On model compatibility with referees and contexts

Thomas Kühne

Abstract A model-based engineering discipline presupposes that models are organised by creating relationships between them. While there has been considerable work on understanding what it means to instantiate one model from another, little is known about when a model should be considered to be a specialisation of another one. This article motivates and discusses ways of defining specialisation relationships between models, languages, and transformations respectively. Consideration is given to both structural and behavioural compatibility concerns. Several alternatives of defining a specialisation relationship are considered and discussed. The article furthermore discusses the notions of referee and context in order to validate and define specialisation relationships. The ideas and discussions presented in this article are meant to provide a further stepping stone towards a systematic basis for organising models.

Keywords Model inheritance · Model compatibility · Language engineering · Model evolution · Subtyping · Refinement · Referee · Context

1 Introduction

As model-based engineering techniques gain traction it becomes increasingly important that the creation and handling of models is performed with a “return on investment” perspective. A high-frequency model usage mandates that models are created and handled in a cost-effective manner. For instance, a model should not be created from scratch if another model exists that can be used to derive the new model from it. With models of substantial size, the derivation operation is typically cheaper than the creation operation and validation efforts performed on the existing model can partially be carried over to the derived model.

Independently of whether or not two models were derived from each other, it pays off to avoid treating models in isolation from each other and capitalise on any relationships between them. Organising models in a network of relationships can aid model retrieval, support questions regarding model compatibility, and help to megamodel big systems [17].

Megamodels have models as their modelling elements and arrange them in a relationship graph. The meaning of an instance-of relationship between two models is well-understood but the same cannot be said for specialisation relationships between models [20]. In the following, I will often use the term “model inheritance” instead of “model specialisation” to avoid any connotation that the term “specialisation” carries. Just as “inheritance” between classes has many drastically differing interpretations [7,21], “inheritance” between models can also refer to a number of very different relationships between a supermodel and its submodels.

While it is possible to relate token models—models that only represent their subjects but are not types for any instances—to each other, this article assumes that “model inheritance” implies that type models [20,29] are related to each other. There is no need to distinguish between ontological or linguistic type models [20], as both have instances that conform to their respective type models [6]. A linguistic type model—often referred to as a “metamodel”—can be regarded as defining the syntax and static semantics of a language [18] which is why the following discussion is
relevant for language engineering just as well as for domain modelling, model evolution, and transformation definitions.

Language engineers can use model inheritance to derive one language definition from another. Such an approach does not only boost productivity but is also highly recommended for creating domain-customised languages, i.e., domain-specific languages that aim to maintain compatibility to the general-purpose language they were derived from [4].

In domain modelling it also makes sense to identify commonalities between domain models, for instance in the form of upper ontologies [30]. An application of this type of domain model inheritance is the layered refinement of partial domain models within an ontological modelling level in the “unified modelling library” approach [3].

In the context of model evolution, models and their versions have a natural derivation relationship with each other and the main concern is to understand and optimise the degree of compatibility between model versions and their instances [13].

Finally, transformation definitions can also be regarded as models [5] and it is worthwhile attempting to define relationships between such transformation models.

One of the most valuable properties that a relationship between models can guarantee is a certain level of compatibility between the models. This is why this article first discusses model compatibility (Sect. 2) and then looks at variability between the models. This is why this article first between models can guarantee is a certain level of compatibility between such transformation models.

The discussion of related work (Sect. 6) precedes the conclusion (Sect. 7).

2 Model compatibility

A very important criterion for judging the utility of model inheritance variants is the degree to which instances of a submodel \( M' \) are compatible with the supermodel \( M \) and vice versa. The direction of the specialisation relationship in Fig. 1 only signifies that \( M' \) is a new model that is derived from the old model \( M \). No other semantic constraints should be assumed; we want to consider any kind of derivation including those where \( M' \) is a reduced, less expressive version of \( M \).

Note that the terminology chosen for Fig. 1 uses the perspective of the type models \( M \) and \( M' \). The diagonal dependencies can be read “conformsTo” upwards, while their labels indicate the respective type of compatibility regarding their downwards meaning. We would have to use the terminology in reverse, if we used the perspective of the instances.

Figure 1 makes the notions of forward- and backward-compatibility look more straightforward than they actually are. Before we can proceed to evaluate forms of model inheritance in Sect. 3, we need to gain a deeper understanding of these notions which involves to precisely define what a model is in this context.

2.1 Formal foundation

I assume a model to be an instance of (i.e., a sentence in) some modelling language. A model is therefore just data. To turn that data into information, i.e., in order to be able to discuss properties of a model, we have to look at the meaning of a model. Formally, I regard the meaning of a model \( \mu(M) \) to be a projection \( \pi \) of the subject \( S \) the model represents, i.e., \( \mu(M) = \pi(S) \). In other words, I regard the meaning of a descriptive model to be an abstraction of the subject it describes. It is of course possible to assign multiple additional semantics to such a model and it makes sense to regard the latter as multiple, coexisting meanings.

The important part for the following discussion, however, is that we can associate both an intension and an extension [8] to the meaning of a type model. The intension \( \iota(\mu(M)) \) of a model can be thought of to be a predicate that determines whether another model is to be considered an instance of \( M \) or not. If \( M \) represents a language definition then in practical terms a metamodel and its associated constraints fulfil the role of the intension. The intension \( \iota(\mu(M)) \) thus is a characteristic predicate defining the extension \( \varepsilon(\mu(M)) \) of the type model, i.e., the set of instances which conform to the type model: \( \varepsilon(\mu(M)) = \{ x \mid P(x) \} \), where \( P = \iota(\mu(M)) \).

A supermodel \( M \) is forward-compatible with respect to a submodel \( M' \), if \( M' \)-instances conform to \( M \). Likewise, a submodel \( M' \) is backward-compatible with respect to a supermodel \( M \), if \( M \)-instances conform to \( M' \). It therefore stands to reason to establish the following formal definitions (using “\( \equiv \)” as logical implication):  

**Definition 1** Given \( M' \prec M \) (i.e., \( M' \) inherits from \( M \)),

\[
\text{forwardComp}(M, M') \equiv \iota(\mu(M)) \leftarrow \iota(\mu(M'))  \tag{1}
\]

\[
\equiv \varepsilon(\mu(M)) \supseteq \varepsilon(\mu(M'))  \tag{2}
\]