MOF Compliant Fundamentals for Multi-Domain System Modeling and Simulation

Tino Jungebloud, Sven Jäger, Ralph Maschotta and Armin Zimmermann
System and Software Engineering Group
Faculty of Computer Science and Automation
Ilmenau University of Technology
Ilmenau, Germany

***Abstract*—Modern software frameworks for Model-Based Systems Engineering (MBSE) methodologies are proprietary and commonly restricts to a poor choice of domain specific languages. Furthermore, the interoperablity and model interchange support between frameworks is a technically hard challenge. The industrial strength standard OMG MOF (Meta-Object Facility) is a solid approach for the definition of domain specific languages. Applying meta-modeling techniques to existing modeling and simulation frameworks reduces obscurity of proprietarity and implicitely language characteristics. On the other hand, standard compliance improves product openness/transparency, reliability and confidence as for primary modes of operation. This paper presents a methodology to incorporate methods of meta-modeling and standard compliant domain specific languages in an existing modeling and simulation framework. First results are emerged and exemplified for an effective modeling dialect called MML (MLDesigner Modeling Language). Future prospects reviews already accomplished goals in this project, address several alternatives and improvements and gives an overview of ongoing development.**

I. Introduction

Applied model-based engineering approaches are able to increase the chance of success in all medium and large-scale software engineering projects. State of the art software applications for model-driven software engineering uses standardized notations like OMG UML [12]. This specific language aimed at software development generally enables the description of a systems structure and behavior. In spite of the confinement to available diagram types and modeling elements, UML is well suited for model-driven software engineering. On the contrary, methods of model-based systems engineering are grown in the last decade but their application to industrial projects is still sparse. Executable models enables conceptual design, breakdown, refinement, validation and also verification of complex mixed hardware/software systems. Various specific modeling languages are used for the description of system architecture and behavior. Modern frameworks for model-based systems engineering are proprietary in the majority of cases. Each tool considered by its own supports a poor choice of domain specific languages and restricts the engineer to the offered elements, languages and methods. To overcome these drawbacks, a typical workflow induces the need for multiple tools, accepting downsides like inconsistent, redundant, inaccurate and incomplete system specification. This paper investigates possibilities of using meta-models for the description of existent domain specific languages. As a first starting point MLDesigners modeling languages are reengineered and based on MOF. The results demonstrates the first move with meta-modeling and promises to make model transformation tasks from or to MLDesigner models as easy as possible. The papers structure is organized as follows. First, related work and state of the art is discussed in detail. This comprises meta-modeling projects as well as new directions in specification. The next section outlines benefits of domain specific modeling and the advantages of MOF compliance for modeling and simulation frameworks adopted in the area of industrial projects. Thereupon the workflow for extracting a hidden meta-model from a proprietary existing modeling language is outlined. Future prospects of this lasting project summarizes the paper.

II. Related Work

Compliance to the OMG MOF standard always involves processes such as meta-modeling, grammar definition and parser/generator development. The approach of meta-modeling aims to equip language definitions with an additional but powerful part called *abstract syntax*, in contrast to its correspondent and possible diverse *concrete notations*. One direction, depicted as *forward engineering* is demonstrated by AsmM and AsmM-CS [3]. Here a human-readable textual notation was developed to ease the creation of concrete state machines which conformes to the Asm formalism given by the AsmM meta-model. Grammar definitions are manually produced for all model packages (i.e. *Structure*, *Definition*, *Terms* and *Rules*) using the EBNF (extended Backus-Naur Forms) formalism. A further method following this approach is given by the Anti-Yacc project [1]. MOF-to-Text is enhanced by grammar rules, extraction rules and lexical rules. This way, Anti-Yacc enables the production of almost any output and is therefore close to the mode of operation known from XSLT [18]. Methods for *roundtrip engineering* are given by HUTN [10] and the OMG XMI specification itself [14]. Using those MOF-to-X methods permits both direction, rendering a textual notation and regenerate a meta-model/model from its textual counterpart. One drawback of HUTN and XMI is given by general design. Models are rendered in textual representation or XML-DOM format without any reduction. The target grammar is always predefined by means of the

model and cannot conform to arbitrary XSD/DTD schemas. In contrast to the other approaches they are therefore suitable for model storage and model exchange.
A common framework for the development of domain specific languages, editors and generators is the Eclipse Modeling Project (EMP), including the Graphical Modeling Framework (GMF), Eclipse Modeling Framework (EMF) and M2T (model to text) [20]. DSL's are defined with elements of the Ecore metamodel, the Eclipse implementation of MOF based on EMOF (essential MOF). By combining a DSL's XMI serialization with a graphical notation, GMF automatically generates a model editor for this paticular DSL. Model to text transformation for generating artifacts from user defined models can be done by defining templates or grammars. The framework also supports mapping based model to model transformation.

III. Benefits of MOF Compliance for System Modeling And Simulation

Many efforts exist to reasonable introduce modeling and simulation into industrial engineering processes. [2] This comprises MOF-based General Purpose Language (GPL) such as UML [12] or SysML [15]. Modeling capabilities and diagram types of UML/SysML are not strict enough and less formal. To overcome this disqualification for system modeling and simulation, restrictions are required by the means of UML itself and/or provided by the particular framework. Profiles such as UML MARTE [16] has the capability to address these issues but they are less popular today.
The model-based system modeling and simulation methodologies uses formal and problem aligned diagram types and semantic models along with diverse models of computation to build executable system specifications. Comprehensive system modeling workflows also considers formal verification, testing and the handover of demonstrators, models or essences such as implementation artifacts. Certainly, this implies the cooperation of approved solutions because no single modeling and simulation framework is able to accomplish every task in a satisfactory quality.
With OMG MOF exists a standardized compilation of modeling concepts for the development of information models. Unfortunately, todays existing modeling and simulation frameworks are proprietary software. Applying meta-modeling techniques to those frameworks eliminates hidden language characteristics. Standard compliance also improves product openness/transparency, reliability and confidence as for primary modes of operation.
To achieve these benefits for modeling and simulation, it is essential to address the following areas. (a) Provide MOF-based domain specific languages. Using the OMG MOF meta-meta model, any kind of domain specific language can be described with basic means such as *Package*, *Class*, *Attribute*, *Assiciation*. This area also includes a deliberate language design. The MOF type system with basic (i.e. integers, strings, enumerations) and structured types [11][12] are also adequate to cover compositions frequently used in simulation activities. (b) Model Storage and Interchange. With user models conforming to MOF based meta-models the storage is technically simple. Model interchange can be done with XMI or any other concrete syntax, as long as the meta-model and the specific schema definition/grammar is undisclosed. (c) Support Model Transformation. With the interoperability in mind, language or model transformation capabilities are essential. Models conforming to the MOF standard can be transformed to each other regardless of its sense. This in turn implies a well structured language description as denoted in (a).

IV. A System Modeling And Simulation Environment

The previous section summarizes the benefits of MOF compliance in the area of system modeling and simulation which are model storage, model interchance and model transformations between certain domain specific languages. For implementation purpose of those details the modeling and simulation environment MLDesigner [5] was choosen. MLDesigner combines the description of architecture and behaviour with well-established simulation domains for system simulation in an integrated graphical environment. It provides the ability to develop executable specifications with different graphical languages such as block-diagram, finite state machine and also primitive types described by C/C++ code. The meta-model of MLDesigners modeling language MML is implicitely given by a proprietary and entirely hidden parser/generator library. Model storage is implemented by XML which contains model structure and presentation. As every simulation environment, the models of computation (or operational semantics) are realized by a model execution engine. This component interprets the model considering different simulation domains such as *Discrete Event*, *Synchronous Data Flow*, *Finite State Machine* or *Continuous Time/Discrete Event*.

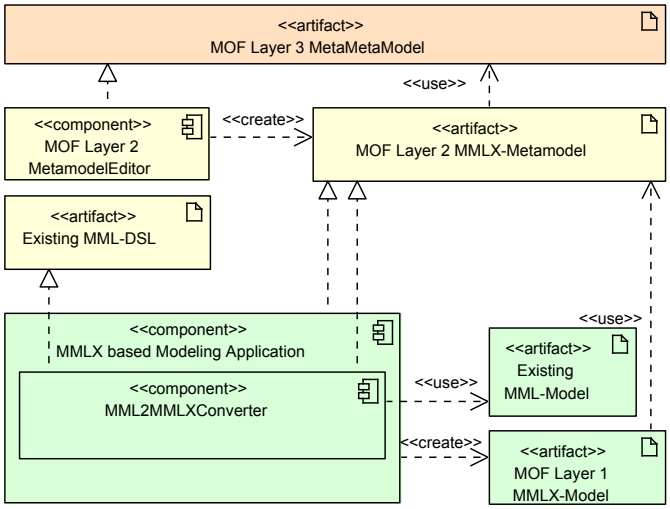

Fig. 1. Overview of the workflow of refacturing proprietary modeling languages on the example of MLDesigner MML

V. Refacturing Workflow

In this section the workflow of reengineering proprietary domain specific languages is described. The example is taken from MLDesigners modeling language MML and its hidden implicit meta-model. For traceability purpose, one language group called *Module* is used in the description. Figure 1 illustrates the overall conceptual view. Starting with existing MML model descriptions, information about the grammar is assembled. According to this, an up-to-date and possibly complete meta-model description can be *generated*. If altering is needed in this model e.g. for naming convention or not reflected characteristics, it can be easily modified with an arbitrary editor approriate for the model representation. Afterwards, the "new" meta-model maybe used in applications for modeling purpose or transformation tasks.

Listing 1. MML user model in XML syntax. Element type Module

```
<class name="Example" version="3.0.0" xmlns="http://www.
    mldesigner.com/mld">
 <property class="String" name="Logical Name" value="
     Example"/>
 <property class="String" name="Version" value="0.0.1"/>
 <port class="float" name="Input1" type="input" />
 <entity class="AddFloat.mml" classgid="xxx" name="AddFloat
     #1">
  <port class="float" label="summand1" name="Input1" type="
      input" />
 </entity>
 <relation name="Relation1" />
 <link port="Input1" relation="Relation1"/>
 <link port="AddFloat#1.Input1" relation="Relation1"/>
</class>
```

A. Breakdown Language

At the first point all elements of the existing language are collected and deconstructed. This task was challenging in this way, that no resilient grammar nor complete or up-to-date information [4] [6] of MMLs meta-model was available at some place. Related work was repeated in several student research projects investigating diverse language details [7]. The MML format is proprietary and implements a kind of lazy storage mechanism. That means, one particular model representation contains only those meta-model instance elements which are deviant from its default. By deconstructing the concrete syntax, a grammar description was built with means of XSD, the XML schema language recommendation of W3C [19]. This step was applied manually in an iterative loop, utilizing plenty of more or less complex example models and a schema validator to test grammar coverage and dertermine the state of progress. Listing 2 shows a subset of XSD rules for type *Module*.

In comparison to user-model driven language breakdown, another feasible approach to obtain the effective meta-model, is source code analysis. Here, the reverse-engineering of C/C++ source code into UML diagrams can be performed with various software products. While the effort is small in all likelihood, this method was intentionally not used because its application is generally not possible for proprietary software.

After breakdown completion was indicated, the results are utilized to deduce a meta-model description.

Listing 2. XSD fragment for language elements in type Module

```
<xs:schema version="1.0">
 <xs:element name="module" type="mmlx:Module"/>
 <xs:complexType name="Module">
  <xs:element name="property" type="mmlx:Property"
      minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="port" type="mmlx:Port" minOccurs="0"
      maxOccurs="unbounded">
  <xs:element name="entity" type="mmlx:Entity" minOccurs="0
      " maxOccurs="unbounded"/>
  <xs:group name="LinkRelationG">
   <xs:sequence>
    <xs:element name="relation" type="mmlx:Relation"
        minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="link" type="mmlx:Link" minOccurs="0"
        maxOccurs="unbounded" />
   </xs:sequence>
  </xs:group>
 </xs:complexType>
</xs:schema>
```

B. Build Language Meta-Model

The grammar created in the first step comprises all information which is needed to define the meta-model. Thus, an approach was needed to carry this information into a meta-model specification. An earlier experiment attempts to model everything using the UML but detailed diagrams became complex and thus not manually manageable. [9] We decided to use EMOF, a subset of MOF along with its reference implementation Ecore. Tool support was provided by EMF and the contained generator engines [11] [20]. This transformation applies a mapping for each XSD structural type to its Ecore meta-model counterpart where possible. Listing 3 illustrates the simplified meta-model for type *Module*.

Listing 3. ECORE XMI description for the language element Module (abstract presentation)

```
<ecore:EPackage xmi:version="2.0">
 <eClassifiers xsi:type="ecore:EClass" name="Module">
  <eStructuralFeatures xsi:type="ecore:EReference" name="
      property" eType="#//Property" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="
      port" eType="#//Port"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="
      entity" eType="#//Entity"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="
      relation" eType="#//Relation" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="
      link" eType="#//Link" />
</ecore:EPackage>
```

C. Concrete User Model

Based on this abstract syntax it was now possible to construct user-models conforming to all syntactical rules. To create such models, several ways are possible. The EMP/EMF is a very popular way to generate language bindings and program libraries to handle DSLs. In very short time the generator capabilities provides a specialized editors for the previously defined Ecore meta-model. The simple editor fuctions

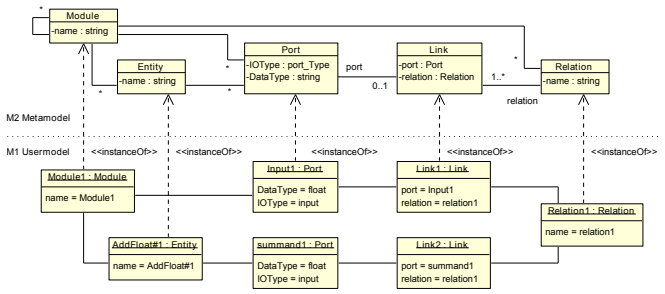

Fig. 2. Illustration of MML meta-model and exemplary user-model instance of type Module (simplified)

assists the development of models concerning the valid model structure as desired. For testing purpose, the resulting models comply with the original MML syntax and it is also possible to persist user-models using the XMI. Another way could be the use of common UML modeling tools. This is possible if the program can import foreign meta-model. Theoretically, this comprises the definition of suitable sterotypes with the means of lightweight UML extension standards (profiles). Thereafter the modeling editor facilitates the development of enhanced UML class diagrams using stereotypes and the tagged value concept. Amongst others, the implementation of means for the creation of models is subject of current work.

VI. Conclusion and Future Prospects

With this paper, we have presented the intermediate results of ongoing projects. Our approach was used to enable MOF-compliance of an existing non MOF-conforming domain specific language called MML. We showed how an existing and proprietary language can be analyzed and deconstructed. With the gathered information a grammar was produced. Then, the language meta-model was derived with EMF support resulting in a Ecore-based meta-model. A generated DSL editor was then utilized to create valid user-models which conforms to a subset of MOF called EMOF.

The general benefit of the achieved compliance is interoperability between different applications and the possiblity to define transformation at both layers, the meta-model layer and user-model layer. One use-case to apply our approach with MLDesigner is the interchange of behavior description with UML activity diagrams such as the Activity Diagram and State Machine diagram. A long-term perspective is the support for the OMG standard called *Foundational Subset For Executable UML Models* (FUML) [17]. An important field of reseach are enhancements to MLDesigner itself. Replacing the proprietary meta-model and data storage combined with undisclosed schema description can improve openness and transparency. Using OMG MOF is often correlated with several reference implementation such as EMF and GMF which are based on the Java programming language. With the improvement of MLDesigner in mind, we tend to other programming languages, so alternative implementations has to be analyzed soon. Forthermore, the knowlege of meta-modeling principles and techniques can be used to implement the definition process, semantical mapping and utilization of so called "User Languages" in MLDesigner. All these topics are subject of ongoing research and development and/or future work.

References

[1] D. Hearnden, K. Raymond, J. Steel. *Anti-Yacc: MOF-to-text*. CRC for Enterprise Distributed Systems (DSTC) University of Queensland, Australia. 2003.

[2] T. Johnson, C. Paredis, R. Burkhart *Integrating Models and Simulations of Continuous Dynamics into SysML* Systems Realization Laboratory, The G. W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, 2008

[3] A. Gargantini, E. Riccobene, P. Scandurra. *Deriving a textual notation from a metamodel: An experience on bridging Modelware and Grammarware*. University of Bergamo,Milan, Catania, Italy. 2006.

[4] Maik Hauguth. *Entwurf eines XML-basierten Dokumentenformates für Blockdiagramme*. Faculty of Computer Science and Automation, Ilmenau University of Technology, Germany, 2001.

[5] MLDesign Technologies, Incorporated. *MLDesigner Modeling and Simulation Framework*, 2013.

[6] Felix Kupsch. *Konzeption und Realisierung einer Schnittstelle zwischen XMI-basierten UML-Werkzeugen und MLDesigner*. Faculty of Computer Science and Automation, Ilmenau University of Technology, Germany, 2008.

[7] Björn Freiherr. *Analyse der Modelldatenstruktur von MLDesigner zur interaktiven Simulation via Webinterface*. Faculty of Computer Science and Automation, Ilmenau University of Technology, Germany, 2010.

[8] Christoph Preuss. *Entwicklung technischer Konzepte zur semi/vollautomatischen Transformation von UML 2 Diagrammen in MLDesigner Modelle*. Faculty of Computer Science and Automation, Ilmenau University of Technology, Germany, 2010.

[9] Qing Liu. *Untersuchungen zur Realisierung einer OMG MOF-basierten Beschreibungssprache für ausführbare Spezifikationen komplexer Systeme*. Faculty of Computer Science and Automation, Ilmenau University of Technology, Germany, 2012.

[10] Object Management Group. *Human-Usable Textual Notation, Version 1.0*. Technical report, Object Management Group, 2004.

[11] Object Management Group. *Meta Object Facility Core Specification, Version 2.4.1*. Technical report, Object Management Group, 2011.

[12] Object Management Group. *Unified Modeling Language Infrastructure, Version 2.4.1*. Technical report, Object Management Group, 2011.

[13] Object Management Group. *Unified Modeling Language Superstructure, Version 2.4.1*. Technical report, Object Management Group, 2011.

[14] MOF 2 XMI Mapping *MOF 2 XMI Mapping, Version 2.4.1* Technical report, Object Management Group, 2011.

[15] Object Management Group. *Systems Modeling Language, Version 1.3*. Technical report, Object Management Group, 2012.

[16] Object Management Group. *The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems*. http://www.omgmarte.org, 2011.

[17] Object Management Group. *Semantics Of A Foundational Subset For Executable UML Models* http://www.omg.org/spec/FUML, 2013

[18] The World Wide Web Consortium (W3C). *Extensible Stylesheet Language, XSLT, W3C Recommendation*. http://http://www.w3.org/TR/xslt20. 2007.

[19] *XML Schema Primer, Structures, Datatypes, W3C Recommendation* http://www.w3.org/TR/xmlschema-0. 2004.

[20] The Eclipse Foundation. *Eclipse Modeling Project*. http://www.eclipse.org/modeling. 2013