The Impreciseness of UML and Implications for ModelicaML

Jörn Guy Süß[1] Peter Fritzson[2] Adrian Pop[2]

[1]ITEE, The University of Queensland, Australia, jgsuess@itee.uq.edu.au
[1]Linköpings Universitet, Sweden, {petfr,adrpo}@ida.liu.se

Abstract

The Modelica community has long pursued the vision of Integrated Whole Product Modelling. This implies the ability to integrate best practice modelling languages and techniques. With ModelicaML a first step towards an open integration within the sphere of the Eclipse Modelling Framework exists. This paper argues for a development direction of ModelicaML that creates a small core with well-defined semantics, instead of the current version that is based on an extension of SysML. To this end, modelling standards and their practicabilities are discussed and exemplified through a usage scenario.

Keywords EMF, UML, ModelicaML, design

1. Introduction

Modeling in general and object-oriented modeling in particular have shown themselves to be useful tools on a software engineer's workbench. With ever more software being produced for increasingly complicated application areas, adequate semantics and languages gain in importance. The term Model is heavily overloaded, even in software engineering. In general, a model is a purpose-built abstraction of something. It exhibits properties that are essential to the abstraction and can hence be treated like the modelled object with regards to those properties.

Very often, a model is used as for simulation. Here, we take 'simulation' to mean an experiment carried out on a model, as opposed to the modelled subject itself [11]. For example, the sentence "a heart is an open book" defines a model for human emotional behaviour that may imply a certain mode of access, changeability, etc. As the heart is not a book, the model can only be used to make predictions for the intended purpose, i.e. to explain human emotional behaviour. This model will fail as a simulation of biological behaviour. The example also shows that the frame of reference or school of thought defines the allowable shapes of a model and the predictions it will make. In the example, whatever *our* mind expects of an open book, we will ascribe to the emotional behaviour of the heart. People will differ in their expectation of the structure and behaviour of books. Hence their mental simulations of the behaviour of the human heart will differ.

While the contemplation of human emotions based on such private mental models is useful and enjoyable, systems engineering is a collaborative task, involving a lot of interaction. The communication about the subject has to be adequate for this use. If we communicate about systems in order to build them effciently, safely and correctly, our school of thought or language to express our models needs to be unambiguously standardised. It also needs to practically usable, which implies terseness and focus on the task at hand.

In this spirit this paper proposes a refactoring of the ModelicaML modelling language that is used to fashion engineering models based on the Modelica language in order to simulate them. We propose that in order to expose Modelica in models, a specific object-oriented approach known as Meta-modeling should be used to create the interface. This involves both creation of visual editors, and storage facilities. We will introduce Meta-Modelling and the associated Meta-Object Facilities as we go along.

The rest of the paper is structured as follows: Section 2 provides an overview of the current version of ModelicaML. Section 3 explains MOF and metamodelling in broad terms. Section 4 describes metamodelling in more detail and argues for a specific product as the basis of ModelicaML. Section 5 gives reasons why certain parts of the current version of ModelicaML should be removed. Section 6 describes how ModelicaML can be used to model an engineering problem and predict relevant properties. Section 7 summarizes the recommendations.

2. Overview of ModelicaML

The current version of ModelicaML is a customisation or profile of SysML, which in turn is a customisation or profile of UML.

In industrial practice, the term profile is taken to mean an alteration of an existing modelling language, that is transitively related to UML through through profiling; ModelicaML is a profile of SysML, which is a profile of UML.

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

Every alteration step can affect the semantics or the visual representation of concepts. They can narrow, refine, extend and change concepts of the profiled source language. Effectively, the term profile only means that the new language is somehow related to UML, in order to increase its popularity. As a result, there are no technically standardised means to capture this wider notion of profiles. The stricter notion of profiles is discussed in Subsection 5.5.

With respect to SysML, ModelicaML reuses, extends and provides several new diagrams. The ModelicaML diagram overview is shown in Figure 1. Diagrams are grouped into four categories: Structure, Behavior, Simulation and Requirement. The ModelicaML profile is presented in [1], [29], and [30]. The most important properties of the ModelicaML profile are the following:

- The ModelicaML profile supports modeling with all Modelica constructs and properties i.e. restricted classes, equations, generics, variables, etc.
- Using ModelicaML diagrams it is possible to describe most of the aspects of a system being designed and thus support system development process phases such as requirements analysis, design, implementation, verification, validation and integration.
- The profile supports mathematical modeling with equations since equations specify behavior of a system. Algorithm sections are also supported.
- Simulation diagrams are introduced to model and document simulation parameters and simulation results in a consistent and usable way.
- The ModelicaML meta-model is consistent with SysML in order to provide SysML-to-ModelicaML conversion.

The current version of ModelicaML is based on SysML, which in turn is promoted as a variant of UML. This paper proposes that the next revision of ModelicaML should be reduced and made independent of SysML and UML. To support this argument, we give an introduction to the relevant technologies and then turn to a usage example.

3. Why expose Modelica through MOF?

This section of the paper proposes that Modelica should be represented through MOF (Meta Object Facility) for all its technical external representation, because MOF is easy to understand, and hence to integrate and is backed by solid technology.

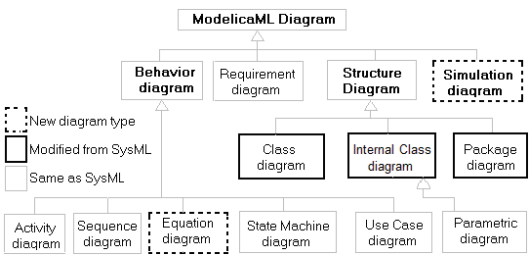

Figure 1. ModelicaML diagram overview.

The MOF is a technical framework built on the assumption that engineering languages can be described well in the ontological terms that underlie object orientation: The world consists of complex objects with primitive attributes. These belong to homogenous classes. The class may be related through an is-a relationship (sub-classing), an is-part-of relationship (aggregation), or via simple association. The latter two may be constrained by cardinalities.

With this tenant, every engineering language can be described as a class diagram. Figure 2 shows such a diagram for a database. An instance of this meta-model, would be a model for a database. Some details of the diagram read like this: ForeignKey, Column, Table and Key are all ModelElements and hence have the attributes 'name' and 'kind'. A Table is made up of ForeignKeys, Columns and Keys. A ForeignKey is associated with a Key. The API generated by a MOF service would allow to create the four mentioned elements and to set their relationships.

Essentially, MOF can be thought of as a standard for defining the cores of domain-specific software engineering tools, much like SQL is a standard for defining cores of relational databases. With MOF, schemas are compiled and the resulting machinery provides a low-volume storage facility and a rich API-based interface. We will describe MOF in more detail in the next section.

MOF combines the natural 'ontological' way of describing a technical area with a powerful generator framework which reduces the cost of building and maintaining a domain specific modeling tool. The ontological description can be represented in diagrams, which is easier to understand than the code of a hand-written modelling tool. In MOF the model-handling code of the tool is generated directly from the diagrams, which avoids errors and tedious manual translation work. For this reason we argue, that Modelica should primarily be exposed through a MOF metamodel for integration with other software engineering tools; This metamodel should be used in the future as its external representation for tool integration purposes.

4. Why should EMF be Modelica's MOF?

This section of the paper proposes that ModelicaML should be based on the MOF variant provided by the Eclipse Modelling Framework, EMF. Before we unfold the argument for this proposition, we will need to inspect MOF in greater detail, as the previous section only presented MOF in fairly abstract terms. This section will provide a bit more detail about the history and practicabilities of MOF. Like SQL, all MOF variants are basically similar, but differ in detail. Hence tools connected to and data defined based on different versions of a MOF variant, say MOF 1.2 and MOF 1.4, are incompatible if one assumes the use of all expressive features. Traditionally the dependencies of MOF are shown as a pyramid, as seen in Figure 3. Normally, this pyramid has three levels. For this paper, we have added a

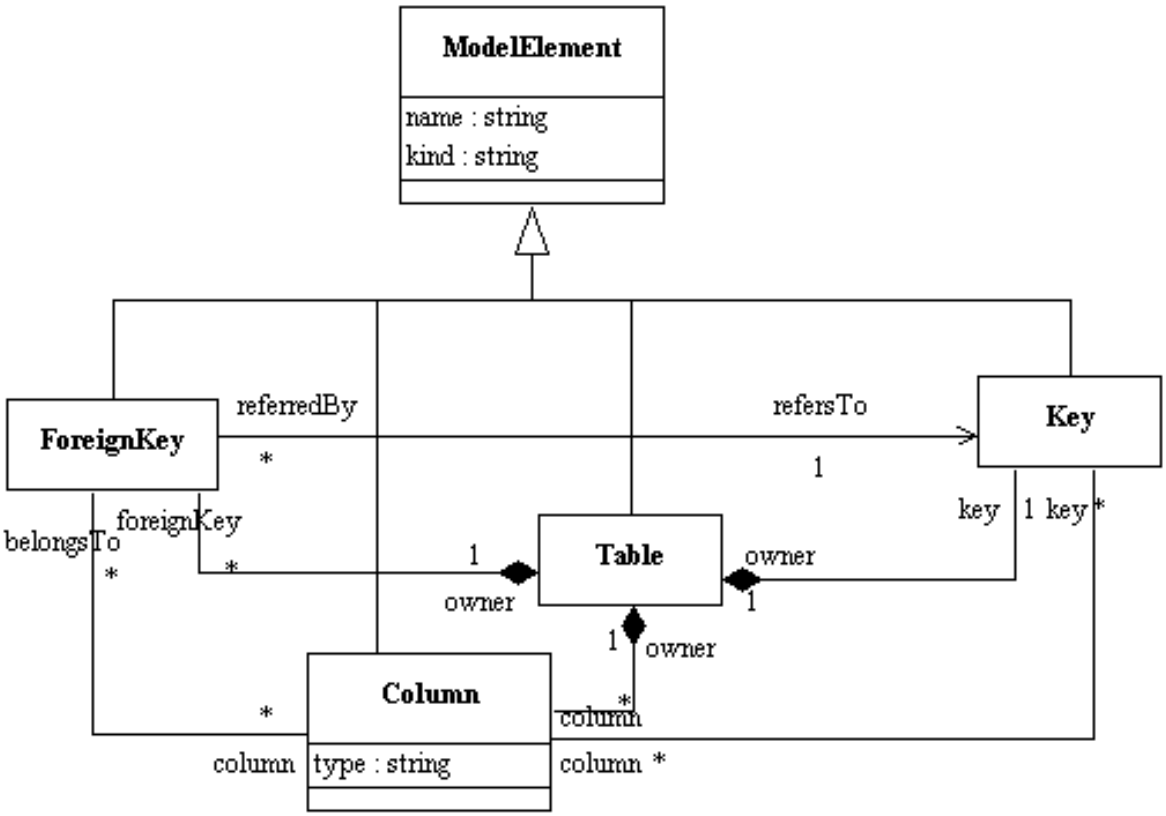

Figure 2. Example of a simple database metamodel. (INRIA)

fourth level as a means to explain the versioning problems that MOF suffers from.

4.1 Philosophy, Standards and Products

At the top of the stack at meta-level four (M4) we find the idea that was introduced in the previous section: object orientation is good for presenting domain-specific languages. This level does not have an equivalent in software. It is purely philosophical and we will call it the philosophy level.

Inspired by this philosophy, developers create implementations of MOF services. Some of the implementation guidelines they adopt are shared in the technical standards of the OMG. However, these standards have never substantially guaranteed exchangeability of services or interchange of artefacts and are thus omitted from the diagram [32, 12]. The OMG is a vendor-based organization. Modelling tools, including MOF implementations, were and are high-price margin software and hence vendors are not interested in interoperability in order to achieve customer lock-in. This is reflected in the nature of the OMG standards, which usually exclude concrete technical detail. Hence, standards in the OMG sense are not standards in a product compliance sense, rather guidelines and inspiration for implementation. In order to submit a standard, a vendor has to show an implementation, but that implementation is not required to work together with reference implementations of related standards. We will call this meta-level three the product level, rather than the standards level. The differences between the implementations usually result in incompatible storage formats, incompatible primitive data types, and incompatible event models of APIs. Thus, A MOF implementation from IBM, from SUN and from SAP would likely be incompatible in these respects.

With the help of a MOF product, software architects can express the domain-specific languages (DSLs) in metamodels. This level is known as the meta-level two, and we will call it the language level. The meta-models of the DSLs are expressed using the features and semantics of a particular MOF implementation that is in turn defined at the product level. Consequently, DSLs cannot generally be exchanged among MOF services.

Once an API has been generated from a metamodel with the help of a MOF product, it can be dressed up with a user interface and serve as a modelling tool for an engineer who is familiar with the corresponding DSL. Following the example above, we could now write the graphic interface to a database design tool. The engineer can now model an engineering problem – the design of a particular database, e.g. for a library - and store and retrieve that model from disk. The well-formedness conditions that have been defined for the engineering language in form of the constraints on the languages class schema help the engineer to design a correct model by vetoing models that cannot have an equivalent in the practice of that engineering domain. Following the example, the engineer could not store a model with a

column that was not part of a table, because the composition relationship enforces this behavior. This level is known as meta-level one and we will call it the model level.

If the engineering language actually describes physical artefacts, then a model can have a correspondence in the real world. For example, for the model of a car, one could point to an actual car in the parking lot. Or alternatively, for the modelling of a non-physical, but still existing, software library, one could identify and point to a software library installed inside an actual computer. This meta-level zero is also known as the real-world level. It is often represented in literature, but is actually an artifact of the underlying philosophy of class-based thinking, which implies instantiation.

Figure 3 aims to summarize the four levels by giving examples of popular models[1] at different levels. The original MOF was developed more then ten years ago for server-side use, but recent developments in the last five years have made its desktop use ubiquitously feasible. The Eclipse Modeling Framework (EMF) is the most recent and arguably most popular variant of MOFand is well-integrated with the Eclipse IDE.

4.2 EMF Offers Generated User Interfaces

In addition to the manipulation and storage API generated by all MOF products, EMF provides a library for interactive manipulation that plugs directly into the Eclipse IDE. This substantially reduces on the time it takes to create an interactive graphical editor for a specific DSL. For this reason the current implementation of ModelicaML already uses EMF as its basis.

In addition, EMF offers a related facilities knowns as the Graphic Modelling Framework (GMF). GMF allows the definition of diagrammatic editors in a model-based fashion. In other words, the tool-designer creates a model of how the diagram editor should look for a specific DSL, and the generator builds the diagrammatic interface. For our example, Tables could be defined as being shown as boxes, Attributes as text lines within the boxes and Keys as arrows pointing among the Attributes.

4.3 EMF has the largest base of reusable editors

As a result the number of (graphical) editors built on EMF is far greater than that of any previous MOF product. This also affects the availability of developers that can create Modelica integrations, if ModelicaML is based on the same platform.

4.4 EMF has the most Models

Because there are more EMF-based tools in actual use, the number of models transitively defined based on EMF via different meta-models also outnumbers those in any other MOF product. The atlantic EMF model zoo is a great example of this http://www.eclipse.org/gmt/am3/zoos/atlanticZoo/ We hence have a good base of engineering models that could be connected to Modelica models.

4.5 EMF can Glue Large (Meta-) Models easily

In addition to this, the architecture of EMF contains a design feature that resolves an issue that plagued previous MOF products. How do you integrate elements of two pre-existing meta-models? The OMG MOF standards do not address this issue, as it is seen as a technicality. Earlier MOF products required the meta-models to be loaded and all references resolved before models could be created. This practically limited the use of large combined models; EMF is lazy and knows the concept of a proxy, which only resolves if necessary. EMF also introduces a mapping between external resources and the model contents based on the concept of a Uniform Resource Identifier (URI). Exploiting this EMF can glue heterogenous models: A model stored in several files appears as one structure to the user of the model interface API. In this way, existing model files can be linked to new model files that reference them. This simplifies version control, as it avoids duplicate storage of data.

4.6 Other MOF Versions Do not Offer as much MDA

The critical mass of EMF use has fostered the growth of general facilities for the manipulation of models. For example, given a model of a building's floor plan and a model of a wiring plan for buildings, it would be feasible to derive a partial wiring plan from the floor plan. This model transformation could be written as a special-purpose Java program using nested iteration loops, but maintenance effort of this solution would be high. In databases, the maintenance problem has led to the development of SQL-DML, where the desired manipulation to the table entries are declaratively expressed by referencing the defined table structure. Within the MOF philosophy a generalizing approach can be adopted to create a general purpose model transformation language that allows to express the desired manipulation of instance models with reference to their asscoiated meta-models.

With such general facilities at hand the development of specialised engineering tools turns from a programming task into the art of defining a set interrelated models, their relationships and connecting model transformations and their visualizations. This idea of small interrelated models is at the heart of the Model-Driven Architecture (MDA) [15, 35]. However, general facilities are expensive tools to build, so the practice of MDA is strongly linked to EMF and its critical mass.

For the reasons presented, we see EMF as the best basis for the representation and integration of Modelica models.

[1] The diagram also clarifies why different versions of the popular modeling language UML cannot actually be exchanged, leading to a breakup in the space of artefacts. The next section will treat UML in more detail.

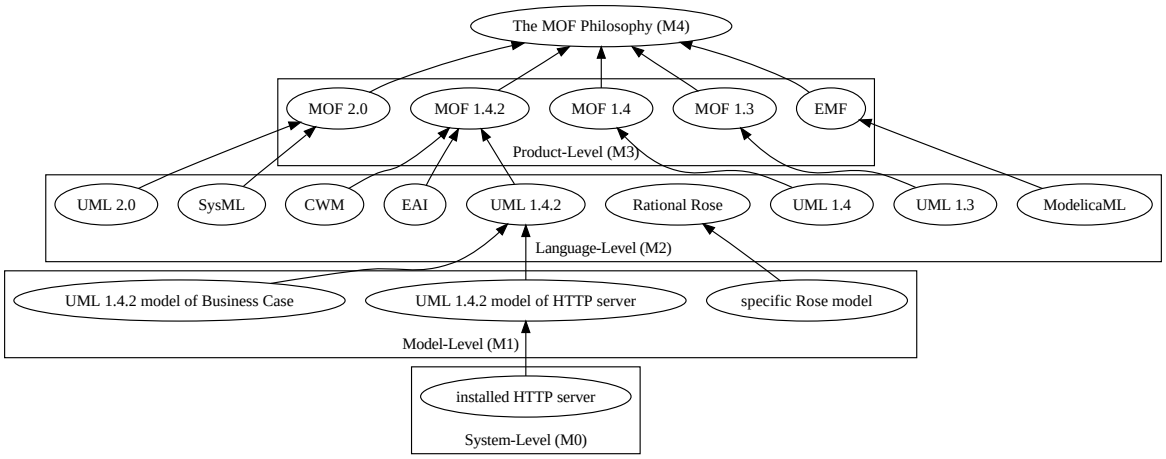

Figure 3. Four layer metamodel pyramid.

5. Why should ModelicaML not be UML?

When people talk about modeling in the context of software engineering they very quickly turn to talk about the Unified Modeling Language (UML). What they usually mean, is the language of class diagrams which only accounts for a very small part of the of the UML. UML comprises a number of modeling languages; It was conceived as a vehicle for the reuse of object-oriented models and a commercially-motivated merging approach to the three competing OO-modeling philosophies of Booch, Rumbaugh and Jacobsen. UML has been very successful in unifying the different graphical notations of these languages; Today's software engineers will always represent class is a rectangle with three partitions: name, attributes and methods.

In the following subsections we will present a number of arguments why ModelicaML should not reuse UML, but rather be a small well-defined core.

5.1 UML Standard Models are not Available, Exchangeable or Reusable

UML was conceived independently of MOF. Consequently UML tools were not built on any MOF core. They were implemented directly ad hoc. As a result, the semantics, the file formats and the APIs of the model manipulation facilities all became different. Since the tools themselves were expensive and specialized, there was little incentive for third parties to produce extensions that would allow the use of models for any purpose in the further development process. Hence the models were cut off from the rest of the development process and remained artifacts of documentation, typically created in the initial phases of projects. Consequently, errors found in later phases of projects were not fed back into the models and the quality of the models remained low. Finally, due to the high cost of their creation, models were treated as expensive intellectual property and hence not made available outside companies using UML. Although UML's inventors used class diagrams to describe UML's semantics, and this approach allowed the use of MOF products for the implementation of UML tools, most UML tools are still custom-built; Low availability, exchangeability, and reuse is hence typical for UML tools these days. The tool list available at `http://galaxy.andromda.org/docs/case-tools.html` gives a good impression of this situation. While an exchange of UML models is defined in a standard called XMI, this standard only exchanges the model data, but not the corresponding diagrams, and the document format varies by UML version and MOF product. Consequently, the exchange of data among tools is almost impossible. We want ModelicaML models to be easily exchangeable and reusable; So these properties of UML are undesirable and a reason not to base ModelicaML on UML.

5.2 UML is too Big

UML is the result of a merger of three modelling approaches, combining their diagram techniques and modelling concepts. Even at the start UML was already a sizable specification. From version 1.3, the first OMG-authorized release, to the current version 2.1 the size of the specification has expanded almost fourfold to over 2000 pages. Figure 4 shows this development. As a result of its substantial size, the UML standards are very hard to implement. Developers simply get lost in the text that mixes models, semi-formal specification in Object Constraint Language (OCL) and informal text descriptions. Very often, developers implement the bits they think they understand, and ignore the rest.

5.3 UML is not defined for EMF

None of the UML standards released through the OMG reference EMF as the underlying MOF version. For UML2.0 an implementation for MOF exists, but it is not officially endorsed as compliant with the standard. Instead, the current version of UML is defined against a sub-standard of the MOF 2.0 standard known as Complete MOF (CMOF). CMOF has a number of complex features [2]. To this date, the author knows of no working implementations of CMOF

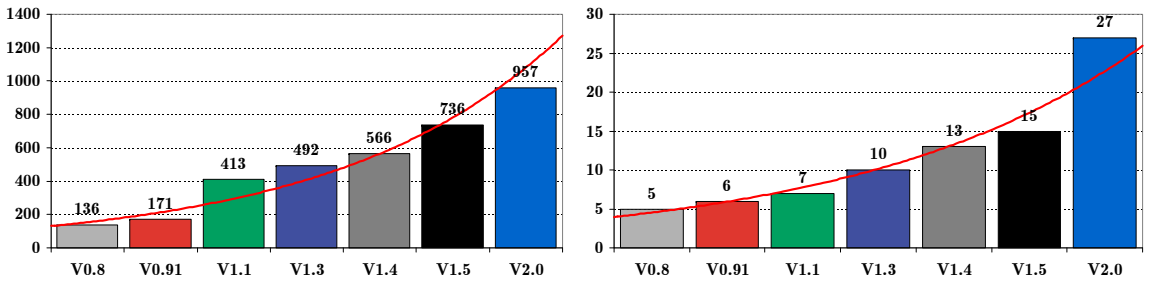

Figure 4. UML Version Development: size of the standard in pages and chapters.

or implementations of UML based on such a MOF variant. If ModelicaML is based on EMF, it cannot be based on the current specification version of UML at the same time.

5.4 UML is Semantically Unsound

Further, the UML standards shy away from defining several important technological matters, which would be necessary to make UML models reusable and exchangeable. For example, the standards do not successfully define what a valid UML model is, and at what point a UML tool would need to assert such validity. There is no test suite and no clearing house. The precise UML group, a think-tank of computer scientists, has worked for ten years on defining the precise meaning and consistency of UML and has yet to deliver a joint and clear statement. As Modelica requires precise semantics, it would suffer from problems of inconsistency and ambiguity, whenever a UML model was to be reused.

5.5 UML's Profiles are a Problem

The UML contains a mechanism known as "Profile". Profiles were originally defined as a mechanism to constrain the relatively weak semantics of UML through the application of a constraint language at the level of the UML meta-model. Unfortunately the Profile concept is often misused, in that additional semantics are added that are not necessarily compatible with UML.

According to the original definition of a profile as a constrained UML subset, the predicates of that constraint language would be evaluated against the content of a model. If a predicate was violated, the modeller would be provided with appropriate feedback. It is important to note, that this approach does *not change* the UML meta-models at all. For this reason, profiles are called light weight meta-model extensions. Hence the technical application of a profile implies that an OCL interpreter is present in every UML tool that can use a profile. The authors to this date know of no tool that satisfies this requirement. Consequently, anything called a profile is either relying on a proprietary tool extension, or it is a mere paper artifact, that does not actually support portable modeling.

The next problem with profiles is the desire of certain UML users to change, rather than to constrain, the meta-models of the UML. In other words, these users, often academics, want something that *looks* like UML, but is semantically incompatible with the UML metamodel . The profile becomes as a means of advertising one's own modelling language under the sales label of the UML. The UML standard version 2.0 vaguely describes mechanisms to alter the meta-model via a profile. There has been little interest in investigation or treatment of the resulting problems, as the mere size of UML 2.0 defeats a complete implementation in a tool anyway. During the closing panel of the last UML conference, the experts agreed that no tool was ever going to implement the UML 2.0 completely.

Today, profiles are mostly perceived as a means to describe visual alterations to UML diagram types. This use of profiles is even further removed from practical portable modeling, as there is no standardized algorithm that describes the allowable graphical renderings of a UML model in diagrams. Hence, there is also no portable amendment interface to this rendering algorithm. If profile-based alterations to rendering would be portable, they would need to be standardized parameters to the well-defined rendering algorithm. Hence even the use of amended diagram features is either proprietary or a suggestion on the manual use of drawing tools.

If ModelicaML was based on UML, it would need to implement a correct UML infrastructure including the ill-defined profile mechanism. This effort seems to be unjustified for the ModelicaML project, which aims at effective technical integration.

5.6 UML's Sublanguages are a Problem

In addition to the meta-model, the UML standard also includes two additional languages. The previously mentioned Object Constraint Language is used to define predicates against class diagrams and for pre- and post-conditions of methods. The Action Language is a slightly abstracted imperative language intended for a more precise description of behavior of systems. In order to be a sub-language of UML, ModelicaML would need to implement these other two languages as well. Both languages have little to do with Modelica's philosophy of equation-oriented modelling.

5.7 UML's children are not UML

Because standard-based UML is practically unmanageable in a tool for the reasons outlined above, vendors have begun to offer tools that offer reduced domain-specific languages

with well-defined syntaxes that are shown in diagram types resembling those of the UML. For the embedded systems domain, these are primarily class diagrams, sequence diagrams and state charts. In order to describe constraints and state transitions, the tools are often augmented with a textual constraint and action language.

Executable UML (xtUML) is such a sub-language. It revives and implements the OO-methodology of Shlaer-Mellor. This methodology was the fourth important contender at the time of UML's inception. However, it did not find its way into the commercial UML venture. xtUML is seeing increasing use in industry as a specification language. For example it is used for the specification of embedded controller software at Saab Bofors Dynamics AB (SB). Other UML sublanguages for system design are SysUML (Boeing, Saab) and RealTimeUML (RTUml) (Ericsson).

It is important to understand that none of these UML-named languages are actually UML since they are not based on CMOF or MOF. Their metamodels differ from that of the UML, and their models would not be interchangeable with standard-compliant UML tools or products, if these existed. They are proprietary engineering languages, branded "UML" for sales purposes. Apart from this misnomer, they can be used well for systems engineering and in collaboration with Modelica.

The current version of ModelicaML is based on SysML. As with any UML derivative, the metamodel of SysML is very large and contains a number of concepts that do not have a correspondence in Modelica. These elements do not form the focus of Modelica. They are superfluous and hence should not be part of ModelicaML. If they were kept, they would need to be fully expressed in diagrams, and their associated well-formedness rules from the SysML standard would need to be enforced. Further, the semantics of Modelica and SysML differ in some core areas. There, the SysML metamodel was altered to support Modelica semantics. As a result, standard SysML cannot be imported or exported from ModelicaML. Also, names of meta classes are all taken from SysML, even though the concepts may carry different names in Modelica. Consequently, there is a semantic mismatch and a long-time Modelica user will not easily find familiar concepts in the ModelicaML API, because they carry SysML names.

It seems advisable that the next version of ModelicaML should be defined via a meta-model that is as small as possible and independent of that of SysML or UML. Such consolidation will also help to improve quality, as the same amount of maintenance time will be applied to fewer artefacts and less code.

6. Usage Scenario and Proposal

The previous sections have outlined the next generation of ModelicaML as the integration interface of Modelica regarding EMF-based tools and IDEs. The revised ModelicaML will be based on EMF, but it will be smaller, and independent of SysML, UML and its children. What can we do with such an interface? How can it help Modelica to collaborate with other engineering languages?

The following sections examine a hypothetical scenario of collaboration between modelling tools using xtUML and Modelica. xtUML and Modelica have different strengths , and we will highlight these differences first. The rest of the section sketches a scenario around a real application of xtUML present at Saab Bofors.

6.1 xtUML and Modelica

xtUML describes state transitions of a model in the way most software engineers find natural: The state of the system is changed by explicitly defined actions and kept consistent by declarative constraints that should never be violated during its existence. Modelica on the other hand, due to its origin as a simulation framework, describes the behavior of a system's parts through equations. Apart from this, Modelica and xtUML know the same concepts of local attributes, generalization and aggregation, as they exist in all other object-oriented languages.

Summarily, xtUML is a language used to *explicitly* describe and drive the behavior of a system, Modelica is a language to *implicitly* describe and observe the behavior of a system. xtUML's strength is construction, Modelica's strength is analysis. Of course, both languages can be used in the respective other domain, but they will be less natural. For example, Modelica can also be used for expressing algorithmic or block-based controller code.

6.2 Missile Control at Saab-Bofors

Concretely, Saab-Bofors uses xtUML to describe the programs that drive the control of anti-aircraft missiles. The process is a typical application of MDA. It begins with a set of models related through model transformations and kept sound by validation procedures. Saab-Bofors uses an xtUML tool called Bridgepoint to create its models of the anti-aircraft missile software, validate it and to generate ADA program code that can be compiled into object-code that can be linked into an executable. The approach is special because software is flexibly apportioned to programmable hardware or controller software. The artifact flow is shown in an informal diagram in Figure 5.

6.3 Testing using Modelica

As stated above, Modelica is very useful for simulation. Testing of engineerng systems regularly involves building simulations of complex reactive environments of systems to explore system behaviour in different scenarios [34]. Among other things Modelica has been used in the past to simulate aerodynamic behavior of military aircraft. Modelica is special in comparison with other simulation systems, because Modelica can blend physical, electrical and electronic characteristics of a system seamlessly. So, Modelica lends itself well to designing and simulating the functions

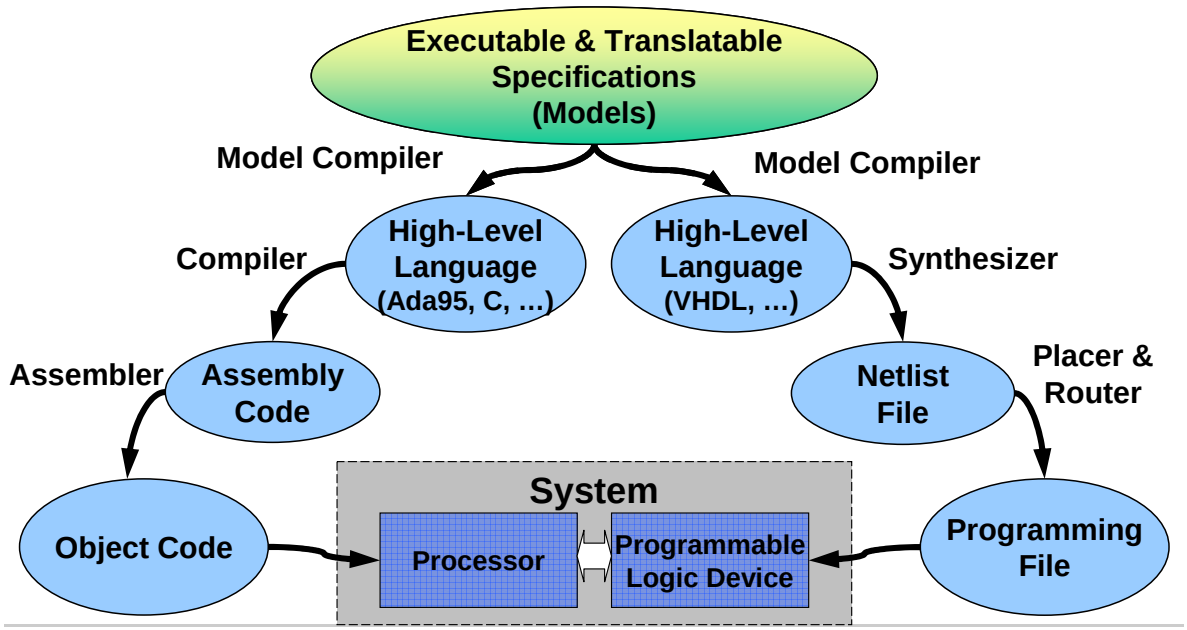

Figure 5. Model-Driven Process at Saab-Bofors. [36]

of a robotic arm, including torque, step motors, sensors, and control.

The following subsection describe three strategies for using Modelica with an engineering language like xtUML: Simulation of the environment, co-simulation of the subject-under-test, generation of simulation parameters, and implementation.

6.3.1 Linking Modelica to xtUML as an Environment

To integrate Modelica with a different modelling technique, the integration has to be modelled and rendered to executable code. Figure 6 shows one example of how xtUML and Modelica would collaborate: Bridgepoint is an Eclipse-based product, which is also internally built on EMF. As a result, its metamodels can be exported and referenced as part of other metamodels, implicitly making the data in the xtUML model available by navigation from other models. On the other side, a Modelica model of the environment of the missiles processor is prepared. This model only uses standard Modelica features. Its description uses the efficient diagrammatic features that Modelica users are familiar with. As is customary with Modelica, the model is translated into C code, which can also be compiled into object code for linking. Now, the features of the two software components have to interface. This connection information is encoded in a link-model that references features in the xtUML model and relates them to the corresponding features in the Modelica model. Now, a source code for the overall simulation can be generated, using this referential information. The source code is translated and the object codes of the environment simulator and missile control logic tied in. The resulting binary can then be run and will produce a simulation with good performance characteristics, due to the compilation of the code, as opposed to model interpretation. This approach can be extended for the special case of hardware-software splitting by providing a Modelica model of the programmable logic in addition to the processor model.

6.3.2 Modelica as Co-Simulator

In the previous scenario Modelica is used exclusively to describe the environment. However, Modelica could also be used to produce a specification of the expected test response: A sort of test oracle. This is also known as a peer model. A peer model is a model, that describes the SUT by different modelling means, in order to validate the behaviour. In the case of a peer model of Modelica for an xtUML component, the Modelica component describes the *expected* behaviour of the xtUML component in mathematical terms. The peer model can be used to check margins of error on the behaviour of the component, while going through the scenarios.

6.3.3 Modelica as Scenario Generator

Scenario coverage and test driving can be another target of Modelica modelling. This involves fashioning a model to describe what scenario initialisation data should be created and in which order the simulations should be run. In this context, Modelica is used to create mathematical models of the variant parameters.

6.3.4 Modelica as Implementation

Finally, Modelica could be used to provide implementations for components of interest. Modelica's semantics provide good clarity for all mathematical interactions, and its flow concepts allow natural modelling of component connections. This makes Modelica useful for the definition of components that resemble filters, pipes and streams. The Modelica standard is open and technically well-defined. As

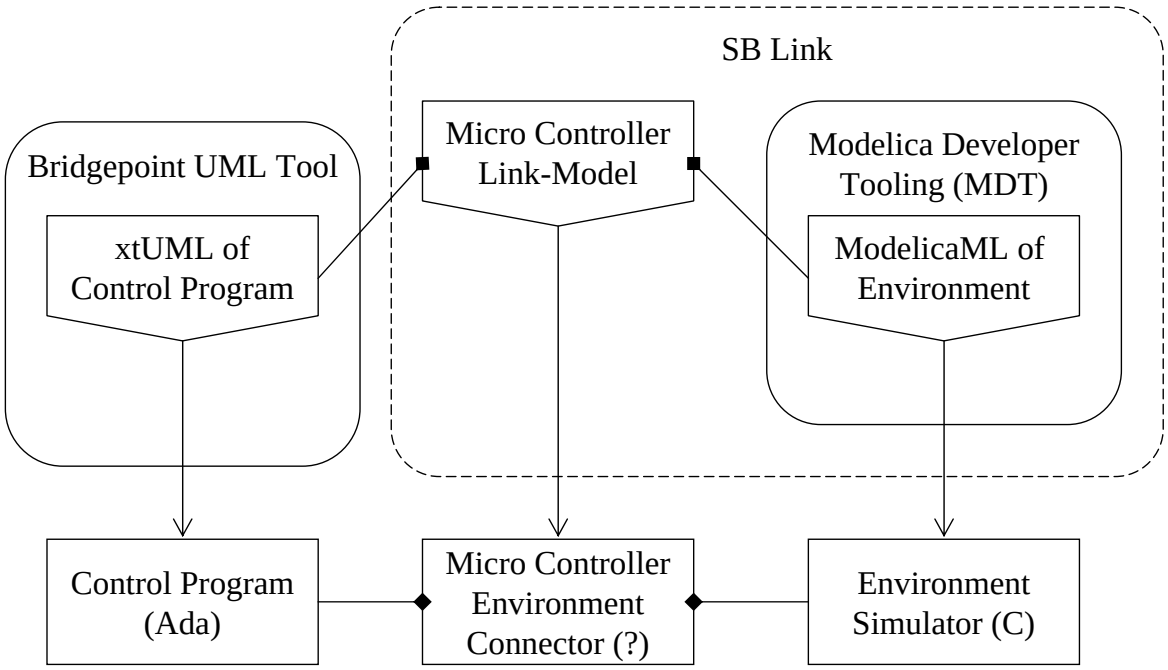

Figure 6. Link between ModelicaML and xtUML in the Saab-Bofors Scenario.

a consequence, model compilers can be written with confidence. For the Saab-Bofors scenario, part of the code for the driver application could be modelled in an imperative xtUML style, another part could be modelled in a reactive fashion in Modelica. The resulting object-codes could subsequently be linked and executed.

7. Summary and Outlook

In this paper we have reasoned about the further development strategy for ModelicaML and its core metamodel. We have argued that EMF is the the most effective choice as the implementation framework, but have discarded the use of full UML and its descendants and profiles for practical reasons. Instead, we have proposed an architecture based on a direct reflection of Modelica with a small footprint. Finally we have discussed four integration strategies in the context of the motivating example of the Saab-Bofors model-based workbench.

We will bring these considerations into the Modelica community as a basis for further discussion towards the standardization of the higher-level Modelica tooling and integration interface.

References

[1] David Akhvlediani. Design and implementation of a UML profile for Modelica/SysML. Technical Report LITH-IDA-EX–06/061-SE, Linköpings Universitet, April 2007. Final Thesis.

[2] Marcus Alanen and Ivan Porres. Difference and union of models. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML*, volume 2863 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2003.

[3] Marcus Alanen and Ivan Porres. Differences and Union of Models. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, volume 2863 of *LNCS*, pages 2–17. Springer, 2003.

[4] Alex E. Bell. Death by UML Fever. *ACM Queue*, 2(1):72–80, March 2004.

[5] Fadi Chabarek. Development of an OCL Parser for UML Extensions. Diplomarbeit, Technical University Berlin, Computation and Information Structures, TU Berlin Fak.IV Franklinstraße 28/29 · D-10587 Berlin, March 2003.

[6] Dan Chiorean and Dragos Cojocari. Implementation of OCL Support in UML CASE Tools - the ROCASE Experience. Information Systems Modelling ISM '01, May 9 - 11, 2001 Hradec nad Moravicí, Czech Republic, November 2001.

[7] Andy Evans. Making UML Precise. In Luis Andrade, Ana Moreira, Akash Deshpande, and Stuart Kent, editors, *Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.

[8] Martin Fowler. What Is the Point of the UML? In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, volume 2863 of *LNCS*, page 325. Springer, 2003.

[9] Robert B. France, Sudipto Ghosh, Trung Dinh-Trong, and Arnor Solberg. Model-Driven Development Using UML 2.0: Promises and Pitfalls. *IEEE Computer*, 39(2):59–66, 2006.

[10] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The

OpenModelica Modeling, Simulation, and Development Environment. In *Proceedings of the 46th Conference on Simulation and Modeling*, pages 83–90, 2005.

[11] Peter Fritzson and Peter Bunus. Modelica-A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation. In *Annual Simulation Symposium*, pages 365–380. IEEE Computer Society, 2002.

[12] Anna Gerber and Kerry Raymond. MOF to EMF: there and back again. In Michael G. Burke, editor, *OOPSLA Workshop on Eclipse Technology eXchange*, pages 60–64. ACM, 2003.

[13] Martin Gogolla, Jean-Marie Favre, and Fabian Büttner. On Squeezing M0, M1, M2, and M3 into a Single Object Diagram. In Thomas Baar, Dan Chiorean, Alexandre Correa, Martin Gogolla, Heinrich Hußmann, Octavian Patrascoiu, Peter H. Schmitt, and Jos Warmer, editors, *Proc. MoDELS'2005 Workshop Tool Support for OCL and Related Formalisms*. In: Satellite Events at MoDELS'2005 Conference. Jean-Michel Bruel (Ed.). Springer, LNCS 3844. Long Version: EPFL (Switzerland), Technical Report LGL-REPORT-2005-001, 2005.

[14] Martin Gogolla and Brian Henderson-Sellers. Analysis of UML Stereotypes within the UML Metamodel. *Lecture Notes in Computer Science*, 2460:84–99, 2002.

[15] Object Management Group. *OMG Unified Modeling Language 2.0*. OMG, http://www.omg.com/uml/, 2005.

[16] The Precise UML Group. The Precise UML Group Homepage.

[17] Mario Jeckle. UML Profiles und sonstige UML-bezogene Aktivitäten. http://www.jeckle.de/uml_spec.htm, 2004.

[18] Mario Jeckle. Unified Modeling Language (UML) Tools. http://www.jeckle.de/umltools.html, 2004.

[19] Mario Jeckle, Chris Rupp, Barbara Zengler, Stefan Queins, and Jürgen Hahn. UML 2.0 - Neue Möglichkeiten und alte Probleme. *Informatik Spektrum*, 27(4):323–331, 2004.

[20] Cris Kobryn. UML 2001: A Standardization Odyssey. *Communications of the ACM*, 42(10):29–37, October 1999.

[21] Cris Kobryn. Will UML 2.0 be agile or awkward? *Commun. ACM*, 45(1):107–110, 2002.

[22] Haohai Ma, Weizhong Shao, Lu Zhang, Zhiyi Ma, and Yanbing Jiang. Applying OO Metrics to Assess UML Meta-models. In Thomas Baar, Alfred Strohmeier, Ana Moreira, and Stephen J. Mellor, editors, *UML 2004 - The Unified Modeling Language. Modeling Languages and Applications. 7th International Conference, Lisbon, Portugal, October 2004, Proceedings*, volume 3271 of *LNCS*, pages 12–26. Springer, 2004.

[23] OMG. *Requirements for UML Profiles*, 1.0 edition, June 1999.

[24] OMG. *Model Driven Architecture (MDA)*, July 2001.

[25] OMG. *Meta Object Facility(MOF) Specification*, April 2002. Version 1.4.

[26] OMG. *Unified Modeling Language Specification, Version 1.3*, March 2003.

[27] OMG. *Unified Modeling Language Specification, Version 1.4*, July 2004.

[28] OMG. *SysML*, May 2006.

[29] Adrian Pop, David Akhlevidiani, and Peter Fritzson. Towards Unified System Modeling with the ModelicaML UML Profile. In *EOOLT'2007*, Berlin, July 2007.

[30] Adrian Pop, David Akhvlediani, and Peter Fritzson. Integrated UML and Modelica System Modeling with ModelicaML in Eclipse. In *The 11th IASTED Int. Conf on Software Eng. and Appl. (SEA 2007)*, Cambridge, MA, USA, Nov 19-21 2007.

[31] Arnor Solberg, Robert France, and Raghu Reddy. Navigating the MetaMuddle. In *Proceedings of the 4th Workshop in Software Model Engineering (WiSME 2005)*, Montego Bay, Jamaica, 2005.

[32] Prawee Sriplakich, Xavier Blanc, and Marie-Pierre Gervais. Collaborative software engineering on large-scale models: requirements and experience in modelbus. In Roger L. Wainwright and Hisham Haddad, editors, *SAC*, pages 674–681. ACM, 2008.

[33] Jim Steele. UML2 gripes, May 2004. Blog on MOF2 inconsistencies. Snapshot on 11/19/04.

[34] Jörn Guy Süß, Adrian Pop, Peter Fritzson, and Luke Wildman. Towards integrated model-driven testing of scada systems using the eclipse modeling framework and modelica. In *Australian Software Engineering Conference*, pages 149–159. IEEE Computer Society, 2008.

[35] Jos Warmer. MDA Explained. Addison-Wesley, to appear, 2003.

[36] Erik Wedin. Model-Based Development of Embedded Systems with MDA and xtUML. Presentation at the MOD-PROD Workshop on Model-based Product Development at the University of Linköping, Sweden, Feb 2007.