Semantic Model Integration for System Specification

Creating a Common Context for Different Model Types

Oskar von Dungern

adesso AG, Rotherstrasse 19, 10245 Berlin, oskar.dungern@adesso.de

Abstract: Models from different methods and tools are integrated semantically. An integration model represents certain aspects of interest found in the individual models. It is built with the fundamental entities Event, Actor and State. Sharing entities between models improves coherence and interrelating entities adds semantic value. Some formal rigor and a common vocabulary helps getting insight and allows for automated checking. The resulting semantic net can be validated in terms of quality mre easily before the system is realized.

1. Introduction

The following goals are taken from a request for proposal recently issued by a well known car manufacturer:

- Create a common documentation site for business processes, IT applications, IT infrastructure and corporate standards – which are confined to individual tools, so far.
- Provide capability for cross-model navigation and search: Navigate along semantic relationships and find functions, application modules and/or requirements – independently of the authoring tool.
- Improve collaboration of different departments and enhance completeness and consistency, thus quality of the models.
- For feedback, let users comment on any diagram or requirement seen – establish an agreement process between stakeholders using a common platform.
- Offer an integrated requirement management related to system components and/or process steps.
- For better traceability, keep concepts and requirements in a "single-source-of-truth" – used as reference by derived work-products such as tasks or testcases.

The paper shows how individual models can be mapped to a common information model. Certainly, the mapping of individual models to an abstract model is hard work. The gain in semantic value and conceptual insight, however, is worth the effort.

2. The Problems of Model-based System Specification

A thorough analysis of popular modelling techniques regarding semantic value is still missing. To our observation the graphical representation is emphasized, while little attention is given to semantics and the logic interdependency between diagram types. The more elements are found in a toolbox and the more entries are offered in dropdown lists, the more difficult it is to make an unambiguous choice. The specification on SysML [Omg12], for example, distinguishes 163 named graphical node types, 22 data types, 211 metaclasses and 25 stereotypes. The authors state that the "Use of more formal constraints and semantics may be applied in future versions to further increase the precision of the language" [Omg12, p.19].

Standards such as XMI [Omg13] have been developed to transfer models from one tool to another, but in practice such interchange is rarely successful.

Semantic vagueness leaves the attribution of meaning at the discretion of tool-makers and users. The value for communicating concepts is diminished, as interpretation differs between modelers and readers – further explanation is required.

3. SpecIF Meta-model

Before discussing domain-specific aspects of model integration, a suitable meta-model is proposed, called Specification Integration Facility (SpecIF). A similar meta-model has been applied and proven in various projects; it may be improved with future practical experience.

The major design-goals are:

- Technology-neutral to allow for system interoperability and innovation.
- Dynamic model for adaption and extrension in a specific project context.
- Sufficiently simple to be fully implemented in the respective tools.

Similarly to UML, the SpecIF meta-model is defined in terms of the OMG MetaObject Facility (MOF) [Omg15]. In fig. 1 on the next page, all elements shown as a box are instances of ‚Class', while all connectors are instances of ‚Association' where ‚Association Ends' define the characteristics at each end, most importantly direction and multiplicity.

The meta-model defines both the *types* (fig. 1 to the left) and the *instances* thereof (fig. 1 to the right) for building system models:

- *ObjectType* and *Object* are used for diagrams and model elements, such as an activity diagram, a system component or a requirement.
- *RelationType* and *Relation* are used for logical relations between *Objects*, such as a system component *satisfies* a requirement. All relations are bilateral and directed. This allows for assertions according to first order predicate logic. They can be easily mapped to many technologies such as ReqIF and OSLC/RDF.
- *HierarchyType* and *Hierarchy* are used for the root of a tree whose leaves point to an *Object* each. Thus, *Hierarchies* with their *Nodes* define a logical order of *Objects*, such as a bill of material (BoM) or a document outline. A given *Object* may be referenced by none, one or multiple *Nodes* in one or more *Hierarchies*.

- A *DataType* may be defined based on any of the data types known from UML. In addition, some parameters defining the value range are available, such as minimum and maximum value of an integer or real number, the accuracy of a real number, a string length or a set of enumerated values.
- Every *ObjectType*, *RelationType* and *HierarchyType* can have an individual set of *AttributeTypes*, each of which is uniquely defined by a *DataType*. For example, a requirement might have three AttributesTypes, such as a „Title“ with *DataType* „String of max. length 96“, a „Description“ with *DataType* „XHTML of max. length 8192“ and a „Priority“ with *DataType* „Enumeration with a single-choice of [„1_high“, „2_medium“, „3_low“]“.
- An *Object*, *Relation* or *Hierarchy* is an instance of an *ObjectType*, *RelationType* or *HierarchyType* respectively. Each instance usually has a set of *Attributes* (i.e. values) corresponding with the *AttributeTypes* of its type.

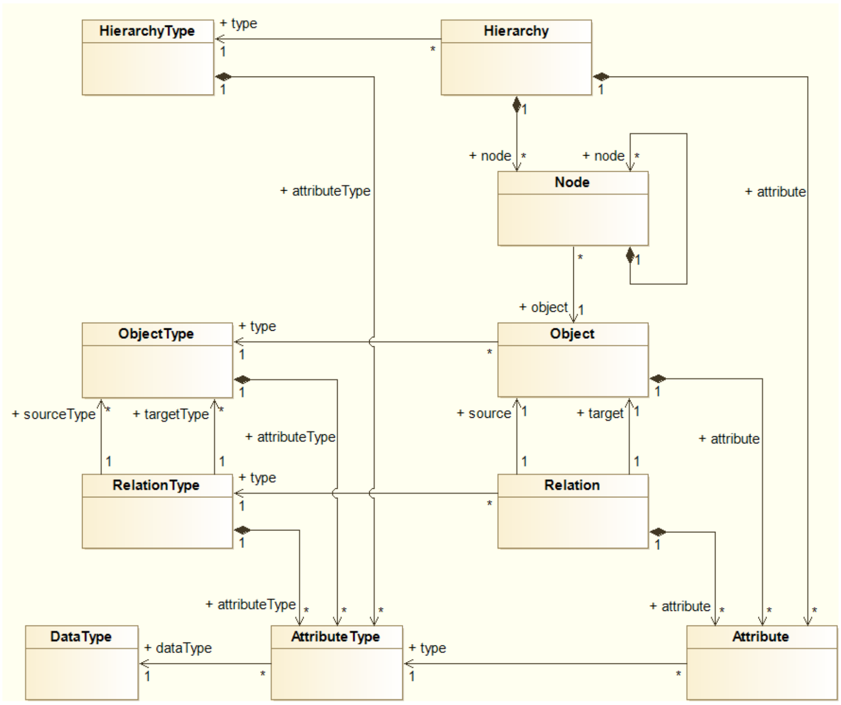

Fig. 1: The SpecIF meta-model derived from the OMG meta-object facility (MOF).

The meta-model consisting of types and instances allows us to dynamically define object and relation types with their attribute types at runtime. Also, when a system model is interchanged between tools, the type information is included. This concept is very similar to the ReqIF meta-model [Omg11]. In a subsequent chapter we will see which concrete types are proposed for SpecIF.

4. Model Integration

Few methods span all needed aspects of a complex system. In large organizations, business process management, IT architecture management and requirement management, all being recognized fields of specialization and expertise, use different methods and tools. The fact is equally apparent in mechatronic system development, where mechanical, electrical and software engineers are designing a common system with their respective methods. Design alternatives should be discussed and weighted in a common effort of all disciplines. But how to document the results? An overarching modelling approach is needed: Different models must be semantically integrated.

It is no viable approach to ask all stakeholders to use the same method or even the same tool. Our approach keeps the proven methods and tools for the different disciplines: A team may choose the most beneficial method and tool – without compromise.

Added value is created by setting the results of ‚any' modelling effort into a common context. So far disparate models shall be semantically integrated in certain aspects of interest. Specific model elements are mapped to abstract ones in the integration model. By exploring the results in a common context it is possible to get insight into mutual dependencies and to uncover inconsistencies.

We propose a rather simple approach to comprehensively represent system specifications, which is being used successfully in several industry projects. The following steps are taken towards model integration:

1. Separate View and Model
2. Abstract Model Element Types
3. Share Model Elements between Views
4. Interrelate Model Elements

Separate View and Model

First, let us distinguish between ‘View' and ‘Model' in the domain of system specification:

- A *Model* is a (simplified) representation of a system in focus. Conceptual models are used to help us know, understand, or simulate the subject matter they represent.
- A *View* exposes a selective aspect of the model to a target audience and with a communication purpose. Well known and comprehensible diagram types shall be used.

In other words, a model diagram is not a *Model*. A *Model* is the logic behind, where a ‚good' *Model* is comprehensive, coherent and interrelates the *Views*. In practice how-ever, it is no trivial task to identify common model elements to assure model coherence, especially in larger teams.

Abstract Model Element Types

Imagine the results of different modelling techniques can be assessed side-by-side in a common context. Which elements are conceptually the same and are comparable, therefore? How to identify the same entities in different model views?

A variety of graphical notations and model element types is used in different methods. There are many conceptual similarities, though. Based on the Fundamental Modelling Concepts [Knö05] and considering widely used model elements in system specification, the following abstract model element types (*ObjectTypes*) are proposed for SpecIF:

- A ▣ *View* is a model diagram with a specific communication purpose, e.g. a business process or system composition.
- An ■ *Actor* is a fundamental model element type representing an active entity, be it an activity, a process step, a function, a system component or a user role.
- A ● *State* is a fundamental model element type representing a passive entity, be it a value, an information store, even a color or shape.
- An ♦ *Event* is a fundamental model element type representing a time reference, a change in condition/value or more generally a synchronisation primitive.
- A ✶ *Feature* is an intentional distinguishing characteristic of a system, often a so-called ‘Unique Selling Proposition’.
- A ↯ *Requirement* is a singular documented physical and functional need that a particular design, product or process must be able to perform.

Initially, the following widely used modelling techniques are considered:

- A Business Process, often using BPMN or EPK notation, is built using *Events*, Process Steps represented by *Actors* plus documents or messages represented by *States*.
- A System Composition, for example in SysML, UML or FMC notation, illustrates the system structure with its components and communication channels. System components are often distinguished according to their ‘processing’ and ‘storing’ nature, thus may be mapped to *Actors* and *States*. A communication channel is considered a *State*, because it transmits information from one *Actor* to another – conceptually it is irrelevant whether they communicate using messages or a database, for example.
- An entity-relationship diagram or UML class diagram details the information from business objects to database records and defines their relations. It is all about *States*.
- A finite State Machine consists of *States* and Transitions, where the latter are considered *Actors*.
- There are many other useful model types, for example Petri Net or User Interface Mockup. All of these may just as well be mapped to the proposed abstract entities.
- A tree (hierarchical list) may reference any model element, plus any diagram. Widely used are Feature List, Bill of Material or Part Breakdown. And obviously the outline of the system specification document itself is a tree, as well.

When using this convention, it is impossible (and not even desirable) to map all details of a specific model to the integration model. On the abstract level, only those model elements and relations are of interest which are common to different methods and lend themselves for sharing and interrelating.

Share Model Elements

Employing just a few fundamental model element types lets us share entites between diagrams. Fig. 2 and fig. 3 shall be interpreted in terms of the SpecIF meta-model: The boxes represent *ObjectTypes* and the connectors represent *RelationTypes*. Fig. 2 shows four *ObjectTypes* used for views at the top and three *ObjectTypes* used for model-elements at the bottom.

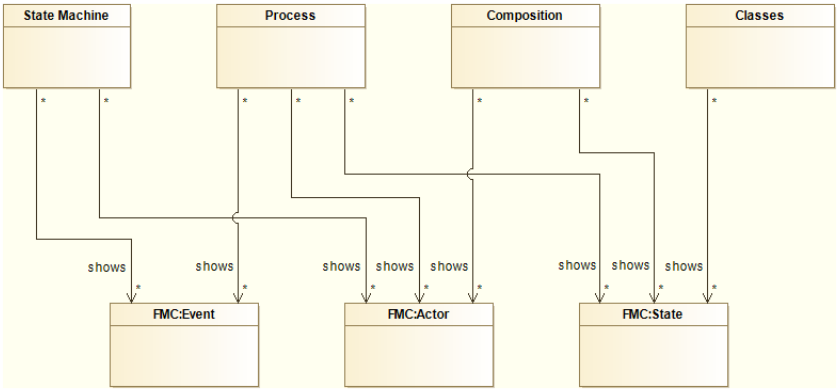

Fig. 2: SpecIF system model derived from the SpecIF meta-model with interrelated *ObjectTypes* used for diagrams and model-elements

Also, eight *RelationTypes*, all named *shows*, between specific diagram types and specific model-element types are visible. Of course, certain constaints apply. Depending on the diagram type, different model elements are allowed:

- A state-machine diagram may show *Events* and *Actors*.
- A process diagram (often SysML Activity Diagram or BPMN) may show *Events*, *Actors* and *States*.
- A composition diagram may show *Actors* and *States*.
- A class diagram (often UML/SysML) or an entity-relationship diagram (Chen) may just show *States*.

Interpreted in a different way, we see which model element types may be shared by which diagram types:

- An *Actor* may be employed both by Processes and Compositions.
- A *State* may be used by Processes, Compositions and Classes.

- An *Event* may be shared between Processes and State-machines.

Sharing model elements between views means to reuse existing *Object* instances when creating a view (a priori) or to consolidate those which are actually the same (a posteriori). A model-element appearing in very few views only is a potential 'loose end' and candidate for consolidation with others. If multiple instances of the same model element are found in federated repositories, they are tied together with a *sameAs* relation.

Interrelate Model Elements

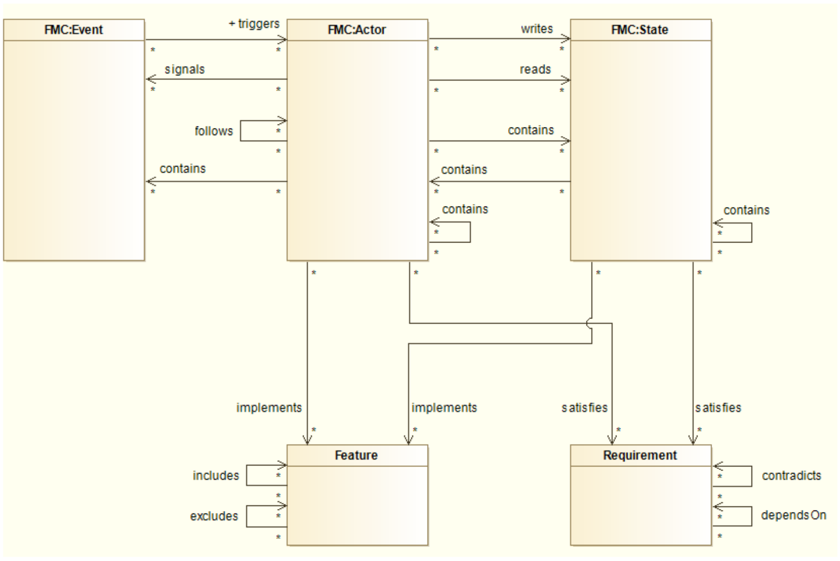

Fig. 3: *ObjectTypes* and *RelationTypes* derived from the SpecIF meta-model

The model elements may be connected with relations, thus contributing to the semantic value of the specification model. Fig. 3 shows the most important *ObjectTypes* and *RelationTypes* of a SpecIF integration model:

- An *Actor* writes a *State*: In a system composition a function writes a value.
- An *Actor* reads a *State*.
- An *Actor* contains an *Actor*: In a process a role is responsible for a process step. In a system composition a component is part of another or a component has a function.
- An *Actor* may also contain a *State* as well as an *Event*.
- A *State* may contain an *Actor* or a *State*.
- An *Actor*, representing a system component or function, satisfies a *Requirement*.
- A *Requirement* contradicts or dependsOn a *Requirement*.
- An *Actor* implements a *Feature*.

- A *Feature* excludes or includes a *Feature*.
- An *Actor* follows an *Actor*: In a Process a process step follows a process step in a control flow.
- An *Event* triggers an *Actor*: A process step is initiated when an event occurs.
- An *Actor* signals an *Event*.

The system model in consideration, the concrete project data, is represented by *Objects* and *Relations* being instances of the *ObjectTypes* and *RelationTypes* shown in fig. 3.

Many *Relations* can be derived from the model diagrams. For example, if a system component is drawn within another, a relation *contains* may be automatically created. Other *Relations*, such as a System Component *satisfies* a requirement, must be created manually.

To maintain consistency of the model, logical constraints are applied. For example, when creating a new block on a diagram, only those are offered for reuse which have the same abstract model-element type. The resulting model structure can be logically analyzed, for example to discover events which are signalled, but never used to trigger anything. Similarly it can be checked, whether every transition in a state machine has a corresponding activity. Thus, completeness and consistency can be analyzed for quality assurance.

5. Example

The mapping of a SysML diagram to SpecIF is shown as an example. Consider the simple activity diagram in fig. 4, where two devices implement an anti-lock brake (ABS).

The relevant logical content is captured with the following SpecIF mapping:

Model element (*Object* instance)	**SysML type**	**SpecIF meta-type**	**SpecIF type**	**Description**
PreventLockup	Activity diagram	*ObjectType*	Process	This SysML model diagram is a ▣ *View* with link to the original diagram as SVG, PNG or similar.
TractionDetector	Activity partition	*ObjectType*	FMC:Actor	A SysML activity partition is an ■ *Actor*.
BrakeModulator				
DetectLossOfTraction	Activity	*ObjectType*	FMC:Actor	A SysML activity is an ■ *Actor*.
ModulateBrake				
TractionLoss	Object	*ObjectType*	FMC:State	A SysML object is a ● *State*
	Initial node	*ObjectType*	FMC:Event	A SysML initial node is an ♦ *Event*
	Final node	*ObjectType*	FMC:Event	A SysML final node is an ♦ *Event*
		RelationType	shows	Relates a ▣ *View* and a shown model-element (*Object*) of type FMC:Actor, FMC:State or FMC:Event.

		Relation Type	contains	Relates an activity partition of type FMC:Actor with a contained activity of type FMC:Actor.
	Control flow	*RelationType*	follows	Relates an activity of type FMC:Actor with the subsequent activity of type FMC:Actor.

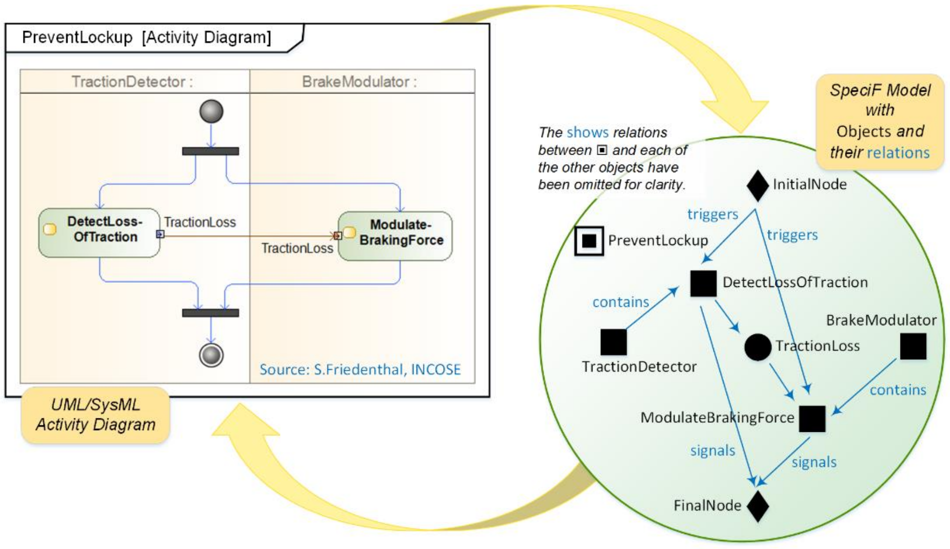

Fig. 4: An example activity diagram in SysML notation (left) and the corresponding SpecIF model (right).

6. Benefits

Once the views are integrated by shared or related model-elements, it is much easier to check and analyse the overall concept:

- Is every process step covered by a role?
- Are adequate system functions allocated to a process step?
- Which system functions are actually used by which process? And which ones are not (any more) used at all?
- Are there defined activities for every transition in a State-machine?
- Have all roles access to the required systems?
- How many systems are involved in a process?
- Does a system support a process seamlessly or is there any ‘hand-carried’ data?

When relating the Requirements to System Components or Process Steps, additional insight is obtained. It is very difficult, if not impossible to assess a list of many hundred requirements in terms of completeness, consistence, feasibility and other quality criteria.

But when requirements are distributed among system components, the system composition acts as a map giving rise to useful cross-checks:

- Have certain components none or too few requirements to be fully understood by a developer? The questions of a developer may lead to further requirements collection or elicitation.
- Is the set of requirements applying to a system component consistent – or are there any contradictory statements?
- Which requirements cause excessive effort?
- Are there any requirements which cannot be attributed to a system component and may thus not be satisfied?

These questions are only some examples. Interrelating model elements really helps in getting insight and in checking the model quality. In real projects, the approach has shown to stimulate discussions between experts of different background. Many interesting aspects are brought up long before the development work starts.

Literature

[Wen01] Wendt, S.: Ein grundlegender Begriffsrahmen für das Wissensmanagement im Software-Engineering. In Proceedings „Knowtech“ Dresden 2001. http://www.community-of-knowledge.de/fileadmin/user_upload/attachments/f25.pdf.

[Knö05] Knöpfel, A.; Gröne, B.; Tabeling, P.: Fundamental Modelling Concepts – Effective Communication of IT Systems. ISBN-13: 978-0-470-02710-3. John Wiley & Sons, Chichester, 2005.

[Poh11] Pohl, K.; Rupp, Ch.: Basiswissen Requirements Engineering. ISBN 978-3-89864-771-7. dpunkt.verlag, Heidelberg, 2011.

[Omg11] Object Management Group: Requirements Interchange Format (ReqIF). http://www.omg.org/spec/ReqIF/

[Omg12] Object Management Group: OMG Systems Modeling Language (SysML™), Version 1.3, http://www.omg.org/spec/SysML/1.3/, June 2012.

[Omg14] Object Management Group: XML Metadata Interchange (XMI) Specification, http://www.omg.org/spec/XMI/

[Omg15] Object Management Group: MetaObject Facility (MOF) Specification, http://www.omg.org/spec/MOF/

[Kau15] Kaufmann, U., Pfenning, M.: 10 Theses about MBSE and PLM, http://gfse.de/Dokumente_Mitglieder/ag_ergebnisse/PLM4MBSE/PLM4MBSE_Position_paper_V_1_1.pdf

[Dun15] Dungern, O.v.: Integration von Systemmodellen mit fünf fundamentalen Elementtypen. TdSE Tag des Systems Engineering der GfSE, Ulm, November 2015. http://enso-managers.de/files/resources/enso-m/documents-de/TdSE-2015_Dungern_Modellintegration-mit-fuenf-fundamentalen-Elementtypen_(Text).pdf