Modelica language extensions for practical non-monotonic modelling: on the need for *selective* model extension

Christoff Bürger[1]

[1] Dassault Systèmes AB, Sweden, Christoff.BUeRGER@3ds.com

Abstract

A Modelica language extension for structural non-monotonic model variation is presented. It enables *selective* model extension: the well-defined refinement of models by deselecting components and connections not of interest or inappropriate for a new design. The need for such variations is explained by the example of Modelica Synchronous, whose adaptation is suffering from crosscutting synchronous decompositions that cannot be *anticipated* when continuous models are designed; instead, contradicting model structure has to be removed when an *actual* sampling is desired. Besides synchronous, further applications for selective model extension are investigated using our prototype implementation in Dymola.

Keywords: Modelica, model variation, synchronous

1 Introduction

Of key importance for Modelica is model variation support, enabling simulation of design alternatives and their step-wise refinement from idealistic prototypes to physically-detailed solutions. To that end, Modelica provides many different abstraction and variation techniques, like model extension, replaceable components, parameters and component modifications.

Having a strong heritage from object-oriented programming however, Modelica's model variation constructs are monotonic with respect to model structure because components, connections or equations can only be added but not removed when extending models. An unfortunately overlooked consequence of flatting is however, that such a structural-monotonic type-strictness, as known from class inheritance in traditional strongly typed object-oriented programming languages like Java or C++, is not required in Modelica. In Modelica, models are flattened before simulation. Flattening essentially reduces the design space of a set of models to a fixed number of instances according to a given parameterization and replaces the resulting instances with their corresponding fixed equation system. The difference to traditional strongly typed object-oriented programming is striking: all instances are known before runtime, such that they can be statically constructed. There exists no *runtime* control-flow in Modelica that may cause different instantiations of entities; dynamic dispatch is not required, ultimately neglecting object-oriented polymorphism and the type-system restrictions that typically come with it[1]. As a consequence, Modelica's current restriction that sub-models must inherit all components and connections of their base-models when extending – that model extension must be monotonic with respect to model structure – can be dropped.

Leveraging on this observation, the paper presents a new Modelica-language extension for non-monotonic modelling: *selective model extension*. Selective model extension can be used to exclude components and connections in a *well-defined* way from inheritance when extending models. Its semantic can be fully understood in terms of model-diagram edits, such that tools can support a convenient graphical user interface for structure-wise non-preserving model variation. The main contribution of selective model extension therefore is to enable unforeseen structural variability without requiring deliberately prepared base-models.

The paper starts with an evaluation on the need for non-monotonic model variation in Modelica (Section 2). To that end, the application of Modelica Synchronous to refine continuous models for discrete use-cases is chosen which requires non-monotonic modeling to handle the crosscutting clock-partitions of different synchronous designs. Based on the non-monotonic modeling requirements elaborated throughout that discussion, an exact syntax and semantic for selective model extension is presented (Section 3). A demonstration of general practical modelling-benefits, not only for Modelica Synchronous, follows (Section 4). A prototype implementation in Dymola is used on a sophisticated example taken from the Modelica Standard Library to show how selective model extension enables model-development along the lines of real engineering processes – i.e., in terms of step-wise model variation and adaptation – avoiding model variation inconsistencies and artificial intermediate models without physical meaning.

[1]Object-oriented languages typically require monotony of inheritance to ensure the functionality of entities is well-defined for all usage-contexts, independent of control-flows determining instantiation. If sub-classes could drop base-class functionality – i.e., inheritance could be non-monotonic – runtime errors are possible whenever base-class functionality is called on sub-class objects. Static type-systems enforce monotonic inheritance to avoid such errors in the first place.