Multi-Paradigm Language Engineering and Equation-Based Object-Oriented Languages

(keynote abstract)

Hans Vangheluwe

School of Computer Science, McGill University, Montréal, Canada
Hans.Vangheluwe@mcgill.ca

Abstract

Models are invariably used in Engineering (for design) and Science (for analysis) to precisely describe structure as well as behaviour of systems. Models may have components described in different formalisms, and may span different levels of abstraction. In addition, models are frequently transformed into domains/formalisms where certain questions can be easily answered. We introduce the term "multi-paradigm modelling" to denote the interplay between multi-abstraction modelling, multi-formalism modelling and the modelling of model transformations.

The foundations of multi-paradigm modelling will be presented. It will be shown how all aspects of multi-paradigm modelling can be explicitly (meta-)modeled enabling the efficient synthesis of (possibly domain-specific) multi-paradigm (visual) modelling environments. We have implemented our ideas in the tool AToM^3 (A Tool for Multi-formalism and Meta Modelling) [3].

Over the last decade, Equation-based Object-Oriented Languages (EOOLs) have proven to bring modelling closer to the problem domain, away from the details of numerical simulation of models. Thanks to Object-Oriented structuring and encapsulation constructs, meaningful exchange and re-use of models is greatly enhanced.

Different directions of future research, combining multi-paradigm modelling concepts and techniques will be explored:

1. meta-modelling and model transformation for domain-specific modelling as a layer on top of EOOLs;
2. on the one hand, the use of Triple Graph Grammars (TGGs) to declaratively specify consistency relationships between different models (views). On the other hand, the use of EOOLs to complement Triple Graph Grammars (TGGs) in an attempt to come up with a fully "declarative" description of consistency between models to support co-evolution of models;
3. the use of graph transformation languages describing structural change to modularly "weave in" variable structure into non-dynamic-structure modelling languages.

Keywords Multi-Paradigm Modelling, Meta-Modelling, Model Transformation, Equation-Based Object-Oriented Languages, Consistency, Variable Structure

1. Multi-Paradigm Modelling

In this section, the foundations of Multi-Paradigm Modelling (MPM) are presented starting from the notion of a *modelling language*. This leads quite naturally to the concept of *meta-modelling* as well as to the explicit modelling of *model transformations*.

Models are an *abstraction* of reality. The structure and behaviour of systems we wish to analyze or design can be represented by models. These models, at various *levels of abstraction*, are always described in some *formalism* or *modelling language*. To "model" modelling languages and ultimately synthesize (visual) modelling environments for those languages, we will break down a modelling language into its basic constituents [4]. The two main aspects of a model are its syntax (how it is represented) on the one hand and its semantics (what it means) on the other hand.

The syntax of modelling languages is traditionally partitioned into *concrete syntax* and *abstract syntax*. In textual languages for example, the concrete syntax is made up of sequences of *characters* taken from an *alphabet*. These characters are typically grouped into *words* or *tokens*. Certain sequences of words or *sentences* are considered valid (i.e., belong to the language). The (possibly infinite) *set* of all valid sentences is said to make up the language.

For practical reasons, models are often stripped of irrelevant concrete syntax information during syntax checking. This results in an "abstract" representation which captures the "essence" of the model. This is called the *abstract syntax*. Obviously, a single abstract syntax may be represented using multiple concrete syntaxes. In programming language compilers, abstract syntax of models (due to the nature of programs) is typically represented in *Abstract*

2nd International Workshop on Equation-Based Object-Oriented Languages and Tools. July 8, 2008, Paphos, Cyprus.

Syntax Trees (ASTs). In the context of general modelling, where models are often graph-like, this representation can be generalized to *Abstract Syntax Graphs* (ASGs).

Once the syntactic correctness of a model has been established, its meaning must be specified. This meaning must be *unique* and *precise*. Meaning can be expressed by specifying a *semantic mapping function* which maps every model in a language onto an element in a *semantic domain*. For example, the meaning of a Causal Block Diagram (e.g., a Simulink diagram) can be specified by mapping onto an Ordinary Differential Equation. For practical reasons, semantic mapping is usually applied to the abstract rather than to the concrete syntax of a model. Note that the semantic domain is a modelling language in its own right which needs to be properly modelled (and so on, recursively). In practice, the semantic mapping function maps abstract syntax onto abstract syntax.

To continue the introduction of meta-modelling and model transformation concepts, languages will explictly be represented as (possibly infinite) sets as shown in Figure 1. In the figure, insideness denotes the sub-set relationship. The dots represent model which are elements of the encompassing set(s).

As one can always, at some level of abstraction, represent a model as a graph structure, all models are shown as elements of the set of all graphs Graph. Though this restriction is not necessary, it is commonly used as it allows for the design, implementation and bootstrapping of (meta-)modelling environments. As such, any modelling language becomes a (possibly infinite) set of graphs. In the bottom centre of Figure 1 is the abstract syntax set A. It is a set of models stripped of their concrete syntax.

1.1 Meta-models

Meta-modelling is a heavily over-used term. Here, we will use it to denote the explicit description (in the form of a finite model in an appropriate meta-modelling language) of the Abstract Syntax set. Often, meta-modelling also covers a model of the concrete syntax. Semantics is however not covered. In the figure, the Abstract Syntax set is described by means of its meta-model. On the one hand, a meta-model can be used to *check* whether a general model (a graph) *belongs to* the Abstract Syntax set. On the other hand, one could, at least in principle, use a meta-model to *generate* all elements of the language.

1.2 Concrete Syntax

A model in the Abstract Syntax set (see Figure 1) needs at least one concrete syntax. This implies that a concrete syntax mapping function κ is needed. κ maps an abstract syntax graph onto a concrete syntax model. Such a model could be textual (e.g., an element of the set of all Strings), or visual (e.g., an element of the set of all the 2D vector drawings). Note that the set of concrete models can be modelled in its own right.

1.3 Meaning

Finally, a model m in the Abstract Syntax set (see Figure 1) needs a unique and precise meaning. As previously discussed, this is achieved by providing a Semantic Domain and a semantic mapping function M. Rule-based Graph Transformation formalisms are often used to specify semantic mapping functions in particular and model transformations in general. Complex behaviour can be expressed very intuitively with a few graphical rules. Furthermore, Graph Grammar models can be analyzed and executed.

1.4 Formalism Transformation

In an attempt to mimimize accidental complexity [2], modellers often transform a model in one formalism to model in another formalism, retaining salient properties.

2. Domain-specific Modelling

Domain- and formalism-specific modelling have the potential to greatly improve productivity as they [5].

- match the user's mental model of the problem domain;
- maximally constrain the user (to the problem at hand, through the checking of domain constraints) making the language easier to learn and avoiding modelling errors "by construction";
- separate the domain-expert's work from analysis and transformation expert's work.
- are able to exploit features inherent to a specific domain or formalism. This will for example enable specific analysis techniques or the synthesis of efficient (simulation) code exploiting features of the specific domain.

The time required to construct domain/formalism-specific modelling and simulation environments can however be prohibitive. Thus, rather than using such specific environments, generic environments are typically used. Those are necessarily a compromise. The above language engineering techniques allow for rapid development of domain-specific (visual) modelling environments with little effort if mapping onto a semantic domain (such as an EOOL) is done.

3. Consistency/Co-evolution of Model Views

In the development of complex systems, multiple views on the system-to-be-built are often used. These views typically consist of models in different formalisms. Different views usually pertain to various partial aspects of the overall system. In a multi-view approach, individual views are (mostly) less complex than a single model describing all aspects of the system. As such, multi-view modelling, like modular, hierarchical modelling, simplifies model development. Most importantly, it becomes possible for individual experts on different aspects of a design to work in isolation on individual views without being encumbered with other aspects. These individual experts can work mostly *concurrently*, thereby considerably speeding up the development process. This realization was the core of Concurrent Engineering. This approach does however have a cost associated with it. As individual view models evolve, inconsistencies between different views are often introduced. Ensuring consistency between different views requires periodic con-

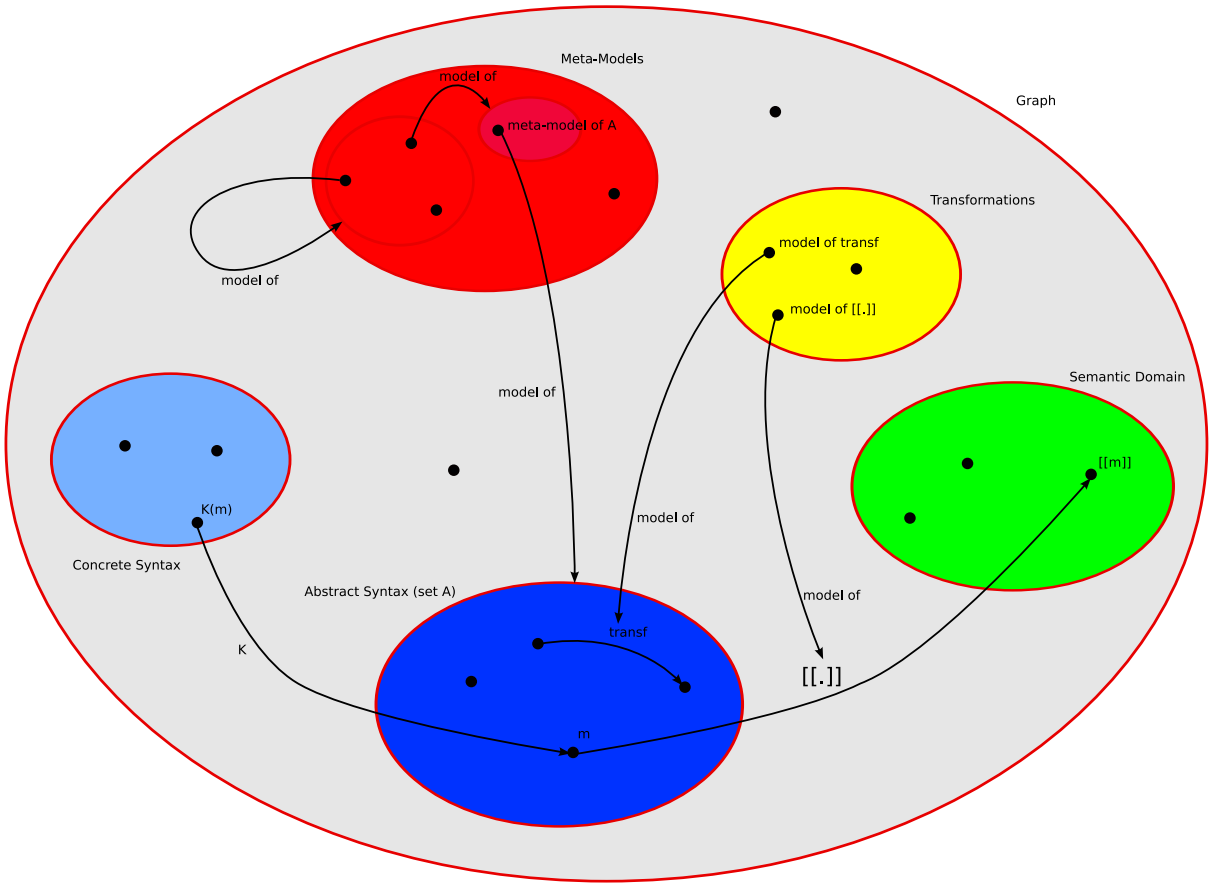

Figure 1. Modelling Languages as Sets

certed efforts from the model designers involved. In general, the detection of inconsistencies and recovering from them is a tedious, error-prone and manual process. Automated techniques can alleviate the problem. Here, we focus on a representative sub-set of the problem: consistency between geometric (Computer-Aided Design – CAD) models of a mechanical system, and the corresponding dynamics simulation models. We have selected two particular but representative modelling tools: SolidEdge for geometric modelling [8], and Modelica [1] for dynamics and control simulation.

The core geometric entities are Assemblies. SolidEdge Assemblies are composed of other Assemblies, Parts and Relationships. Relationships describe mechanical constraints between geometric features of two distinct parts, and there can be many such relationships between parts.

On the dynamics side, to represent an equivalent structure in Modelica, we have a model which can be hierarchically composed of other models, bodies, relationships and geometric features. This last type of model element is introduced to have a counterpart to represent the geometric information which is intrinsic to a SolidEdge part.

Associations (correpondences) that must exist between SolidEdge and Modelica models are shown in a meta-model triple in Figure 2. Note that this model is *declarative* as it does not specify how and what to modify to correct possible inconsistencies. Triple Graph Grammar theory [7] introduced by Schür provides a procedure for automatically deriving operational update transformations (in the form of triple graph rewrite rules) from the declarative meta-model [6]. If either the geometry or dynamics models change, the association model can be used to determine what has been added or deleted from either side.

On the other hand, the use of EOOLs to complement Triple Graph Grammars (TGGs) in an attempt to come up with a fully “declarative” description of consistency between models to support co-evolution of models;

4. Modelling of Variable Structure

Various formalisms have been devised to describe the discontinuous change of the structure of systems. The rule-based description of graph transformations is ideal to elegantly describe structural change. A rule’s left-hand-side describes the conditions under which a state-event occurs. In modelling languages for hybrid systems, crossing conditions on variable values are used to specify *when* a state-event occurs. The handling of a state-event may introduce discontinuous changes in the value of variables. The rule-based approach adds detection of particular object configurations to the low-level variable-value conditions. A rule’s right-hand-side describes the handling of the state-event. This may not only include variable value changes, but also creation/destruction of entities and their interconnections. A promising avenue for future research is the modular

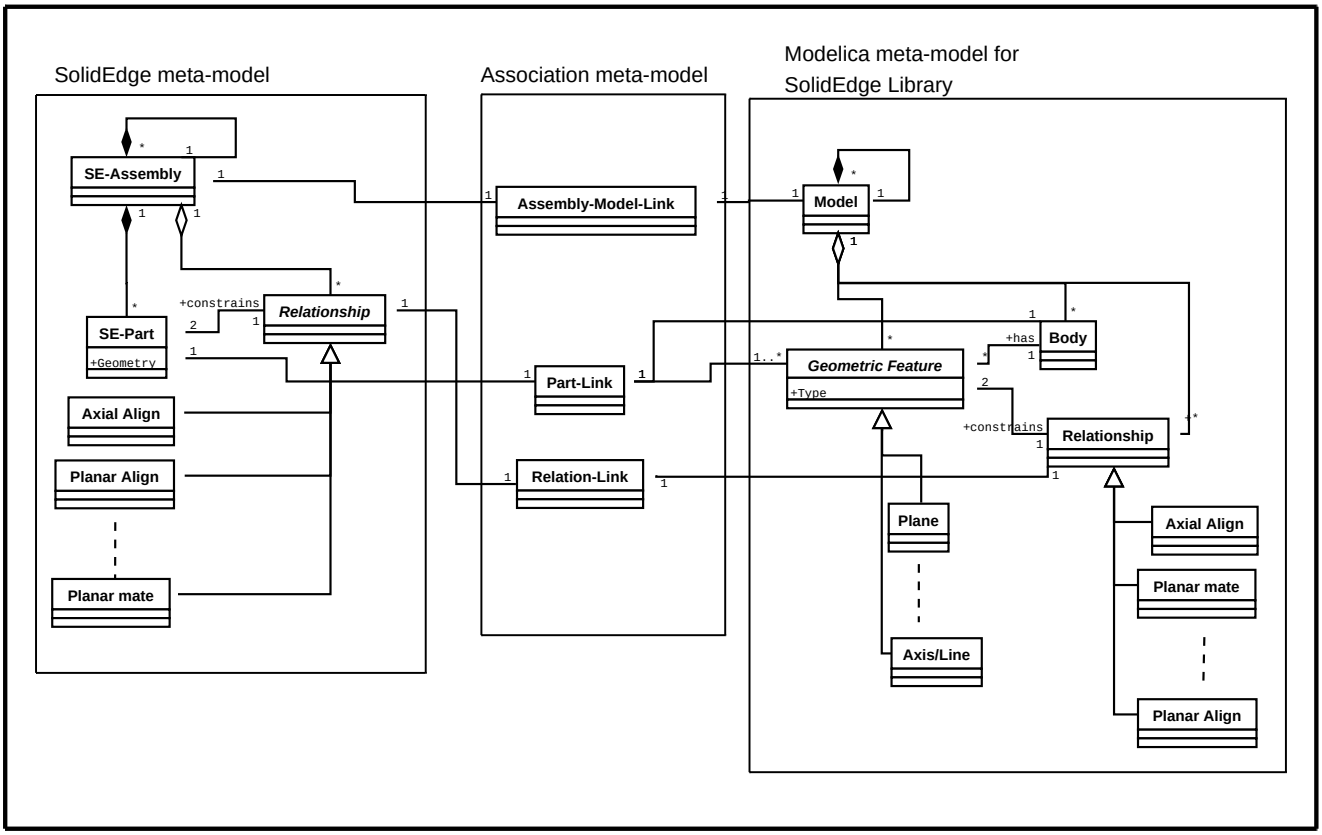

Figure 2. Relating SolidEdge and Modelica models

"weaving in" of rule-based variable structure description language constructs into non-dynamic-structure modelling languages such as EOOLs.

References

[1] Modelica™ Association. A unified object-oriented language for physical systems modeling. Modelica homepage: `www.modelica.org`, since 1997.

[2] F. P. Brooks. No silver bullet: Essence and accidents of software engineering. *Computer*, 20(4):10–19, 1987.

[3] Juan de Lara and Hans Vangheluwe. AToM^3: A tool for multi-formalism and meta-modelling. In *European Joint Conference on Theory And Practice of Software (ETAPS), Fundamental Approaches to Software Engineering (FASE)*, Lecture Notes in Computer Science 2306, pages 174 – 188. Springer, April 2002. Grenoble, France.

[4] D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff, part i: The basic stuff. Technical report, Jerusalem, Israel, 2000.

[5] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation.* Wiley, 2008.

[6] Alexander Königs. Model Transformation with Triple Graph Grammars. In *Model Transformations in Practice Satellite Workshop of MODELS 2005, Montego Bay, Jamaica*, 2005.

[7] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, editor, *WG'94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science (LNCS)*, pages 151–163, Heidelberg, 1994. Springer Verlag.

[8] SolidEdge®. `www.solidedge.com`.

Short Biography

Hans Vangheluwe is an Associate Professor in the School of Computer Science at McGill University, Montreal, Canada. He heads the Modelling, Simulation and Design (MSDL) research lab. He has been the Principal Investigator of a number of research projects focused on the development of a multi-formalism theory for Modelling and Simulation. Some of this work has led to the WEST++ tool, which was commercialised for use in the design and optimization of bioactivated sludge Waste Water Treatment Plants. He was the coordinator of the "Simulation in Europe" Basic Research Working Group. He was also one of the original members of the Modelica design team. His current interests are in domain-specific modelling and simulation and more in general, tool support for Multi-Paradigm modelling. MSDL's tool AToM^3 (A Tool for Multi-formalism and Meta-Modelling) developed in collaboration with Prof. Juan de Lara uses meta-modelling and graph grammars to specify and generate domain-specific environments. He has applied model-driven techniques in a variety of areas such as modern computer games, dependable and privacy-preserving systems (the Belgian electronic ID card), embedded systems, and to the design and synthesis of advanced user interfaces.