2012 Eighth International Conference on the Quality of Information and Communications Technology

A Multimodel for Integrating Quality Assessment in Model-Driven Engineering

Javier González-Huerta, Emilio Insfran, Silvia Abrahão
ISSI Research Group, Department of Information Systems and Computation
Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain
{jagonzalez, einsfran, sabrahao}@dsic.upv.es

***Abstract*— The development of complex software systems following the Model-Driven Engineering (MDE) approach relies on the use of different models for describing the system (e.g., structure, behavior). These models should be specified first separately but then their inter-relationship must be established since they represent complementary aspects of the system. Besides, MDE development processes are mostly focused on functionality, and do not give proper support to the quality aspects of the system. In this paper, we present a generic multimodel and the process for its construction, allowing the representation of the different viewpoint models of a software system and the relationships among elements on these viewpoints. This multimodel is a means for integrating a quality viewpoint in MDE processes, allowing the quality attributes to become a decision factor in the choice among design decisions in transformation processes. The feasibility of this approach is illustrated through the use of the multimodel in a specific example for Software Product Line development.**

Keywords: Model Driven Development; Quality Atttributes; Software Product Lines

I. INTRODUCTION

Model Driven Engineering (MDE) relies on the use of models at different levels of abstraction as a way for describing the system, which will later be obtained by transforming these models into the final code. The use of well-defined modeling languages and automated transformations raises the level of abstraction, improves the time to market and enhances the software quality. The models used in MDE development may represent different and complementary aspects of the system (e.g., structure, behavior, quality) and should be specified first separately, due to the separation of concerns [10]. However, the relationships among their constituent elements should be also established to have a complete representation of the whole system. The dependencies, relationships, and constraints that exist among these elements should be considered and explicitly represented to address the real complexity of MDE projects. Nevertheless, most MDE approaches only focus on the system functional requirements when modeling the system, without integrating the non-functional requirements (i.e., quality attributes) in the MDE process [1].

In this paper, we present a generic multimodel for representing the different viewpoints of a software system and the process required for its construction. This multimodel allows the establishment of relationships among elements on different system viewpoints, including the relationships among the quality attributes on the quality viewpoint, and elements on other viewpoints. With those relationships the quality attributes can become an additional decision factor when selecting among design decisions in model transformation processes. In general, these inter-viewpoint relationships defined by a domain expert represent *explicit* knowledge in the problem domain that can be used for constraining the solution space. In traditional MDE approaches the inter-model relationships are defined *implicitly* in the model transformation definitions whereas in our approach, we provide a mechanism to capture this knowledge as part of the multimodel.

Many approaches dealing with the definition of multiple related models have been defined in recent years [5], [9], [11], [12]. These proposals deal with megamodels [5], [9], [11], or macromodels [12] which are both defined as a model composed of related models, containing relationships between them [10]. The differences between the term megamodel and macromodel do not reside on its structure or semantic but on its intention [10]. Megamodels are mainly used to model management [5], to design software architectures where the design decisions are encoded as part of the model transformations associated with the relationships between models [11], and to reason about the types of relationships that can exist in MDE [9]. Macromodels are mainly used to capture how models are related to each other and to later analyze if those relationships are satisfied [12]. However, these approaches define relationships between models (i.e., *conformsTo*, *isComposedBy*), which allow analyzing, through the application of metrics, the inter-model dependencies, the change impact, or the completeness of a system description [10]. In our approach, the expressiveness is increased by explicitly representing relationships among model elements on different viewpoints instead of defining the relationships among models. This provides a more precise and complete representation of the system, while allows performing more complex analysis (e.g., to determine if it is possible to obtain a valid product configuration given the required functional and non-functional requirements).

The remainder of this paper is structured as follows. Section II introduces the role of multimodels in MDE. Section III describes the process for defining a multimodel, applied to a specific example of software product line development. Finally, the Conclusions and future work are presented in Section IV.

II. The Role of Multimodels in MDE

A *multimodel* is a set of interrelated models that represents the different viewpoints of a particular system. A *viewpoint* is an abstraction that yields the specification of the whole system restricted to a particular set of concerns and it is created with a specific purpose in mind. In any given viewpoint it is possible to make a model of the system that contains only the objects that are visible from that viewpoint [2]. Such a model is known as a *viewpoint model*, or a *view* of the system from that viewpoint. The multimodel also allows the definition of relationships among model elements on those viewpoints, which captures the missing information that the separation of concerns could introduce.

Fig. 1 shows the generic metamodel for multimodels. A multimodel contains viewpoint models, which are composed of entities and relationships among those entities. The multimodel also contains additional entities (*MultimodelEntity*) that extend entities on the different viewpoint models defined so that they hold inter-viewpoint relationships *(MultimodelRelationship*). These relationships involve two multimodel entities, which extend model entities on different viewpoints. Both *MultimodelEntity* and *MultimodelRelationship* can contain *MultimodelAttributes* for storing the multimodel information.

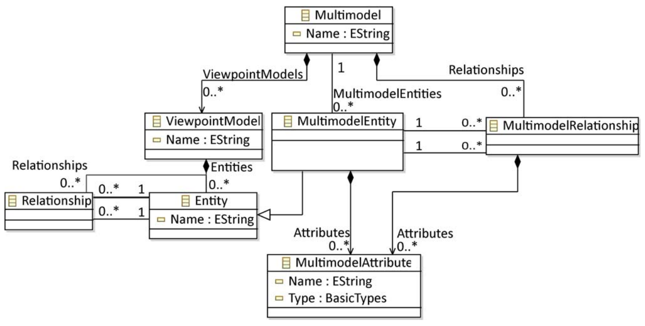

Figure 1. Generic Metamodel for Representing Multimodels

III. The Process for Defining a Multimodel

The process for defining a multimodel includes the following steps (shown in Fig. 2): A) Establishing the goal of the multimodel; B) Identification of the multimodel viewpoints; C) Selection of formalisms; D) Construction of the viewpoint models; E) Definition of inter-viewpoint relationships; and F) Inter-consistency validation. In order to illustrate this process, we have used an example of the software product line development in the automotive domain. A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features developed from a common set of core assets in a prescribed way [6].

A. Establishing the goal of the multimodel

To establish the goal of the multimodel a Goal/Question/Metric (GQM) template for goal-oriented software measurement is applied [3]. GQM is a technique that allows organizations to establish measurement models and mechanisms for feedback and evaluation in a structured and effective way. The result of the application of GQM is a measurement **system** that comprises three levels: goal, question, and metrics. In this specific application of the GQM we are interested in defining the goal and the questions that together define the set of viewpoints that constitute the multimodel.

A goal is defined based on the policy and strategy of the organization, the description of the products, the organization's processes and the organization's information model. The derivation of each goal results in a set of meaningful questions that characterize the goal in a quantifiable way.

In terms of the GQM template for goal-oriented software measurement, the *goal* of the multimodel in our example is to **represent** a software product line in the automotive domain, **for the purpose of** using this multimodel **with respect to** its ability to develop high-quality software products **from the point of view** of both i) *researchers* evaluating how expressive the multimodel is, and ii) the *domain engineer* evaluating the possibility of adopting the multimodel in his/her organization. The **context** concerns a model-driven engineering approach.

B. Identification of the multimodel viewpoints

The goal can be decomposed into different questions that we can use to derive the set of relevant multimodel viewpoints. Those viewpoints allow organizations to define and model the products they want to build.

In our example, the *questions* are: i) What quality attributes are relevant in our domain?; ii) Does a particular component impact on the quality attributes of the final product?; iii) Which products in the SPL better satisfies the quality requirements?; iv) How variability is supported by software components?; and v) Does the application of a given model transformation affect the quality of the products?

Fig. 3 shows the viewpoints that we need to define in the multimodel to express the relevant concerns identified in those questions: the *variability viewpoint*, which is introduced since the goal of the multimodel is the representation of a software product line, and in this domain the variability management is crucial [6]; the *quality viewpoint*, which is introduced since the main purpose of using the multimodel is the explicit representation quality information and the product quality improvement; the *functional viewpoint,* which is introduced since it is a way to represent the structure and components that the SPL comprises, and finally, the *transformations viewpoint* since the context of the multimodel is the MDE development approach and, in this scenario, model transformations are a key means for developing software systems.

C. Selection of formalisms

Once the viewpoints have been established the next step is to select the formalisms in which each viewpoint will be expressed. Each formalism should have the capability to express the concepts associated with each concern. In each case the different formalisms, or modeling languages, available for expressing the different concerns should be considered, and the one that best fits the organizational and technological needs should be selected.

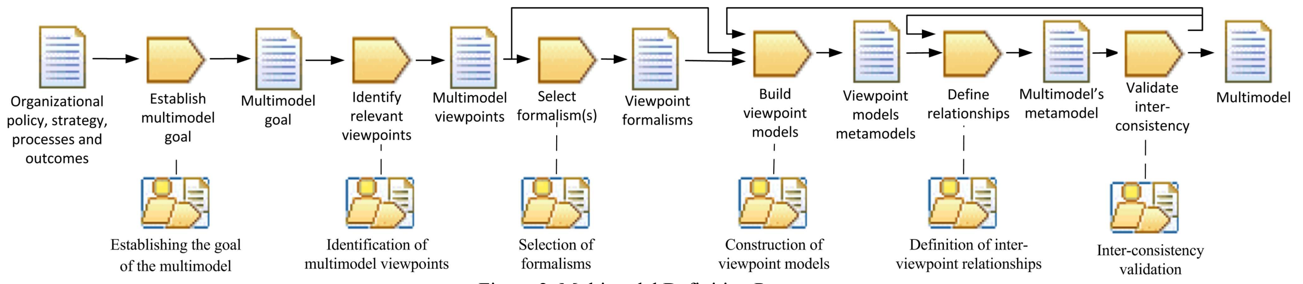
Organizational policy, strategy, processes and outcomes
Establish multimodel goal
Multimodel goal
Identify relevant viewpoints
Multimodel viewpoints
Select formalism(s)
Viewpoint formalisms
Build viewpoint models
Viewpoint models metamodels
Define relationships
Multimodel's metamodel
Validate inter-consistency
Multimodel
Establishing the goal of the multimodel
Identification of multimodel viewpoints
Selection of formalisms
Construction of viewpoint models
Definition of inter-viewpoint relationships
Inter-consistency validation

Figure 2. Multimodel Definition Process

In our example, if we are considering the quality viewpoint we can select existing *quality models* (e.g., SQuaRE or QUAMOCO [7]), *measurement ontologies,* or define a *metamodel* to express the quality concerns of an organization. If we are dealing with variability in a SPL then we can select one of the existing *feature modeling languages* (e.g., the cardinality-based feature model or the orthogonal variability model), or build our own *domain specific language (DSL)* to represent variability. If we are considering the architectural or functional viewpoint, then we can select an *Architectural Description Language (ADL)* (e.g., AADL, SYSML, UML or EAST-ADL) to describe the software architecture and its components; finally, if we are dealing with the transformation viewpoint, we can define a viewpoint to model the different transformation processes that occurs during the development process. Fig. 3 shows the formalisms selected for each viewpoint: a cardinality-based feature model for the *variability viewpoint*, the AADL specification for the *functional viewpoint*, the SQuaRE based quality model for the *quality viewpoint*, and finally, the transformation language to express the different model transformations for the *transformation viewpoint*.

D. Construction of the viewpoint models

Once the formalisms have been selected the next step is to define the structure and scope of each viewpoint, identifying its relevant entities. In some cases this step also involves the creation of DSLs for expressing a concern that is not supported by existing modeling languages, including the intra-model relationships among model elements inside each viewpoint.

In our example, we identify the relevant entities in each viewpoint: the *features* (i.e., user-visible aspects or characteristics of a system [6]) in the variability viewpoint (Fig. 3 top left), the *entities* representing the functional components in the selected ADL (Fig. 3 top right), the *entities* representing the quality attributes in the quality model (Fig. 3 bottom left), and finally, the *entities* needed for representing the transformation processes (Fig. 3 bottom right).

E. Definition of inter-viewpoint relationships

After building the viewpoints the relevant inter-viewpoint relationships among them must be established. Domain experts need to identify how the different viewpoints identified are interrelated, and establish relevant relationships among elements of the different viewpoints required for answering the GQM questions. The multimodel holds inter-viewpoint relationships defined among elements on different viewpoints (e.g., functional, quality), which can have different semantics:

- *Implication:* A model element *A* in a viewpoint implies an element *B* in another viewpoint.
- *Co-implication:* A model element *A* in a viewpoint implies an element *B* in another viewpoint and vice versa.
- *Exclusion:* A model element A in a viewpoint excludes the model element B in another viewpoint (e.g., a component in the functional viewpoint can exclude a transformation in the transformation viewpoint).
- *Decomposition (composite/part):* A model element A in a viewpoint can be decomposed into elements B, C… in other viewpoints. This follows the definition of δ edges introduced in [9] for describing how entities can be decomposed into more elementary parts.
- *Representation of relationships:* A model element *A* in a viewpoint can be represented by a set of elements {A...N} in another viewpoint. It is an adaptation of the definition of μ edges introduced in [8] for describing how systems and models can be represented in other models.
- *Impact relationships:* A model element *A* in a viewpoint impacts on an element *B* on another viewpoint. (e.g., a given entity in a viewpoint impacts on a quality attribute from the quality viewpoint). Additional attributes may be associated with this kind of relationship for the quantification of the impact.
- *Constraints:* The multimodel must be capable of expressing more complex relationships at the multimodel level. Those constraints can be similar to OCL restrictions. The language, or formalism, used to express those constraints will depend on the formalisms selected for expressing both the multimodel and the viewpoints.

In our automotive example, the relevant relationships needed to relate the different viewpoints are: i) the *decomposition* relationships among features from the variability view, and functional components in the functional view (e.g., in Fig. 3 the relationship between *ABS* and the *antilock_brake_system*); ii) the *impact* relationships that features have on the quality attributes of the final product (e.g., in Fig. 3 the *stability control* feature impacts on the *cost* quality attribute); iii) the *impact* relationships that functional components have over the quality attributes of the final product (e.g., in Fig. 3 the *cruise_control_system* impacts on the *resource consumption* quality attribute); and iv) the *impact* relationships that transformations have over the quality attributes of the final product (e.g., in Fig. 3 the *Watchdog pattern* impacts on the *Fault tolerance* quality attribute).

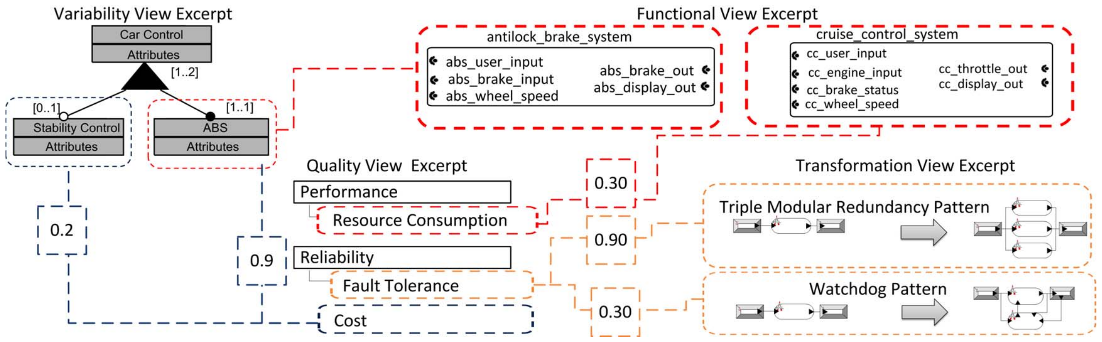

Figure 3. Example of Multimodel Relationships

In Fig. 3 the relative importance of the impacts is quantified by means of a weight ranging from 0 to 1, the higher the value the better it supports the quality attribute.

With these relationships the decision to select the features or apply transformations can be taken based on the functional requirements together with the quality attributes to be fulfilled (e.g., if we want to reduce the cost then we select the ABS since it better supports *Cost*, or if we want to apply transformations to improve the fault tolerance then we should select the *Triple Modular Redundancy pattern*).

F. *Inter-consistency validation*

Finally, after defining the relationships among the viewpoints a multimodel consistency analysis should be performed to analyze if the multimodel is expressive enough to cope with the organizational needs identified using the GQM template for goal definition. This analysis can be automatized by using automatic solvers like the FAMA [4] tool, that we can use for validating variability and quality relationships, or by using OCL solvers for constraints defined using OCL at the metamodel level, or by translating the inter-viewpoint relationships and constraints into other formalisms like propositional logic or constraint programing to check syntactic and semantic consistency. This activity provides feedback both to the *construction of the viewpoint models* and the *definition of inter-viewpoint relationships* activities to solve the eventual inconsistences identified.

IV. Conclusions and Future Work

This paper presents a multimodel approach to represent the different viewpoints of a software system developed following the MDE approach, and the process for its construction. The multimodel allows the integration of the quality viewpoint and the establishment of relationships among elements on different viewpoints. Those inter-viewpoint relationships allow us to define constraints that affect the whole system, to detect violations on those constraints, and to define transformation processes in which the selection among design decisions are taken considering not only the functional requirements but also the quality attributes that the product must fulfill.

This approach improves upon traditional MDE practices by providing a more comprehensible and flexible mechanism for modeling the relationships among elements of different viewpoint models, rather than introducing this information directly into the model transformation definitions.

As future work, we plan to formalize and validate the relationships described in this paper, to provide more detailed techniques for inter-consistency validation, and to refine the multimodel construction process by applying the approach to other domains.

Acknowledgment

This research is supported by the MULTIPLE project (MICINN TIN2009-13838) and the ValI+D program (ACIF/2011/235).

References

[1] D. Ameller, X. Franch, J. Cabot, "Dealing with Non-Functional Requirements in Model-Driven Development" In: 18th IEEE Int. Conf. Requirements Engineering, 2010, pp.189-198

[2] E.J. Barkmeyer, A.B. Feeney, P. Denno, D.W. Flater, D.E. Libes, M.P Steves, E.K. Wallace, “Concepts for Automating Systems Integration” NISTIR 6928, National Institute of Standards and Technology, U.S. Dept. of Commerce, 2003

[3] V. Basili, H Rombach, “The TAME Project: Towards Improvement-Oriented Software Environments,” IEEE Transactions on Software Engineering, 14, 1998

[4] D. Benavides, S. Segura, P. Trinidad, A. Ruiz-Cortes, “FAMA: Tooling a framework for the auto- mated analysis of feature models.” In: 1st Int. Workshop on Variability Modelling of Software-intensive Systems, 2007

[5] J. Bezivin, F. Jouault, P. Rosenthal, P. Valduriez, ”Modeling in the Large and Modeling in the Small” In: European MDA Workshops: Foundations and Applications, LNCS 3599/2005, 2005, pp. 33–46

[6] P. Clements, L. Northrop, “Software Product Lines: Practices and Patterns”, Addison-Wesley, Boston, 2007

[7] F. Deissenboeck, L. Heinemann, M. Herrmannsdoerfer, K. Lochmann, S. Wagner, “The quamoco tool chain for quality modeling and assessment.” In: 33th Int. Conf. on Software Engineering, 2009, pp. 1007-1009

[8] J.M. Favre, “Foundations of Model (Driven) (Reverse) Engineering. Episode I: Story of The Fidus Papyrus and the Solarus”. In: Dagsthul Seminar on Model Driven Reverse Engineering, 2004

[9] J.M. Favre, “Megamodelling and Etymology”, In: Dagstuhl Seminar 05161 on Transformation Techniques in Software Engineering, Dagstuhl, Germany, 2005

[10] R. Hebig, A. Seibel, H. Giese, “On the Unification of Megamodels” In 4th International Workshop on Multi-Paradigm Modeling, 2010

[11] D. Perovich, M.C. Bastarrica, C. Rojas, “Model-Driven approach to Software Architecture design” In ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK '09), 2009, pp. 1-8

[12] R. Salay, J. Mylopoulos, S. Easterbrook, “Using Macromodels to Manage Collections of Related Models” In Proc. of 21st Int. Conf. on Advanced Information Systems, LNCS 5565, 2009