Towards a Tool-Supported Quality Model for Model-Driven Engineering

Parastoo Mohagheghi, Vegard Dehlen, Tor Neple

SINTEF, P.O.Box 124 Blindern
N-0314 Oslo, Norway
{Parastoo.Mohagheghi, Vegard.Dehlen, Tor.Neple}@sintef.no

Abstract. This paper reviews definitions of model quality before introducing six properties of models that are important for building high-quality models. These are identified to be correctness, completeness, consistency, comprehensibility, confinement and changeability. We have earlier defined a quality model that separates intangible quality goals from tangible quality-carrying properties and practices that should be in place to support these properties. A part of that work was to define a metamodel for developing quality models with MDE in mind. In this paper we analyze existing literature in order to extract model quality properties and to build a quality model with focus on the quality of models. For this purpose the metamodel is implemented in a tool that allows us to model quality models. The advantage of defining the metamodel is learning how to precisely define quality elements and relations in the quality model, and building models that may be used to generate documentation, guidelines or checklists. The disadvantage is mainly in the research phase where the metamodel is not stable and undergoes changes.

Keywords: Quality model, model-driven engineering, metamodel, model quality.

1 Introduction

Model-Driven Software Development (MDSD) or Model-Driven Engineering (MDE) is an approach to software development that emphasizes using models when specifying, developing, analyzing, verifying and managing software systems[1]. Since MDE requires a model-centric software development approach, researchers have also worked on specific quality issues in MDE such as identifying characteristics of models that are important depending on the modeling purpose and how to achieve and evaluate them.

The work described in this paper also aims at identifying quality attributes and approaches to improve the quality of models. We have earlier defined a quality model with concepts and their relations to be able to build quality models for different domains and different targets of quality; for example models, languages or

[1] We use the term MDE in the remainder of this paper to cover these approaches.

transformations. This work is explained in [1] and [2]. We have also developed a prototype tool in Eclipse built on the metamodel that is described in [2]. This paper builds an instance of the quality model with focus on the quality of models. The goal is two-folded: 1) identifying quality-carrying properties of models and relating them to practices needed to achieve them; 2) presenting the tool and metamodel to the audience of this workshop to get feedback.

The remainder of this paper is organized as follows. Section 2 discusses concepts important for discussing quality of models and provides a classification of model quality-carrying properties. Section 3 introduces the metamodel and the prototype tool while Section 4 presents a model on the quality of models which is developed by using the concepts presented in previous sections, an extensive literature search and by using the tool. Section 5 is discussion and conclusion.

2 Model Quality

Discussing approaches to improve the quality of models is not possible without discussing what model quality means. Software quality in general has been subject of extensive research, as reflected in the various quality models and standards with their definitions of quality. We have discussed some existing quality models in [2]. Here we just provide a short description of our quality model and quality properties identified for models.

2.1 A Quality Model MDE

In [1] and [2], we discussed the need for defining a quality model[2] in MDE as a means to integrate quality work related to MDE. Our approach is built on the Dromey's quality model, which has three main principles: high-level quality attributes or *quality goals*, product properties that are important for achieving quality goals (called *quality-carrying properties*), and links between quality-carrying properties and quality goals [3]. Dromey's focus is on component-based systems and in order to establish the links, Dromey has identified four properties of components that impact software quality. These are:

- Correctness properties; related to the deployment of components and that rules are not violated; either internally or associated with their use in the context;
- Internal properties; how well a component is deployed according to its intended use or requirements, covering both correctness and other properties;
- Contextual properties; how to compose components in a context;
- Descriptive properties; requirements, design, implementations and user interfaces must be easy to understand and use for their intended purpose.

Although Dromey's model emphasizes separating intangible quality goals from tangible quality-carrying properties, it does not focus on how to construct these

[2] We called this a quality framework in earlier publications.

properties in a product. Therefore we have added the concept of *practice* to our quality model that helps achieving a quality-carrying property and maps to the concept of "means" in the work of Lindland et al. [4]. Adapting the approach to MDE requires identifying properties of models that impact software quality; equivalent to the four properties identified by Dromey for components. We discuss these properties in the next section.

2.2 Model Quality Goals and Properties

UML 1.5 defines a model as "an abstraction of a physical system with a certain purpose" [8]. Daniels defines three kinds of models based on their purposes:

- *Conceptual models* describe a situation of interest in the world, such as a business operation or factory process;
- *Specification models* define what a software system must do, the information it must hold, and the behavior it must exhibit. They assume an ideal computing platform;
- *Implementation models* describe how the software is implemented, considering all the computing environment's constraints and limitations.

MDA separates between CIM (Computational Independent Model), PIM (Platform Independent Model) and PSM (Platform Specific Model). Comparing with the Daniels definitions, a CIM is a conceptual model of the domain, sometimes called a domain model, and a vocabulary that is familiar to the practitioners of the domain is used in its specification. Thus it should be evaluated for its understandability by users and validity and completeness related to the domain ontology. PIMs may be both specification of what a system does and a platform-independent solution, while PSMs are clearly implementation models and are in the solution domain. PIMs and PSMs should be evaluated for other characteristics such as their correctness and consistency with the concepts in the CIM model. Other models that may be evaluated are pattern models, transformation models or even metamodels as models of models.

Various models may be used for communication, implementation, generation and execution, or documentation. Thus the users are either human beings (developers, customers, business analysts etc.) or tools that should interpret the models. There are some definitions of quality goals for various types of models and various purposes of modeling that we refer to here. We cannot provide an extensive review of model types and quality goals due to the limited space of this paper but refer only to the related work that is used as basis for our definitions.

One of the earliest and widely referred works on model quality goals is the article by Lindland et al. [4]. They refer to earlier work on the quality of requirement models that contain quality goals such as:

- Appropriate: specifications should be free of implementation concerns, concepts must be suitable for the domain and the systems role in the environment such as data processing.
- Conceptually clean: covers notions such as simplicity, clarity and ease of understanding.

- Complete: contain all needed information.
- Expressive economy: least number of statements.
- Unambiguous: every requirement has only one interpretation.
- Consistent: Not in conflict with one another.

We realize that the above goals are applicable to other types of models as well. There are also quality goals that are especially relevant for requirement models such as being verifiable, having a traceable identification of requirements and being testable.

Lindland et al. have defined a framework for the quality of conceptual models which relates model quality to modeling language, domain and the audience interpreting the models. They have further borrowed three linguistic concepts to classify quality goals of models. These quality types are:

- *Syntax*. Relates the model to the modeling language by describing relations among language constructs without considering their meaning. Syntactic quality is therefore how well the model corresponds to the language. Syntactic errors happen if a model contains symbols not defined in the language or a model lacks constructs or information to obey the language's grammar. Having a formal syntax helps prevention and detection of syntactic errors.
- *Semantics*. Relates the model to the domain by considering not only syntax but also relations among statements and their meaning. Semantic quality is how well the model corresponds to the domain or the knowledge of people from the domain. There are two semantic goals: validity and completeness. Validity means that all statements made by the model are correct and relevant to the problem. Completeness means that the model contains all the statements about the domain that are correct and relevant. The authors write that semantic quality is difficult to achieve and it is best to relax the requirements and agree therefore on feasible validity and completeness. Consistency, unambiguity and being minimal are also types of semantic quality that are covered if models are valid and complete. Semantic means (or practices as we called them) are adding or removing statements to make a model complete or valid, and consistency checking.
- *Pragmatics*. Relates the model to the audience's interpretation of it. There is one goal of pragmatic quality: comprehension. The comprehension requirement may be relaxed since it takes a lot of effort to develop a large model where every part is comprehensible by everyone. Pragmatic means are those that make a model easier to understand; for example inspection, visualization (use of graphical models instead of textual), filtering (hiding details, using different languages for different parts, and we may also add the MDA concept of models at different abstraction layers), and executable models that can be simulated to help understanding.

Fig.1 shows the quality goals related to each of the quality types and means identified to achieve them.

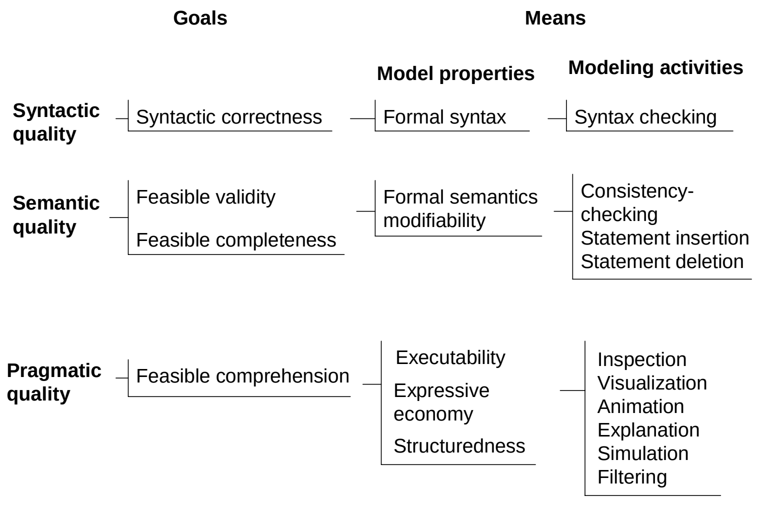

Fig. 1. Proposed quality goals and means in the framework of Lindland et al. [4]

This framework is later extended by others, among them Krogstie [5] who has added other quality goals to the framework such as *organizational quality* (defined as whether a model fulfils the goals of modeling and that all the goals of modeling are addressed through the model) and *technical pragmatic quality* defined as being interpretable by tools. Solheim and Neple have added *productivity* to the organizational goals and have defined two other quality goals relevant for MDE; i.e., transformability and maintainability [12].

Unhelkar has also used Lindland et al.'s framework but replaces the pragmatic quality with *aesthetics* [6]. He therefore defines three quality goals for UML models:

- *Syntax* with focus on correctness, for example does classes have correct attributes and operations, does attributes have correct types, etc. His definition of syntax also covers documentation, packaging and other issues related to understandability of models that Lindland et al. defined as pragmatic quality.
- *Semantics* or meaning with focus on completeness, consistency and representing the domain: does elements represent entities as they should, are dependencies correct, are models consistent with one another? Inspection of models is often the best way to check semantic quality since not all models are executable.
- *Aesthetics* with focus on symmetry and consistency: does a class have too many responsibilities, how many actors and use cases are shown in one diagram, how many activity diagrams are associated with a use case etc. These checks are both to improve the look and to help understanding.

His classification is different from Lindland et al., and we chose to use the definitions of Lindland et al. when classifying quality goals in the remainder of the paper.

Haesen writes that the boundary between syntax and semantics is sometimes blurred [10]. For example in [13], Harel and Rumpe state that as soon as a constraint can be automatically checked, it is a syntactic constraint, while a semantic constraint is formulated in natural language and cannot be checked automatically. This implies that if a modeling language has been well-formalized, more constraints can be automatically checked and are considered to be syntax, whereas for a badly formalized language, almost everything becomes semantic checking. Another view on syntax versus semantics is that syntax refers to the well-formedness of a single diagram or view, whereas semantics refers to the existence of a (mathematical) model defining the ”meaning” of a diagram and allowing reasoning and inference on specifications. Therefore although we keep the notions of syntax, semantics and pragmatics, we realize that we need a more precise definition of quality goals for models that is based on the experience of developers who rather talk of properties such as consistency or correctness.

Based on the above discussions and the results of an extensive literature search that we have done (some results are already published in [1] and [2], while we work on publishing the others), we have identified five basic quality properties of models that are shared between various models and are widely used in literature. These are important for achieving several quality goals such as maintainability of models and reliability of the generated software. They are also useful when discussing the impact of approaches to improve the quality of models. Others call them quality goals for models. The selection of terminology depends on the vocabulary used. These five quality properties of models are:

- *Correctness*; as including correct elements and correct relations between them and not violating rules and conventions; for example adhering to language syntax, style rules, naming rules or other conventions. Thus it covers both syntactic correctness (right syntax or well-formedness) and semantic correctness (right meaning and relations relative to the knowledge about the domain).
- *Completeness*; as having all the necessary information and being detailed enough; according to the goals of modeling. Completeness is a semantic quality.
- *Consistency*; as no contradictions in the models, related to semantic quality. It covers consistency between views that belong to the same level of abstraction or development phase (horizontal consistency), and between views that model the same aspect, but at different levels of abstraction or in different development phases (vertical consistency). The model should also be unambiguous; i.e. not allowing multiple interpretations.
- *Comprehensibility*; as being understandable by the intended users, related to the pragmatic quality. For human users, several aspects impact comprehensibility such as aesthetics of diagrams, organization of a model, model simplicity (or complexity which may be reduced by for example generalization and refactoring), conciseness (expressing much with little), and using domain concepts (Mitchell et al. call this *continuity* defined as carrying the structure and behavior of a problem domain into system and design models [7]). For tools,

having a formal syntax and semantic helps analysis and generation. Krogstie calls comprehensibility by human users as social pragmatic quality and comprehensibility by tools (or interpretability) as technical pragmatic quality.

- *Confinement* ; as being in agreement with the purpose of modeling and the type of system, and being restricted to the modeling goals, such as being at the right abstraction level and not having information not required (for example including implementation details in analysis models). Confinement is related to semantic quality.

A sixth property which has not received fair attention by literature so far but is of importance in a dynamic world is the *changeability* of models when the domain or our understanding of it changes or the solution must evolve because of changing requirements. We call these the *C6 properties* and think that they are fundamentally important in MDE and in any model-centric development process. Fig. 2 shows the quality properties in the transition from real world to software. The figure is inspired from a figure in [11] but has added new elements and quality properties to it.

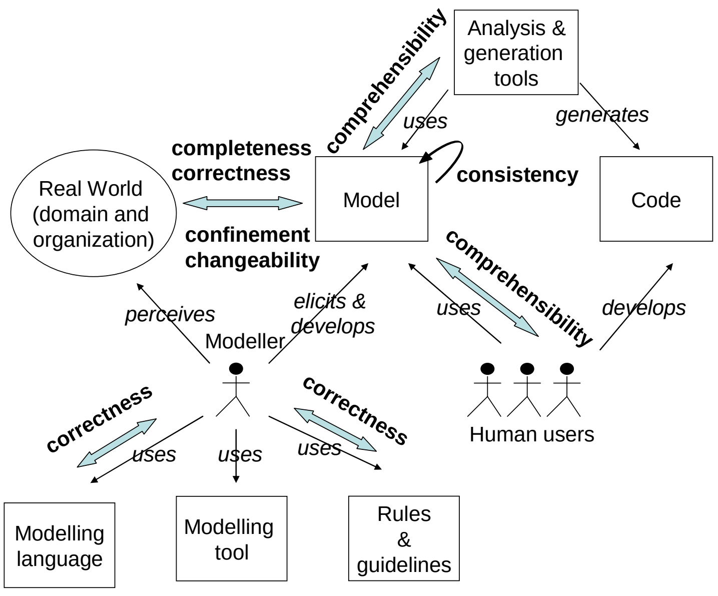

Fig. 2. Model-centric development with transformation of real world to running software

A model is a representation of a system and should be complete relative to the system it wants to represent and according to the modeling goals defined by the organization. The elements in the model should also be relevant to the domain and have correct relations relative to the knowledge we have of the domain. All these properties depend on the perception of the model developer from the domain and the modeling goals. Properties that are related to the hard assets (hard assets are shown with rectangles in Fig. 2, such as modeling language) may be evaluated automatically, while those that are related to the soft assets (real world and human users) depend on perceptions; i.e., how a developer perceives the real world and how human users interpret the model. These require other means of evaluation such as experiments and inspections. We may also add reverse engineering to the figure: tools may generate models from the code. But this is not relevant for the discussion here.

The order of the first five properties is important when evaluating them: correctness should be in place before one can evaluate completeness, and both are necessary for consistency. Comprehensibility requires correct, complete and consistent models. Finally it is difficult to match a model against the real world to evaluate confinement if it is not comprehensible. Changeability presents another dimension and is required, otherwise the models cannot evolve with changes in the real world and get outdated.

We think that other quality goals of models can be defined as a combination of these properties. For example, maintainability requires almost all of the above properties, while reusability requires models that are comprehensible and changeable such as being well-organized. Other properties may be added if we discover the need since we want a combination of properties that addresses the following requirements: 1) completeness, as including all basic properties, 2) parsimony or being minimal, meaning that all of the properties are necessary, and 3) independence or orthogonality of the properties [9]. All of the above properties can be achieved by practices and be evaluated by appropriate approaches. We discuss them further in Section 4.

3 Metamodel and Tool

In the context of our work on quality in MDE, we have created a metamodel and supporting tool for the definition of quality models. The tool provides a graphical syntax that allows developers to define the important quality concepts of a given domain – in our example here, quality of models – and the relationships between these concepts. The metamodel can be seen a library of the most common artifacts that should be considered for any quality model, guiding the quality engineer in her/his task. Furthermore, once a quality model has been built, we can use transformations to generate useful artifacts from the model, like questionnaires, guidelines or checklists. In the following, we will provide an overview of our metamodel and tool.

3.1 The Metamodel

To support the ability of creating quality models for different targets (such as models or languages) and domains, we defined a metamodel. A part of this metamodel, representing the core concepts, is depicted in Fig. 3[3]. For a thorough explanation of the metamodel concepts we refer the reader to [2]. A brief summary follows.

[3] Missing are several subclasses of these core elements along with some detailing of relationships.

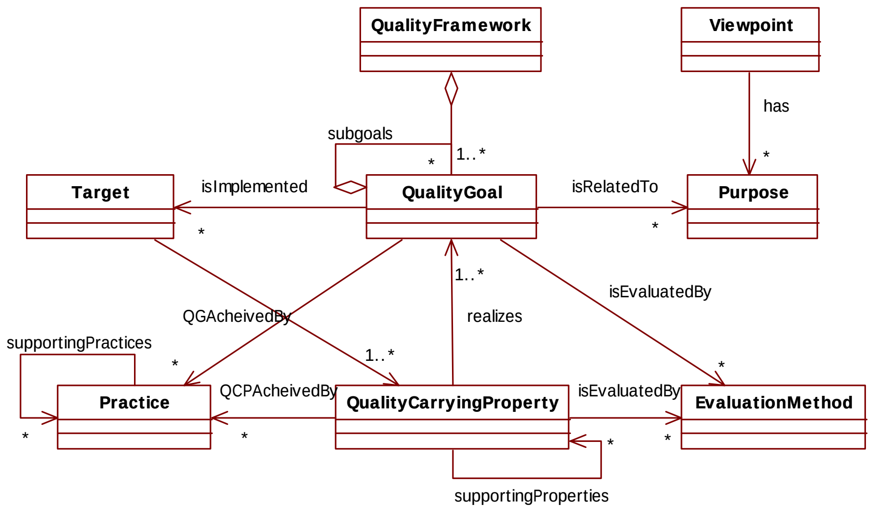

Fig. 3. Main constructs of the metamodel. Note that for improving the readability of the model we have omitted some elements as well as most of the aggregation relationships between QualityFramework and the other elements.

- A *QualityFramework* (or quality model) is a collection of quality entities and their relations. Quality frameworks can have *scope*. An example is whether a quality framework is generic or domain-specific.
- A *QualityGoal* is a clear and understandable definition of what quality means to a stakeholder such as users of a model, developers or managers. An example is to "improve software quality" of, for example, generated code. There may be a hierarchy of quality goals.
- A *QualityCarryingProperty* is some tangible property of an artifact or activity that is needed to achieve a quality goal. The purpose is to break down intangible quality goals into tangible properties of targets that can be evaluated. For example, understandability of models depends on them being simple and consistent. Quality-carrying properties may be supported by other properties.
- A *Target* is the artifact or activity that contains the property required to achieve a quality goal. For example, the quality of artifacts developed in an MDE approach depends on the quality of models, metamodels, tools, languages, transformations, modeling process and the expertise of people involved [1]. Therefore these elements are defined as target elements in the MDE quality framework.
- *Viewpoint* is used to indicate stakeholders of a quality goal such as system analyst, model developers or managers.
- *Purpose* describes what purpose a stakeholder, represented by a viewpoint, has in a given quality goal. For example, the purposes of modeling may be generation of code or documentation and these are related to specific quality goals.

- *Practice* is the means to achieve a required quality-carrying property. For example, using modeling conventions is a practice that can lead to developing correct and consistent models. Practices may be supported by other practices.
- Every property should be evaluated either quantitatively or qualitatively as defined in the *EvaluationMethod*. For example, including domain knowledge in a domain specific code generator (a practice in our model) will result in less error-prone code (a property) that can be evaluated by the reduction in the number of defects (a metrics).

All of the metamodel elements have the attributes name, definition (a textual description), type (in order to classify them if necessary) and evidence (connecting the element to literature).

3.2 Tool Support

The main point of the tool is, as mentioned, to support the use of the metamodel. Our goal is to allow people to define graphical quality models on different targets of a model-driven process, such as the tool, language, model, etc. We envision several benefits of our tool. Not only does such a quality model define what constitutes a high-quality target, but more importantly it systematically defines:

1. How high quality can be achieved during the development process, and
2. How to measure or evaluate whether or not this has been achieved.

Additionally, we plan to use such a model as the basis for:

1. Generating artifacts, like questionnaires, guidelines or checklists, by the use of model transformations.
2. Connecting evaluation methods, like metrics, model checking and simulation, to the quality model for evaluation purposes.

An early version of this tool has been implemented on the Eclipse platform using the Graphical Model Framework (GMF). Eclipse is widely used as a tool and development platform within academia and consequently provides several benefits; (1) people are experienced in using the environment, (2) using it promotes interoperability and allows our models to be used by other EMF-based tools, and (3) many plug-ins exist for possible reuse. The GMF plug-in, for example, allows one to create a concrete syntax in a graphical editor. Currently, our concrete syntax uses a UML profile-like syntax and consists of; a box, the concept name inside guillemots, a color and the name of the instanced concept itself. We also support showing the semantics of connectors by differentiating them either through graphics or tags.

This syntax is considered temporary, and our intention is to experiment with an increased use of graphics to differentiate concepts. The flexibility of GMF in defining graphics and icons is also a key reason for choosing GMF over UML-profiling for our tool solution. Additionally, a smaller metamodel is much better to use for model transformation and model validation purposes. Having an early implementation of the tool, our plan is to test it in a use case in order to gain experience with its usability and ability to model quality models. These experiences will also be the basis of the following tool iterations, as well as implementing transformations and model evaluation techniques mentioned in the start of this sub-section.

4 An Instance of the Quality Model with Focus on the Quality of Models

We have performed a review of literature concerning approaches to improve the quality of models. This work is going to be submitted to a journal soon. Different publications have focused on different properties of models while our goal is to integrate these into a model that shows the C6 properties as discussed before and their relation to the practices proposed to improve the quality of models. This model is also used to examine the relations in the metamodel depicted in Figure 3. The target in our discussion in this paper is generally models.

There are three viewpoints when discussing the quality of models:

- Developers who need to understand models for the purpose of implementation and modification. For them, models should be defect-free in the first place and of course understandable;
- Tools that should interpret and analyze models or generate other artifacts from them. For them, models should be defect-free and technically comprehensible;
- Others who use models for the sake of communication. They need models that are understandable in the first place, and in less degree the defect-free aspect.

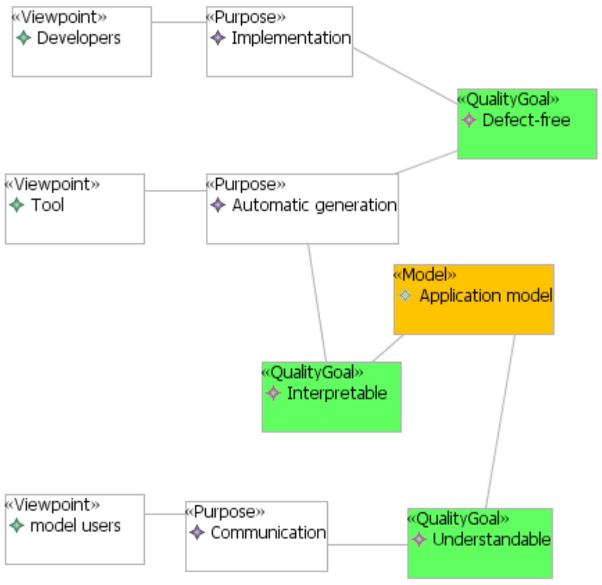
«Viewpoint» Developers
«Purpose» Implementation
«QualityGoal» Defect-free
«Viewpoint» Tool
«Purpose» Automatic generation
«Model» Application model
«QualityGoal» Interpretable
«Viewpoint» model users
«Purpose» Communication
«QualityGoal» Understandable

Fig. 4. Viewpoints, purposes of modeling and related quality goals

Thus we have defined three high-level quality goals: defect-free, interpretable by tools and understandable for human users (the last two are aspects of comprehensibility). Fig. 4 shows these quality goals. The quality model is developed by using the tool described in Section 3.2.

Being confined is also important if an organization has defined goals of modeling and one of its practices (using multiple views) might help understanding as discussed later. To be defect-free, a model should be correct, complete and consistent. To be understandable, a model should be appropriate relative to the domain, be well-organized, aesthetic and be close to the structure people have in mind of the problem domain. We have not shown comprehensibility as a property in the model since understandability and interpretability are defined as quality goals. Fig. 5 shows correctness and completeness properties important for a defect-free model. The relations between elements such as AcheivedBy and SupportingPractices are defined in the model while we have not assigned separate and specific graphics to them yet.

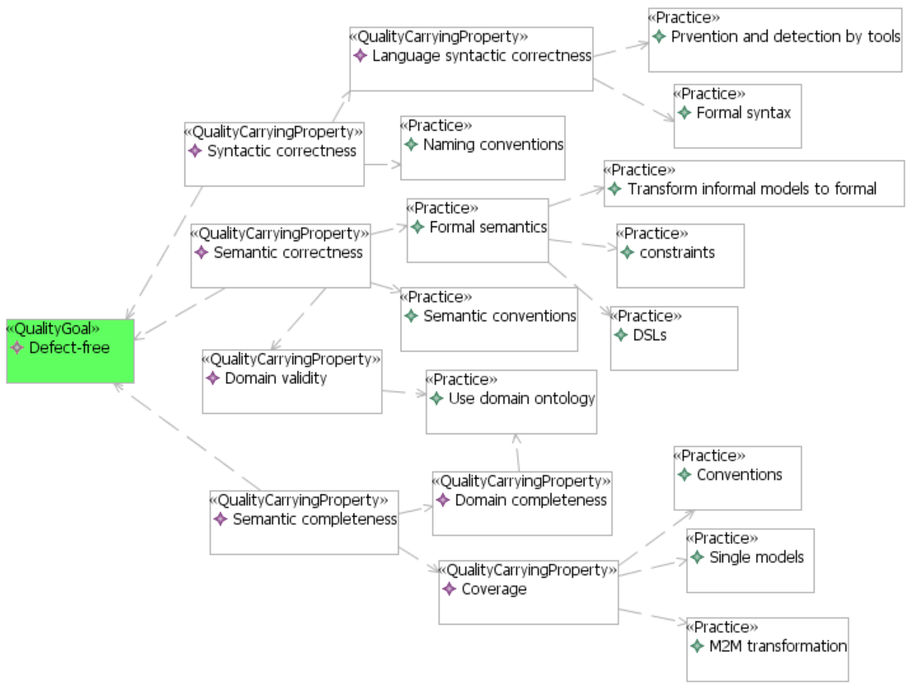
«QualityGoal» Defect-free
«QualityCarryingProperty» Language syntactic correctness
«Practice» Prvention and detection by tools
«Practice» Formal syntax
«QualityCarryingProperty» Syntactic correctness
«Practice» Naming conventions
«Practice» Transform informal models to formal
«Practice» Formal semantics
«QualityCarryingProperty» Semantic correctness
«Practice» constraints
«Practice» Semantic conventions
«Practice» DSLs
«QualityCarryingProperty» Domain validity
«Practice» Use domain ontology
«QualityCarryingProperty» Domain completeness
«QualityCarryingProperty» Semantic completeness
«Practice» Conventions
«Practice» Single models
«QualityCarryingProperty» Coverage
«Practice» M2M transformation

Fig. 5. A defect-free model should be correct (syntactically and semantically) and complete. The third property important is being consistent which is not shown in the figure.

The model also includes evaluation methods but these not included in the figure to improve readability. Language syntactic correctness and some semantic correctness and coverage may be evaluated by tools. Other properties such as domain validity and domain completeness are usually verified by inspecting the models.

Some of the practices we have identified to achieve quality properties are:

- Syntactic correctness may be improved if a modeling language has formal (or precise) syntax and by using different types of conventions such as naming conventions. Conventions are either provided as checklists or are enforced by tools.
- Semantic correctness: Domain validity (all the statements are relevant for the domain) may be improved by involving domain experts and using a domain ontology. Also using a DSL or UML profile, semantic conventions, semantic constraints, and transforming informal models into formal languages to allow analysis are practices to achieve semantic correctness.
- Completeness may be defined as domain completeness and coverage (for example all states are covered in the processes). We do not know of any practices to achieve domain completeness other that involving domain experts and using domain ontology. Coverage may be improved by having modeling conventions and generating models from other models (model-to-model transformations).
- Consistency may be improved by defining consistency constraints or conventions, performing Model-to-Model (M2M) transformation, and having a formal semantics.
- Understandability by human users or comprehensibility may be improved by well-organized models (using packaging conventions and possibility by limiting the number of diagrams), improving aesthetics (by layout conventions) and domain-appropriateness (by using concepts of the domain). Interpretability by tools will be improved by having a formal syntax and semantics.
- Confinement may be improved by having conventions on modeling, and clear definitions of the goals of modeling.

Finally, improving the modeling process impacts all of the quality goals by means of different practices such as defining goals of modeling.

Fig. 6 is a crosscut of the quality model related to understandability. The properties are best evaluated by inspections or experiments, while some layout issues may be evaluated by collecting metrics such as the number of classes in a class diagram.

Some practices help several quality properties. For example using stereotypes or DSLs (shown as DSLs in Fig. 5 and Fig. 6) allows adding formal semantics to models and helps domain-appropriateness and understandability, plus preventing errors in modeling and achieving semantic correctness. Other practices may help one property and have negative impact on another property. For example modeling a system from multiple views may help understanding but introduces consistency problems. Therefore we have defined relations such as “helps”, “breaks” and “depends” in our metamodel. However, showing all the details in the above figures would make them difficult to read.

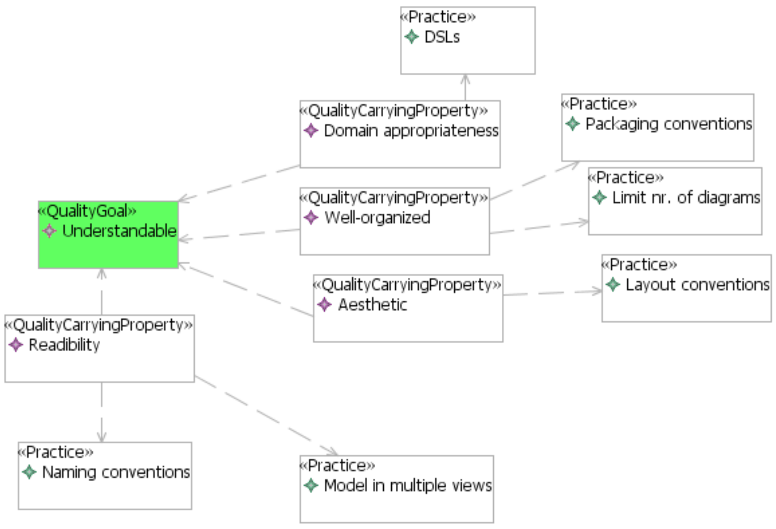
«Practice» DSLs
«QualityCarryingProperty» Domain appropriateness
«Practice» Packaging conventions
«Practice» Limit nr. of diagrams
«QualityGoal» Understandable
«QualityCarryingProperty» Well-organized
«Practice» Layout conventions
«QualityCarryingProperty» Aesthetic
«QualityCarryingProperty» Readibility
«Practice» Naming conventions
«Practice» Model in multiple views

Fig. 6. Quality-carrying properties and practices proposed to improve the understandability of models

5 Discussion and Conclusion

This paper covers our attempt to identify quality properties of models and integrate earlier work on the quality of models. It also shows how to use a developed tool to build a quality model that allows us to specify and reason about the quality of models. Our quality model thus covers quality goals, properties to measure if the goals have been achieved and practices that help us in actually achieving them, in addition to evaluation methods. The most important properties for measuring the quality of models have been identified as correctness, completeness, consistency, comprehensibility and confinement.

The purpose of our work is two folded: Identifying concepts and practices related to the quality of models and other aspects important in model-drive engineering such as languages and transformations, and identifying general concepts that are needed to build quality models. Our metamodel serves the second purpose and defines a language that may be used when defining quality models. It defines the elements of quality models and their relations to one another and is built by integrating earlier work. We have so far built two quality models using this metamodel and tool related to the first goal; i.e., specifying quality goals in MDE. The first model with focus on the quality of domain-specific languages was presented in [2] while the second one with focus on the quality of models is presented in this paper. Building the metamodel and tool also allows us to experiment with concepts and relations and gain experience on the aspects we cover. Our metamodel and models should also be evaluated according to the quality goals we identify.

Future work will cover enhancing the tool and inserting literature results in various models. Having an early implementation of the tool, our plan is to test it in a use case in order to gain experience with its usability and ability to model an organization's quality goals and identify relevant properties and practices. These experiences will also be the basis of the following tool iterations.

Acknowledgments. This work has been funded by the Quality in Model-Driven Engineering project (cf. http://quality-mde.org/) at SINTEF.

References

1. Mohagheghi, P., Dehlen, V.: Developing a Quality Framework for Model-Driven Engineering. Models in Software Engineering, LNCS vol. 5002, pp. 275--286. Springer, Heidelberg (2008)
2. Mohagheghi, P., Dehlen, V.: A Metamodel for Specifying Quality Frameworks in Model-Driven Engineering. In: Proceedings of the Nordic Workshop on Model Driven Engineering, Engineering Research Institute, University of Iceland, pp. 51—65 (2008)
3. Dromey, R.G.: Concerning the Chimera. IEEE Software 13 (1), pp. 33--43 (1996)
4. Lindland, O.I., Sindre, G., Solvberg, A.: Understanding Quality in Conceptual Modeling. IEEE Software 11(2), pp. 42--49 (1994)
5. Krogstie, J.: Evaluating UML Using a Generic Quality Framework. Chapter in UML and the Unified Process, Idea Group Publishing, pp. 1--22 (2003)
6. Unhelkar, B.: Verification and Validation for Quality of UML 2.0 Models. Wiley-Interscience (2005)
7. Mitchell, R.: High-Quality Modeling in UML. In: The 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), p. 388 (2001)
8. http://www.omg.org
9. Moody, D.L., Sindre, G., Brasethvik, T., Sølvberg, A.: Evaluating the Quality of Information Models: Empirical Testing of a Conceptual Model Quality Framework. In: The 25th International Conference on Software Engineering, pp. 295—305 (2003)
10. Haesen, R., Snoeck, M.: Implementing Consistency Management Techniques for Conceptual Modeling. In: 3rd International Workshop, Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships, pp. 99--113 (2004)
11. Nelson, H.J., Monarchi, D.E.: Ensuring the Quality of Conceptual Representations. Software Quality Journal 15(2), pp. 213--233 (2007)
12. Solheim, I., Neple, T.: Model Quality in the Context of Model-Driven Development. In: 2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS'06), pp. 27--35 (2006)
13. Harel D., Rumpe B.: Modeling Languages: Syntax, Semantics and All That Stuff. Technical paper number MCS00-16, The Weizmann Institute of Science, Rehovot, Israel (2000)