Achieving Practical Genericity in Model Weaving through Extensibility*

Author version of the accepted manuscript. The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-642-38883-5_12.

Max E. Kramer[1], Jacques Klein[2], Jim R. H. Steel[3], Brice Morin[4], Jörg Kienzle[5], Olivier Barais[6], and Jean-Marc Jézéquel[6]

[1] Karlsruhe Institute of Technology, Karlsruhe
[2] University of Luxembourg, Luxembourg
[3] The University of Queensland, Brisbane
[4] SINTEF ICT, Oslo
[5] McGill University, Montréal
[6] IRISA-INRIA, Triskell, Rennes

Abstract. Many tasks in Model-Driven Engineering (MDE) involve cross-cutting model modifications that are bound to certain conditions. These transformation tasks may affect numerous model elements and appear in different forms, such as refactoring, model completions or aspect-oriented model weaving. Although the operations at the heart of these tasks are domain-independent, generic solutions that can easily be used and customized are rare. General-purpose model transformation languages as well as existing model weavers exhibit metamodel-specific restrictions and introduce accidental complexity. In this paper, we present a model weaver that addresses these problems using an extensible approach that is defined for metamodelling languages and therefore generic. Through examples of different formalisms we illustrate how our weaver manages homogeneous in-place model transformations that may involve the duplication, merge, and removal of model elements in a generic way. Possibilities to extend and customize our weaver are exemplified for the non-software domain of Building Information Modelling (BIM).

1 Introduction & Motivation

In Model-Driven Engineering (MDE), various activities require the modification of several areas of a model that satisfy specific properties. Such activities may take the shape of refactoring tasks or search-and-replace tasks similar to those supported in textual editors of Integrated Development Environments (IDEs). Others appear as model-completion transformations or aspect-oriented model weaving. These activities are composed of atomic add, change, and remove operations similar to Create, Read, Update, Delete (CRUD) operations of databases. Although these operations are problem-independent, generic solutions that can be *easily* reused and customized for arbitrary domains are rare. Existing solutions are restricted to certain types of models, do not support conditional application of changes, ignore domain-specific properties, or introduce accidental complexity.

General-purpose model-to-model transformation languages, for example, have not been designed specifically for homogeneous in-place refinement transformations, but support a multitude of scenarios. As a result, domain experts wanting

* This work is supported by the Fonds National de la Recherche (FNR), Luxembourg, under the MITER project C10/IS/783852.

to add or change model details have to make efforts to master these powerful, yet general-purpose, transformation languages. They have to reason about languge technicalities that are not central to their task, such as copying elements.

Model weaving approaches provide specific constructs for model changes that cross-cut the system's main decomposition. Currently available model weavers, however, tend to complicate these simple tasks just as general-purpose transformation languages do. The complexity results from the need for detailed weaving instructions, preparatory transformations of input models to weaving-supporting formalisms, or incomplete automation. Nevertheless, industrial domain-specific applications of model weaving, e.g. for communication infrastructure [4] or robustness modelling [1], suggest that these shortcomings can be overcome.

This paper presents a generic, extensible, and practical model weaver, called GeKo [5], together with a demonstration of its use in different domains. Our approach is generic because it is defined on top of a metamodelling language. It can be applied to all instances of arbitrary metamodels that were defined using this metamodelling language. Our approach is extensible because domain-specific solutions can be used without modifications of the generic core weaving logic. Finally, it is practical because it can be used together with existing MDE tools. It is not necessary to learn new notations or to understand new frameworks in order to apply the weaver. The presented approach evolved from earlier work on generic model weaving [22]. We added extension support, automated customization steps and improved the join point detection mechanism, the weaving implementation and the formalization. Our weaver was used to integrate building specification information into models of buildings. It is currently being integrated into the Palladio [3] IDE for model-driven development of component-based systems.

The contributions of this paper are:

- The presentation of a generic model weaver proving that practical generic model weaving can be defined on the level of metamodelling languages.
- The illustration of an extension mechanism for this weaver, showing that little work is needed to customize the generic approach to specific domains.
- The detailed description of challenging weaving scenarios for examples of two formalisms that illustrate the atomic metamodel-independent operations.

The remainder of this paper is structured as follows. Section 2 provides the background for our work. In Section 3, we present the key characteristics and the individual weaving phases. Section 4 explains how we ensure that our concepts and implementations are generic and extensible. The customization capabilities are illustrated in Section 5 through an application of our approach to Building Information Modelling (BIM). Section 6 details the generic realisation of atomic duplication, merge, and removal operations during model composition. Section 7 presents related work and Section 8 draws some final conclusions.

2 Foundations

2.1 Model Weaving & Aspect-Oriented Modelling

Aspect-Oriented Modelling (AOM) provides explicit language constructs for cross-cutting concerns. Many AOM techniques use constructs similar to those of Aspect-Oriented Programming (AOP). A *pointcut* describes at which points of a model an aspect should be applied. An *advice* defines what should be done whenever a

part of a model matches the description of a pointcut. Together, pointcut and advice form an *aspect*. The points in a base model that match a pointcut are called *join points*. After identification of these points, the changes described in an advice can be executed at these points. This process of incorporating advice information into a base model is called *model weaving*.

Other approaches to model composition, e.g. [6], do not provide new constructs such as pointcuts as they merge models expressed using the same notation.

2.2 Building Information Modelling

The term Building Information Modelling (BIM) [7] refers to models of buildings that contain semantic information in addition to three-dimensional geometric information. BIM started to replace two-dimensional models in the last decade, but is still not completely widespread [10]. Most BIM design tools use proprietary formats to represent and render models. For interoperability these tools usually provide import and export functionalities for a standard format called Industry Foundation Classes (IFC) [11]. The weaver presented in this paper was used together with a framework that bridges the technological spaces of BIM and MDE [27] in order to apply MDE techniques to models of buildings. Such an application of MDE presents challenges in terms of scalability and integration as many stakeholders use partial models of significant size and complexity.

A common technique to avoid adding the same details at several places in a model of a building is to define them in a document called a *building specification*. As building specifications, like all natural-language texts, can be ambiguous and open to different interpretations, it is hard to use them in automated processes. Nevertheless, building specifications and models are used as the main inputs for analysis tasks like cost estimation. These analyses would be easier if cross-cutting specification concerns were directly woven into models of buildings [17].

3 Overview

In this section we introduce our approach to model weaving. First, we describe five key features that characterise our approach in addition to the genericity and extensibility explained in Section 4. Then, we outline the main weaving phases.

3.1 Key Characteristics

Asymmetric Weaving of Ordinary Models In our approach, aspects that are defined by a pointcut model and an advice model are woven into a base model. This kind of approach is called asymmetric as the arguments have different *roles* (base, pointcut and advice), in contrast to symmetric approaches, such as [6], which weave entities that are not distinguished using *roles* or types.

Implicit Join Points allow Direct Use Our approach uses implicit join points that are identified using a join point detection mechanism. This means that points at which a model should be changed can be defined using an ordinary model snippet, which serves as a detection pattern. No preparatory steps, such as manually annotating a model or executing transformations that mark elements to be changed, are needed as is the case for other approaches [13,26].

Aspect Definition using Familiar Syntax In our approach, pointcut and advice models are defined using relaxed versions of the original metamodel. In

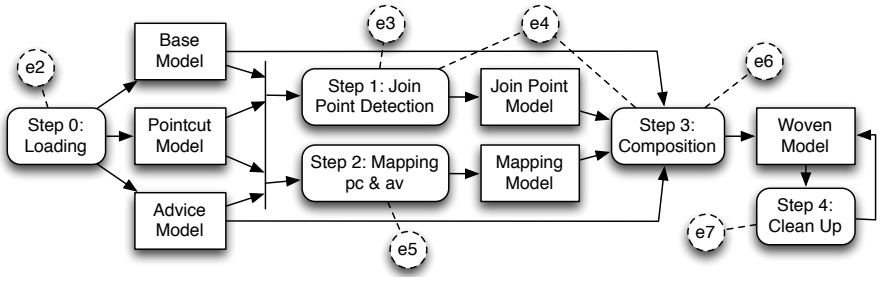


Fig. 1. The models, phases, and major extension points of the weaving process.

these metamodels, constraints, such as lower bounds and abstract metaclasses, are relaxed in order to allow the definition of incomplete model snippets. Such a relaxed metamodel is a supertype of the original metamodel as every model conforming to the original metamodel also conforms to the relaxed metamodel. Therefore, aspects can be defined with existing tools that only have to be slightly modified in order to support instantiations of abstract metaclasses and allow violations of lower bounds. Relaxed metamodels were previously presented [24], but they have not been realised in an automated, metamodel-independent way.

Declarative Mapping from Pointcut to Advice In our approach, users declaratively define which elements of the pointcut correspond to which elements of the advice. This indirect weaving specification relieves the user from the need to explicitly specify weaving steps as they are inferred from the mapping. In most cases the mapping can even be determined automatically. In contrast to declarative transformation languages like QVT-R, this mapping is metamodel-independent. The foundations of such declarative weaving instructions have been presented previously [22], and continue to be a unique feature of GeKo.

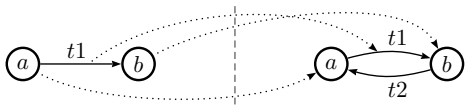
Metamodel-independent Operations Our generic model weaver is able to process instances of arbitrary metamodels. This is possible because weaving operations are based on the properties of the metamodel to which the model conforms. These metamodel properties are automatically retrieved for every metamodel and not hard-coded for a specific metamodel. They can be attributes, which store primitive types, or references to complex types. Attributes and references are part of various metamodelling languages, such as the standard EMOF 2.0 or KM3 [12]. Therefore, our approach can even be used for different metamodelling languages. Metamodel-independent operations have already been proposed [22], but have never been realised in a completely generic way. Our current implementation [5] is based on the metamodelling language *Ecore*, which is a variant of EMOF 2.0.

3.2 Weaving Phases

Our approach consists of five different phases, shown in Fig. 1. Six out of the seven extension points discussed in Section 4 are also displayed.

0) Loading Makes the relevant base, pointcut, and advice models available.

1) Join Point Detection The first phase of weaving identifies all locations of the base model that match the model snippets defined in the pointcut model. As an intermediate result, we obtain for each matched location a one-to-one mapping from pointcut elements to base elements, which we call *join points*. Depending



(a) pointcut model (b) advice model

Fig. 2. An example of a pointcut and advice model with an unambiguous mapping (dotted arrows) from pointcut to advice elements that can be automatically inferred.

on the structure and size of base and pointcut models, this preparatory step can dominate the overall time required for weaving. For this reason, we decouple it completely from the other phases of weaving. This allows for different matching algorithms as well as for domain-specific pointcut matching optimisations that are independent of the remaining weaving steps.

In our current implementation, join point detection is fully automated by generating rules targeting the business logic integration platform Drools, which implements the Rete algorithm [8]. This is similar to the SmartAdapters approach [23]. The main difference, however, is that we do not generate advice instantiation rules but decoupled this from the advice-independent join point detection in order to separate steps that are subject to different evolution pressure.

2) Inferring a Pointcut to Advice Mapping In order to know how elements before weaving correspond to elements after the weaving we need a mapping from pointcut to advice elements. This mapping is a model consisting of entries that list references to pointcut and advice elements. It can be defined independent of the way the pointcut and advice model itself are defined. To relieve the user from as much complexity as possible, the weaver automatically infers the mapping and skips ambiguous cases. Unambiguity is given, if every pointcut element matches at most one advice element of the same type having the same primitive attributes. Fortunately, this happens to be the case for many weaving scenarios such as the one presented in Fig. 2. The mapping inference algorithm matches pointcut elements to advice elements that exhibit all attributes of the pointcut element. Therefore, it rather produces false negatives than false positives. If an automatically inferred mapping is incomplete, only the remaining unmapped elements have to be mapped manually. As the m-to-n mapping may relate multiple pointcut elements with multiple advice elements, it may induce duplication and merge operations, which are discussed in detail in Section 6.2 and Section 6.3.

3) Model Composition The central weaving phase composes the base and advice models by merging the property values of their elements. Property values of the advice are used to replace or complete base property values, but removal operations are deferred to the last phase. At the end of the composition phase, newly introduced elements are added to containers using the involved containment references. A detailed description of the composition phase is given in Section 6.

4) Removal & Clean-up In the last phase of weaving, base elements that correspond to pointcut elements, but that do not correspond to any advice elements, are removed. In order to keep the model consistent, references to these elements need to be removed as well. If model elements violate the lower bounds of reference properties as a result of these removals, then they are removed as well. This is necessary to guarantee that woven models still conform to their metamodel. An example for this removal of inconsistencies is presented in Section 6.4.

4 Genericity & Extensibility

In this section we explain the techniques used in order to provide a generic and extensible approach, which can be customized for arbitrary metamodels.

4.1 Genericity

The key design decision that makes our approach generic is to transform models solely by operations formulated on the meta-metamodel level. These operations allow us to add, change, and remove elements of a metamodel instance using the properties of the metamodel that in turn conforms to a meta-metamodel. Let us illustrate this using a small example. Suppose a single join point element j in a base model matches a pointcut element p that corresponds to an advice element v. Such a match leads to a woven model in which j exhibits the properties of v. In order to perform this weaving it is irrelevant whether the model elements j and v are entities of a UML diagram or elements of a construction plan of a building. It is sufficient to inspect and update the values of the properties that are defined in the metamodel for the metaclasses of j and v.

To make this metamodel-independent approach work, we give users the ability to formulate pointcut and advice model snippets as instances of automatically derived metamodel variants with relaxed constraints. We already described the derivation and use of these relaxed metamodels in Section 3.1. A convenient consequence is that users can express weaving instructions using the familiar syntax for ordinary models. No domain-specific aspect languages are needed.

4.2 Extensibility

Our generic approach may not handle all weaving circumstances for all metamodels in the way desired by its users. Therefore, we give users the ability to reuse parts of our generic weaver and to customize them to obtain a domain-specific weaver. In this section, we briefly present the customization capabilities and in Section 5 we show an exemplary customization for the domain of BIM.

Some of the extension points that we provide can be used to change the default weaving behaviour of GeKo. Others can be used to perform additional work before or after general weaving operations. In some cases we provide two extension points for the same task in order to give users the ability to provide simple as well as more elaborate extensions. In the current implementation of our approach the customization possibilities are realised as Eclipse extension points that can be extended without directly modifying the original plug-ins.

We will now briefly describe the customization facilities in their order of use:

EP 1: During the preparatory derivation of relaxed metamodels for pointcut and advice models the default generator model can be modified. It specifies how Java classes that realise the metaclasses of the metamodel are generated.

EP 2: The process of loading and storing models before and after the actual weaving can be customized using a simple and a detailed extension point.

EP 3: Join point detection can be completely customized as its result is an ordinary one-to-one mapping from pointcut to base elements for every join point.

EP 4: It is possible to ignore specific properties of metaclasses during join point detection and model comparison using another extension point.

EP 5: For the automatic inference of a mapping from pointcut elements to advice elements the calculation of unique identifiers can be customized. These identifiers are used to match pointcut elements to advice elements.

EP 6: The introduction of new base elements corresponding to advice model elements that do not have associated pointcut elements can be customized.

EP 7: The determination of containment references can be customized for advice elements that are not unambiguously contained in another element.

All of these extension points, except EP 3 and EP 5, are used for the BIM customization of our weaver, which we present in the next section. EP 3 is required, for example, when model semantics have to be considered during join point detection. For behavioural models, such as sequence diagrams, it is possible that join points do not appear explicitly with the same syntax in the base model. In the presence of loops, for example, the first part of the join point can appear in a first iteration of the loop, whereas the second part of the join point occurs on a second iteration of the loop [14]. In such a case, the join point detection mechanism has to be extended to account for the semantics of such elements.

5 Customizing GeKo to Support BIM Weaving

To give the reader a better idea of the extension capabilities of our generic approach we present a set of weaver extensions for IFC models of buildings.

The first two extensions to our model weaver are necessary because we cannot use default XMI serialisation for IFC models. We load IFC models serialised in ASCII text files as instances of an Ecore metamodel using a technological bridge [27]. In our first extension (EP 1) we propagate the IFC-specific changes in the bridge's code generator into the code generator for the relaxed pointcut and advice metamodels. The second extension (EP 2) customizes the resource loader to retrieve content model elements from wrapping elements that model the serialisation format. Because the serialisation and the domain metamodel are defined using Ecore, we do not have to provide all loading and storing infrastructure but can reuse most of the generic facilities of GeKo and EMF.

The third extension (EP 4) ensures that the weaver ignores specific values of properties of metaclasses during join point detection and model comparison. Specifically, when creating a pointcut model and specifying that a property's value is irrelevant, it is important to avoid that the default value specified by IFC is applied. During join point detection, there are certain properties, such as globally unique identifiers, which cannot be omitted from the pointcut (for reasons specific to the IFC metamodel) but which we do not wish to detect in join points. The third extension allows us to ignore values for these properties.

The fourth extension (EP 6) ensures that model elements that are not convenient to express in the aspect are included in the woven model. For example, every IFC element is required to include an "owner history", which details the person responsible for making changes to the model. It is inconvenient to repeat this information for every pointcut and advice element, so this extension makes it possible to have this information propagated implicitly.

The last extension (EP 7) for IFC models applies at the very end of the weaving process. It ensures that all elements added to the base model during the weaving that are not yet contained in any building element are added to the

main container using the correct containment reference. This extension illustrates an advantage of our approach resulting from the decision to support pointcut and advice definition using incomplete model snippets. IFC models may exhibit deeply nested hierarchies. A window, for example, may be part of a hierarchy that starts with a storey and includes a building container, building, site container, site, project container, and project. If pointcut and advice models were complete models, the whole hierarchy beginning with the building project would have to be specified. In our approach, however, it is possible to refer to arbitrarily nested elements at the first level of pointcut and advice models. If new elements are added during the weaving, we can use information available at the join points to hook these new elements into the containment hierarchy.

Given the practical experience of providing a set of domain-specific extensions to our own generic approach, we are convinced that this strategy is generally suitable for modifying domain-specific models. The fact that less than 10% additional code (0.5 KLOC customization code, 5.1 KLOC generic code) was needed to customize the weaver for IFC models suggests that applying our generic approach requires less effort than the development of domain-specific model weavers. This, however, needs to be confirmed by future experiments that involve new extensions for other DSMLs.

6 Composition: Duplication, Merge & Removal

This section illustrates some model composition operations executed during the application of our generic weaving approach. First, we provide a short description of the formalisation upon which all composition operations are built. Second, we exemplify duplication, merge and removal operations using examples for Labelled Transition Systems (LTS) and Building Information Modelling (BIM). We chose a well-known formalism to ease the understanding and provide examples from the construction domain to illustrate the metamodel-independence of the operations.

6.1 Weaving Formalisation

We present the essential concepts of a set-theoretic formalisation of our approach. The input to our weaving algorithm is a set of base-model elements B, a set of pointcut-model elements P, and a set of advice-model elements A. From these, a join point mapping from pointcut to base elements $j : P \to B$, and a mapping from pointcut to advice elements $m : 2^P \to 2^A$ are calculated as intermediate results in steps 1) and 2) of our weaving process (see Section 3.2). Finally, the woven model is obtained using three sets and a bidirectional m-to-n mapping that we present in this section. A visualisation of the presented formalisation is displayed in Fig. 3. The interested reader is referred to an initial [22] and complete [15] description of our formalisation.

The first set contains all base-model elements that have to be removed during the weaving. These are all elements of the base model that correspond to an element of the pointcut model with no corresponding element in the advice model. More formally, given B, P, A, j and m as defined above, we define $B_{remove} := \{b \in B \mid \exists\, p \in P : j(p) = b \wedge m(\{p\}) = \emptyset\}$.

We define a second set that contains all base-model elements to be updated during the weaving. These are all base elements that correspond to at least one

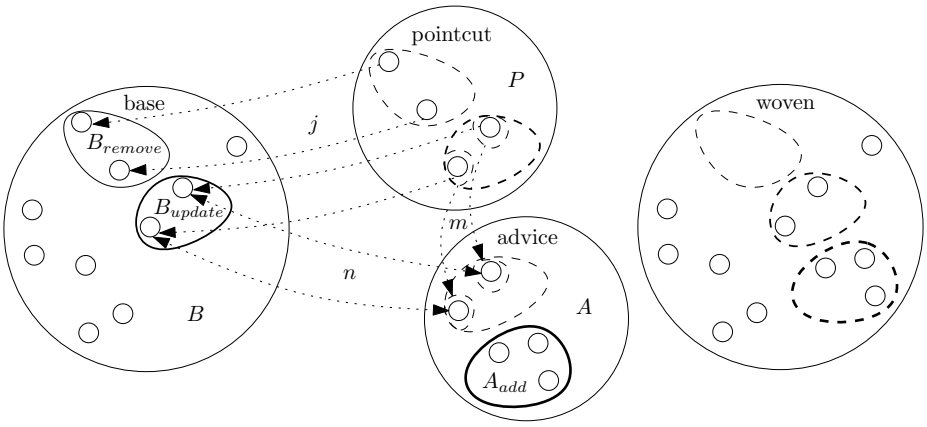


Fig. 3. Formalisation visualisation showing involved models, sets and mappings.

pointcut element with a corresponding advice element. In the same context as B_{remove} we define $B_{update} := \{b \in B \mid \exists\, p \in P : j(p) = b \wedge m(\{p\}) \neq \emptyset\}$.

The third set contains all advice-model elements that have to be added to the base model during the weaving. It is independent of a join point and contains all advice elements that correspond to no pointcut model element. More formally, given the input as above, we define $A_{add} := \{a \in A \mid \nexists\, p \in P : a \in m(\{p\})\}$.

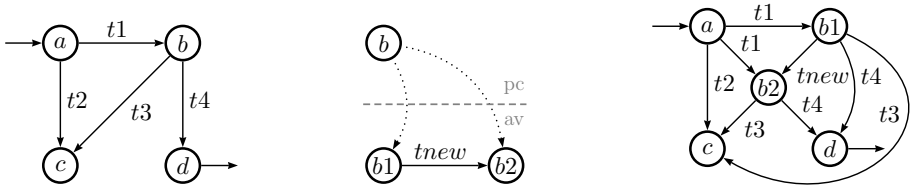
The bidirectional m-to-n mapping relates base-model elements with the corresponding advice-model elements using the detected join-point mapping from pointcut to base elements and the mapping from pointcut to advice elements. In the same context as for B_{remove} and B_{update} we define the mapping $n_{base-advice}$ as $b \mapsto \{a \in A \mid \exists\, p \in P : j(p) = b \wedge a \in m(\{p\})\}$ and the mapping $n_{advice-base}$ as $a \mapsto \{b \in B \mid \exists\, p \in P : j(p) = b \wedge a \in m(\{p\})\}$ as compositions of j and m.

Our approach is inspired by graph transformations but different: In contrast to other approaches [28,20] our formalisation and implementation [5] uses sets that directly contain model elements. No translation to nodes, edges and their types and attributes is performed. Relations between set members are only handled different than other attributes after removal operations. The mapping from pointcut to advice elements can also be non-injective and not right-unique.

6.2 Duplication

The first weaving scenario that we present involves the duplication of a base model element. Such a duplication is needed if a pointcut element corresponds to more than one advice element (m is non-injective). The consequence for each join point is as follows: All the base elements representing the advice elements that are involved in the duplication have to be updated. After the duplication, these base elements have to exhibit all properties of the base element that corresponds to the pointcut element of the duplication. This is achieved by introducing the attribute and reference values of the base element that corresponds to the pointcut element into the base elements that correspond to the advice elements.

Fig. 4 illustrates such a duplication with example models of a LTS. The pointcut element b corresponds to the two advice elements $b1, b2$ (Fig. 4(b)). The only possible join point maps this pointcut element b to the base element b. More



(a) base model (b) pointcut and advice model (c) woven model

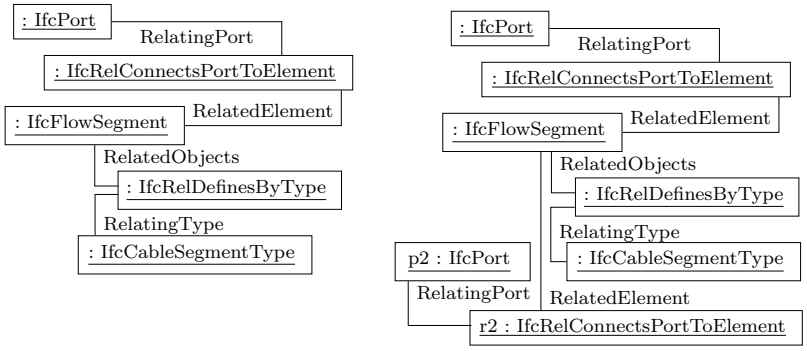
Fig. 4. Weaving an aspect into a LTS while duplicating the base element b.

formally, $n_{base-advice}(b) = \{b_1, b_2\}$. As a result, all incoming transitions $t1$ and all outgoing transitions $t3, t4$ of b are duplicated for $b1$ and $b2$ during the weaving (Fig. 4(c)). The transition $tnew$ from $b1$ to $b2$ is newly introduced independent of this duplication operation as $tnew \in A_{add}$.

Fig. 5 illustrates a duplication scenario for models of buildings. The purpose of the aspect is to duplicate cable ports. In IFC (see Section 2.2) a cable port is represented as an `IfcPort` that is related via an `IfcRelConnectsPortToElement` to an `IfcFlowSegment` that is typed using `IfcCableSegmentType` (Fig. 5.a). To achieve a duplication of such ports, the advice model (Fig. 5.b) contains the same elements as the pointcut model and an additional IfcPort together with an additional relation. The mapping from pointcut to advice elements relates the single `IfcPort` of the pointcut to both `IfcPorts` of the advice and the single `IfcRelConnectsPortToElement` to both instances of the advice. All other pointcut and advice elements have a one-to-one correspondence. We do not visualise this mapping or a woven example as this would require too much space.

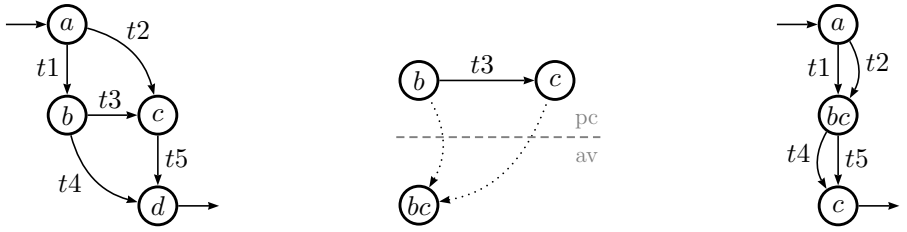
6.3 Merge

A scenario that can be seen as the dual to duplication occurs if more than one pointcut element corresponds to an advice element (m is not right-unique). The resulting merge has to ensure that the relevant advice elements exhibit all properties of all corresponding pointcut elements. This is realised by introducing all attribute and reference values of the base elements corresponding to the pointcut elements into the base element corresponding to the advice element.



(a) pointcut model (b) advice model

Fig. 5. An example aspect for IFC models which duplicates cable ports.



(a) base model (b) pointcut and advice model (c) woven model

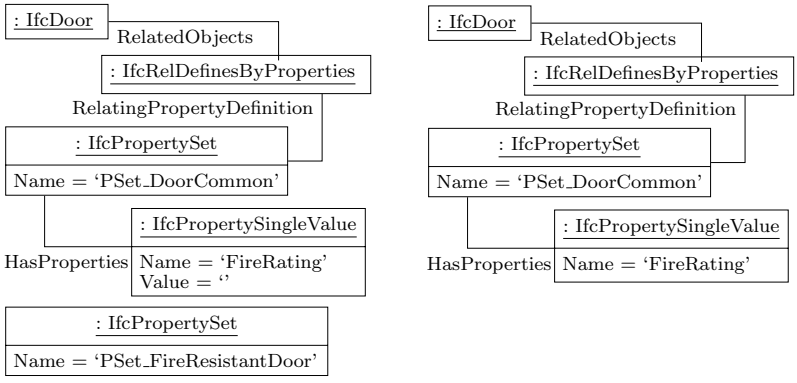
Fig. 6. Weaving an aspect into a LTS while merging the base elements b and c.

The merge weaving scenario is illustrated for LTS in Fig. 6. The two pointcut elements b and c correspond to the advice element bc (Fig. 6(b)). The only possible join point maps these pointcut elements b and c to the elements with the same names in the base model. More formally $n_{base-advice}(b) = \{bc\} = n_{base-advice}(c)$. During the weaving of this example b's incoming transition $t1$ and c's incoming transition $t2$ are merged into the resulting element of the woven model bc (Fig. 6(c)). The same applies for b's outgoing transition $t4$ and c's outgoing transition $t5$. Independent of this merge operation the transition $t3$ from b to c is removed as it is bound to the transition $t3$ of the pointcut model but has no correspondence in the advice model ($t3 \in B_{remove}$).

A similar merge scenario for IFC models is shown in Fig. 7. The aspect ensures that every door with an unspecified fire rating obtains the properties of a fire resistant door. To achieve this, the property set that contains the unspecified fire rating value (and other property values which should be preserved) is merged with a property set of fire resistance properties. Instead of listing all these properties (e.g. fire rating = "AS 1905.1", smoke stop = true), the corresponding property sets are listed in the pointcut and mapped to a single property set in the advice.

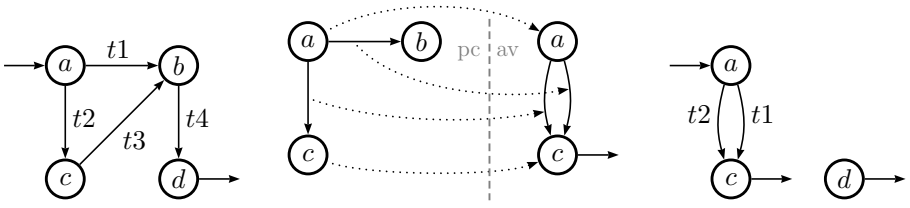
6.4 Removal

The last scenario that we discuss in detail involves the removal of base elements and illustrates the final clean-up phase. As explained in Section 6.1, a base element has to be removed during the weaving at a join point if this join point binds the base element to a pointcut element without a correspondence in the advice. After



(a) pointcut model (b) advice model

Fig. 7. An example aspect for IFC models which merges properties.



(a) base model (b) pointcut and advice model (c) woven model

Fig. 8. Weaving an aspect for a small LTS while removing the base element b.

removing these unmatched elements it may be that other base elements that referred to a removed element violate lower bound constraints of the metamodel. Therefore, we have to detect these inconsistent elements and remove them too. Because this removal of inconsistencies can produce new inconsistencies, we have to continue the clean-up until all constraints are satisfied.

We illustrate a removal scenario using LTS example models in Fig. 8. The pointcut model element b corresponds to no advice-model element (Fig. 8(b)). Thus $b \in B_{remove}$ and therefore b is removed from the woven model (Fig. 8(c)). As a result, the transition $t3$ that originally went from c to b violates the lower-bound constraint for its mandatory target attribute as it refers to no element. The same applies for the source attribute of the transition $t4$ that originally went from b to d. During the clean-up phase of the weaving both $t3$ and $t4$ are removed. Note, however, that although no element refers to it, the state d is not removed during the clean-up as it does not violate any constraint of the metamodel. Because the transition from a to b in the pointcut is mapped to the transition corresponding to $t1$, the target of $t1$ is changed. Without this mapping $t1$ would have been deleted and inserted with a possible loss of further attributes. This target change and the addition of the final attribute to c are independent of the removal and clean-up operations. We do not provide another example for IFC models as building specifications do not specify removals.

7 Related Work

In this section we discuss approaches to homogeneous in-place transformations that are generic in the sense that they can be applied to different metamodels.

SmartAdapters is a model weaving technique for which join points had to be specified manually in the initial version. First, it had to be tailored to specific metamodels, such as Java Programs [19] and Class Diagrams [18]. Then, it has been generalised to support arbitrary metamodels [21]. Later efforts focused on its use for adapting component models at runtime so that initially generic weaving functionality can no longer be separated from advanced concepts for component-based systems. Despite this specialisation, SmartAdapters shares various concepts with GeKo. A major difference, however, is the representation of weaving instructions. In addition to a declarative pointcut and advice model, SmartAdapters needs a *composition protocol* with imperative weaving instructions. It supports sophisticated weaving operations that cannot be expressed with GeKo, but it also requires explicit definitions for very basic weaving tasks.

MATA [28] is a concept for generic model weaving based on graph transformations. It converts a base model into an attributed type graph, applies graph rules obtained from composition rules, and converts the resulting graph back to the

original model type. Composition rules are defined as left-hand-side (pointcut) and right-hand side (advice), but can also be expressed in a single diagram. Although the approach is conceptually generic, we are only aware of an application in which composition rules are defined using the concrete syntax of the UML. An aspect is defined using a UML profile with stereotypes to mark elements that have to be created, matched, or deleted. In contrast to our approach, MATA does not directly operate on the input models but requires conversions and does not provide extension possibilities for domain-specific weaving.

Almazara [25] is a model weaver that generates graph-based model transformations from Join Point Designation Diagrams (JPDDs) using transformation templates. These diagrams are defined using a UML Profile and support various selection criteria, such as indirect relationships. The generated transformations collect runtime information, evaluate dynamic selection constraints and realise the weaving. This is very different from snippet-replacing approaches as it heavily integrates matching and weaving. Although JPDDs provide specialised constructs for behavioural weaving, the authors state that Almazara can be used with any modelling language. We are, however, not aware of such non-UML applications.

The Atlas Model Weaver (AMW) was developed to establish links between models that can be stored as so called *weaving models*. The links are created semi-automatically and can be used for comparing, tracing, or matching related elements. Published applications of AMW [6] use the links for heterogeneous model transformations and model comparison, but they can also be used to weave elements into a model instance. An unpublished example [2], in which attributes are woven into metaclasses, shows that join points have to be specified manually as no means are provided for pointcut definition or join point detection.

The Reuseware Composition Framework [9] provides a generic mechanism for composing modelling languages. In order to compose languages they either have to be manually extended so that they represent a component description language or a non-invasive extension has to be provided using OCL-expressions. The authors state that it is possible to reuse much composition logic once a language was made composable. Nevertheless, they do not describe an automated way to retrieve such language extensions. Furthermore, the focus of Reuseware is rather permanent language modularity than transitional composition of instances.

The Epsilon Merging Language (EML) [16] can be used to merge heterogeneous models using a syntax that is similar to declarative model transformation languages like QVT-R. These general-purpose languages support various transformation scenarios and are not specialized for in-place asymmetric homogeneous weaving according to property-based conditions. As a result, basic weaving operations, such as merging two instances of the same metaclass, have to be redefined for every application domain. This disadvantage can be mitigated using advanced transformation approaches. Higher-Order Transformations (HOTs) [13], for example, adapt transformation patterns to a domain and to individual model parts. Similarly, Generic Model Transformations [26] provide transformation templates that can be bound to specific metamodels. These approaches only support restricted pattern matching and need to be explicitly instantiated. Furthermore, users have to express transformations using constructs of the approaches and cannot describe their tasks solely with concepts of their domain.

We summarize our discussion of related work in three points. First, only our approach offers a declarative and domain-specific notation for homogeneous in-place transformations that is automatically derived in a generic way. Second, no other approach reduces the verbosity and complexity of weaving instructions and sophisticated weaving scenarios such as duplication or merge like we do it with pointcut to advice mappings. Last, related work neither separates matching from modifying logic to allow for combinations of different approaches nor provides it explicit extension points to support domain-specific customizations.

8 Conclusions & Future Work

In this paper we have presented GeKo, a generic model weaver working purely on metamodelling language constructs. We have shown that GeKo is both practical and generic because it uses declarative aspects formulated in existing notations and because it can be applied on instances of any kind of well-defined metamodel. With a selection of extension points for the refinement of weaving behaviour we have also shown that GeKo is easily extensible. This feature is crucial for a generic approach, in that it allows for customizations for domain-specific needs while reusing generic core operations. Finally, we have shown how the formalisation of GeKo allows the management of challenging weaving scenarios such as duplication, merge, and removal. With examples based on BIM and LTS we have illustrated that the operations induced by related pointcut and advice snippets can solve the problems of these scenarios in a generic way.

Further application of GeKo to weaving problems in other domains will assist in evaluating the sufficiency and usefulness of currently available extension points, and, if necessary, the identification of new ones. Also, it will be interesting to investigate alternative engines and concepts for the detection of join points, e.g. to ensure scalability in the presence of large base models and numerous join points or to allow for join point detection based on pointcut semantics.

References

1. Ali, S., Briand, L., Hemmati, H.: Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. Software and Systems Modeling pp. 1–38 (2011)
2. AMW Use Case: Aspect Oriented Modeling, eclipse.org/gmt/amw/usecases/AOM
3. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. Journal of Systems and Software 82, 3–22 (2009)
4. Cottenier, T., van den Berg, A., Elrad, T.: The motorola WEAVR: Model weaving in a large industrial context. In: Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD'06). ACM (2006)
5. Current prototype: code.google.com/a/eclipselabs.org/p/geko-model-weaver
6. Didonet Del Fabro, M., Valduriez, P.: Towards the efficient development of model transformations using model weaving and matching transformations. Software and Systems Modeling 8, 305–324 (2009)
7. Eastman, C., Teicholz, P., Sacks, R., Liston, K.: BIM Handbook. Wiley (2011)
8. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1), 17 – 37 (1982)
9. Heidenreich, F., Henriksson, J., Johannes, J., Zschaler, S.: On language-independent model modularisation. In: Transactions on Aspect-Oriented Software Development VI, LNCS, vol. 5560, pp. 39–82. Springer (2009)

10. Howard, R., Björk, B.C.: Building information modelling - experts' views on standardisation and industry deployment. Advanced Engineering Informatics 22(2), 271 – 280 (2008)
11. Industry Foundation Classes: (IFC2x Platform), ISO/PAS Standard 16739:2005
12. Jouault, F., Bézivin, J.: Km3: A DSL for metamodel specification. In: Formal Methods for Open Object-Based Distributed Systems, LNCS, vol. 4037, pp. 171–185. Springer (2006)
13. Kapova, L., Reussner, R.: Application of advanced model-driven techniques in performance engineering. In: Computer Performance Engineering, LNCS, vol. 6342, pp. 17–36. Springer (2010)
14. Klein, J., Hélouet, L., Jézéquel, J.M.: Semantic-based weaving of scenarios. In: Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD'06). ACM (2006)
15. Klein, J., Kramer, M.E., Steel, J.R.H., Morin, B., Kienzle, J., Barais, O., Jézéquel, J.M.: On the formalisation of geko: a generic aspect models weaver. Technical Report (2012)
16. Kolovos, D., Paige, R., Polack, F.: Merging models with the epsilon merging language. In: MoDELS, LNCS, vol. 4199, pp. 215–229. Springer (2006)
17. Kramer, M.E., Klein, J., Steel, J.R.: Building specifications as a domain-specific aspect language. In: Proceedings of the seventh workshop on Domain-Specific Aspect Languages. pp. 29–32. DSAL '12, ACM (2012)
18. Lahire, P., Morin, B., Vanwormhoudt, G., Gaignard, A., Barais, O., Jézéquel, J.M.: Introducing variability into aspect-oriented modeling approaches. In: MoDELS, LNCS, vol. 4735, pp. 498–513. Springer (2007)
19. Lahire, P., Quintian, L.: New perspective to improve reusability in object-oriented languages. Journal of Object Technology (ETH Zurich) 5(1), 117–138 (2006)
20. Mehner, K., Monga, M., Taentzer, G.: Analysis of aspect-oriented model weaving. In: Transactions on Aspect-Oriented Software Development V, LNCS, vol. 5490, pp. 235–263. Springer (2009)
21. Morin, B., Barais, O., Jézéquel, J.M., Ramos, R.: Towards a generic aspect-oriented modeling framework. In: Models and Aspects workshop, at ECOOP 2007 (2007)
22. Morin, B., Klein, J., Barais, O., Jézéquel, J.M.: A generic weaver for supporting product lines. In: EA '08: Proc. of the 13th international workshop on Early Aspects at ICSE'08. pp. 11–18. ACM (2008)
23. Morin, B., Klein, J., Kienzle, J., Jézéquel, J.M.: Flexible model element introduction policies for aspect-oriented modeling. In: MoDELS, LNCS, vol. 6395, pp. 63–77. Springer (2010)
24. Ramos, R., Barais, O., Jézéquel, J.M.: Matching model-snippets. In: MoDELS, LNCS, vol. 4735, pp. 121–135. Springer (2007)
25. Sánchez, P., Fuentes, L., Stein, D., Hanenberg, S., Unland, R.: Aspect-oriented model weaving beyond model composition and model transformation. In: MoDELS, LNCS, vol. 5301, pp. 766–781. Springer (2008)
26. Sánchez Cuadrado, J., Guerra, E., Lara, J.: Generic model transformations: Write once, reuse everywhere. In: Theory and Practice of Model Transformations, LNCS, vol. 6707, pp. 62–77. Springer (2011)
27. Steel, J., Duddy, K., Drogemuller, R.: A transformation workbench for building information models. In: Theory and Practice of Model Transformations, LNCS, vol. 6707, pp. 93–107. Springer (2011)
28. Whittle, J., Jayaraman, P.: Mata: A tool for aspect-oriented modeling based on graph transformation. In: Models in Software Engineering, LNCS, vol. 5002, pp. 16–27. Springer (2008)