML Meta Model

In [19] and [20], the TMUML meta model has been presented, including a detailed description as well as the according class diagrams. It is a meta model for Topic Maps, directly derived from the UML meta model. It does not make use of all of the UML meta model components, only those which are also relevant for Topic Maps (like `Association` or `GeneralizableElement`). Note that topics are divided into the meta classes `TopicType` (for topics used as types for other topics), `TopicObject` (topics not functioning as types but as instances of types), `AssociationTopicType` (similar to the UML `AssociationClass`, representing types for association instances) and `OccurrenceType` (types for occurrences). The `TopicAssociationEnd` defines, like the `AssociationEnd` in UML, the allowed `TopicType` for a role within the according `TopicAssociation`, and also

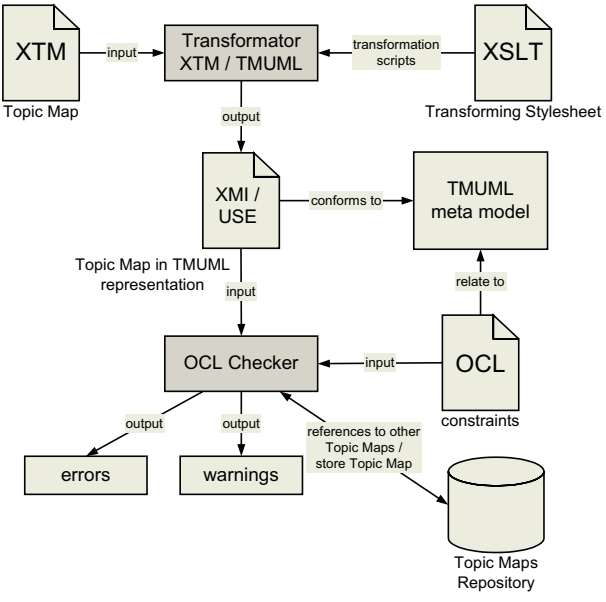

Fig. 1. Topic Maps validation process

its multiplicity. Base names and its subvariants are all represented by `TopicName` (which may be part of a name hierarchy). See [19] and [20] for a detailed description of the TMUML meta model. The TMUML meta model enables our process for validating the well-formedness of a topic map. Fig. 1 schematically shows our approach in the context of validating an XTM topic map before storing it within a repository.

First, the XTM topic map is transformed into an UML representation. For the transformation, XSLT would be an appropriate method, while for the representation, the XMI metadata interchange format [10] or the USE syntax (see [5]) could be used. While the first alternative provides more openness, the second will fit perfectly if using the USE tool for validating OCL constraints. The OCL checker is the component which is responsible for validating the UML representation of the topic map against the OCL constraints that make up well-formedness. For our testing purposes, we used the USE tool, but any other tool capable of OCL validation may be used instead. The OCL checker is then responsible for generating appropriate error and warning messages. After proving a topic map to be valid, it can be inserted into a repository. Also, reading repository data may be important for the OCL checker, in case topics could be merged according to the name based merging principle (see section 5).

The following example shows the definition of a simple OCL constraint, stating that no `Topic` shall be identified by itself.

```
context Topic
 inv C_Topic_identifiedByItself:
  not self.identificator()->exists(e | e = self)
```

It makes use of the operation `identificator()`, which is defined in the context of `Topic` and retrieves all `Identificators` that are related to the context `Topic` via an `Identification` relationship in the TMUML meta model. An `Identificator` may be a subject constituting resource, a subject indicating resource or another topic (see [12]).

```
identificator() : Set(Identificator) =
 self.identifiedBy->collect(i : Identification | i.identificator)->asSet
```

3 Related Work

An early work on constraining Topic Maps is [14], where several suggestions for constraints at the instance layer of a topic map are given (without a formal method). An overview about the idea behind constraints for Topic Maps in general can be found in [6]. In [11], Ontopia describes their own solution for a Topic Maps constraint language, called OSL (*Ontopia Schema Language*). It allows for a more constrained and precisely defined Topic Maps schema definition, but again is not usable for constraining the Topic Maps meta model. With OSL, the types and roles allowed within certain association types may be specified, but, for example, the effects of subclassing association types in general are not treated (the semantics of inheritance). Our method is best suitable for solving the second kind of problem, while it may also be used for the first kind. A similar approach is sketched by the TMCL [13], the *Topic Maps Constraint Language*, although it has not yet surpassed the level of a requirements suggestion. [2] presents a formal model for Topic Maps, consisting only of topics and associations, which are contained in a *homogenous* hypergraph. Meta associations like instancing, subclassing or scoping are contained in a *shift* hypergraph, that compounds components of the homogenous hypergraph and thus establishes semantic layers (e.g. separating role types, topic types or association types). Although this approach applies formal semantics to Topic Maps using hypergraphs, it does not mention additional constraints on the Topic Maps meta model or how they can be applied. Our approach goes beyond that and provides means for the automation of semantic constraints using OCL and USE (or comparable OCL tools). Moreover, the hypergraph model does only distinguish between topics and associations, while names, occurrences or resources have to be modelled by topics. Actually, names have different properties compared to topics in the Topic Maps meta model. Names can not be merged or have occurrences or have names etc. Further, there may be difficulties due to the fact that the semantics of the associations in the shift hypergraph (denoted as θ) is only determined by the semantic layers of the connected topics. For example, there may be different meanings of connections from a role type to a topic type, one may mean a "subclass" relation, the other may mean a type constraint for the topic members playing the role type in the according association type.

Additional information on Topic Maps can be found in [8], [18], [1] and [3]. The first gives an introduction to the fundamental concepts of Topic Maps and introduces a system architecture for applications based on a distributed

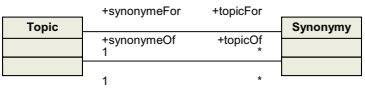

Fig. 2. The Synonymy relation

Topic Maps repository. The second describes a webbased search engine based on a Topic Maps repository, from the conceptual approaches up to a pseudocode, including a prototypic query language. The third publication discusses a range of possible application areas for Topic Maps, while the fourth presents preliminary guidelines to the design of certain topic maps. Detailed information on OCL can be found in the according specification [9] and in [17]. In [4], uncertainties in the semi-formal OCL specification are discussed. In [16], a formal semantics for the OCL is introduced.

4 The Synonymy Relation

XTM defines that two topics T_1 and T_2 shall be merged to one single topic T_3, that inherits all characteristics of T_1 and T_2 (names, occurrences, association memberships, types), if T_1 and T_2 meet some merging criteria and can therefore be deemed to reify (stand for) the same subject. Unfortunately, this would mean that they could not be separated from each other. This should be possible in the context of a Topic Maps repository: in the case one wants to remove the topic map TM_1, which contains T_1, T_2 which belongs to the topic map TM_2 should remain in the repository, in the state it was before the merging with T_1. For this reason, we will keep topics which shall be merged separated in the TMUML meta model, but interrelate them via a `Synonymy` relationship, an equivalence relationship which we have to introduce in our TMUML meta model. Fig. 2 shows the additional class `Synonymy` and its associations.

When removing a previously merged `Topic` T, the topics which have been merged with T can still remain in the repository, only their `Synonymy` link to T will be removed.

In our set of constraints, every time we refer to a Topic T or its characteristics, we also have to refer to the characteristics of all the `Topics` $TSYN_i, i = 1, .., n$, which are directly or transitively connected to T via a `Synonymy` relation. We do this by providing an OCL operation called `closure()` yielding the transitive closure of a `Topic` T, formally
$\{T \cup \{TSYN_i | TSYN_i$ is (transitively) synonymous to $T; i = 1, .., n\}\}$.

```
Topic::closure() : Set (Topic) =
 let strt : Set(Topic) = oclEmpty(Set(Topic)) in
  self.synonymesRec (strt)
```

`strt` denotes a local variable of the type `Set(Topic)`, initialized with an empty set of `Topics`. `synonymesRec` is an operation in the context of `Topic` that takes a set S of `Topics` as argument and returns for a `Topic` T a set of `Topics`

containing T and the Topics returned by recursively calling synonymesRec (which takes S extended by T) for all Topics directly connected to T via the Synonymy relation.

```
Topic::synonymesRec(s : Set(Topic)) : Set(Topic) =
 Set{self}->union (
  self.topicOf.synonymeFor->select (f |
   s->excludes(f)).synonymesRec (s->including(self))
  )->union (
  self.topicFor.synonymeOf->select (f |
   s->excludes(f)).synonymesRec (s->including(self))
  )->asSet
```

Thus, by calling T.closure(), we receive T and all other Topics that are synonymous to T. Further, to determine whether two Topics are equal, we can not simply compare the two Topics, we have to find out if one is contained within the closure of the other. We therefore provide the operation Topic::equals(), which is defined as follows.

```
Topic::equals(t : Topic) : Boolean =
 self.closure()->exists (c_t | c_t = t)
```

5 Merging Conditions

XTM defines two merging conditions. The *naming constraint-based merge* [12] says that two Topics T_1 and T_2 have to be merged if T_1 has a base name N_1 in the scope S_1, and T_2 has a base name N_2 in scope S_2, and N_1 and N_2 are equal strings and S_1 and S_2 are equal sets of Topics. We define this condition as an OCL operation called Topic::nameBasedMerge(b: Topic), which checks whether the context Topic may be merged with b due to the naming constraint-based merge.

```
Topic::nameBasedMerge(b : Topic) : Boolean =
 (self.closure().nameForTopic->exists (s_n |
  b.closure().nameForTopic->exists (b_n |
   s_n.value = b_n.value and s_n.equalScope (b_n))))
```

Here, ScopableElement::equalScope (s: ScopableElement) is used to compare the scopes of two ScopableElements, which are names in our case (Occurrences and Associations are also ScopableElements. It uses two other operations, which are also defined within the context of ScopableElement.

```
ScopableElement::scope() : Set(ScopingElement) =
 self.scopedBy.scope->asSet
ScopableElement::compareScope (other: ScopableElement) : Boolean =
 self.scope()->forAll (
  self_t | other.scope()->select(
   other_t | other_t.oclAsType(Topic).equals(self_t.oclAsType(Topic))
  )->size = 1)
ScopableElement::equalScope (s : ScopableElement) : Boolean =
 self.compareScope (s) and s.compareScope (self)
```

Note that `Topic.nameForTopic` yields all `TopicNames` for a `Topic`, `TopicName.value` is the lexical represenation of a name, and `ScopableElement.scope()` yields the set of `Topics` (which are `ScopingElements`) making up the scope of a `ScopableElement`.

The other XTM merging condition is called *subject-based merge* [12] and says that two `Topics` T_1 and T_2 have to be merged if T_1 has an `Identificator` I_1, and T_2 an `Identificator` I_2, where the URIs of I_1 and I_2 are equal or $T_1 = I_2$ or $T_2 = I_1$. Note that also in this operation we have to excessively use `Topic::closure()`.

```
Topic::subjectBasedMerge(b : Topic) : Boolean =
 (self.closure().identificator()->select (i |
  i.oclIsTypeOf (Topic)).oclAsType(Topic).closure()->includes (b))
 or (self.closure().identificator()->exists (
   s_i | b.closure().identificator()->exists (
    b_i | b_i.URI = s_i.URI ))
  )
 or (self.closure().identificator()->select (i |
   i.oclIsTypeOf (Topic)).oclAsType(Topic).closure()->exists (
    s_i | b.closure().identificator()->select (i |
     i.oclIsTypeOf (Topic)).oclAsType(Topic).closure()->exists (
      b_i | b_i.URI = s_i.URI ))
  )
```

Further, note that the determination of URI equality is not discussed in this paper and therefore, for the time being, simplified by comparing the URI strings. Instead, the information objects that can be reached by the specific URIs should be compared, and also time (timestamps) may play an important role. See also [15] for more details on URIs.

We will call two `Topics` T_1 and T_2, which meet one of the two XTM merging conditions, *candidates for merging*, as they should be merged according to XTM, but this merging and its consequences has to be examined and possibly prohibited due to some resulting complications.

A situation where merging is still not possible, even if one of the aforementioned conditions would be met, is, when the two candidate `Topics` T_1 and T_2 both have a `SubjectConstitutingResource` (which means, the resource is the subject itself) as `Identificator` - call them SCR_1 and SCR_2 - and SCR_1 is different from SCR_2 (they have different URIs). In XTM, only one `SubjectConstitutingResource` is possible for a `Topic`, which is quite intuitive. Fig. 3 shows this issue schematically. There, T_1 is synonymous to T_2 due to having an equal base name in the same scope $\{T_s\}$. Because they both exhibit a `SubjectConstitutingResource` as `Identificator`, differing from each other, the OCL checker has to raise an error.

We have to check this situation in our validation process and therefore suggest an appropriate operation called `Topic::noDifferentSCR (b:Topic)`. It uses an operation called `Topic::ownSCRIdentificators()`, which is defined in the following.

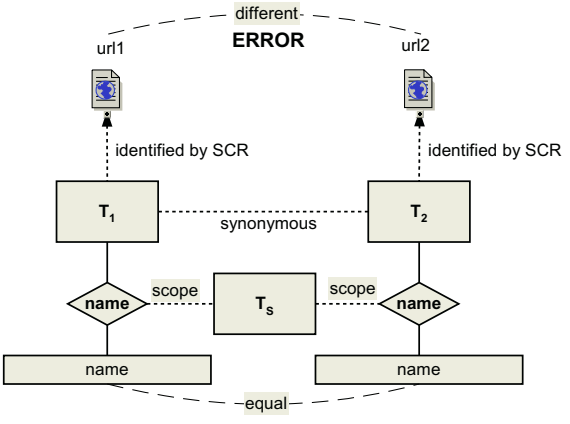

Fig. 3. Different SubjectConstitutingResources as Identificator

```
Topic::noDifferentSCR(b : Topic) : Boolean =
 not self.ownSCRIdentificators()->exists (i_self |
  b.ownSCRIdentificators()->excluding(i_self)->exists (i_b |
   i_self.URI != i_b.URI ))

Topic::ownSCRIdentificators () : Set (SubjectConstitutingResource) =
 self.closure().identificator()->select (i |
  i.oclIsTypeOf(SubjectConstitutingResource)).
   oclAsType(SubjectConstitutingResource)->asSet
```

6 Merging Different Kinds of Topics

What is yet unconsidered in the merging criteria is the fact that the two `Topics` T_1 and T_2, which are candidates for merging due to one of the merging conditions, can be of a different kind. The possible kinds are: `TopicType (TT)`, `TopicObject (TO)`, `AssociationTopicType (AT)` and `OccurrenceType (OT)`. We will use the abbreviations (in brackets) for the four kinds of `Topics`.

In the following, we will examine many of the possible combinations (refer to our further work for a full description of all combinations, including an extensive description of the combination AT - AT) and find out for any combination, if merging shall be prohibited due to the possibility of a semantic inconsistence or not.

6.1 TT - TT, TO - TO, OT - OT

First, we will examine the merging of two `TopicTypes` TT_1 and TT_2. A `TopicType` TT_1 may define a `TopicAssociationEnd` of an AT, where the `multiplicity_min` (the lower bound of the multiplicity) is greater than 0. Look at the example in fig. 4. Some constraint, which will not be shown in this work

Fig. 4. Example for a Multiplicity Constraint on an AT

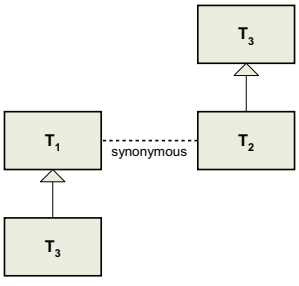

Fig. 5. Circular Inheritance After Merging

due to space reasons, will have to guarantee that every instance of "person" is connected to at least one company via a `TopicAssociation` of type AT_1. Thus, when merging the `TopicType` "person" with some other `TopicType` T_2, all instances of T_2 will also have to have at least one `TopicAssociation` of type AT_1. We will call this the *MultiplicityMinProblem*. This can be realized either by formulating an operation that does this check before merging (called from inside the preconditions of the merge operation), or by an invariant that has to hold any time and yields false after merging.

A further problem may arise when TT_1 and TT_2, which are candidates for merging, define the roles of the two `TopicAssociationEnds` TAE_1 and TAE_2 of the same AT. When desiring unique roles within the same AT, the appropriate OCL invariant will yield false after merging TT_1 and TT_2. We will call this the *UniqueRolesProblem*. Finally, the *CircularInheritanceProblem* may arise after merging, so that TT_1 and TT_2 are then in a circular generalization relationship chain. Fig. 5 shows this schematically.

A circular inheritance prohibiting constraint, like the one adopted by the TMUML meta model from the UML `GeneralizableElement` (constraint 3 for `GeneralizableElement` in [9]), would yield false after merging. Again, these problems can also be checked by OCL operations before performing the merge operation.

By merging two `TopicObjects` TO_1 and TO_2, TO_1 is afterwards instance of its own TTs and the TTs of TO_2. The *MultiplicityMinProblem*, as described in the previous section, may also arise here. `TopicObjects` may also define the role of some property of an AT, so in this case, the *UniqueRolesProblem* also may arise.

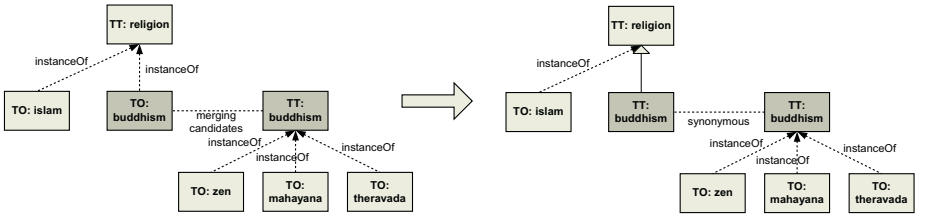

Fig. 6. Merging TT - TO

At the moment, there is no mechanism to constrain an OT to be only applicable to several TTs (like the `TopicAssociationEnds` do for ATs). Therefore, the merging of two `OccurrenceTypes` OT_1 and OT_2 should be no problem.

6.2 TT - TO, AT - TO, OT - TO

When merging two `Topics` of a different kind, the question arises whether the resulting `Topics` should be of the one kind or of the other. Similarly, when establishing a `Synonymy` relation in our TMUML meta model for two different kinds of `Topics`, we have to agree upon one kind for both `Topics`. In the case of merging TT_1 with TO_2, it is not possible to change TT_1 to TO_1, because TT_1 could have subtypes and may be used in `TopicAssociationEnds` as type constraints. But changing TO_2 to TT_2 would be an appropriate measurement, requiring every class-instance relation from TO_2 to some TT_{2super} to be transformed by the merging operation to a subtype relation from the resulting TT_2 to the original TT_{2super}. This works, because a TO does not have any further capabilities in the TMUML meta model than a TT has, and a TT may also play a role within an association instance (see also [19]). Fig. 6 shows an example. There, the `Topics` "TT:buddhism" and "TO:buddhism" are candidates for merging. "TO:buddhism" has to be converted to "TT:buddhism", and the class-instance relation between "TO:buddhism" and "TT:religion" is converted to a subtype relation.

Note that also in the TT-TO case, a *UniqueRolesProblem* may occur after merging. The following OCL operation is called `Topic::changeToTT (t: TopicType)`, where the context `Topic` is the TO, t is the TT and `TopicObject._class` is a TMUML relation to the TTs of the context TO.

```
Topic::changeToTT(t : TopicType)
 pre changeToTT_pre:
  self.oclIsTypeOf (TopicObject)
  and self.closure()->includes(t)
 post changeToTT_post:
  self.oclIsTypeOf (TopicType) and
  self.oclAsType(TopicType).parent().oclAsType(TopicType)->
asSet->includesAll(
    self@pre.closure()->excluding(t).oclAsType(TopicObject)._class)
```

After calling this operation after the merge operation, which already set the `Synonymy` relation between the TO and TT, no `Topic::closure()` contains both a TT and a TO. Note that this strategy is a suggestion - of course, other strategies can be realized with OCL constraints. The merging of a TT and TO could in general be prohibited, which would be a more performant, but also a more restrictive solution. The two cases of merging an AT with a TO or an OT with a TO can be treated similar to the TT - TO case. As a TO has no instances and no more capabilities in the TMUML meta model than an AT, the TO can be converted to and AT after merging the TO and the AT. Note that an AT can also be a member of a `TopicAssociationEndInstance` (see [19]). The same is true for an OT and a TO. For space reasons, the operations `Topic::changeToAT(t : AssociationTopicType)` and `Topic::changeToOT(t : OccurrenceType)` are not shown here.

6.3 TT - AT, TT - OT

Converting an AT_2 to a TT_2, when merging TT_1 and AT_2, will often be impossible, as AT_2 may have a supertype AT_{2super}, and TT_1 cannot be subtype of an AT (a TT has no `TopicAssociationEnds`). Intuitively, in this situation we will try to specialize TT_1 and convert it to AT_1. As $\texttt{AT}_1\texttt{.properties}()$ will (after the conversion from a TT to an AT) obviously be empty, AT_1 simply will have the same properties as AT_2 after the merge (for the definition of `AT.properties`, which yields the `TopicAssociationEnds` of an AT, taking inheritance into account, see [19] or [20]). On the other hand, TT_1 and all its subtypes must not have any instances (TOs), because this would mean these TOs should actually be `TopicAssociations` after the merge, which is not possible. Moreover, all supertypes of TT_1 are neither allowed to have any instances, because after converting TT_1 to AT_1, each direct or transitive supertype TT_x of AT_1 has to be abstract (if not so, TT_x would be a class containing both `TopicObjects` and `TopicAssociations`, which are fundamentally different constructs in Topic Maps). Thus, this kind of merging will rarely be possible, which seems intuitive, as it would mean a mixing of topics and associations, which are fundamentally different meta constructs within Topic Maps. Nevertheless, the following shows the formal representation of our suggested strategy, represented by the OCL operation `Topic::checkATTT(b : Topic)`, which yields true if the context `Topic` is a `TopicType` and all its supertypes and subtypes, which are also `TopicTypes`, have no instances. This operation is used inside the merge operation to determine the possibility of the merging of two candidate `Topics`. Note that `allParents()` recursively yields all supertypes of a `TopicType`, with respect to its closure, and `allChildren()` does so for its subtypes.

```
Topic::checkATTT(b : Topic) : Boolean =
 self.oclIsTypeOf(AssociationTopicType)
 and b.oclIsTypeOf(TopicType)
 and b.oclAsType(TopicType).allTTInstancesSuperAndSub()->isEmpty

TopicType::allTTInstancesSuperAndSub() : Set (TopicObject) =
 self.allTTInstances()->union (self.allTTparentInstances())
```

```
TopicType::allTTInstances() : Set (TopicObject) =
 self.typeClosure()._instance->union(self.allChildren()->select (c |
 c.oclIsTypeOf(TopicType)).oclAsType(TopicType)._instance)->asSet

TopicType::allTTparentInstances() : Set (TopicObject) =
 self.typeClosure()._instance->union(self.allParents()->select (p |
 p.oclIsTypeOf(TopicType)).oclAsType(TopicType)._instance)->asSet

TopicType::typeClosure() : Set (TopicType) =
 self.closure()->select (c |
  c.oclIsTypeOf(TopicType)).oclAsType(TopicType)->asSet
```

Note that `TopicType._instance` yields the set of direct instances (TOs) of the context TT.

6.4 AT - OT

Converting AT_1 to an OT_1 may only work in the case all supertypes and subtypes of AT_1, which are also ATs, have no `TopicAssociationEnds`, which would be a rather theoretical situation. Converting OT_2 to an AT_2 would only work, if OT_2 had neither instances (Occurrences) nor subtypes. Thus, we will simply prohibit merging of AT_1 and OT_2. At last, the OCL checker should report a warning that the merging of two candidate `Topics` was avoided.

7 Merging Inside the OCL Checker

In the previous sections, we have discussed the merging of different kinds of `Topics` and its effects on the consistency and well-formedness of a topic map and presented several additional constraints to avoid merging in certain situations. Finally, we will take a short look at the processing steps inside the OCL checker, which is not only responsible for checking all OCL invariants, it also has to perform merging by establishing the according `Synonymy` relations. Merging inside the OCL checker can be divided into the following three steps, where the set of processed topic maps may be, for example, all the topic maps within a repository and a new topic map to be stored into the repository:
For all Topics T_1, T_2 in the set of processed topic maps

1. check, if T_1 and T_2 are candidates for merging and - if they are of a different kind - if they can be merged
2. if T_1 and T_2 should be merged, establish a `Synonymy` relation between T_1 and T_2
3. if T_1 and T_2 should be merged, and T_1 is of a different kind than T_2, change the kind of T_1 or T_2 so that T_1 and T_2 are of the same kind

The core of the first step is the following operation `Topic::canMerge (b : Topics)`, which simply calls the already presented check operations.

```
Topic::canMerge(b : Topic) : Boolean =
 (subjectBasedMerge(b) or nameBasedMerge(b))
 and typeOfMergePossible(b) and noDifferentSCR(b)
```

While the first two operation calls are for checking the XTM merging conditions, and the last one checks an additional condition implied by the XTM merging conditions, `Topic::typeOfMergePossible(b : Topics)` encapsulates our suggested strategies for merging different kinds of `Topics`. It contains calls to `Topic::tOMP (b: Topic)` in both directions, which is the actual application of our suggestions. There, operations like `checkATTT()` are called. Note that in this version, issues like the *UniqueRolesProblem* (section 6.1) are not checked in advance, merging is optimistically performed and may lead to an according constraint violation afterwards, which is reported by the OCL checker. Performing all checks before establishing the `Synonymy` relation, would call for additional OCL operations to be called from within `tOMP()`.

```
Topic::typeOfMergePossible(b : Topic) : Boolean =
 self.tOMP (b) or b.tOMP (self)

Topic::tOMP(b : Topic) : Boolean =
 b.oclIsTypeOf(TopicObject)
 or (self.oclIsTypeOf(TopicType) and b.oclIsTypeOf(TopicType))
 or (self.oclIsTypeOf(OccurrenceType) and b.oclIsTypeOf(OccurrenceType))
 or (self.checkATTT (b))
```

If $\mathtt{T_1.canMerge(T_2)}$ yields true, the `Synonymy` relation can be established by $\mathtt{T_1.merge(T_2)}$. This operation has to be implemented by an according method, in OCL we can only formulate its pre- and post-conditions.

```
Topic::merge(b : Topic)
 pre merge_pre:
  self.canMerge(b) and not self.closure()->excluding(self)->includes(b)
 post merge_post_Synonymy:
  self.closure()->excluding(self)->includes(b)
```

For the third step, a method `Topic::change (b : Topic)` has to be implemented, which checks the kinds of T_1 and T_2 and subsequently calls the appropriate `changeToXX()` operation, if the kind of T_1 or T_2 has to be changed. One example for this would be the `changeToTT(b: TopicType)` operation presented in section 6.2. The postcondition of `change()` has to state that all `Topics` in the closure of the context `Topic` have to be of the same kind. We omit this for space reasons.

Fig. 7 schematically shows the processing steps of the OCL checker. At first, the previously introduced three merging steps are performed. Afterwards, all other constraints (OCL invariants for inheritance, instancing, etc.) are checked. If errors are to be reported, the OCL checker does so and stops further work, the topic map is considered not well-formed (and therefore can not be stored into a repository). Otherwise, if some merging has happened, the whole processing has to be repeated, as one merging may cause a further merging to happen. This situation is shown in fig. 8.

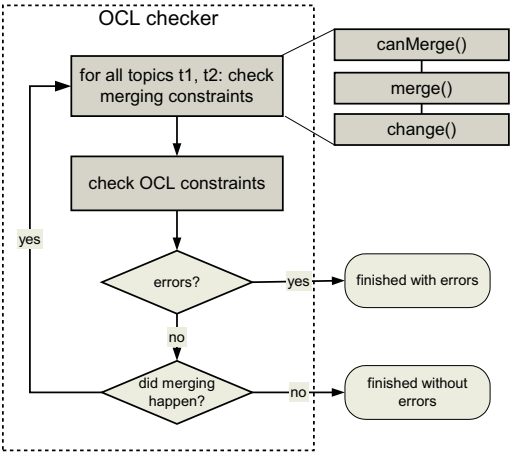

Fig. 7. Process Steps of the OCL Checker

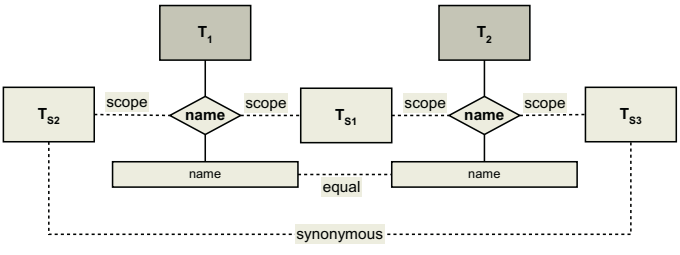

Fig. 8. One Merge Causing a Further Merge

There, T_1 has a name in the scope $\{TS_1, TS_2\}$, and T_2 has the same name in the scope $\{TS_1, TS_3\}$. In a first processing cycle of the OCL checker, T_1 and T_2 are considered to stand for different subjects and are therefore not merged. Suppose that TS_2 and TS_3 are merged (due to identifying the same subject). Then, T_1 and T_2 had the name in the same scope and could also be merged, but only in the second processing cycle of the OCL checker. These processing cycles have to be implemented by a fix-point algorithm, stopping if either an error occurred (a constraint was violated) or a complete cycle has run without performing a single merge. This is shown in fig. 7.

8 Conclusion

In this paper, we have treated the merging of topics in XTM in detail. We have shown how the XTM merging conditions can be realized with the TMUML meta model and presented the according OCL constraints and operations. These operations and constraints can be tested straightforwardly with the USE tool

[5], which helps creators of Topic Maps systems trying out different strategies and their implications. We identified further difficulties when merging different kinds of topics, including association types, occurrence types, topic types and topic instances. After presenting suggestions to overcome these difficulties using OCL, we sketched the processing steps of the OCL checker, which is the central component for validating the well-formedness of topic maps.

All in all, the more voluminous and extensive a Topic Maps system is designed to be, the more important is the issue of merging. At this point, we want to emphasize the duality of merging. On one hand, it provides a great gain in extending existent knowledge by additional facts and information. Two topics may be part of topic maps of different authors. When their associations, occurrences, names and types are combined due to merging, the resulting topic holds the knowledge about a subject of both of the authors. One may learn from the other, and the quality of search queries against topic maps as well as the navigational access on topic maps may enormously benefit from the combined information about a subject.

On the other hand, merging that is applied "silently", simply checking the existing conditions, may always cause subjectively undesired effects. Consider the case two authors both created a topic with the same name, but the second author added another name that the first author would not recognize as a valid name for the topic. No merging process could automatically detect and avoid such situations. Therefore, a notification system seems very important to be integrated with a Topic Maps repository. Every merging should be reported (e.g. by e-Mail, a newsletter system or at least a common logging file or database) to the originators of the merged topics, so that they can verify the semantic impacts of the merge and possibly correct undesired effects (or even manually take back the merge). Another strategy can be adding a special attribute, let us call it "mergeable", to each topic, providing a flag that may avoid merging the topic in any case, even if it would have to be merged according to XTM. The drawback would be that possibly merging would happen quite rarely, and it would be a proprietary extension to XTM.

Our further work will contain an explicit description of the transformation of XTM to a UML compliant, intermediate format like XMI [9] or the USE syntax. The implementation of this transformation together with the implementation of the OCL checker will make up a validator for well-formed XTM Topic Maps, where OCL constraints can dynamically be added or modified in the according USE file and need not be hard-coded into the OCL checker.

References

1. Ahmed, K.: Topic Maps for repositories. Proceedings XML Europe 2000. GCA, Paris (2000).
2. Auillans, P., de Mendez, P.O., Rosenstiehl, P., Vatant, B.: A formal model for Topic Maps. International Semantic Web Conference (ISWC) 2002. http://www.mondeca.com/english/publications_-_doc.htm, last visited 23.2.2003

3. Baird, C.: Topic Map Cartography - a discussion of Topic Map authoring. Proceedings XML Europe 2000. GCA, Paris (2000)
4. Gogolla, M., Richters, M.: On constraints and queries in UML. In: Schader, M., Korthaus, A. (eds.): The Unified Modeling Language - Technical Aspects and Applications, p. 109-121. Physica, Heidelberg (1998)
5. Gogolla, M., Richters, M.: Development of UML Descriptions with USE. In: Tjoa A.M., Shafazand, H., Badie K. (eds.): Proc. 1st Eurasian Conf. Information and Communication Technology (EURASIA'2002). LNCS, Springer Verlag, Berlin Heidelberg New York (2002)
6. Gronmo, G.O.: Creating semantically valid topic maps. Proceedings XML Europe 2000. GCA, Paris (2000)
7. ISO/IEC 13250: Information technology - SGML Applications - Topic Maps. International Organization for Standardization, Geneve, Switzerland (1999)
8. Mueck, T.A., Widhalm, R.: Schlagwort - Topic Maps. Wirtschaftsinformatik, 3:297-300. Verlag Vieweg, Wiesbaden (2001)
9. Object Management Group: OMG UML Specification Version 1.3. http://www.omg.org./uml/, last visited 1. 8. 2002
10. Object Management Group: XMI XML Metadata Interchange, Version 1.1. http://cgi.omg.org/docs/ad/99-10-02.pdf, last visited 2. 8. 02
11. Ontopia: The Ontopia Schema Language Reference Specification, Version 1.3. http://www.ontopia.net/omnigator/docs/schema/spec.html, last visited 23. 10. 2002
12. Pepper, S., Moore, G. et al: XML Topic Maps (XTM) 1.0. Topic Maps Authoring Group (2001). http://www.topicmaps.org/xtm/1.0/, last visited 19. 6. 2001
13. Pepper, S.: Draft requirements, examples and a "low bar" proposal for Topic Map Constraint Language. ISO/IEC JTC 1/SC34/WG3 (2001). http://www.y12.doe.gov/sgml/sc34/document/0226.htm, last visited 1. 8. 2002
14. Rath, H.H.: Technical Issues on Topic Maps. Proceedings MetaStructures 99. GCA, Alexandria, VA (1999)
15. IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax. IETF (1998). http://www.ietf.org/rfc/rfc2396.txt, last visited 17. 1. 2003
16. Richters, M., Gogolla, M.: On formalizing the UML object constraint language OCL. In: Tok-Wand, L. (ed.): Proc. 17th Int. Conf. Conceptual Modeling (ER'98), p. 449-464. LNCS Vol. 1507, Springer Verlag, Berlin Heidelberg New York (1998)
17. Warmer, J., Kleppe, A.: The object constraint language - precise modeling with UML. Addison-Wesley (1999).
18. Widhalm, R., Mueck, T.A.: Topic Maps - Semantische Suche im Internet. Springer Verlag, Heidelberg (2002).
19. Widhalm, R., Mueck, T.A.: Web metadata semantics - on the road to well-formed topic maps. In: Ozsu T. et al. (eds.): Proc. 2nd Int. Conf. On Web Information Systems Engineering (WISE), vol. 2, p. 141-150. IEEE Computer Society, Los Alamitos, CA (2002).
20. Widhalm, R., Mueck, T.A.: Well-formed Topic Maps. Submitted to SoSyM, Journal on Software & System Modeling. Springer Verlag, Berlin Heidelberg New York (2003)