A M3-Neutral Infrastructure for System Engineering

Olivier le Merdy (olivier.lemerdy@free.fr)

Sodius SAS (Nantes, www.sodius.com) and Ecole des Mines de Nantes (www.emn.fr)

Abstract

In this paper we report on some of the research activities at the Sodius Company in the domain of model-based system engineering. We start from the idea that even if Systems Engineering and Software Engineering, it is possible to create bridges at the highest level of abstraction and thus create correspondence at lower levels. The main message of this paper is that it is possible to consider software engineering and system engineering as two similarly organized areas, based on different metametamodels (M3-level). Consequently building bridges between these spaces at the M3-level seems to offer some significant advantages that will be discussed in the paper. We illustrate the space of system engineering with the well established CORE set of standards.

1 Introduction

Model engineering (or MDE for Model Driven Engineering) is being considered as an important departure from traditional techniques in such areas as software engineering, system engineering and data engineering. In software engineering, the MDA™ approach proposed by OMG in November 2000 allows separation of platform dependent from platform independent aspects in software construction and maintenance. More generally MDE is proposing to use models to capture specific aspects of a system under construction or maintenance, not only the business and platform aspects.

In the system engineering domain, a similar organization has been used for the last twenty years, mainly based on the TRW standard. However the overall organization was more implicit than explicit.

This paper describes one ongoing project at the Sodius Company in Nantes. The goal is to define a generic experimental advanced model management platform for system engineering. The idea is to consider that we have similarly organized technical spaces (MDA, CORE, Step/Express, Grammarware, XML, DBMS, XML, etc.). For each of these we have an implicit or explicit so-called M3-level. The MOF notation for MDA or the EBNF notation for grammarware play this role of defining, with different precision, the representation system for the entire technical space. In addition to this general M3-level organization, each space offers, at the M2-level, a rich set of specific domain specific languages (DSLs). These DSLs may be called grammars, metamodels, ontologies, DTDs, XML schemas, etc. Since these DSLs are used to capture specific aspects of systems, their relations or combinations is presently an important research concern. Transformation of programs written in various DSLs is one current very active research activity.

In this paper we propose the idea that it should be possible to establish generic coordination between different technical spaces by making explicit the M3-level properties and providing domain-independent transformation facilities at this level. This would be more efficient than providing ad-hoc, case by case transformation between various DSLs belonging to the same or different technical spaces.

This paper is thus organized as follows. In section 2 we introduce some general considerations on the three layer conjecture. Section 3 presents the domain of system engineering and the CORE set of standard. In Section 4, we show how the idea of defining bridges between these spaces at the M3-level may bring a lot of significant economies and other advantages. Finally we conclude by summarizing the project goals and sketching possible extension paths.

2 The 3-Layer Conjecture

In this section we recall the main characteristics of the three layer conjecture and we introduce one important technical space, namely the software engineering (MDA).

2.1 The OMG MDA Space

Each technical space is organized on a metametamodel (explicit or implicit) and a collection of metamodels. For the OMG/MDA the MOF and the collection of standard metamodels and UML profiles play this role.

In November 2000 the OMG proposed a new approach to interoperability named MDA™ (Model-Driven Architecture) [8]. MDA is one example of a much broader approach known as Model Driven Engineering encompassing many popular research trends like generative programming, domain specific languages, model-integrated computing, model management and much more.

The basic assumption in MDE is the consideration of models as first class entities. A model is an artifact that conforms to a metamodel and that represents a given aspect of a system. These relations of conformance and representation are central to model engineering [1]. A model is composed of model elements and conforms to a unique metamodel. This metamodel describes the various kinds of contained model elements and the way they are arranged, related and constrained. A language intended to define metamodels and models is called a metametamodel.

The OMG/MDA proposes the MOF (Meta Object Facility) as such a language. The Eclipse metametamodel is part of EMF and is compatible with MOF 2.0. This language has the power of UML class diagrams complemented by the OCL assertion and navigation language.

2.2 Technical spaces

There are other representation systems that may also offer, outside the MDA strict boundaries, similar model engineering facilities. We call them technical spaces [7]. They are often based on a three level organization like the metametamodel, metamodel and model of the MDA. One example is grammarware [7] with EBNF, grammars and programs but we could also consider XML documents, Semantic Web, DBMS, ontology engineering, etc. A Java program may be considered as a model conforming to the Java grammar. As a consequence we may consider strict (MDA)-models, i.e. MOF-based like a UML model but also more general models like a source Java program, an XML document, a relational DBMS schema, etc.

The main role of the M3-level is to define the representation system for underlying levels. The MOF for example is based on some kind of non-directed graphs where nodes are model elements and links are associations. The notion of association end plays an important role in this representation system. Within the grammarware space we have the specific representation of abstract syntax trees while within the XML document space we have also trees, but with very different set of constraints.

Associated to the basic representation system, there is a need to offer a navigation language. For MDA the language that plays this role is OCL, based on the specific nature of MDA models and metamodels. OCL for example know how to handle association ends. For the XML document space, the corresponding notation is XPath that takes into account the specific nature of XML trees. As a matter of fact OCL is more than a navigation language and also serves as an assertion language and even as a side-effect fee programming language for making requests on models and metamodels.

At the M3-level when the representation system and corresponding navigation and assertion notations are defined, there are also several other domain-independent facilities that need to be provided. In MDA for example generic conversion bridges and protocols are defined for communication with other technical spaces:

- XMI (XML Model Interchange) for bridging with the XML space
- JMI (Java Model Interchange) for bridging with the Java space
- CMI (Corba Model Interchange) for bridging with the Corba space

Obviously these facilities may evolve and provide more capabilities to the MDA technical space. We may even see many other domain-independent possibilities being available at the M3-level like general repositories for storing and retrieving any kind of model or metamodel, with different access modes and protocol (streamed, by element navigation, event-based, transaction based, with versioning, etc.).

3 System engineering

The system engineering technical space will be illustrated here by the CORE set of standards.

We provide in this section a metametamodel of this space and describe some specific DSLs by metamodels based on this CORE M3-level facility.

First assumption is that Systems Engineering gets very specific challenges in comparison to Software Engineering.

The role of the Laws of World: Systems are ruled by laws of Physics and Sociology. The influence of the System on its own context has to be taken into account.

The multiplicity of the disciplines and cultures: Systems involve lots of different actors who can have different interpretations of the same notions (e.g. Interface, Function).

The stake of the design vs integration: It is nearly impossible to test Systems at implementation level, for various physical, social or political reasons. Systems have to be validated at design level, before implementation.

The management at the Life Cycle level: The system desing shall take into account the evolution and the future ruptures and transitions within the life cycle.

Assuming these fundamental differences in terms of challenges, M2 level languages are also completely different. However, it is possible to identify for each of these sets of languages some common properties allowing to specify a compliant meta-meta-model. The comparison between M3-level language of Systems Engineering and Software Engineering shows similarities and thus bridgeability.

It is thus possible to define mapping rules between meta-meta-models in order to make metamodels transformation automatic.

The idea of metamodel agnostic systems has been accepted. We suggest here the idea that metametamodel agnostic systems are not much more difficult to handle and that they could bring significant advantages.

Furthermore we are presently convinced that the technological level has reached the point where it should be feasible to build a common open model engineering platform capable of handling artifacts based on different meta-meta-models.

3.1 CORE meta-meta-model (M3)

See Appendix A for a UML diagram of CORE meta-meta-model

CORE is based on the entity-relation-attribute approach and thus provides a number of meta-meta-model elements:

- The *Schema* is the enclosing element of CORE meta-meta-model. A *Schema* instance represents the meta-model itself.
- The *ModelElement* entity represents the basic element of a given CORE Schema. It is an abstract supertype containing common fields of all meta-model elements, like “name” or “creator”.
- A *Facility* instance represents a group of *Class* instances. A given Class instance can be owned by multiple *Facility* instances.
- The *AttributedElement* entity is an abstract supertype representing the ability to own *Attributes* (see thereafter).
- A *Class* instance represents a given concept in a meta-model.
- A *Relation* instance represents a link between two *Class* instances. Each *Relation* instance has a complement, which is the reverse *Relation*.
- An *Attribute* instance represents a property of a given *AttributedElement* instance.
- A *PossibleValue* instance represents a certain value that can be taken to given *Attribute* instance.
- A *Target* instance comes with a *Relation* instance and gives every *Class* instance reachable through this relation from a given *Class* instance.

3.2 CORE meta-model (M2)

The basic CORE Schema is based on the metamodel TRW and provides a broad set of elements usable in modeling systems. This Schema can be further enriched by adding, modifying or deleting elements – classes, possible values, relations… – specific to a given domain. Such an enriched Schema can then be considered as a DSL and as a specific meta-model.

For instance, specific metamodels exist for C4ISR (***C****ontrol* ***C****ommand* ***C****ommunication* ***C****omputer* ***I****ntelligence* ***S****urveillance* ***R****econnaissance*) and

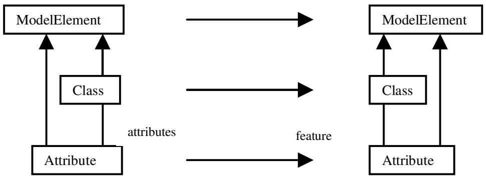
DODAF (***D**epartment **o**f **D**efense **A**rchitecture **F**ramework*).
As a DSL, a specific CORE meta-model can own a large number of elements spread between "essential" – elements common to every meta-model and undeletable – and "non-essential" ones. Essential elements cover classes necessary to any meta-model, such as the "System" whose instance would represent the real system which is modeled.

4 Bridging spaces

We describe here how the previous infrastructure may be used to define generic bridging facilities between these spaces.

4.1 M3 to M3 mapping

A *Schema* instance represents the meta-model itself and thus can be mapped in UML by a *Model* instance. Indeed, we should keep in mind that a meta-model can be considered as a model expressed in a meta-model that would be the meta-meta-model.
There is a correspondence between the notions of CORE *ModelElement* and UML *ModelElement*. Similarly, there is a correspondence respectively between notions of CORE *Attribute* and UML *Attribute* and between notions of CORE *Class* and UML *Class*.

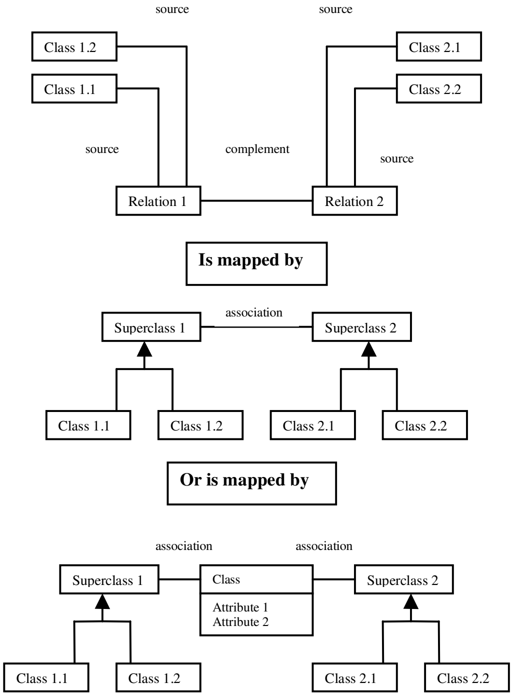


4.1.1. Schema of direct correspondences

Some of the links between and fields of these elements get their equivalent in UML representation:

- CORE *Class* "parent" link becomes a UML *Generalization*.
- CORE *Attribute* "initialValue" field becomes a UML *Expression* linked to the corresponding UML *Attribute* through the "initialValue" link.
- CORE *ModelElement* "abstract" field data is stored in the equivalent UML *ModelElement* "isAbstract" field.
- CORE *ModelElement* "schema" link which links each *ModelElement* instance to the top-level *Schema* is mapped by a "namespace" link between the corresponding UML *ModelElement* and the top-level UML *Model*.

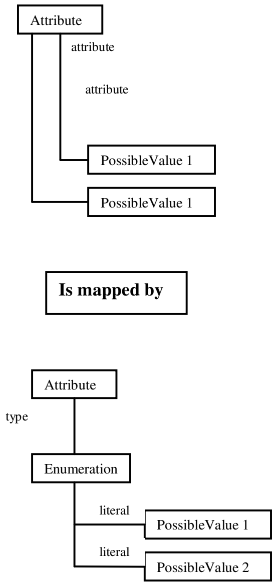
A CORE *Facility* can be mapped with a UML *Package*. UML *Class*es corresponding to this *Facility*'s CORE *Class*es are nested in this Facility through a UML *Dependency*.
Mapping a *Relation* involves to take into account the CORE *Relation* itself and its complement. Each of this relation is mapped by a super-class of all *Class*es sources of this relation, and another super-class of all *Class*es source of the complement. The link between super-classes and UML *Class*es is done through a UML *Generalization*.
Depending on whether the couple relation-complement owns *Attribute*s or not, the mapping is a direct UML *Association* between the two super-classes or an intermediary UML *Class* owning the UML *Attribute*s



4.1.2. Schema of Relations mapping

Properties of CORE *AttributedElement* are transferred to corresponding UML *Class*es and attributed *Relation*s. The relation "owner-attributes" is mapped by a UML *Association* "owner-feature".

CORE *PossibleValue*s are mapped with UML *EnumerationLiteral*s. These literals are attached to an *Enumeration* typing the *Attribute*.



4.1.3. Schema of Possible Values mapping

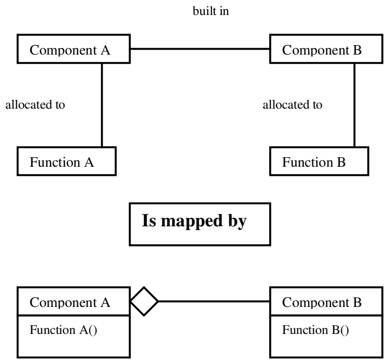
CORE elements fields that do not get their equivalent in UML are stored in UML *Tagged-Value* attached to these CORE elements equivalent in UML. For example, CORE *ModelElement* “alias” field will be stored in a UML *Tagged-Value* named “alias” tagged to the corresponding UML *ModelElement*.

4.2 Applications of M3 to M3 mapping

Mapping CORE M3 Infrastructure with MOF M3 corresponding level allows a broad field of applications. The main purpose is the capacity of automatic meta-model translation by allowing definition of translation rules from one meta-meta-model to the other. This means significant economies in terms of time, making M2-level manual mapping useless. This also means significant gains in terms of quality of the resulting meta-model, thanks to automatic translation.

This allows then to work with automatically generated M2-level meta-models and, possibly, automatically generated M2-level mapping between both meta-models, with corresponding M1-level transformation tools.

Example of this M2 level transformation would be automatic transformation of CORE *Component*s and attached CORE *Function*s through a link of “allocated to” in stereotyped UML *Class*es with attached UML *Operation*s.



4.2.1. Schema of M2-level mapping for Component and Functions

As seen is the precedent section, working at M3-level allows to write clear and simple transformation rules thanks to the high level of abstraction and the fewer types of elements.

4.3 Benefits

Discussed mapping and applications offer a number of benefits:

- Seamless system to software process-communication
- Increase traceability and reliability.
- Direct interface model and code generation, since the interface definition belongs to the system level.

5 Conclusions

We have presented here some work in the application of MDE ideas to the domain of system engineering. MDA is probably now the most advanced and visible technical space of MDE in software engiennerin, with practical tools like Eclipse EMF being defined and becoming widely available. We believe it is possible to conciliate the best of both worlds (software engineering and system engineering) by a clear and regular framework based on the idea of technical spaces. Building generic bridges at the representation level (i.e. the M3-level) seems a very promising engineering practice. We have provided some illustrations in support of this hypothesis. There is still much work to be done in this area. However if the general framework is shown feasible in these areas of system and software engineering, it may probably also be applied to many other areas as well.

Acknowledgements

This work has been benefited from the constant support and encouragements of Jean-Philippe Lerat, head of the Sodius Company and of other staff of the company.

About the Authors

Olivier le Merdy is a student at the Ecole des Mines de Nantes. He has been working for several months at the Sodius Company.

References

[1] Sodius. Available from www.sodius.com

[2] OMG/MOF: Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14, September 1997. Available from www.omg.org

[3] OMG/RFP/QVT: MOF 2.0 Query/Views/Transformations RFP, OMG document ad/2002-04-10. Available from www.omg.org

[4] OMG/XMI: XML Model Interchange (XMI) OMG Document AD/98-10-05, October 1998. Available from www.omg.org

[5] Bézivin, J.: In search of a Basic Principle for Model Driven Engineering, Novatica/Upgrade, Vol. V, N°2, (April 2004), pp. 21-24, http://www.upgrade-cepis.org/issues/2004/2/upgrade-vol-V-2.html

[6] Booch G., Brown A., Iyengar S., Rumbaugh J., Selic B.: The IBM MDA Manifesto The MDA Journal, May 2004, http://www.bptrends.com/publicationfiles/05-04%20COL%20IBM%20Manifesto%20-%20Frankel%20-3.pdf

[7] Kurtev, I., Bézivin, J., Aksit, M.: Technological Spaces: An Initial Appraisal. Int. Federated Conf. (DOA, ODBASE, CoopIS), Industrial track, Irvine, 2002.

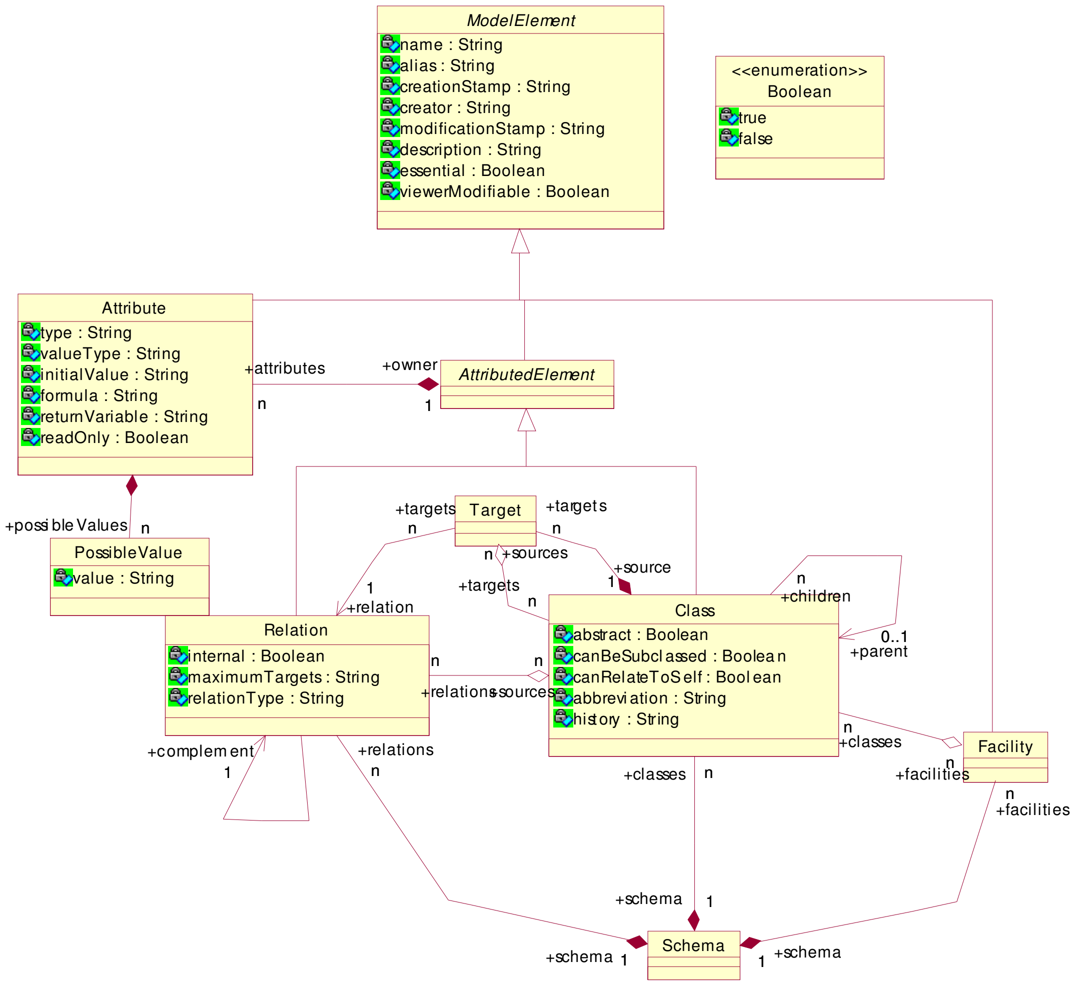
[8] Soley, R. & the OMG staff: MDA, Model-Driven Architecture, (November 2000), http://www.omg.org/mda/presentations.htm

[9] Vitech Corporation, founder of CORE tools. Available from www.vtcorp.com

[10] SysML. Available from www.sysml.org

[11] AP233. Available from http://step.jpl.nasa.gov/AP233/

Appendix A: UML Diagram of Core meta-meta-model


ModelElement
name : String
alias : String
creationStamp : String
creator : String
modificationStamp : String
description : String
essential : Boolean
viewerModifiable : Boolean
<<enumeration>>
Boolean
true
false
Attribute
type : String
valueType : String
initialValue : String
formula : String
returnVariable : String
readOnly : Boolean
+attributes
n
+owner
1
AttributedElement
+possibleValues
n
PossibleValue
value : String
+targets
n
Target
+targets
n
n
+sources
+source
1
+targets
n
1
+relation
n
+children
0..1
+parent
Relation
internal : Boolean
maximumTargets : String
relationType : String
n
+relations
n
+sources
Class
abstract : Boolean
canBeSubclassed : Boolean
canRelateToSelf : Boolean
abbreviation : String
history : String
n
+classes
Facility
n
+facilities
+complement
1
+relations
n
+classes
n
n
+facilities
+schema
1
Schema
+schema
1
+schema
1