On the Relationships Between Subsetting, Redefinition and Association Specialization

Daniel Bildhauer[*]
dbildh@uni-koblenz.de

University of Koblenz-Landau

Abstract. UML class diagrams and their variants in MOF and EMF play an important role throughout all steps of modern software development. A feature often used also in the UML definition itself is property redefinition. However, not all aspects of redefinition and especially its interrelation to property subsetting and association specialization are defined and used coherently. This article fills existing gaps in the semantics and shows, that redefinition can be interpreted as a special case of subsetting and that association specialization and subsetting of its ends imply eachother. A sample realization of this semantics treating links as first-class objects is shown as a part of the TGraph library JGraLab.

1 Introduction

In modern software development, UML class diagrams and several often domain specific derivatives are used to describe structural aspects of software systems. Additionally, they are used as the metamodeling language e.g. in MOF and EMF and implemented in (meta-)modeling tools. Such implementations need a precise definition of the language concepts which is given by the UML Infra- and Superstructure for most concepts. However, there are still some open issues such as the interrelation between the specialization of an association and the refinement of its ends by subsetting or redefinition. This article tries to fill the existing gaps and gives a precise and concise definition for association specialization, property subsetting and redefinition, and their interrelation.
Even if the statements in this paper hold also for n-ary associations, we focus on binary ones due to the following reason. Genova et al. [1] have stated, that n-ary associations are of limited use in the way they are currently defined in the UML. They have proposed an adaption of their semantics and extension by additional multiplicities. A detailed inspection of redefinition and subsetting at n-ary associations should consider also these extensions. Finally, this would lead to an analogical extension on redefinition and subsetting which is out of scope of this paper.
An example use case for class diagrams is given in the next section, motivating the need of a precise definition of the refinement concepts

[*] This work is partially funded by the German Research Foundation (DFG) and the EU IST FP6 project ReDSeeDS

and their interrelation. Section 3 presents the description of association specialization and property subsetting and redefinition in the UML and shows relationships defined or explicitly negated. The interrelation suggest by the single definitions and reasonable for implementation purposes is shown in section 4. A proposal how to manifest this in the UML is described in section 5, and the TGraph library JGraLab is presented as a modeling framework implementing this semantics in section 6. Other modeling frameworks and further work on refinement of associations and its ends are listed in section 7. Finally, section 8 concludes the main propositions and presents the examination and implementation of redefinition at n-ary associations as further work.

2 Motivation

In modern software development processes, modeling and model transformation is used in nearly all steps from requirements engineering to implementation. In the ReDSeeDS[1] project, these technologies have been combined with case-based reuse driven by requirements as it was shown in [2]. In this context, a software system together with all its development artifacts such as requirements and models is named a *software case*.
Such cases are developed using model transformation technologies during all development steps from requirements to code. The single software artifacts are connected by traceability links and stored in a *software case repository*. To support reuse of existing solutions, this repository can be searched for cases or parts of them whose requirements are similar to the ones of the case currently under development. It can be assumed, that cases with several similar requirements contain realizing artifacts which can be reused in the current case. The comparison of requirements as well as their transformation to models need a precise description of the requirement language used. Hence, the developed Requirements Specification Language (RSL) was metamodeled using MOF and the refinement concepts of association specialization as well as property subsetting and redefinition. Figure 1 depicts a simplified part of the RSL metamodel. While originating from a special application domain, the model is used as a running example here, since it illustrates the usage of the refinement concepts and their interrelation in a small but realistic usecase. However, the considerations below are not restricted to this special domain but are valid in all models which use association specialization and refinement of association ends.
Requirements of software systems are instances of the class `Requirement` and their specializations `Non-FunctionalRequirement` (which is referred to as `NFR` in the following) and `UseCase`. Requirements can be represented by different `ReqRepresentation`s such as `Scenario`s, which are linked to the `Requirement` by `HasRepr`-links. The set of representations linked to a requirement `r` is usually denoted by `r.HasRepr` or by `r.representation` as an equivalent notation based on the rolename. The `HasRepr` association is specialized by `HasScenario` and `HasTextRep`, linking `UseCases`

[1] www.redseeds.eu

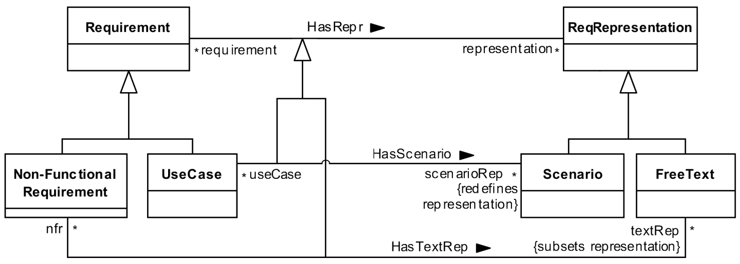

Fig. 1. Requirements metamodel using three refinement concepts

and `NFR`s to their more special representations which also have special roles. The `subsets representation`-constraint at the role `textRep` enforces, that for every `NFR n` the set denoted by `n.textRep` is a subset of the one denoted by `n.representation`. A more restrictive constraint is the `redefines representation`-one at the role `scenario`, which forbids the existence of other representations than `Scenario` to be linked to a `UseCase`.
The RSL metamodel defines the abstract syntax of the artifacts stored in the case repository, which uses the graph-library JGraLab as its internal datastructure. In the context of JGraLab, the metamodeling language called grUML (Graph UML) with an expressive power between EMOF and CMOF is used to describe graph schemas which serve. as metamodels for graphs. To allow subsetting and redefinition to be used in the RSL metamodel, these concepts have been introduced into grUML. During the extension it was noticed, that the three refinement concepts are not coherently defined in the UML and that especially their interrelation is not clear. However, a precise definition of the concepts is necessary for a correct interpretation of metamodels using them and especially for the development of tools. A detailed investigation of the UML documents shows, that such a precise definition is possible and compatible with the usually assumed semantics of the single concepts, even if it leads to interrelations between the refinement concepts which are explicitly negated in the UML description. *The specialization of an association is equivalent to the subsetting of (one of) its ends and redefinition can be treated as a subconcept of subsetting.* In section 6 the implementation of this interrelation is described, treating associations as first-class elements. As the basis for the following descriptions, the core part of the UML metamodel is explained in the next section.

3 Definitions in the UML Metamodel

The UML is defined by a metamodel written in the Meta Object Facility (MOF) [3], 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

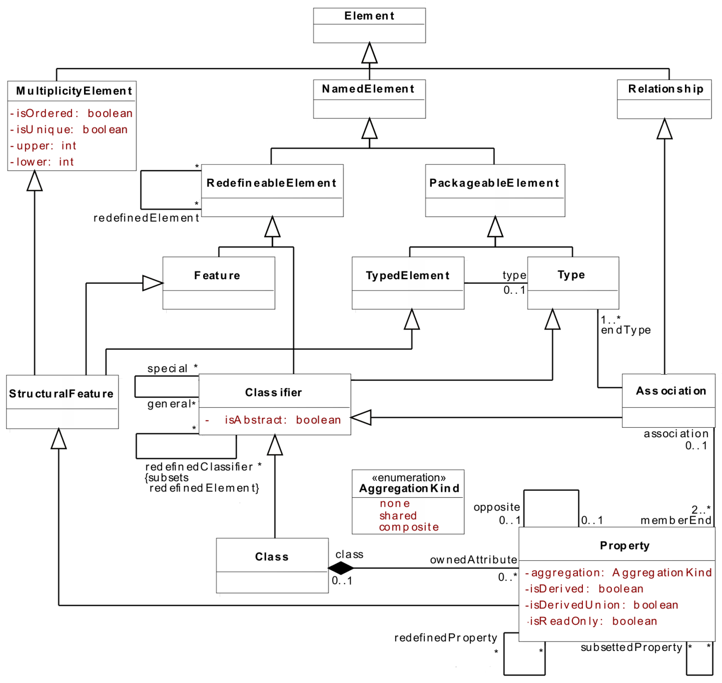

Fig. 2. Relevant part of the UML metamodel

classes and associations as well as subsetting and redefinition of properties are expressed by associations between the relevant classes. A UML model is an instance of the metamodel. For the detailed description of the concepts, the Object Constraint Language (OCL) [4] is used as well as natural language text. 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 with quotations of the UML specification, showing that there is neither a concise nor a coherent definition, especially for the redefinition feature, thus leaving much room for interpretation.

3.1 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`.

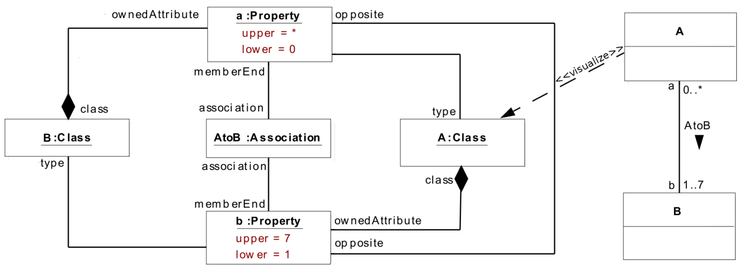

Fig. 3. Class diagram and its object diagram

In its left part, figure 3 shows an instance of the UML metamodel notated as an object diagram. In the right part, the visualization of this object diagram as a class diagram is shown. The two classes `A` and `B` are instances of `Class` and the association `AtoB` is an instance of `Association`. The association ends with the rolenames `a` and `b` are instances of `Property`. Each association end belongs to the association as a `memberEnd` and is connected to both classes at the association. One class defines the `type` of the property, while it belongs as an `ownedAttribute` to the one at the other end of the association. Furthermore, each property is connected to its `opposite` end at the association by a link.
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 based on their descriptions in the UML Super- and Infrastructure, using the model from figure 1 as example.

3.2 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. Therefore, features specified for instances of the general classifier are implicitly specified for instances of the specific classifier. Any constraint applying to instances of the general classifier also applies to instances of the specific classifier. [5, p. 55]

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

> When an association specializes another 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. [5, 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 [6, p. 113].

In the example model depicted in figure 1, the specialization of the association `HasRepr` by `HasScenario` leads to the condition $\forall i \in$ `UseCase` : i.`HasScenario` $\subseteq$ i.`HasRepr`. As each `HasScenario`-link is also an `HasRepr`-link, the set of elements reachable by `HasScenario`-links from a `UseCase` is a subset of the set reachable by `HasRepr`-links.

3.3 Subsetting

Similar to the specialization of classifiers, the refinement of properties is supported in the UML by a concept named *subsetting*. 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[6, p. 113].

In the example model, the constraint `subsets representation` at the role named `textRep` enforces, that for all `NFR`s i it holds i.`textRep` $\subseteq$ i.`representation`. All instances contained in the set reachable by `i.textRep` are also a member of `i.representation`.

3.4 Redefinition

Additionally 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 the description of redefinition is scattered over about 20 places in the Super- and Infrastructure, we try to give an interpretation which is compatible with the implementations of redefinition as realized e.g. by Scheidgen [7] and enables a practical usage of redefinition in our application domain. According to the UML,

> 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. [5, p. 41].
>
> A redefined property must be inherited from a more general classifier containing the redefining property. [5, p. 128].

Similarly to subsetting as described above,

> 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[6, p. 113].

This description is more precise than the other ones on the one hand but it contradicts to another description in the Infrastructure, which states:

> Redefinition prevents inheritance of a redefined element into the redefinition context thereby making the name of the redefined element available for reuse, either for the redefining element, or for some other[6, p. 129].

Both statements describe redefinition as a concept to forbid instances of an inherited property with one difference in the details. The first explains it as similar to subsetting restricting only instances of the redefined property, while the latter declares the redefined property as not existent at the class where it is redefined. With the latter interpretation, there is one major problem, since each subclass should contain all features defined for the superclass as described in section 3.2. If redefinition prevents inheritance, the redefined feature is not present at the subclass and it is not possible to treat the instances of the subclass as instances of the superclass.

It seems to be reasonable to use the first interpretation given above. Redefinition is treated as 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 the example model, the redefinition at the association end named `scenario` assert, that the redefined association end named `representation` must not be used at instances of `UseCase` and it is not possible to link `ReqRepresentations` to `UseCases`. Due to the redefinition, only `HasScenario`-links can be used at `UseCases`, leading to `Scenario` objects only. As cited above ("...the collections denoted by the redefining and redefined ends are the same..."), the collection of elements linked to a `UseCase` as `scenarioRep` is also accessible by `representation`, it holds $\forall i \in \texttt{UseCase} : (i.\texttt{representation} = i.\texttt{scenarioRep} \vee \forall k \in i.\texttt{representation} : k \in \texttt{Scenario})$. Thereby, even if nothing in this direction is stated in the UML, it seems that redefinition and subsetting are somehow similar. A redefinition can be seen as a subsetting with the additional semantics, that the association whose end is redefined is treated as abstract in the context of the class containing the redefining property. Further details on the interrelation between subsetting and redefinition are described in the following paragraph.

3.5 Interrelation of the Concepts

In contrast to the observation above, the relationship between the three refinement concepts is undefined in the UML. In particular, the Superstructure states this explicitly:

> The interaction of association specialization with association end redefinition and subsetting is not defined.[5, p. 41]

In contrast to this, the following is also stated in the Superstructure:

> Association specialization and redefinition are indicated by appropriate constraints situated in the proximity of the association ends to which they apply. Thus:

- The constraint {`subsets endA`} means that the association end to which this constraint is applied is a specialization of association end `endA` that is part of the association being specialized.
- A constraint {`redefines endA`} means that the association end to which this constraint is applied redefines the association end `endA` that is part of the association being specialized[5, p. 18].

In this description the interrelation between association specialization on the one hand and subsetting and redefinition on the other is nearly made explicit. Subsetting and redefinition are described as relationships between ends of an association and its specialization. The semantics suggested by this description is uncovered in the next section.

4 Formal definition

In the previous section it was shown, that the UML definitions of the refinement concepts for associations and their ends are not coherent. There there is an implicitly described interrelation, which is explicitly negated in the Superstructure. In the following, we try to make the interrelation explicit, using the model already shown in figure 1 as an example.

4.1 Mutual implication of subsetting and specialization

Based on the considerations presented up to now, we claim that for two associations `HasRepr` and `HasTextRep`, subsetting of one end of `HasRepr` by an end of `HasTextRep` implies the subsetting of the other end as well as the specialization of `HasRepr` by `HasTextRep`. Furthermore, the specialization of the associations implies a subsetting of both ends.

Proposition 1 (Subsets and specialization imply eachother). *Given a model as depicted in figure 1, the subsetting of `representation` by `textRep` is equivalent to the subsetting of `requirement` by `nfr`, and to the specialization of `HasRepr` by `HasTextRep`.*

Proof: According to the realization of association ends by properties and their definition in UML and the definition of the dot-notation in OCL, we know that for all instances of `Requirement` the sets reachable by the `representation` role and the `HasRepr` links contain the same elements.

$$\forall i \in \texttt{Requirement} : i.\texttt{representation} = i.\texttt{HasRepr} \qquad (1)$$

As `NFR` is a specialization of `Requirement`, its instances can be treated as `Requirement`s and the statement above is true also for `NFR`:

$$\forall i \in \texttt{NFR} : i.\texttt{representation} = i.\texttt{HasRepr} \qquad (2)$$

The same holds also for the association `HasTextRep` and its ends:

$$\forall i \in \texttt{NFR} : i.\texttt{textRep} = i.\texttt{HasTextRep} \qquad (3)$$

As `textRep` is marked as a subset of `representation`, it holds:

$$\forall i \in \texttt{NFR} : i.\texttt{textRep} \subseteq i.\texttt{representation} \tag{4}$$

Putting these formulas together, we get the following:

$$\begin{aligned}
&\forall i \in \texttt{NFR} : \quad i.\texttt{textRep} \quad \subseteq i.\texttt{representation} && (5)\\
\Leftrightarrow\ &\forall i \in \texttt{NFR} : i.\texttt{HasTextRep} \subseteq i.\texttt{representation} && (6)\\
\Leftrightarrow\ &\forall i \in \texttt{NFR} : i.\texttt{HasTextRep} \subseteq i.\texttt{HasRepr} && (7)
\end{aligned}$$

Thus, the set of elements reachable by an `HasTextRep`-link is a subset of the one containing the elements reachable by an `HasRepr`-link. This is not necessarily equivalent to a specialization between the two associations, because it could also be regarded as a constraint, that there must a `HasRepr`-link in every case when there exists an `HasTextRep`-one. As soon as `HasRepr` is marked as abstract, there are no direct instances which enforces a specialization of `HasRepr` by `HasTextRep` in this case. Consequentially, it is obvious to assume such a specialization in all cases and not to treat abstract and non-abstract associations different at this point. Hence, the *subsetting of one association end can be treated as equivalent to the specialization of the association itself.* Since associations are symmetric, it holds:

$$\begin{aligned}
&\forall i \in \texttt{NFR}, k \in \texttt{TextRep} :\\
&\qquad (k \in i.\texttt{textRep}) \Leftrightarrow (k \in i.\texttt{HasTextRep}) \Leftrightarrow\\
&\quad (i \in k.\texttt{HasTextRep}) \Leftrightarrow (i \in k.\texttt{nfr})
\end{aligned} \tag{8}$$

Thus, as the specialization of associations is equivalent to subsetting of an end, the subsetting of one end also implies the subsetting of the other one. This is exactly what was claimed in the proposition above. □

4.2 Redefinition is a subconcept of subsetting

While the relationship between the specialization of an association and subsetting of (one of) its ends has been explained above, the interrelation between specialization and subsetting on the one hand and redefinition on the other hand remains still unclear. Assuming the semantics of redefinition given above to be the desired one, we claim that redefinition of an association end is a special case of subsetting. Thus, also the redefinition of an association end implicitly implies a specialization of the association the end belongs to.

Proposition 2 (Redefinition is a subconcept of subsetting). *Given a model as depicted in figure 1, the redefinition of `representation` by `scenarioRep` implies the subsetting of `representation` by `scenarioRep`, of `requirement` by `useCase` and the specialization of `HasRepr` by `HasScenario`.*

Proof: From the description of redefinition ("...the collections denoted by the redefining and redefined ends are the same"[6, p. 113]) in the Infrastructure we know, that the sets denoted by **`representation`** and **`scenarioRep`** are the same for all instances of **`UseCase`**, i.e. it holds $\forall i \in$ `UseCase` : i.`scenarioRep` $=$ i.`representation`. Thus, it also holds $\forall i \in$ `UseCase` : i.`scenarioRep` $\subseteq$ i.`representation`, which is exactly the equation 4 of subsetting shown on page 9 above. With this equation, it is possible to perform the same steps as above, leading to the equation 7 as a conclusion of redefinition. Thus, *redefinition of association ends can be treated as a subconcept of subsetting and hence implies the specialization of associations*, which is also mentioned at some places in the superstructure as shown above. □

5 Proposal for the metamodel

Based on the considerations above, we propose some changes to the UML metamodel, which does not reflect the relations between the three refinement concepts for associations and its ends. Even if software development guidelines and (OCL) constraint as a workaround allow a usage of these concepts, we believe that a modeling language should be as concise and precise as possible without a need for additional regulations. Our proposal deals with the classes `Property` and `Association` which are depicted in figure 4. As we have shown above, redefinition is a specialization of subsetting. This should be present in the metamodel and is made explicit by the `subsets subsettedProperty` constraint at the association end named `redefinedProperty` at the class `Property`.

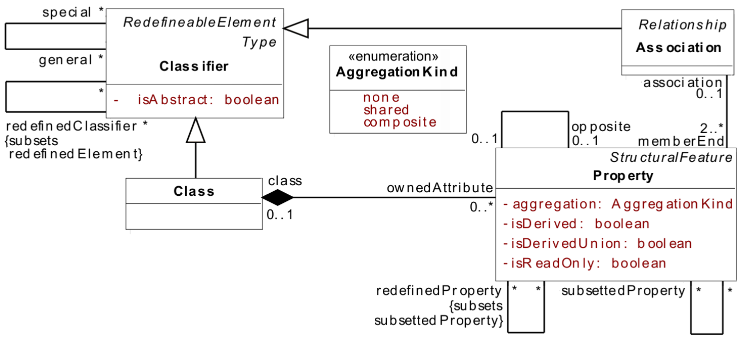

Fig. 4. Metamodel part affected by change proposal

Furthermore, the fact that subsetting and association specialization mutually imply eachother should be expressed appropriately. As UML class diagrams are not expressive enough, we use OCL to formulate this relationship as shown in the listing below.

```
context Association inv:
 self.memberEnd->forall(i |
   i.subsettedProperty->forall(k |
     self.general->contains(k.association)
 ) )
 and
 self.special->forall(k |
   k.memberEnd->forall(i |
      i.association.general->contains(self)
 ) )
```

Listing 1.1. OCL constraint for subsetting and specialization

The first part of this OCL invariant ensures, that subsetting of properties at association ends only occurs, if the association the subsetting property belongs to is a specialization of the association containing the subsetted property. The second part describes the dependency in the other direction. An association may only specialize another if its ends subset the ends of the specialized one. Due to the definition of redefinition as a special case of subsetting in the metamodel there is no explicit constraint necessary for redefinition.

6 Implementation

The practical usability of a (meta) modeling language such as the UML depends on frameworks which allow not only to draw models but also to handle model instances. Often this is achieved by the automatic generation of source code which implements the structure and behavior described by the model. For a UML class diagram, every class in the model is transformed to a class in the desired implementation language, for instance 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) [8] based on the ECore metamodel, which can be seen as a restriction of MOF 1.4. Similar to Essential MOF (EMOF), ECore 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 which contains the references to the appropriate linked objects.

6.1 Realization of roles by links

This realization of links is common also for other modeling environments as the ones mentioned in the related work below. In contrast to ECore and EMOF, some of them support subsetting and redefinition of association ends by synchronizing the relevant sets. An alternative to the implementation of links by coupled association ends is the treatment of

links as first-order elements which are known at the objects they connect. Besides an easy realization of association attributes and generalization, this allows also a convenient implementation of subsetting and redefinition. If the associations defined in the model are implemented as classes, then the links are instances of these classes similar to objects being instances of classes realizing the classes of the model. The classes implementing associations and the rolenames used for the association ends can be matched to each other. This allows the realization of a role access by a traversal of the right association objects.
Assuming that subsetting and redefinition go along with an (implicit) association specialization as shown in section 4, this implementation of links allows to use the specialization concept of Java or other object-oriented languages also to realize subsetting and redefinition. The association matched to a subsetting rolename is a subclass of the one matched to the subsetted rolename due to the dependency between subsetting and association specialization. Thus, the traversal of the links for the subsetted rolename will include the links matched to the subsetting rolename without any need of further synchronization.
As soon as an element should be added to the set denoted by the subsetting rolename, an instance of the association matched to this rolename is created. Due to the specialization, this link is also an instance of the association matched to the subsetted rolename and thus is included in the traversal of links to access this role. Also the redefinition feature can be realized by the specialization easily. The only additional effort necessary is a check when links should be created. The redefinition forbids instances of the general association, thus instances of the association matched to the redefining role need to be created even if elements should be added to the redefined rolename.

6.2 Implementation in JGraLab

An implementation of this ideas which is used in practice is the TGraph-library JGraLab[2], developed at the Institute for Software Technology of the University of Koblenz-Landau. JGraLab is the current implementation of the TGraph-Approach [9], which is based on typed, attributed, ordered, and directed graphs, called TGraphs. A restricted form of UML class diagrams called grUML (Graph UML) with an expressive power between EMOF and CMOF is used to describe graph schemas, which serve as metamodels for graphs.
The classes in a grUML diagram denote the node types while the associations denote types of edges connecting nodes. In JGraLab, Java classes are generated automatically out of graph schemas described in grUML. These classes realize the behavior described above and provide access to collections denoted by rolenames by traversal of the edges whose types are matched to the rolenames. As an example, the diagram depicted in figure 1 is transformed into Java interfaces and implementing classes for the node types `Requirement, UseCase` and so on and similar to interfaces and classes for the edge types `HasRepr, HasScenario` and `HasTextRep`.

[2] jgralab.uni-koblenz.de

Every link of type HasRepr between a node of type Requirement and one of type ReqRepresentation is represented by an instance of the generated class for HasRepr. The read and write access to the rolename representation is realized by the generated code in class Requirement, whose main methods to add and remove elements and to retrieve the list of linked elements are shown in the listing below.

```
public HasRepr addRepresentation(ReqRepresentation o) {
  return (HasRepr)graph.createEdge(HasRepr.class, this, o);
}

public void removeRepresentation(ReqRepresentation o) {
  //get first outgoing edge at vertex
  Edge e = getFirstEdgeOfClass(HasRepr.class, EdgeDirection.OUT);
  // iterate over edges at vertex
  while (edge != null) {
    Edge next = edge.getNextEdgeOfClass(HasRepr.class, EdgeDirection.OUT);
    if (edge.getThat() == o) edge.delete(); //delete edge to o
    edge = next;
  }
}

public List<? extends Representation> getRepresentation() {
  List<ReqRepresentation> adjacences = new ArrayList<ReqRepresentation>();
  Edge edge = getFirstEdgeOfClass(HasRepr.class, EdgeDirection.OUT);
  while (edge != null) {
    adjacences.add((ReqRepresentation) edge.getThat());
    edge = edge.getNextEdgeOfClass(HasRepr.class, EdgeDirection.OUT);
  }
  return adjacences;
}
```

Listing 1.2. Generated code to access rolename representation

In the addReqRepresentation(...) method, an HasRepr edge is created to the ReqRepresentation object which should be added to the Requirement instance in the role representation. Consequentially, these edges are traversed in the getReqRepresentation() method to access the role and removed in the removeReqRepresentation(...) method to delete elements from the set representation. Similar methods are created in the class implementing NFR for the role textRep as well as for the inherited one representation. Depending on the method called, either only HasTextRep or all HasRepr edges are taken into account and thus either the set denoted by the role textRep or the inherited one representation is accessed.
The same methods as shown for Requirement are also created for the class UseCase with one difference in the addReqRepresentation(...) method. To ensure, that only HasScenario links start at UseCase instances as enforced by the redefinition of representation by scenario, an additional test of the element to be added is necessary. If this element is an instance of Scenario, a HasScenario edge is created, otherwise an exception is thrown. The generated Java code is shown below.

```
public HasRepr addReqRepresentation(ReqRepresentation o) {
  if (!o instanceof Scenario)
    throw new SchemaException("Redefinition of role...");
  return graph.createEdge(HasScenario.class, this, o);
}
```

Listing 1.3. Generated code in class C

This kind of implementation ensures, that the constraints imposed by redefinitions of rolenames hold for all model instances. Using these methods, a JGraLab user can easily create and change graphs as instances of metamodels described by grUML class diagrams. As grUML is a restricted form of CMOF, its usage is not restricted to the context of graphs but open to arbitrary application domains, providing an convenient and efficient representation suitable also for a large amount of data, as it was presented in [2].
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.

7 Related Work

Besides JGraLab and EMF there are some other modeling frameworks allowing to deal with model instances by code generation. An implementation for MOF 2.0 supporting subsetting and redefinition is realized by Scheidgen [7] in the (meta-)modeling environment called "A MOF 2.0 for Java". Associations are represented by their ends which are realized by Java references stored in collections. Subsetting and redefinition are 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 he states that "links are usually not used directly. Links are of rather limited use." In section 6 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. [10]. As shown by Scheidgen [7, 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. In [11] the authors have also shown, that subsetting or redefinition of one association end implies subsetting of all other ends. 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. Taking the model from above as an example, `HasRepr`-links between `UseCase`- and `Scenario`-objects would be still valid. Based on this perspective, they propose an additional concept named `equals` to express, that for all instances of `Scenario` all elements of type `UseCase` included in `requirement` are also a member of `useCase`. If the existence of `HasRepr`-links is forbidden between `UseCase` and `Scenario` by the redefinition as it is in our interpretation, then the additional `equals` concept

is not necessary. Its semantics as presented by Amelunxen and Schürr is implicitly expressed by the redefinition. Furthermore, Amelunxen and Schürr 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.
The approached presented by Amelunxen is used for instance in the MOFLON[12] transformation language, while JGraLab and its implementation of subsetting and redefinition by association specialization is a supported repository backend of the MOLA transformation language[13].
Büttner and Gogolla [14] have inspected redefinition in the context of generalization of classes and co/contravariance, focusing on the redefinition of operations. They have shown, that UML subclassing cannot be used to define safe subtyping due to the covariant specialization supported with redefinition of attributes and operations. However, they argue that for the modeling of real world domains this specialization is more adequate than a typesafe contra- or invariant one, a position that is also shared by Ducournau [15].
Olivé [16] 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 [16, p. 230f]. While possibly useful if modeling without associations as first-class elements, this interpretation contradicts to the description in the UML superstructure [5, p. 18].

8 Conclusion and further work

The refinement concepts of association specialization and subsetting and redefinition of association ends allow the detailed modeling of relationships in UML class diagram. Especially if these concepts are used together in one model, a precise definition of them and their interrelation is necessary. In particular, the latter is not given in the UML but rather described as “explicitly undefined”. As a proposal to fill this gap in the UML metamodel it was shown, that redefinition can be treated as a subconcept of subsetting while the latter is equivalent to association specialization. This interpretation allows a high precision in modeling as well as an easy implementation of tools based on models. Treating links as first-class objects with an own identity as it is done in JGraLab, implementations for models using all three refinement concepts can be generated easily.
As ongoing and further work, the proposal of Genova et al. [1] for two kinds of multiplicities at each end of an n-ary association can be transfered to subsetting and redefinition. Allowing the redefinition of an association end for the edge and the vertex independently may lead to an improvement of the applicability of n-ary associations and might be subject of a further publication. A implementation in JGraLab is currently in development and will allow the practical usage of n-ary relationships in a broad application domain.

References

1. 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
2. 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
3. Object Management Group: Meta Object Facility (MOF) Core Specification, Version 2.0. (January 2006)
4. Object Management Group: Object Constraint Language, OMG Availiable Specification, Version 2.0. (May 2006)
5. Object Management Group: Unified Modeling Language: Superstructure, Version 2.2. (February 2009)
6. Object Management Group: Unified Modeling Language: Infrastructure, Version 2.2. (February 2009)
7. Scheidgen, M.: Description of Computer Languages Based on Object-Oriented Meta-Modelling. PhD thesis, Humboldt Universität zu Berlin (October 2008)
8. Steinberg, D., et al.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional (2009)
9. 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
10. Amelunxen, C., Bichler, L., Schürr, A.: Codegenerierung für Assoziationen in MOF 2.0. In: Proc. Modellierung 2004. Volume P-45 of Lecture Notes in Informatics., Bonn, Gesellschaft für Informatik (3 2004) 149–168
11. 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
12. Amelunxen, C., et al.: MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In Rensink, A., Warmer, J., eds.: Model Driven Architecture - Foundations and Applications: Second European Conference. Volume 4066 of Lecture Notes in Computer Science., Heidelberg, Springer (2006) 361–375
13. Kalnins, A., Celms, E., Sostaks, A.: Tool support for MOLA. Electronic Notes in Theoretical Computer Science **152** (2006) 83–96
14. Büttner, F., Gogolla, M.: On generalization and overriding in UML 2.0. In Patrascoiu, O., ed.: OCL and Model Driven Engineering, UML 2004 Conference Workshop, University of Kent (2004) 1–15
15. Ducournau, R.: Real world as an argument for covariant specialization in programming and modeling. In: OOIS '02: Proceedings of the Workshops on Advances in Object-Oriented Information Systems, London, UK, Springer-Verlag (2002) 3–12
16. Olivé, A.: Conceptual Modeling of Information Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)