Associations as First-class Elements

Daniel BILDHAUER [1]
dbildh@uni-koblenz.de
University of Koblenz-Landau

Abstract. Models in general and class diagrams as defined by the UML in particular play an important role throughout all steps of modern software development. However, the UML contains several modeling concepts which are not defined precisely and hence are used either rarely or in varying ways and with different semantics. In this article, an extension of n-ary associations as one of those concepts is presented including their refinement by specialization, subsetting and redefinition. DHHTGraphs are introduced as one realization of this extension treating associations and links as first-class objects. In particular, a metamodel and a possible implementation realizing association end refinement by association specialization are presented.

Keywords. N-ary associations, redefinition, subsetting, specialization, Hypergraph

Introduction

In modern software development processes, *models* are becoming the *central artifacts during all development phases* from requirements engineering to implementation and even maintenance and re-engineering. One of the most used kinds of models are *class diagrams* as defined by the UML [1] or in the variants defined by e.g. the metamodeling languages MOF [2] or EMF [3].

While the expressiveness of MOF and EMF is limited especially regarding associations, the UML provides several advanced modeling concepts such as n-ary associations and several possibilities to refine associations and association ends. While some of them, especially redefinition, are not defined precisely and thus are used in varying ways, others such as associations specialization and n-ary associations are rarely used at all. As stated by Genova et al. [4], n-ary associations are of limited use in the way they are currently defined in the UML. In practice, they are often simulated by relation-like objects.

Based on the work of Genova et al. [4] and the formal description of the semantics of redefinition in [5], this article tries to fill the existing gaps in the definition of these modeling concepts and presents a precise and concise definition of n-ary associations and their refinement by specialization, subsetting and redefinition. Furthermore, is shows how this semantics could be implemented treating links as first-class objects.

The next section presents an example use case, motivating the need of a precise definition of n-ary associations and their refinement. Section 2 introduces n-ary associations, the refinement concepts for associations and association ends. Furthermore, the exten-

[1] This work is partially funded by the German Research Foundation (DFG), project number EB 119/6-1

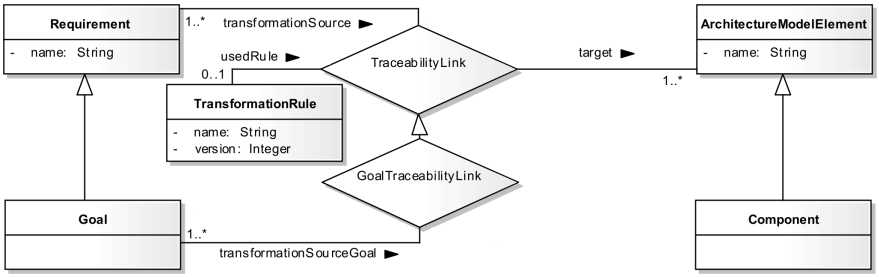

Figure 1. Example for a n-ary association

sion on the semantics of n-ary associations proposed by Genova et al. [4] is shown in this section with some further additions.

In Section 3 the refinement of n-ary associations and their ends is analyzed. A metamodel supporting the extensions of Genova as well as the refinement concepts is presented in Section 4, and a possible implementation is presented in Section 5. Other implementations, further work on refinement of associations and its ends and on n-ary associations are listed in Section 6. Finally, Section 7 summarizes the main propositions of this article.

1. Motivation

Being in modern model-centric development processes, *models* become the *central artifacts* in the software development lifecycle. Thus, their creation, modification and transformation by humans as well as by programs gets more important, while the generation of implementation code tends to be reduced to a primarily technical and automatable step. An appropriate modeling language should therefore be expressive, flexible and natural enough to allow for convenient and detailed modeling of systems. Simultaneously, the language's concepts need to be defined precisely for the language to be automatically processable by programs such as transformations.

Besides *domain specific languages (DSL)* in various variants, *class diagrams* as defined by the Unified Modeling Language (UML) are still one of the most used languages. While known to most modelers and easy to understand in their conceptional semantics and basic model elements, some of their details are defined either not precisely enough or in a way not obvious to all modelers. An example is shown in the following.

In the ReDSeeDS[2] project, model driven development has been combined with case-based reuse driven by requirements [6]. Software systems are developed using model transformation technologies during all development steps from requirements to code, where *traceability links* are used to record the transformations performed together with their affected elements. Besides only linking transformation sources with transformation targets, it may be useful to attach the used transformation rule as a further element to each traceability link. Figure 1 shows a simplified metamodel as an example for n-ary associations.

[2] www.redseeds.eu

Conceptually, the model describes that links of the n-ary association `TraceabilityLink` connect `Requirement`s as `source`s with `ArchitectureModelElement`s as `target`s. Additionally, if a link is a result of a transformation, the `TransformationRule` used should be attached to the link as a further element. `GoalTraceabilityLink`s are a special kind of `TraceablityLink`s, as expressed by the association specialization. Furthermore, as defined by the association end `sourceGoal` and the additional redefinition constraint, `GoalTraceabilityLinks` may not connect to all kinds of `Requirement` but are restricted to `Goal`s.

According to the semantics of n-ary associations defined by the UML, the model does in fact not represent this conceptual idea. As already mentioned by Genova et al. [4], the semantics of n-ary associations as defined by the UML is not very obvious and considers objects to be more important than links. It originates from the cardinalities in entity-relation diagrams, which allow to define the number of relations an element participates in by cardinalities in a flexible way, while no such possibility exists for relations. E.g., in UML as well as in ER diagrams every end of an association or relation denotes, that there is exactly one object connected to this end for each instance of the association while it is not possible to define, that for one end of a relation there may be more elements connected to one instance of the relation. In the example, it is not possible to define that at one `TraceabilityLink` there may be several `Requirement`s. While such a preference of objects may be adequate in prescriptive models for code generation, where associations are only implemented indirectly by (coupled) attributes representing the association ends, such limitations seem to be no longer reasonable if models become the central software artifacts.

Besides the multiplicities, there is a further issue with the *specialization* of associations and the inheritance of association ends. Even if association specialization is rarely used explicitly and usually expressed by subsetting or redefinition of association ends, there is still a benefit of specialization as shown in Figure 1. Conceptually, the specialization of `TraceabilityLink` by `GoalTraceabilitylink` describes, that `GoalTraceabilityLink` is a special kind of `TraceabilityLink` which inherits all features from the superclass. While this semantics is compatible to the well-known and widely used specialization of classes and the descriptions of generalization in the UML Superstructure [1, p.72], it is not compatible to the description of associations in the same document [1, p.40].

Furthermore, as earlier shown in [5], the *semantics of redefinition and subsetting* and its interrelation to *association specialization* is *not precisely defined*. While a proposal for the refinement of binary associations has been given in [5], refinement of n-ary associations has not been tackled and is even more complicated. Probably, these complications are at least one reason, why n-ary associations are rarely used in practice but simulated by relation-like objects. While this workaround enables a more flexible way to define connections of such simulated association and classes on the one hand, it enforces a special treatment of those relation-like objects and their connected links in all algorithms working on the model. As shown later in this article, such a simulation is not necessary, if the concept of n-ary associations defined by the UML is extended and associations are treated as first-class objects. As a basis for this extensions, the single concepts are introduced in detail in the next section.

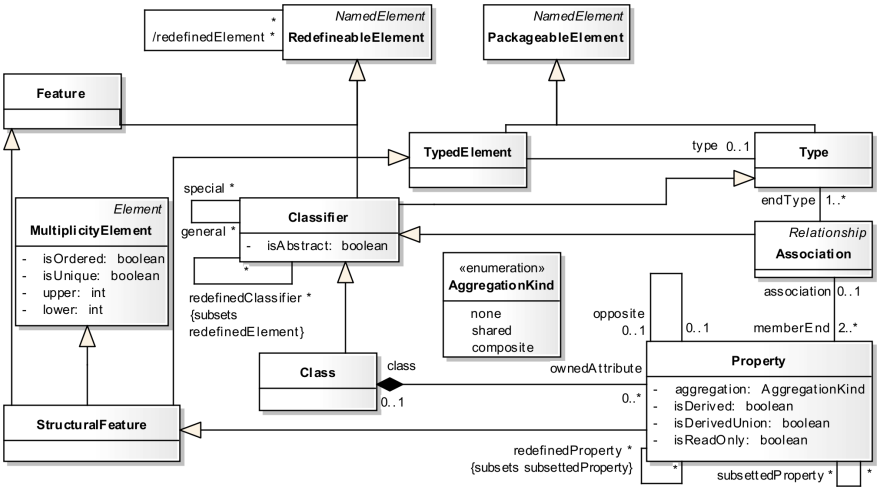

Figure 2. Relevant part of the UML metamodel

2. Definitions in the UML Metamodel

The UML is defined by a metamodel written in the Meta Object Facility (MOF) [2], which is a subset of UML class diagrams. In this metamodel, the UML concepts of classes, associations, association classes, and properties are defined by classes. Their relationships such as specialization of classes and associations as well as subsetting and redefinition of properties are expressed by associations between the relevant classes. An UML model is an instance of the metamodel. Figure 2 shows the relevant part of the UML metamodel with the classes `Class`, `Association` and `Property` and the relevant associations. In the following, the concepts are described in some more detail, using quotations of the UML specification.

2.1. Classes, Associations, and Properties

Classes as representations of entities of the modeled world are instances of the metaclass `Class`. According to the UML superstructure, "a class describes a set of objects that share the same specifications of features, constraints, and semantics." The relations between entities of the modeled world are abstracted by *associations*, which are instances of the metaclass `Association`. The connections of classes and associations, also referred to as *association ends*, as well as attributes of classes and associations are instances of the metaclass `Property`.

As mentioned above, classes and associations can be specialized while properties can be refined by subsetting and redefinition. In the following, these concepts are explained using the model from Figure 1 as example. Furthermore, as n-ary associations have a special semantics shortly introduced above, they are also described in a bit more detail together with the extension proposed by Genova et al. [4].

2.2. Specialization

Classes as well as associations can be specialized and generalized, as indicated by the reflexive association at `Classifier` with the rolenames `special` and `general` in the metamodel. According to the UML Superstructure, the specialization of a classifier means that

> "An instance of a specific Classifier is also an (indirect) instance of each of the general Classifiers. [...] features specified for instances of the general classifier are implicitly specified for instances of the specific classifier." [1, p. 55]

Since `Class` as well as `Association` are both subclasses of `Classifier`, this notion holds for both of them with some additional constraints imposed for associations:

> "An association specializing another association has the same number of ends as the other association. [...] every end of the specific association corresponds to an end of the general association, and the specific end reaches the same type or a subtype of the more general end." [1, p. 40] "The existence of a link of a specializing association implies the existence of a link relating the same set of instances in a specialized association." [7, p. 113]

As described by those statements and conforming to the examples and usages of association specialization in the UML, associations may be specialized, but it is assumed that association ends are not inherited on specialization but need to be defined explicitly for each association. While for binary associations notated as solid lines connecting two classes the ends are defined implicitly anyway, the necessity to define the ends seem to be not reasonable for n-ary associations, as shown by the example in Figure 1. In this model, the explicit definition of two additional association ends at `GoalTraceabilityLink` would make the model mode complicated without any additional gain in precision. Therefore, an extension treating classes and associations equivalently regarding specialization is described later on in this article.

2.3. Subsetting

Similar to the specialization of classifiers, the refinement of properties is supported in the UML by a concept named *subsetting*. While its definition is scattered over various places in the Super- and Infrastructure, a good description is:

> "An end of one association may be marked as a subset of an end of another in circumstances where [...] each of the set of types connected by the subsetting association conforms to a corresponding type connected by the subsetted association. In this case, given a set of specific instances for the other ends of both associations, the collection denoted by the subsetting end is fully included in the collection denoted by the subsetted end."[7, p. 113]

Figure 3 shows an example for subsetting. For all instances of `Goal`, the set `target` contains all elements of the set `targetComponent`. As shown in [5], *subsetting of one end of a binary association and specialization of the association mutually imply each* other. Regarding n-ary associations, this statement needs to be generalized with respect to subsetting between two ends at the same association, as shown in Section 3.

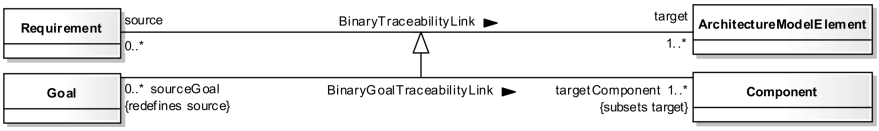

Figure 3. Subsetting and redefinition at a binary association

2.4. Redefinition

In addition to subsetting, the UML provides a concept named *redefinition* for the overriding refinement of properties as association ends and attributes. The redefinition concept is also used for the refinement of operations, but in this case not related to subsetting and association specialization and thus not considered here. While redefinition is not defined coherently in the UML Super- and Infrastructure, we assume the following semantics, which has been justified in detail in [5] and enables a practical usage of redefinition in our application domain.

> "Redefinition is a relationship between features of classifiers within a specialization hierarchy. Redefinition may be used to change the definition of a feature, and thereby introduce a specialized classifier in place of the original featuring classifier, but this usage is incidental." [1, p. 41] "A redefined property must be inherited from a more general classifier containing the redefining property." [1, p. 128] "An end of one association may be marked to show it redefines an end of another in circumstances where [. . .] each of the set of types connected by the redefining association conforms to a corresponding type connected by the redefined association. In this case, given a set of specific instances for the other ends of both associations, the collections denoted by the redefining and redefined ends are the same."[7, p. 113]

As defined in this statements, redefinition is similar to subsetting with an additional constraint. In the context of the redefinition, all instances of the redefined property need to be also instances of the redefining one. All those instances are accessible using the redefining rolename as well as the redefined one. In Figure 3, the redefinition constraint at the association end `sourceGoal` specifies, that for all instances of `Component` the sets `source` and `sourceGoal` are identical. Thus, only `Goal`s could be linked to `Component`s, but they are accessible by the rolename `sourceGoal` as well as by the inherited one `source`.

2.5. N-ary Associations

While the associations usually used in class diagrams are binary with instances connecting exactly one source with exactly one target element, the UML allows for a more general kind of associations with a higher arity, whose instances connect more than two elements. The association `TraceabilityLink` in Figure 1 above is an example for such an *n-ary association*. The UML describes the semantics of an n-ary association as follows:

> "For an association with N ends, choose any N-1 ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplic-

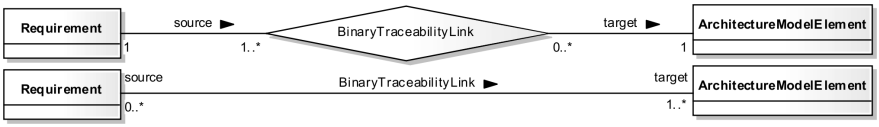

Figure 4. Binary association notated as a solid line and as a diamond

ity of the association end constrains the size of this collection."[1, p. 56] "For n-ary associations, the lower multiplicity of an end is typically 0. A lower multiplicity for an end of an n-ary association of 1 (or more) implies that one link (or more) must exist for every possible combination of values for the other ends."[1, p. 57]

Thus, the semantics of the association `Traceabilitylink` above is, that for each pair of `Requirement` and `ArchitectureModelElement` there is up to one `TransformationRule` linked to the pair and for each pair of `Requirement` [`ArchitectureModelElement`] and `TransformationRule`, there is at least one `ArchitectureModelElement` [`Requirement`] linked to the pair.

As shown by Genova et al. [4], this semantics does not allow for the detailed specification of participation constraints for the single classes connected by an n-ary association nor for the association itself. Genova et al. have proposed to extend the entity-relationship [8] like UML multiplicities by a second kind of multiplicities based on the method Merise [9]. This kind of multiplicity should be notated near the diamond depicting the association and is referred to as "inner multiplicity", while the Chen-like multiplicity of UML is called "outer multiplicity". In Figure 5, these two kinds of multiplicities are depicted at each association end. The semantics of those multiplicities is defined as follows: "The outer multiplicity [...] specifies the potential number of values at the specified end with respect to a combination of values at the other n−1 ends. A minimum outer multiplicity 0 means that the specified link end may be empty. The inner multiplicity [...] specifies the potential number of combinations of values at the other n − 1 ends with respect to one value at the specified end. A minimum inner multiplicity 0 means that the elements at the end may not take part in the association."[4]

While these descriptions are still focused on the classes participating in an association and still considers objects to be more important than links, its is much easier to express the semantics of multiplicities if *associations* and its links are treated as *first-class objects*. In that case, the *inner multiplicities* define the *number of links* that may be *connected to an object* while the *outer multiplicities* defines the *number of objects connected to a link*. Assuming this semantics, the multiplicities at the association end named `source` in Figure 1 specify, that at each `Requirement` there may be an arbitrary number (`0..*`) of `TraceabilityLink`s, while at least one (`1..*`) `Requirement` participates in each such link. As the association is instantiated by a link, each end of the association is instantiated by zero, one or more ends of the link. Thus, each association end denotes sets of connections between one links and several objects and vice versa. The special case of a binary association and its representation as a solid line between the objects omitting the diamond and the multiplicities for the objects connected to a link is shown in Figure 4. Both notations in this figure are equivalent.

Similar to the set of connections at an object, which can be specified in more detail by constraints such as `ordered` or `unique`, also the set of connections at a link should be specifiable using the established constraints. Furthermore, the specialization of asso-

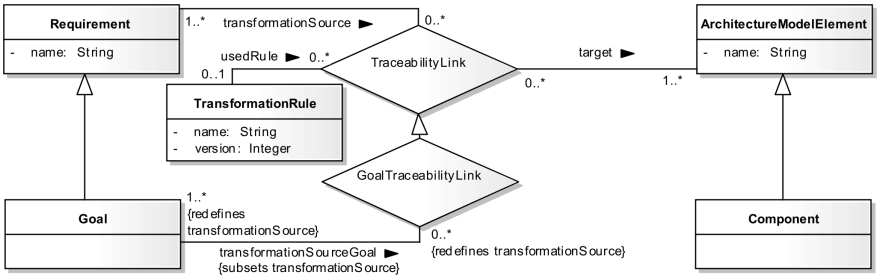

Figure 5. Model from Figure 1 extended by inner multiplicities and refinement

ciations and the refinement of association ends by subsetting and redefinition need to take this new semantics into account.

3. Refinement of N-ary Associations

As argued in Section 1 above, in a model-centric development environment there is no reason why the inheritance of association ends is treated different in the specialization of classes and associations. Treating associations as first-class objects, their properties such as their ends should be subject to specialization as they are for classes. Such an inheritance would allow a modeling as shown in Figure 5, where the association `GoalTraceabilityLink` specializes the association `TraceabilityLink` and inherits all its association ends. Thus, `GoalTraceabilityLink`s also have ends connecting to `ArchitectureModelElement`s and `TransformationRule`s without a need to specify these ends explicitly in the model. This semantics is analogical to the semantics of classes, which inherit the properties from their superclasses.

However, a refinement of a class or a association usually also includes a refinement of some of their properties such as attributed or connected association ends. While this refinement is implicit in the association notation of the UML, the concepts of subsetting and redefinition are used to refine the association ends connected to classes. The later can also be used for associations if an inheritance of their ends is assumed.

3.1. Subsetting

In the UML it is stated, that the ends of an association need to conform to the ends of every specialized association. In [5] it has been shown, that in fact the specialization of an association implies subsetting of all its ends. The semantics of subsetting described in Section 2.3 can be generalized to cover the extension of n-ary associations introduced above. A subset-relationship between two association ends denotes, that for all links of the association with the subsetting end the set of connections denoted by the subsetting end is fully included in the set of connections denoted by the subsetted end. Analogical, at each instance of the class connected to the subsetting end this restriction holds. As an example, the constraint `subsets transformationSource` at the association end `transformationSourceGoal` in Figure 5 defines, that for each instance of `Goal`, the set of `transformationSourceGoal`-connections is fully included in the set of `transformationSource`-ones. Similarly, this holds also

for `GoalTraceabilityLink`. It may be reasonable, not to restrict subsetting to two ends of an association and its specialization, but to allow also a subsetting relationship between two ends of the same association, similar to subsetting between two ends at the same class allowed in the UML. The semantics of subsetting is indeed not changed by this extension.

3.2. Redefinition

While subsetting can be used to define that one association end is a special case of another one, it does not restrict the occurrence of the specialized end. Such a restriction is possible by the usage of *redefinition*. As shortly introduced in Section 2.4, the redefinition of an association end restricts the existence of instances of the redefined end in the context of the redefinition. This enables a covariant specialization of properties and association ends, as it is possible to restrict inherited properties. Expressed in the terms of links connecting objects, it is possible to restrict the links at an object by redefining the other end of the association. The restriction of the objects connected to links is not done by redefinition as it is defined in the UML, but for each association it is specified explicitly which classes it connects without an inheritance of association ends on the specialization of an association. However, if the inheritance of association ends is assumed, the redefinition concept can be extended to restrict also the objects connected to links. An example is included in Figure 5. Similar to the multiplicities, the redefinition is notated separately for the association and the class at each end. While the inner redefinition constraints the links which may be connected to the objects of the class, the outer one restricts the objects which may be connected to the links.

One redefinition at an association ends implies also a subsetting of the end, since the restriction imposed by the redefinition is stronger than the subsetting. Thus, the statement in Section 2.4, that redefinition implies subsetting, is also true for the extended version of n-ary associations. However, we would propose to notate the subsetting explicitly for clarity.

4. Metamodel for N-ary Associations

As argued in Section 1, a precise definition is necessary for modeling concepts to be usable in practice. For the style of n-ary associations described above, such a definition and an appropriate implementation is given by the approach of *Distributed Hierarchical Hyper-TGraphs (DHHTGraphs)*. They are based on typed, attributed, ordered directed graphs, (*TGraphs* [10]), and extend them by means of distribution, hierarchy and hyperedges. In the context of this article, especially the hyperedges as a realization of n-ary links are relevant, while the other features are not considered in detail. In the context of DHHTGraphs, a slightly modified version of UML class diagrams called *Graph UML (grUML)* is used to describe graph schemas, which serve as metamodels for graphs. The diagram shown in Figure 5 is already a valid grUML diagram. The classes `Requirement`, `Goal` and so on denote *vertex types*, while the associations `TraceabilityLink` and `GoalTraceabilityLink` denote types of (hyper)edges connecting vertices. The language grUML is defined by a *metaschema*, the part relevant for the realization of n-ary associations is shown in Figure 6.

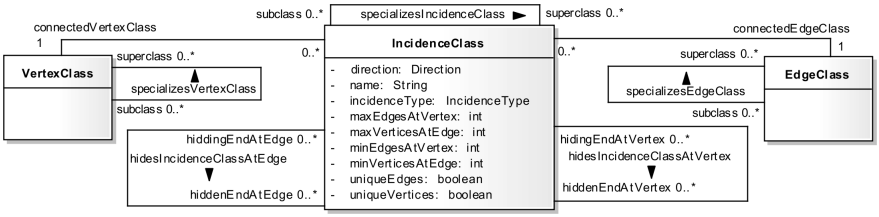

Figure 6. Part of the grUML-Metaschema

The connections points of nodes and their connecting edges are defined by `IncidenceClass`es, which are similar to properties in the UML. Each such one has a name, a direction and a type analogical to the class `Property` shown in Figure 2. Moreover, it stores the two kinds of multiplicities introduced in Section 2.5 which define the number of vertices connected by such an incidence to an edge and vice versa. Similar to the multiplicity semantics of the UML, these multiplicities affect the instances of the respective classes as well as all of their specializations. Furthermore, each `IncidenceClass` defines the uniqueness of the connections at a vertex or an edge by the attributes `uniqueEdges` and `uniqueVertices`. As in DHHTGraphs all incidences at a class as well as at a vertex are ordered by default, this information is not represented in the `IncidenceClass`es.

The possibility to specialize vertex-, edge- and incidence classes as representations of entities, relationships and their connection points is expressed by a relationship in each case. In every specialization, the properties are inherited. To allow a more detailed specification of connections, the two kinds of overriding refinement introduced in Section 3 are supported and expressed by the relationships `hidesIncidenceClassAtEdge` and `hidesIncidenceClassAtVertex`. These relationships represent the redefinition constraints as shown in Figure 5.

5. Implementation

The practical usability of a modeling language depends on frameworks which allow to handle model instances, e.g. by the automatic generation of source code which implements the structure and behavior described by the model. For a class diagram, every class in the model is transformed to a class in the desired implementation language, e.g. Java, and features such as generalization are realized by appropriate concepts of that language. For UML and MOF several such tools are established and used in practical software engineering. The possibly best established one is the Eclipse Modeling Framework (EMF) [3] based on the ECore metamodel. Similar to (Essential) MOF, ECore is restricted to binary associations and does not support associations as attributable and generalizable first-class elements, but reduces them to synchronized pairs of references representing the association ends. Each rolename is implemented by a collection containing the references to the appropriate linked objects. Since n-ary associations are not contained by ECore, there is no support by the code generation, but they need to be emulated by artificial classes. This realization of associations is common also for other modeling environments, even if some of them support subsetting and redefinition of association ends by synchronizing the relevant sets.

An alternative to the implementation of associations by their ends is the treatment of links as first-order elements as realized in the graph-library JGraLab [10] and also used for DHHTGraphs. Besides an easy realization of association attributes and generalization, this allows also a convenient implementation of subsetting and redefinition and especially of n-ary associations. If the associations defined in the model are implemented as classes with the links being instances of these classes, the role access can be realized by a traversal of the respective link objects. Moreover, as shown in [5], this implementation allows to use the specialization concept of the implementation language Java to realize subsetting and redefinition, since the traversal of the links for the subsetted of redefined rolename will include the links matched to the subsetting or redefining rolename without any need of further synchronization. Only at the creation of the links, some additional effort is necessary to deal with redefinition. In the listings below, the implementation of the model depicted in Figure 3 is shown as an example. As a special feature of JGraLab, the method `getThat()` allows to access the element at the other end of a link. The iteration over all `BinaryTraceabilityLink`-links to access the role `target` includes all `BinaryGoalTraceabilityLink`s which are created if a `Component` is added as a `targetComponent` to a `Goal`. The redefinition of the rolename `source` by `sourceGoal` is realized by an appropriate check as shown below.

```
class Requirement {
public BinaryTraceabilityLink addTarget(ArchitectureModelElement o) {
  return graph.createBinaryTraceabilityLink(this, o);
}

public List<? extends ArchitectureModelElement> getTarget() {
  List<ArchitectureModelElement> adjacences = new ArrayList<...>();
  BinaryTraceabiltyLink edge = getFirstBinaryTraceabilityLink();
  while (edge != null) {
    adjacences.add((ArchitectureModelElement) edge.getThat());
    edge = edge.getNextBinaryTraceabilityLink();  }
  return adjacences;
} }

class Component {
public BinaryTraceabilityLink addSource(Requirement r) {
  if (!r instanceof Goal)
    throw new SchemaException("Redefinition of role 'source' ... ");
  return graph.createBinaryGoalTraceabilityLink(r, this);
} }
```

Listing 1 Example implementation of role access on binary links

For n-ary associations, the implementation and in particular the role access is a bit more complicated. At first, as not only two elements but combinations of sets of elements are related by a link, there is the issue if such a role access is possible if only one element participating in a link is given. In the UML, ends of n-ary associations are treated only as member ends of that associations and not as properties of the relevant classes. Due to the UML semantics of n-ary associations described in Section 2.5, an access to a role is possible only if a combination of elements at the other end is given.

As an example, assume an object `r` of the class `Requirement` from Figure 5. An access to the rolename `target` on `r` may be interpreted in two differ-

ent ways and thus lead to two different results. Following the UML semantics, an access to the role `target` is only valid, if a combination of `Requirement` and `TransformationRules` is given but not for a single `Requirement`. However, if the extended semantics of n-ary associations is assumed, it may be reasonable to access the set of `ArchitectureModelElements` related to the `Requirement` by `TraceabilityLink`s. An example implementation of this semantics is shown in the listing below for the classes `Requirement` and `TraceabiltiyLink`.

```
public TraceabilityLink addTarget(ArchitectureModelElement o) {
  TraceabilityLink t = (...) graph.createTraceabilityLink();
  t.addSource(this); t.addTarget(o);
  return t;
}

public List<? extends ArchitectureModelElement> getTarget() {
  List<ArchitectureModelElement> adjacences = new ArrayList<...>();
  TraceabilityLink edge = getFirstTraceabilityLink();
  while (edge != null) {
    adjacences.addAll(edge.getTarget());
    edge = edge.getNextTraceabilityLink();   }
  return adjacences;
}
```

Listing 2 Implementation of role access on n-ary link in class `Requirement`

```
public void addTarget(ArchitectureModelElement o) {
  //create target incidence of IncidenceClass TraceabilityLink_target
  TraceabilityLink_target incidence = new TraceabilityLink_target(this, o);
  //add element to set of incidences at this edge and object o
  incidences.add(incidence);  o.addIncidence(incidence);
}

public List<? extends ArchitectureModelElement> getTarget() {
  List<ArchitectureModelElement> adjacences = new ArrayList<...>();
  for (Incidence inc : incidences)
    if (inc instanceof TraceabilityLink_target)
      adjacences.add(inc.getVertex());
  return adjacences;
}
```

Listing 3 Implementation of role access on n-ary links in class `TraceabilityLink`

As shown in line two of Listing 2, adding an element to a set denoted by a rolename results in an edge to be created and the edge object returned by the create operation can be used to further attach elements to the relation represented by the edge. Special care needs to be spent on the refinement of association ends by subsetting between two ends at the same association. In general, refinement may be implemented either by synchronized sets, with all its advantages and disadvantages described in detail in the related work below, or by the usage of the specialization concept provided by the implementation language. In the latter case, each association end needs to be represented by a separate class and each link end by an appropriate object as realized in the examples above. While the main advantages of this implementation is the full support of the

model semantics, its main disadvantages is the increasing number of objects necessary to represent model instances. However, for the case of binary associations the approach has been proven in several projects which model instances containing several millions of objects and links and it can be assumed, that also for the case of n-ary associations the performance is sufficient. Furthermore, as shown by Scheidgen [11], all approaches based on coupled association ends and their synchronization are not able to reflect the semantics of subsetting and redefinition correctly without any further history.

The generated code allows to use the semantics of association specialization as well as of property subsetting or redefinition. The representation of links as edges which are first-order elements enables the usage of association attributes and thus offers modeling concept not present in other modeling frameworks like EMF. While in those modeling approaches the representation of relations carrying further information results in artificial objects and a special treatment of them in the algorithms, the TGraph approach keeps algorithms simple as it does not enforce such a distinction.

6. Related Work

Besides JGraLab and EMF there are some other modeling frameworks allowing to deal with model instances by code generation. However, the established ones are restricted to variants of EMF or EMOF without a support of n-ary associations. In the "MOF 2.0 for Java" implementation by Scheidgen [11], associations are represented by their ends which are realized by Java references stored in collections with subsetting and redefinition realized by synchronization of updates of those sets. Scheidgen shows, that this synchronization is not trivial, and proposes additional dependency information for each element to solve this issue. Scheidgen also discusses the alternative of implementing the links, but assumes them to be of limited use. In Section 5 we have shown, that this is not the case and that links can be used to implement subsetting and redefinition easily and without any need for additional dependency information.

A similar approach without the additional dependency information is presented by Amelunxen et al. [12]. As shown by Scheidgen [11, p 45], this approach has a slightly different update semantics where deletion of an element previously added to a set does not completely restore the previous state on all subsetted sets. The authors have also shown, that subsetting or redefinition of one association end implies subsetting of all other ends. However, in contrast to our interpretation of redefinition, which affects the association itself rendering it abstract in the context of the redefinition, only the classes are affected by the redefinition in their interpretation. Furthermore, the authors show, that `union` on an association end implies an abstract association and vice versa. This semantics is also supported by the implementation in JGraLab described above.

Olivé [13] identifies and formally defines different forms of refinement. He assumes the redefinition of a property (as representation of the refinement of a relationship participant) to be only a constraint on the connected elements of a relationship but independent of an association specialization [13, p. 230f]. While possibly useful when modeling without associations as first-class elements, this interpretation contradicts to the description in the UML superstructure [1, p. 18].

While several approaches for n-ary links or hyperedges have been developed, their usage and value has been discussed controversially. Engels and Schürr [14] propose to

keep links a simple as possible. They argue, that otherwise the separation of edges and vertices vanishes, leading to hyperedges pointing to hyperedges with their ends as new kind of plain edges. As shown above, the separation is actually assured by a uniform and consistent representation of simple edges and any more complex relationship.

7. Conclusion

If *models* become the *central artifacts* in the software development, the modeling languages used need to be expressive, flexible and natural enough to allow for convenient and detailed representation of entities as well as their relationships. However, even the widely used modeling language UML is not expressive enough for n-ary associations. In particular, besides the definition of multiplicities for n-ary associations, there are some issues regarding the refinement of associations and the inheritance of their properties. Based on Genova's [4] extensions to the multiplicities, also the refinement of associations can be improved if associations are treated as first-class objects. Using the established concepts of subsetting and redefinition of association ends in a slightly adopted way, a details specification of relationships is possible by n-ary associations. As a realization of this semantics, DHHTGraphs have been introduced and an exemplary implementation has been presented.

References

[1] Object Management Group: Unified Modeling Language: Superstructure, Version 2.2. (February 2009)

[2] Object Management Group: Meta Object Facility (MOF) Core Specification, Version 2.0. (01 2006)

[3] Steinberg, D., et al.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional (2009)

[4] Génova, G., Lloréns, J., Martínez, P.: The meaning of multiplicity of n-ary associations in UML. Software and System Modeling **1**(2) (2002) 86–97

[5] Bildhauer, D.: On the relationship between subsetting, redefinition and association specialization. In: Proc. of the 9th Baltic Conference on Databases and Information Systems 2010, Riga, Latvia (07 2010)

[6] Bildhauer, D., Horn, T., Ebert, J.: Similarity-driven software reuse. In: CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models, Washington, DC, USA, IEEE Computer Society (2009) 31–36

[7] Object Management Group: Unified Modeling Language: Infrastructure, Version 2.2. (February 2009)

[8] Chen, P.P.S.: The entity-relationship model—toward a unified view of data. ACM Trans. Database Syst. **1**(1) (1976) 9–36

[9] Tardieu, H., Rochfeld, A., Coletti, R.: La m´ethode MERISE, Tome 1: Principles et outils. Les Editions d'Organisation, Paris, France (1985)

[10] Ebert, J., Riediger, V., Winter, A.: Graph Technology in Reverse Engineering, The TGraph Approach. In Gimnich, R., Kaiser, U., Quante, J., Winter, A., eds.: 10th Workshop Software Reengineering (WSR 2008). Volume 126., Bonn, GI (2008) 67–81

[11] Scheidgen, M.: Description of Computer Languages Based on Object-Oriented Meta-Modelling. PhD thesis, Humboldt Universität zu Berlin (October 2008)

[12] Amelunxen, C., Schürr, A.: Vervollständigung des Constraint-basierten Assoziationskonzeptes von UML 2.0. In Mayr, H., Breu, R., eds.: Proc. Modellierung 2006. Volume P-82 of Lecture Notes in Informatics., Bonn, Gesellschaft für Informatik (2006) 163–172

[13] Olivé, A.: Conceptual Modeling of Information Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)

[14] Engels, G., Schuerr, A.: Encapsulated Hierachical Graphs, Graph Types, and Meta Types. In: SEGRAGRA'95, Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation. Volume 2., Elsevier (1995)