Association Types: Motivation, Specification and Implementation with the XModeler$^{ML©}$

Ulrich Frank
Universität Duisburg-Essen
Essen, Germany
ulrich.frank@uni-due.de

Daniel Töpel
Universität Duisburg-Essen
Essen, Germany
daniel.toepel@uni-due.de

ABSTRACT

Associations represent an essential concept of multi-level models. While it is obvious that allowing for an unrestricted number of classification levels of objects is beneficial, associations are usually defined at one level only. There are, however, reasons to enable the definition of association types. Against the background of the explanation of these reasons, we analyze requirements for the semantics of association meta-types and the functionality of corresponding tools. Subsequently, an extension of the FMML^X metamodel is presented that forms the foundation of the implementation of association meta-types. Despite being focused on the FMML^X and the XModeler$^{ML©}$, it should also serve as an orientation for respective extensions of other multi-level languages and modeling environments. The concept and its implementation are evaluated against the requirement and compared to related work.

CCS CONCEPTS

• **Software and its engineering → Object oriented architectures**; **Domain specific languages**.

KEYWORDS

Association, Multi-Level Modeling, Underspecification, FMML^X

ACM Reference Format:
Ulrich Frank and Daniel Töpel. 2024. Association Types: Motivation, Specification and Implementation with the XModeler$^{ML©}$. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3652620.3688210

1 INTRODUCTION

Objects and classes respectively are at the core of multi-level language architectures, since they form the basis for defining essential, undisputed characteristics of multi-level models: every class is an object and there is an arbitrary number of possible levels. In addition, attributes and corresponding slots (see [9]) as well as, to a lesser extent, operations received considerable attention, too. They allow the definition of deferred instantiation, which is a mandatory feature for making use of higher level classes as powerful abstractions. In contrast, associations seem to have been rather neglected in modeling research. Or, in other words, traditional conceptions of associations may have been regarded as sufficient for multi-level modeling, too. This was, at least, the assumption the specification of the FMML^X ("Flexible Multi-Level Modeling Language") [4, 5] was based on.

Nevertheless, it has been obvious for some time that associations deserve more attention [1]. This is for various reasons. The terminology commonly used, including by us, is slightly misleading. Often, one speaks of associations that are "instantiated" into links, indicating correspondences of associations and classes and of links and objects. However, usually associations are not specified as classes, nor are links specified as objects. Therefore, it is, e.g., not possible to ask a link for the association it has been "instantiated" from. Also, and more important, there are differences regarding the specification of associations, which so far have been analyzed to a limited degree only (for a detailed analysis of specific differences see [14]). While, for example, the LML allows associations to be defined only between classes located at the same level, this restriction does not apply to other approaches (for a respective comparison of the LML and the FMML^X see [10]). Hence, there is no common notion of association.

At the same time, a closer look at associations leads to a number of open questions concerning possible requirements and their implementation. Töpel presents a comprehensive list of requirements [14]. Among others, it comprises a demand for underspecification and for enabling dependencies between associations. Underspecification refers, e.g., to the possibility to not completely specify multiplicities of associations defined between classes at higher levels. Defining an association as dependent of another association allows to restrict the set of links it is an abstraction of. On a more general level, Töpel called for a concept of association that supports reuse and integrity by including domain-specific semantics.

This paper ties in with the last two points. It is focused on meta-associations, that is, on a concept that allows for the specification of association types. To that end, we will at first motivate the need for association types through selected use cases. After that, related requirements are presented. Against this background, we will then discuss design options and present a specification of association types and their implementation in the XModeler$^{ML©}$. Subsequently, the presented solution is compared to related work. Note that we use a special terminology for expressing multi-level concepts that was inspired by [12] and described in more detail in [6]. Of particular relevance are the following terms. Instead of "instantiation", we speak of "concretization" in cases where traditional instantiation is accompanied by inheritance. Also, we refer to the tree of classes that are concretized from a class and its concretizations as

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MODELS Companion '24, September 22–27, 2024, Linz, Austria
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688210

a "concretization subtree". Properties of a class like attributes or associations that are meant for deferred instantiation are called "intrinsic". To avoid confusion, we use an "L" for marking levels instead of the "M" known from the MOF.

2 MOTIVATION

In recent years, our work on multi-level languages and models led to a growing need for the possibility to specify association types. The previous implementation did not allow for that. The core FMML^X metamodel did not cover associations. Instead, the use of associations was addressed only by additional classes such as **`Association`** or **`End`** at L1. In other words: associations were restricted to one generic, predefined association type. There were only two exceptions of this limitation. As a measure to avoid the counter-intuitive use of specialization relationships, we added delegation as a an additional association type [8]. It was, however, specified and implemented separately from the default association type. The only way to express further, domain-specific "types" was through the attribute **`type`** in **`Association`**, see Fig. 1.

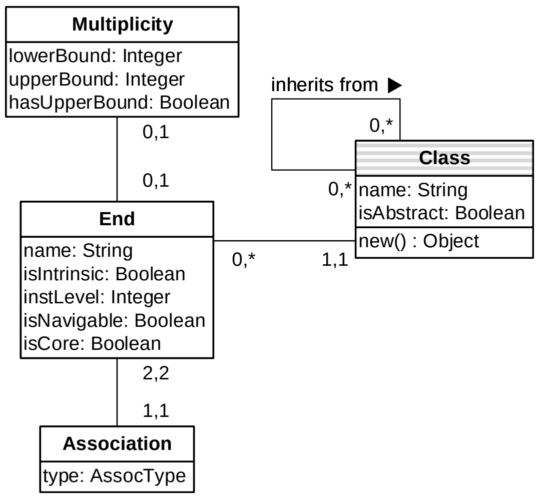


Figure 1: Previous specification of associations within the FMML^X

We first ran into the limitations of such a workaround when we designed multi-level modeling languages (DSMLs) with a specific concrete syntax. The concrete syntax of a DSML may require different kinds of associations to be distinguished in order to improve a diagram's comprehensibility. For example, a language for modeling IT infrastructures may include "uses" associations between classes representing software systems, and "runs on" associations between classes representing software systems and other classes representing platforms. Without some kind of abstraction over associations of a certain kind, it is not possible to specify a common graphical notation shared by all these associations. Furthermore, the different kinds of associations also show clear semantic differences. For example: while an application system may use a DBMS, it must not run on it. Since this constraint applies to all associations of this kind, it should be possible to specify it only once, which was not possible in the previous implementation.

This limitation of the previous implementation was confirmed by other use cases. To give one example only: a language for modeling organizational structures should include an association to express a relation between a line manager and her staff members. First, such an association is characterized by a strict 1..1 multiplicity on the side of the manager class. Second, apart from being restricted to classes that qualify to represent managers, it may also imply further semantic restrictions, e.g. concerning the age or the formal qualification of a manager in relation to the qualification of the corresponding staff members.

A further aspect that motivated our work on association types relates to dependencies between associations. Assume, for example, a multi-level model of cars would include an association between **`CarModel`** and **`TireType`**, both at L2, that allows assigning approved tire types (both at L1) to a certain car model. A further association, also defined at L2 but instantiated at L0 that serves the description of the particular tires mounted on a particular car would then be dependent on the first association in the sense that only those tires may be mounted on a car the types of which are approved for the car's model. In the previous version, this kind of dependency needed to be expressed by additional constraints (see example in [7]. It would be an obvious contribution to reuse, if there was a language concept that allowed for expressing this dependency. At the same time, such a concept could jeopardize model integrity, if it allowed for defining dependencies between any kind of associations. Without meta associations, this could hardly be prevented. A more precise definition of dependencies will be given in Section 4.

Finally, association types may help with model analysis. Imagine, for example, one needs to search for certain kinds of associations within one or multiple models. Without associations types that correspond to these kinds of associations, specifying such a search would be a complex undertaking.

An association type serves the specification of properties shared by a set of associations. First, it constrains the classes between respective associations. Second, it may constrain the multiplicities allowed for these associations. Furthermore, it may comprise further more specific constraints. In addition to these semantic characteristics, an association type may also serve the definition of a concrete syntax that applies to all respective associations. In Sections 3 and 4 this preliminary concept of association type will be refined.

3 REQUIREMENTS

The quest to enrich the FMML^X with meta associations led to a few general questions:

- What is an association and what are links?
- What is an association type or an association meta type?
- Related to the previous one: what are specific, invariant characteristics of certain kinds of associations that could be abstracted into an association type?
- Should an association meta type also allow to be instantiated into generalization/ specialization or delegation?
- How many levels of association types are required?

In general, associations between classes serve the specification of the set of possible links between instances of these classes. Within the UML, associations may be defined between more than two classes. The UML also allows for links to have a state. We do not follow the UML here, because we assume that a clear distinction of classes and associations contributes to a more comprehensive and consistent language design. That leads to a preliminary conception of an association as a relationship between *two* classes that specifies the set of possible links between instances of these classes. A link between two objects allows one object to navigate to the other. In case of a unidirectional association, only one of the two can navigate to the other, while a bidirectional linkage allows for navigating in both directions. There are different approaches to implement links, which are not essential for the notion of an association.

The use of associations in multi-level models suggests to extend this definition. The design of a multi-level model recommends expressing knowledge about the targeted domain at the highest level possible in order to avoid conceptual redundancy [6] or, in other words, *accidental complexity*. To this end, the FMML^{X} provides intrinsic associations that serve the definition of associations that can be concretized to at a lower level. When we, e.g., define classes such as **`Computer`** and **`PeripheralDevice`** at L3, we already know that particular computer models, represented at L1, may be linked to particular peripheral device models at L1. This knowledge can be expressed by defining that the association **`uses`**, even though defined at L3, represents a set of links at L1 only. Note that the FMML^{X} allows associations between classes at different levels, since this has proven to be extremely useful.

An association meta type, on the other hand, goes clearly beyond intrinsic associations, since it allows to define semantics shared by a set of congenial associations. Aspects of association type semantics include the classes, associations of this type may connect, the multiplicities, as well as further constraints. For example, an association type **`eligible`** may be defined for connecting classes that are concretized from the classes **`Position`** and **`OrganizationalRole`** (and only from these!), e.g, between the classes **`SoftwareAnalyst`** and **`SoftwareReviewer`**.

Hence, an association type requires the specification of two classes A and B, with A at Ln and B at Lm, where n > 1 and m > 1 (m and n do not have to be equal). An association of a particular association type can then be defined between classes C and D, where C is element of A's concretization tree, and D is element of B's concretization tree. In addition, an association type may comprise further constraints that refer to the state of associated or linked objects.

While regarding generalization/specialization as specific types of a general association meta type may seem like an attractive option, there would be little benefit of such an abstraction and, at the same time, a considerable growth of complexity with respect to its specification. Obviously, the commonalities between "regular" association types and generalization/specialization are very limited, while the semantic differences are substantial. Therefore, we decided against this option.

Different from generalization/specialization, there are clear commonalities between delegation and "regular" association types. At the same time, however, delegation comprises specific peculiarities, which are not shared by other association types. In addition, the XModeler$^{ML©}$ already includes a well working implementation of delegation [8]. Therefore, we did not see the need to make it an element of the targeted association meta types.

It turned out that determining the number of required association levels quickly exceeded the scope of our imagination. To address this question and to develop more specific requirements, we performed a methodical analysis of examples. To that end, we referred to a repository of multi-level models we created over the last years. In addition, we created new examples with specific emphasis on associations. The following list represents the result of the requirements analysis. It distinguishes between requirements that concern language semantics and those that relate primarily to a corresponding model tool. The first are numbered with "Sn", the latter with "Tn".

Requirement S1: It should be possible to define specific association types as language extensions. *Rationale*: Enabling the definition and use of specific association types fosters reuse, hence modeling productivity, integrity, and contributes to maintainability, too.

Addressing this requirement implies a conception of association type, that is, of the characteristic features that are subject of an association type specification.

Requirement S2: To enable the specification of an association type, it should be possible to restrict the set of classes, between which corresponding associations may be established. It also calls for the specification of multiplicities and further constraints. *Rationale*: These are characteristic features of associations.

Our analysis of various example models revealed that requirement 2 needs to be refined to fit the peculiarities of multi-level models. For instance, IT management may have defined the policy that a maximum of two printers may be attached to a computer, but only one scanner (both at L0). Within a corresponding multi-level model, any multiplicity defined with the intrinsic association at L3 between classes like **`Computer`** and **`PeripheralDevice`** could not cover all cases adequately, and, hence, would allow for a violation of the policy.

Requirement S3: It should be possible to underspecify the definition of multiplicities that characterize an association type. *Rationale*: Depending on the classes concretized at lower levels, multiplicities may vary. Underspecified multiplicities indicate the need for refinement at a lower level.

Note that S3 applies only in cases where the principle character of an association does not differ too much. Otherwise, the idea of an association type would be compromised.

Requirement T1: Upon the creation of an association the type of which includes underspecified multiplicities, a modeling tool should guide users with properly filling the remaining specification gaps. *Rationale*: This is a prerequisite for the consistent use of underspecified multiplicities.

Constraints are an important instrument to refine the semantics of modeling concepts. That leads to requirements regarding the specification of constraints.

Requirement S4: The specification of an association type should allow for the definition of constraints that refer to the state of linked objects. *Rationale*: The proper use of associations of a certain type may restrict the permitted states of objects that are linked according to an association of this type.

Assume, for example, an association type **`approvedBy`** that is defined for concretizations of the classes **`EducationalInstitution`** and **Accreditation**, both at L2 (see Fig. 2. While the range of classes, associations of this type may be applied to, is restricted only through the respective classes, links that may be created as "instantiations" of these associations are only permitted in those cases where an educational institution is older than 5 years. Apparently, this constraint can be applied only to classes that define intrinsic properties that are initialized at the level where links would be created.

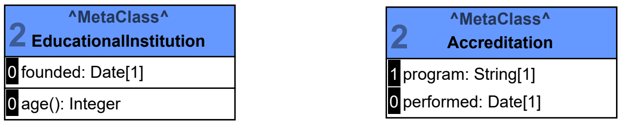


Figure 2: Restrictions concerning states of linked objects

Requirement T2: The specification of an association type should allow for restricting the range of objects it is applicable to. *Rationale*: There are cases, where an association type is restricted to specific classes that are concretized from generic classes, where it would be misleading to define a specific constraint for all possible concretizations. Assume, you want to introduce an association type **marriedTo**. Its use should be restricted to classes representing human beings. Furthermore, assume there is no reason to introduce a further specific (meta) class of a **Person**. Instead, **Person** is instantiated from the generic class **Class**. Apparently, it would make no sense to allow for using **marriedTo** between all possible instances of **Class**.

In multi-level models, the legitimate use of an association of a certain type may also be restricted by the state of the classes participating in this association, which leads to the following requirement:

Requirement S5: The specification of an association type should allow for the definition of constraints that refer to the state of classes participating in an association of this type. *Rationale*: The example in Fig. 2 is suited to illustrate this case, too. A certain accreditation type at L1, e.g., for approving Master's programs in computer science, may be associated only with those institutions at L1 that offer Master's programs.

Requirement S6: It should be possible to define that only associations of a certain type may depend on associations of another certain type. *Rationale*: While allowing for defining dependencies between any kind of associations promotes flexibility, it may also be a threat to model integrity, since there are many cases where a dependency would be completely absurd. Therefore, modelers should be given the choice between the two alternatives.

To further refine this requirement, a more precise definition of dependencies between associations is required. For an association c1 between the classes A1 at Ln and B1 at Lm to depend on an association c2 between the classes A2 at Li and B2 at Lj, the following prerequisites need to be satisfied: n <= i and m <= j. Then, for every link "instantiated" from c1 there must be a corresponding link "instantiated" from c2. The examples in Fig. 3 illustrate what "corresponding" may mean.

Assume that in the above example the association **issuesOrder** depends on the association **registeredAt**. For a link created between two objects p1 and s1 created from **issuesOrder** to be valid,

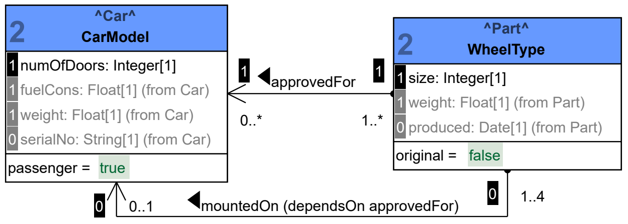

2
^Car^
CarModel
1 numOfDoors: Integer[1]
1 fuelCons: Float[1] (from Car)
1 weight: Float[1] (from Car)
0 serialNo: String[1] (from Car)
passenger = true
2
^Part^
WheelType
1 size: Integer[1]
1 weight: Float[1] (from Part)
0 produced: Date[1] (from Part)
original = false
1 approvedFor 1
0..*
1..*
0 0..1 mountedOn (dependsOn approvedFor) 0 1..4

Figure 3: Illustration of dependencies between associations

there needs to be a corresponding link between both objects that was created from **registeredAt**.

The following requirements concern the design of corresponding tooling.

Requirement T3: It is required that a modeling tool supports the consistent use of association types. That includes detecting and, if possible, preventing violation of implicit and explicit constraints. *Rationale*: Since semantics of association types can become rather complex, tool support is mandatory to avoid integrity problems.

Requirement T4: A modeling tool should handle association types like any other language concept, e.g., by including them in the palette upon their creation. *Rationale*: A natural integration of association types with other language concepts facilitate the convenient (re-) use of association types.

Further requirements regarding the modeling tool relate to support for model management, that is for supporting consistent change operations.

Requirement T5: It should be possible to edit the specification of an association type after associations of this type have been created. *Rationale*: This kind of change is hard to avoid. If it happens, tool support is important to cope with complexity.

Relaxing the constraints that define the semantic of an association type will usually be less challenging than further restricting them.

Requirement T6: A modeling tool should allow for deleting association types. Since this suggests deleting corresponding associations first, the tool should support identifying these. *Rationale*: In case an association type turns out as a misconception, it should no longer remain in a system.

Concerning the required number of abstractions over associations, that is, association type, association meta type, etc., we found a few cases where additional levels above association types would make sense. One example is marriage. Due to the fact that there are different kinds of constraints that apply to marriage depending on country and culture, one could think of a meta-association that represents the commonalities of all these different types, such as, e.g. that marriage is only possible between human beings. Other cases relate to different legal regulations, too. For example, issuing an order is possible between humans or organizations on the one side, and humans and organizations on the other side. With respect to humans, there exist different regulations concerning age, and maybe, gender, of the customer. We came, however, to the conclusion that these cases do not constitute urgent requirements. At the same time, introducing higher levels of association types would clearly contribute to complexity and, likely, to confusion.

Therefore, we decided to restrict the implementation to one meta-association only. Nevertheless, we accounted for this case with the final requirement.

Requirement S7: The specification of an association meta-type should be extensible in the sense that it allows for adding further levels of association types. *Rationale*: With more cases that may appear in the future, it may turn out that the benefit of additional levels of association types would justify the corresponding implementation effort.

4 DESIGN AND IMPLEMENTATION

As a comprehensive language engineering tool that allows for an arbitrary number of classification levels, the XModeler provides a strong foundation for developing a specification and a corresponding implementation that satisfy the requirements described above. However, the XModeler as well as the FMML^X, which is an extension of XCore, the metamodel, the XModeler is based on, treat associations originally as second class citizens. They are not part of the XCore. Instead, they are defined in an additional package named **`Associations`** (see Fig. 4). Within that package, the class **`Association`** is located at L1. Hence, it can be instantiated to an association, which does not allow for further instantiation (which is the reason, why a link does not qualify as an instance of an association). In order to enable association types, it must be possible to somehow enable an association meta type. With respect to requirement S7, it should furthermore be possible to later add further levels of association types.

In the following, we first describe the modifications made to the metamodel, to then elucidate selected details of the corresponding implementation. Subsequently, the implementation it illustrated by a demonstration of corresponding features of the XModeler$^{ML©}$, which is the version of the XModeler that implements the FMML^X.

4.1 Metamodel

To enable additional levels of association types, we made use of the recursive construction of XCore that enables multiple levels of classification. As a default, **`Class`** is located at L2. Hence, creating an instance of **`Class`** produces a class C at L1. To allow for placing C on whatever classification level you want, the following mechanism is applied. To lift C from L1 to L2, C inherits from **`Class`**. Then C is instantiated to C1 (at L1), which would then inherit from **`Class`**, too, thus lifting C1 to L2 – and C implicitly to L3. By repeating this procedure C can be lifted to any level. The implementation of the FMML^X within the XModeler$^{ML©}$ hides this approach from the user and allows for the direct creation of classes at any level the user wishes for.

Note that assigning levels to associations and association types may cause confusion. From a conceptual perspective, an association is instantiated from an association type and "instantiated" into a link. That would suggest placing association types at L2, associations at L1 and links at L0. However, from an implementation perspective, a particular association is located at L0. A corresponding link is not an instance of an association, but realized through slots that store object references, which is similar to the relation between attributes and slots. Hence, in this view, an association type is at L1. To support a conceptual perspective on associations without compromising the implementation view, we use a specific scheme to describe levels related to associations. Levels in this scheme are numbered with the prefix "AL". According to this view, an association type is located at AL2, an association at AL1.

We adapted this approach to allow for multiple levels of associations, too. At first, we restricted the extension to one additional level, that is, to the definition of association types at L1 or AL2 respectively. That requires an association meta type at L2 (AL3). In order to achieve this, the **`AssociationType`** inherits from **`Class`** (Req. S1). As a consequence, **`AssociationType`** is not only located at L2, it can also be associated with constraints, which are required to refine the semantics of association types.

It is important to know that these classes are not defined with the FMML^X, because the extension should not affect existing models/packages within the XModeler. Therefore, it is not possible to assign explicit levels to these classes and, more important, to define intrinsic properties for deferred instantiation. That created a problem, because certain properties of associations are already known at the type level: every instance of an association requires links to either one (in case of a uni-directional association) or both associated classes (in case of a bi-directional association). Therefore, it should be possible to define them already there to avoid conceptual redundancy. To achieve this, the abstract class **`AbstractAssociation`** was introduced (Req. S2). It is located at L1 and defines the attributes **`source`** and **`target`**. Furthermore, an association should be characterized by a name. Therefore, **`AbstractAssociation`** inherits from **`NamedElement`** and not from **`Class`**, because that would have lifted it to L2.

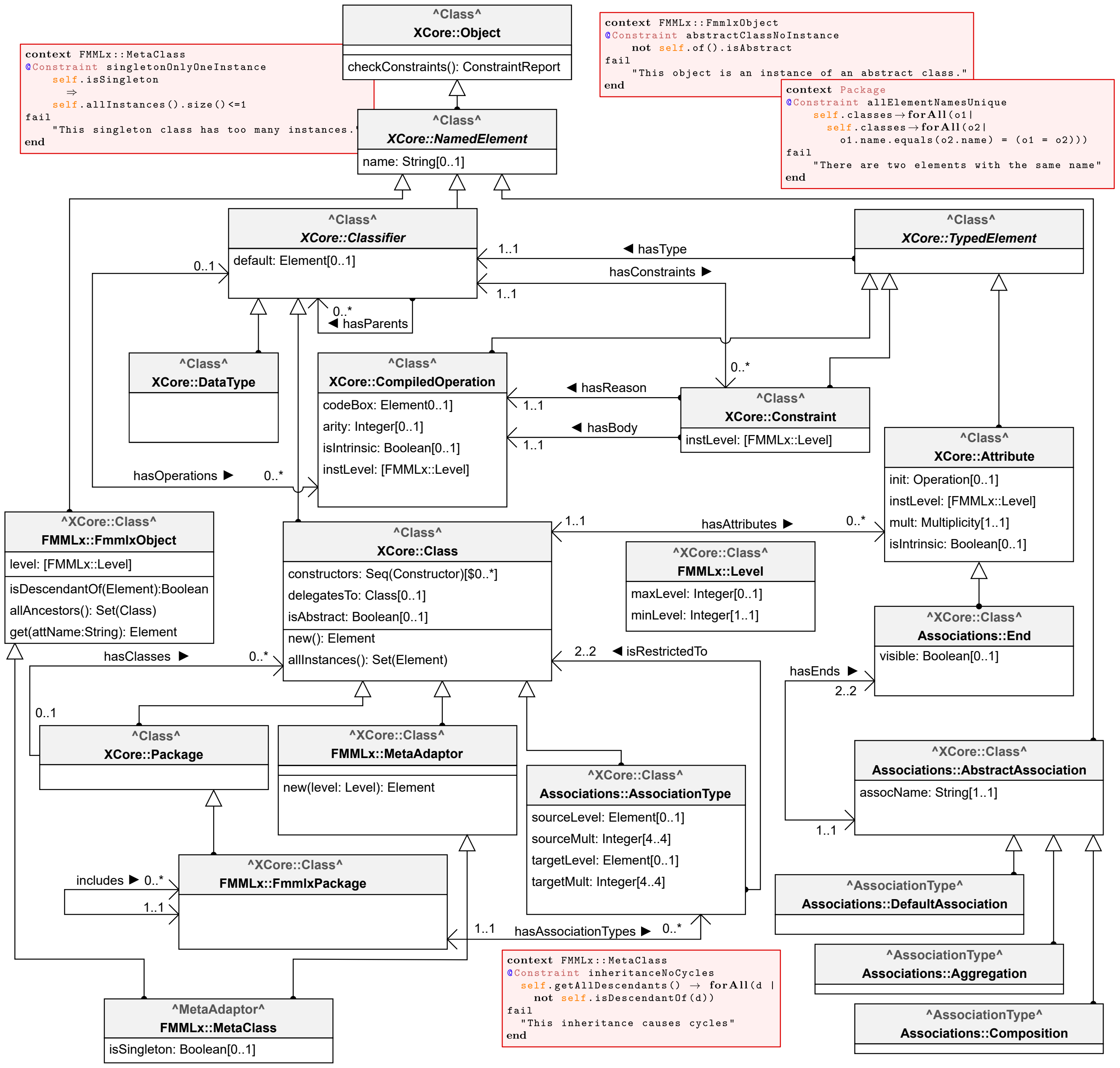
A particular association type is created in a two-step process. First, the association type is instantiated from **`AssociationType`**. Second, the association type inherits from **`AbstractAssociation`**. This is to make sure that every association type (at L1 or AL2) has the attributes **`name`**, **`target`**, and **`source`**, which in turn serve the definition of corresponding slots within a particular association. Fig. 4 shows the extended, slightly simplified metamodel. It is restricted to those classes and properties that are essential for understanding the specification of association types.

The metamodel consists of classes that are part of XCore and those that were added for the definition of the FMML^X. Associations are defined in the additional package **`Associations`**. The class **`AssociationType`** serves the creation of association types. Since it inherits from **`Class`**, it is located at L2 (or AL3 respectively).

4.2 Implementation

The implementation for S1,S2 and S3 mainly consists of four parts. First, the extension of the metamodel had to be implemented. Second, the specification of semantic properties of an association type was addressed by providing users with a template that allows to fill in implicit constraints on multiplicities. Third, constraints had to be added to ensure that associations indeed conform to the restrictions on an association type supplied by a modeler. Fourth, the diagram editor had to be adapted to allow for displaying a specific notation defined for associations of a certain type.

The metamodel extension takes place in the **`Associations`** package as shown in Fig. 4. From the perspective of classes there are



```
context FMMLx::MetaClass
@Constraint singletonOnlyOneInstance
    self.isSingleton
      ⇒
    self.allInstances().size()<=1
fail
    "This singleton class has too many instances."
end
```

```
context FMMLx::FmmlxObject
@Constraint abstractClassNoInstance
    not self.of().isAbstract
fail
    "This object is an instance of an abstract class."
end
```

```
context Package
@Constraint allElementNamesUnique
    self.classes→forAll(o1|
      self.classes→forAll(o2|
        o1.name.equals(o2.name) = (o1 = o2)))
fail
    "There are two elements with the same name"
end
```

```
context FMMLx::MetaClass
@Constraint inheritanceNoCycles
  self.getAllDescendants() → forAll(d |
    not self.isDescendantOf(d))
fail
  "This inheritance causes cycles"
end
```

Figure 4: Simplified version of the extended metamodel

no structural changes, as classes refer to association ends as before. There is now a class **AssociationType** at AL3, which can be instantiated into an association type on AL2. This class holds the constraints on multiplicities and the involved classes defined by the modeler. Currently, this class also holds the specification of the visual appearance of the associations (on AL1) and links (on AL0), which is faded out in Fig. 4,

Due to a lack of deferred instantiation in the core packages, **AssociationType** on AL3 cannot define general properties of associations on AL1. As already mentioned above, these are instead defined in the abstract class **AbstractAssociation** on AL2, which all association types on AL2 have to inherit from. This is ensured by the constructor of **AssociationType** and also by the constraint in Listing 1.

The former class **Association** has been replaced by the new class **DefaultAssociation** (on AL2), which is an instance of the newly created meta-class **AssociationType** and as mandated by Listing 1, a subclass of **AbstractAssociation**. It has no restrictions on the involved classes nor the multiplicities. This is done by setting **Object** as the required class, and setting lower bounds

```
context Associations::AssociationType
@Constraint assocTypeInheritAbstract
  self.parents.contains(Associations::AbstractAssociation)
fail
 "This association type does not inherit from AbstractAssociation"
end
```

Listing 1: An association type on AL2 must inherit from `AbstractAssociation`

of several properties to 0, and not setting an upper bound. The implementation also includes **`Aggregation`** and **`Composition`** on AL2 which also have no restriction on the involved classes but for the multiplicities. They also have a different appearance. To add a customized association type, the modeler needs to supply the respective information, e.g. through the dialog in Fig. 5. These association types are added to the package and become available for the modeler to choose from alongside the three default association types, when they intend to add an association (see Fig. 6).

```
context Associations::AbstractAssociation
@Constraint assocMatchMetatype
  self.source.type.isKindOf(self.sourceRestriction.type)
    and
  (self.source.type.isKindOf(FMMLx::MetaClass) ⇒
    self.source.type.level.matches(self.sourceLevel))
  and
  [ditto for target side]
fail
  "This association cannot be drawn between these classes"
end
```

Listing 2: An association on AL1 can only be drawn between appropriate types

There are two constraints checking whether an association conforms to the association type. The first, in Listing 2, checks whether both involved classes are of the correct type, and if they are FMML^X-classes, the level is checked as well. The other constraint checks the multiplicities. This constraint, which is not shown here, calls an operation producing a list of problems, and checks whether the list is empty. The operation is shown in Listing 3.

The multiplicity restrictions are stored in an array of four numbers. The first two are the upper and lower bounds of the actual lower bound, while the last two restrict the upper bound. The value null indicates infinity. For example, the restriction $[1, 4, 3, null]$ would indicate that the lower bound must be between 1 and 4, while the upper bound must be 3 or higher. This would allow the multiplicity 2..6, but prevent 2..2, as the upper bound is too small or 0..∗, as the lower bound is too small.

To allow for a customized appearance of associations and links, various options are conceivable. To provide for a proof of concept in general, the implementation is currently restricted to a set of predefined shapes to choose from. The current implementation of edges, which include associations and links, is a series of straight lines connected at right angles and joining the nodes, i.e., objects and classes, at so called ports. These may be with or without decorations. User can customize the notation of associations by defining color, width, and form (solid or dashed). Dashed lines are represented by an integer array of even size, indicating the length of the lines and the gaps of the repeating line pattern. Also, end decorations

```
@Operation checkMultiplicities()
  let issues = {} in
    if  self.of().sourceMult = null
      then issues := issues + {"No multiplicity rules for
source supplied."}
    elseif not self.source.mult.isKindOf(Multiplicities::
CollectionMult)
    then issues := issues + {"Source multiplicity must be of
type Multiplicities::CollectionMult."}
    else
      if self.source.mult.lowerBound < self.of().sourceMult.at
(0) then
        issues := issues + {"Source lower bound too small"}
end;
      if self.of().sourceMult.at(1) <> null andthen self.
source.mult.lowerBound > self.of().sourceMult.at(1) then
        issues := issues + {"Source lower bound too large"}
end;
      if self.source.mult.hasUpperBound then
        if self.of().sourceMult.at(2) = null
        then issues := issues + {"Source must not have an
upper bound"}
        else
          if self.of().sourceMult.at(2) > self.source.mult.
upperBound then
            issues := issues + {"Source upper bound too small
"} end
        end
      end;
      if self.of().sourceMult.at(3) <> null then
        if self.source.mult.hasUpperBound
        then
          if self.source.mult.lowerBound > self.of().
sourceMult.at(3) then
            issues := issues + {"Source upper bound too large
"} end
        else
          issues := issues + {"Source must have an upper bound
"}
        end
      end
    end;
    [ditto for target];
    issues
  end
end
```

Listing 3: Operation returning multiplicity issues

have been implemented. The modeler may choose from a number of predefined shapes such as arrows, triangles, diamonds or circles.

The specification of an association type (AL2) requires both, the definition of the notation for respective associations (AL1), and for the corresponding links (AL0).

The definition of domain-specific association types can be part of a DSML provided as an add-on to the XModeler$^{ML©}$. Alternatively, users of the tool may define their own association types. Only aggregation and composition were added to the FMML^X as specific association types. In this case, the implementation was restricted to add constraints on multiplicities of compositions.

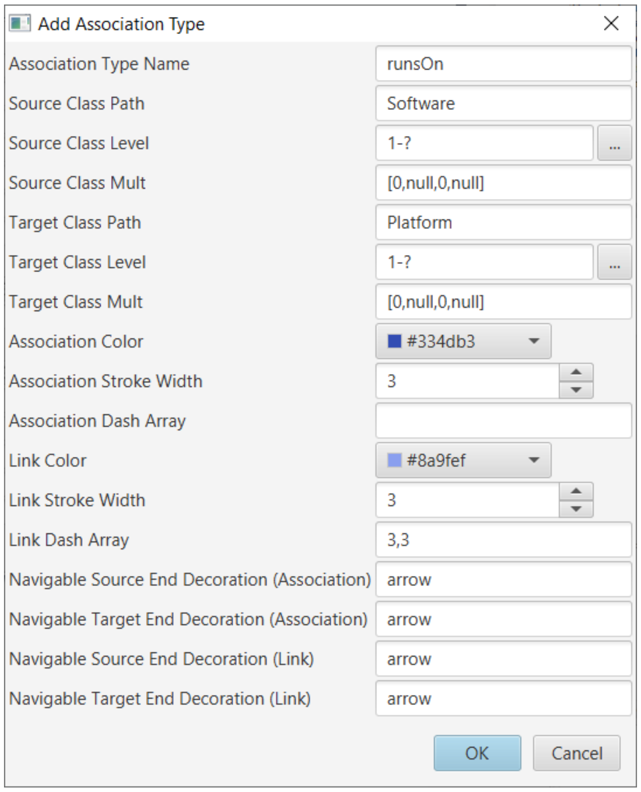
A further issue relates to the persistence of association types and with it its re-usability in other models. In the XModeler$^{ML©}$, one package as a whole can be made persistent. If it does not depend on other packages apart from the core packages such as **`XCore`**, **`FMMLx`** or **`Associations`**, this is sufficient to recreate a package to its state when it was saved. If a package depends on other packages, these have to be loaded first. Also special attention has to be paid to changes of those packages, further packages rely on. To prevent issues with referential integrity, a package which another package makes use of must be explicitly named as such. This serves as a reminder for the modeler to load it first and also populates the palettes and the menus accordingly.

Association types are part of packages and as such, what was said about packages above, equally applies to association types as a part of it. This also allows packages to use association types from other packages.

With respect to S4 and S5, users need to be able to write XOCL constraints. Given the peculiarities of specific constraints, this can hardly be avoided. Also, the constraints have to be added at the console, since the template for specifying association types does not include a feature to add constraints.

4.3 Demonstration

Users can define an association type by specifying certain features within a template (Fig. 5). The template is rather extensive and probably confusing at first sight, since it accounts for multiple aspects including those concerning the concrete syntax. However, once a user has understood the template, the specification is straightforward.

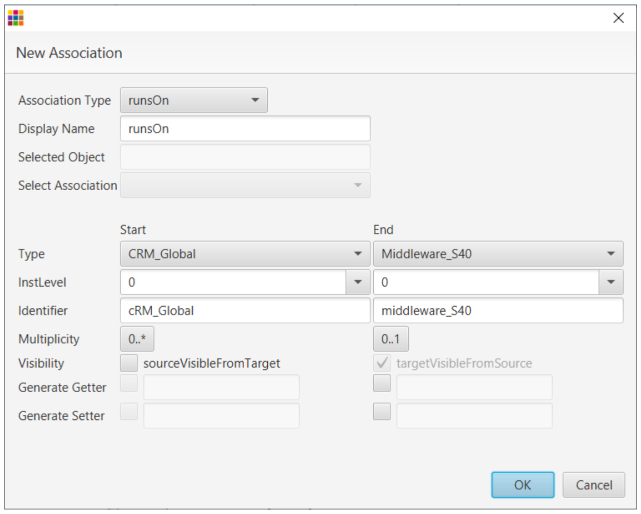


Add Association Type

Association Type Name	runsOn
Source Class Path	Software
Source Class Level	1-?
Source Class Mult	[0,null,0,null]
Target Class Path	Platform
Target Class Level	1-?
Target Class Mult	[0,null,0,null]
Association Color	#334db3
Association Stroke Width	3
Association Dash Array	
Link Color	#8a9fef
Link Stroke Width	3
Link Dash Array	3,3
Navigable Source End Decoration (Association)	arrow
Navigable Target End Decoration (Association)	arrow
Navigable Source End Decoration (Link)	arrow
Navigable Target End Decoration (Link)	arrow

Figure 5: Template to specify association type

The use of association types within the XModeler$^{ML©}$ is demonstrated in Fig. 7. It includes associations of the two types **`runsOn`** and **`uses`**. Due to the semantics of **`runsOn`**, an attempt to define a corresponding association between **`DBMS_4000`** and **`ERP_X200`** causes the violation of a constraint. The classes in the upper part of the figure are located at L1. The two objects at the bottom serve the demonstration of the representation of links at L0. Note that we faded out the representation of a few features from the diagram in order to save space.



New Association

Association Type	runsOn
Display Name	runsOn
Selected Object	
Select Association	

	Start	End
Type	CRM_Global	Middleware_S40
InstLevel	0	0
Identifier	cRM_Global	middleware_S40
Multiplicity	0..*	0..1
Visibility	sourceVisibleFromTarget	✓ targetVisibleFromSource
Generate Getter		
Generate Setter		

Figure 6: Definition of an association of a certain type

The XModeler$^{ML©}$ can be downloaded from www.le4mm.org. A screencast that illustrates the use of association types can be found there, too (Menu "XModeler$^{ML©}$-> Examples 2").

4.4 Preliminary Evaluation

The evaluation of the presented work consists of two parts. First, we compare the solution against the requirements (see Tab. 1). Second, we report on experience gained so far with the use of association types and peculiarities of the implementation.

While the requirements are widely satisfied except for one, two principal aspects of the evaluation still need to be accounted for. The first concerns the question whether the effort to implement (and use) association types pays off. Given the fact that so far most researchers that work on multi-level modeling seem to have not felt the need for association types, this is a valid concern. There are two arguments that invalidate it. First, and most important, providing for the specification of association types is an obvious consequence of the idea of DSMLs. A DSML is supposed to provide domain-specific concepts in order to promote model integrity and modeling productivity. Second, related to the first, our experience with the design of multi-level DSMLs in the context of enterprise modeling clearly showed the benefit of an abstraction over associations.

With respect to the tool, there is one issue that requires further consideration. Currently, association types are defined within a package, that is, within a particular multi-level model. Hence, their reuse within other models requires importing corresponding packages. This may be inappropriate in those cases where only association types are needed (together with the two classes required to define them).

Currently, the graphical notation of association types is specified only once with the association type. A separate specification, as it is featured by the *Concrete Syntax Wizzard* that is part of the XModeler$^{ML©}$, would allow for more flexibility

Table 1: Requirement Satisfaction

Req.		Comments
S1	+	The additional level is achieved by having **AssociationType** inherit from **Class**.
S2	+	In order to satisfy this requirement, every association type inherits from the abstract class **AbstractAssociation** upon its creation.
S3	+	The specification of multiplicities within association types is done by tuples that define the range of possible minimum and maximum values. If the difference between these values is larger than zero, the corresponding multiplicity is underspecified and needs to be refined with the creation of associations of this type.
S4	+	Since **AssociationType** inherits from **Class**, it is possible to assign constraints to association types. The specification of such a constraint can refer to states of concretizations of the classes connected through an association of the respective association type.
S5	+	Similar to the previous one: a constraint assigned to an association type can refer to the states of the connected classes.
S6	○	It is possible to specify that certain kinds of dependencies may be defined only between associations of certain types. Currently, users are not provided with a dialog to fill in the specification. Instead, the specification can be done only via the console.
S7	○	The metamodel allows for adding further levels of association types, e.g., by having newly created types inherit from **AssociationType**. However, currently, there is no need for such an extension. Also, we lack a clear conception of a type of association types.
T1	+	Users are provided with templates to specify and use associations types, see Fig. 5 and Fig. 6.
T2	+	An association type can not only restrict its ends to concretizations of a given class, it can also be restricted to that very class.
T3	+	In some cases, inconsistent states are prevented. In the remaining cases, inconsistent states would violate constraints and are highlighted as such.
T4	–	Extending the palette not yet implemented. Associations types available in dialog for adding associations (see Fig. 6).
T5	+	An association type can be edited in the console. In case, changing an association type violates constraints, inconsistent associations or links will be highlighted.
T6	○	Association types can only be deleted if no longer in use. The respective association have to be deleted or migrated first.
		+ fully implemented ○ partially implemented – not yet implemented

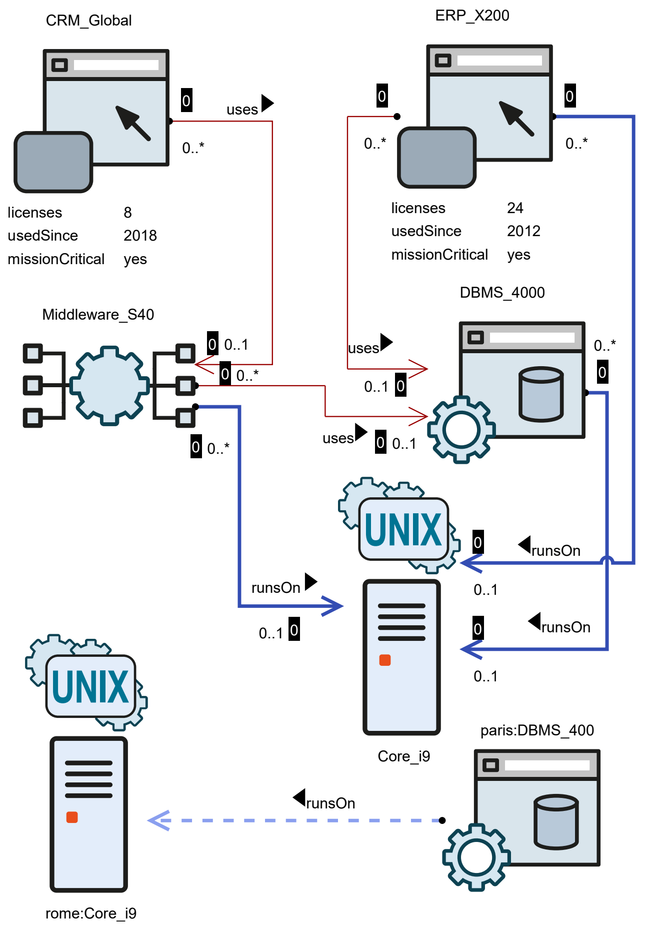


Figure 7: Use of specific graphical notations for association types and corresponding links (below) in the XModeler$^{ML©}$

5 RELATED WORK

To the best of our knowledge, there is only one other approach that addresses association types. In their foundational work on "connections", intended as a abstraction over associations and links, Atkinson et al. [1] speak of "connection types" that are instantiated into associations. However, from the perspective of the FMML^X, the corresponding specification seems to address mainly peculiarities of intrinsic associations. Other approaches address certain aspects of abstraction over associations. They concern deferred instantiation, underspecification, redefinition, and dependencies of associations.

Similar to [1], various approaches to multi-level modeling support the specification of associations at higher levels, the "instantiation" of which is deferred to lower levels. This corresponds to the abstraction that is referred to as intrinsic association in the FMML^X, see Section 3.

Deferred "instantiation" of associations is enabled by various approaches, which, however, differ in some details. Within LML [2], an association has a potency which indicates how many levels are between the association and the links it is to be "instantiated" to. At the levels in between, unless directly adjacent with a potency of 1, associations have to be created to actually forward the association down to the link level. These relationships are neither associations nor links, but rather stand between them, analogously to their notion of clabjects between classes and objects. Obviously, such an approach implies both ends of the association to have the same potency. Therefore, associations must not cross levels, which is different from the FMML^X. "Dual Deep Modeling" ([13] (DDM) allows associations to cross levels. Associations have a dual potency ($m - n$). Each of these potencies serves to define how often the two involved classes may be concretized. Since links do not allow for further "instantiation", their dual potency is $0 - 0$. As in FMML^X, "Flexible Multi-Level Modeling" (FMLM) [11] does not provide for an additional layer of abstraction between associations and links. It allows the potency of an association to be flexible, so that links "instantiated" from it may appear on different levels. However, links on higher levels are not an abstraction of links on lower levels.

Underspecification of association types is not addressed by any approach we would know of. Redefinition of associations, however, provides a mechanism that is in some respect similar. By redefining an association that was defined at a higher level, it is indirectly allowed to relax the higher level specification. Research on the UML produced various approaches that allow for redefining multiplicities of associations. For instance, Costal et al.[3, figure 3.2] present an example where the multiplicities are redefined in subclasses of the class connected by an association. In multi-level modeling, DDM allows for redefinition or overriding of associations at lower levels by restricting the applicable classes. In this case, the potency must be lower by at least 1 respectively. Similar to the FMML^X, this is not possible with FMLM.

Dependencies between associations, not between association types, are sometimes addressed by specialization relationships, presumably inspired by the respective concept offered by the UML. In the UML a specialization of an association a' from an association a implies that the set of linked objects "instantiated" from a' is a subset of the set of linked objects "instantiated" from a. For example, an association like `takesExamsAt` between the classes `Student` and `University` would, in this sense, be regarded as a specialization of the association `isEnrolledAt` between the same classes. The LML features this kind of association specialization, too (see example in [10]). It also provides a graphical notation to represent it. Different from LML, FMML^X allows dependencies to be defined between associations at different levels (see example in Fig. 3). Also, we intentionally decided against using the term specialization, because it does not seem appropriate to regard, e.g., taking an exam as a specialization of enrollment – nor as a generalization the other way around.

6 CONCLUSIONS AND FUTURE WORK

The fact that association types have been neglected for long may indicate that there is no urgent need for this kind of additional abstraction. This is partly in line with our experience. For long, associations have not been a top priority of our work. However, during the last years we experienced various scenarios, especially regarding the development of DSMLs, where association types proved to be very beneficial. Also, the introduction of association types ultimately reflects the core idea of multi-level modeling, which is to promote reuse, integrity, and flexibility through additional abstraction. The current implementation provides a testbed for further investigating the utility of association types and for refining respective requirements. A further aspect of our future work concerns the graphical notation of association types. It is currently restricted to the specification of a few decoration styles only and cannot vary with the use context. We plan to extend the Concrete Syntax Wizzard to provide for a more versatile specification. Also, the current implementation lacks a presentation of association types within the diagram editor. This is a preliminary solution only. We will provide for adding new association types to the palette upon their creation.

REFERENCES

[1] Colin Atkinson, Ralph Gerbig, and Thomas Kühne. 2015. A Unifying Approach to Connections for Multi-Level Modeling. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 216–225. https://doi.org/10.1109/MODELS.2015.7338252

[2] Colin Atkinson, Bastian Kennel, and Björn Goß. 2011. The Level-Agnostic Modeling Language. In *Software Language Engineering*, Brian Malloy, Steffen Staab, and Mark van den Brand (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 266–275.

[3] Dolors Costal and Cristina Gómez. 2006. On the Use of Association Redefinition in UML Class Diagrams. In *Conceptual Modeling - ER 2006*, David W. Embley, Antoni Olivé, and Sudha Ram (Eds.). Springer, Berlin, Heidelberg, 513–527. https://doi.org/10.1007/11901181_38

[4] Ulrich Frank. 2014. Multilevel Modeling: Toward a New Paradigm of Conceptual Modeling and Information Systems Design. *Business & Information Systems Engineering* 6, 6 (2014), 319–337. https://doi.org/10.1007/s12599-014-0350-4

[5] Ulrich Frank. 2018. *Flexible Multi-Level Modelling and Execution Language (FMML^x): Version 2.0: Analysis of Requirements and Technical Terminology.* Technical Report 66. ICB-Research Report.

[6] Ulrich Frank. 2021. Prolegomena of a Multi-Level Modeling Method Illustrated with the FMMLx. In *Proceedings of the 24th ACM/IEEE International Conference on Modell Driven Engineering Languages and Systems: Companion Proceedings*. IEEE.

[7] Ulrich Frank and Tony Clark. 2022. Multi-Level Design of Process-Oriented Enterprise Information Systems: 10:1-50 Pages / Enterprise Modelling and Information Systems Architectures (EMISAJ), Vol. 17. (2022). https://doi.org/10.18417/EMISA.17.10

[8] Ulrich Frank, Tony Clark, Jens Gulden, and Daniel Töpel. 2024. An Extended Concept of Delegation and its Implementation within a Modelling and Programming Language Architecture: Enterprise Modelling and Information Systems Architectures (EMISAJ), Vol. 19. (2024). https://doi.org/10.18417/EMISA.19.2

[9] Thomas Kühne, João Paulo A. Almeida, Colin Atkinson, Manfred A. Jeusfeld, and Gergely Mezei. 2023. Field Types for Deep Characterization in Multi-Level Modeling. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems companion*. IEEE, Piscataway, NJ, 639–648. https://doi.org/10.1109/MODELS-C59198.2023.00105

[10] Arne Lange, Ulrich Frank, C. Atkinson, and Daniel Töpel. 2023. Comparing LML and FMML^X. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, ACM/IEEE (Ed.). IEEE Conference Publishing Services, Los Alamitos, CA, Washington, Tokyo.

[11] Fernando Macías, Adrian Rutle, Volker Stolz, Roberto Rodriguez-Echeverria, and Uwe Wolter. 2018. An Approach to Flexible Multilevel Modelling. *Enterprise Modelling and Information Systems Architectures (EMISAJ)* 13 (July 2018), 10:1–35–10:1–35. https://doi.org/10.18417/emisa.13.10

[12] Bernd Neumayr and Michael Schrefl. 2009. Multi-Level Conceptual Modeling and OWL. In *Advances in Conceptual Modeling - Challenging Perspectives*, Carlos Alberto Heuser and Günther Pernul (Eds.). Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 189–199.

[13] Bernd Neumayr, Christoph G. Schuetz, Manfred A. Jeusfeld, and Michael Schrefl. 2018. Dual Deep Modeling: Multi-Level Modeling with Dual Potencies and Its Formalization in F-Logic. *Software and Systems Modeling* 17, 1 (2018), 233–268. https://doi.org/10.1007/s10270-016-0519-z

[14] Daniel Töpel. 2021. Associations in Multi-Level-Modelling: Motivation, Conceptualization, Modelling Guidelines, and Implications for Model Management. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 502–510. https://doi.org/10.1109/MODELS-C53483.2021.00079