Model Engineering and Traceability

Joel CHAMPEAU, Emmanuel ROCHEFORT
ENSIETA, Brest
Laboratoire Développement des Techniques Nouvelles
Email : emmanuel.rochefort@ensieta.fr
Tel: +33 298 348 839
August 15, 2003

Abstract

Joel
Keywords: Traceability, Requirements, Models, software engineering.

1 Introduction

The part of the software in embedded systems increases more and more in the last decade. The embedded systems grow in complexity due to their scope which became very large. They are mainly based on reactive software with distributing, communicating and embedding constraints. In parallel, if we want to be able to use embedded systems efficiently, we must optimize constraining implementation like response time, memory space and embedding constraints set another type of complexity on software.

In this context, the attention dedicated to the software modeling phase must increase to manage this complexity and to obtain reusable software. In order to organize models during the design step, the separation of concern is used to obtain different models connected to a dedicated concern. In this perspective, UML will be customized to support AOM (Aspect Oriented Modeling) [7] to gain the benefits of code and design reuse of aspect software. In a same way, the MDA (Model Driven Architecture) proposal is used to separate different models, independently to a dedicated platform or mapped to a specific platform. These proposal must take into account the platform models (PDM Platform Description Model) This platform model is very important in the scope of embedded systems due to the constraints which have an important impact on the PSM model. Another aspect, included in the MDA approach, is the greater part of model transformations. In a MDA process, each refinement step is based on model transformation like merging a design pattern with a PIM model or including security concern in a PSM model. In some transformations, the models can be based on different meta models to assume a semantic change for performance evaluations or behavior validation for example.

In this context of MDE (Model Driven Engineering), model transformations will be powerful. But we must assume a global coherence during all the process. In this perspective, stakeholder requirements must be modeled and we must have a continuity between requirements and final system implementation model. Thus the requirements traceability must be done of prime necessity at the model element level. In the case of embedded systems, real time constraints or requirements must be insured all along the process to keep the time constraints in a domain which allows a correct behavior of the system.

In this case of MDE and traceability we suggest in this paper a definition of requirements traceability in a transformation environment which can be apply on real time requirements. In the 2 first sections we present some definitions on traceability and requirements. After we positioned the traceability process in the MDA framework. And finally we suggest a model of concepts for the traceability context which must be powerful for embedded systems.

2 Traceability's meaning

Requirement traceability is the ability to describe and follow the life of a requirement, in both forward and backward direction [6] by defining and maintaining relationships to other artifacts created during system development [12]. With the requirement traceability, we want to know and understand the life of a requirement, but also its *impact*[2] on other elements. In this point of view, the requirement traceability is the ability to describe, follow and understand the life of a requirement and the elements that depend on it. It is to be able to trace or calculate the impact of requirements. The traced elements are to be understood as requirements, and by, like models of the system. And we don't mistake traceability and documentation. The documentation doesn't explain the information collected. But a good documentation is necessary to assure an efficient traceability.

Talking about traceability implies that we talk about traceability's tools. The traceability's tools are business oriented. For each type of business a different tool is used. A programmer and a project supervisor don't use the same tools. Their job are different, their tools too. There are tools to trace documents and their evolution[13] , tools to trace the different version of the documents or source code of system, tools to trace the requirements in the models[4]. These tools are business oriented and no one of these tools can be used during the whole project. So there is a discontinuity in the traceability. The informations collect by the different tools are not the same, and the traceability is not oriented in the same aim. And even if tools like Doors from Telelogic can be used during the whole project, they offer only a little help to analyze the traces. The collect of the trace, and the interpretation of the linkages are left to user.

About traceability tool's deficiencies, we can mention some of them. First, the capture of traces is either too complex, detailed and difficult to use afterwards, or too simple and many informations are forgotten, like rationales[1, 14]. The documentation entities, and relationship to be traced are often unclear.

3 Requirements and models

The requirements are used as textual elements. They could be expressed as formal or non formal text, or as graphical. Each requirement item could be classified in a category, like functional or design point requirement[8]. The relationships between the elements are simple, but no model of these relationships existed. And the relation between requirement and the documents are not clearly expressed. A problem is the translation of the requirements in the UML models. The uses cases diagrams are used to represent the functional requirements , and sequences diagrams modelise the non functional requirements. They don't be present in the others diagrams. It's a poor UML use: only 2 on the 9 types of diagrams. So the requirements are not present in the other models. Only few notes are used to reference few requirements. The reason of this situation is the absence of this notion in UML. The solution could be the creation of an element requirement, like a stereotyped class. In this way, Foisseau in [5] suggests a requirement meta model, which describe the important information to explain a requirement. This model could be use to define a UML profile(cf fig. 1). It is now possible to model the requirements, the information concerned by the requirements and the relations linked to these elements.

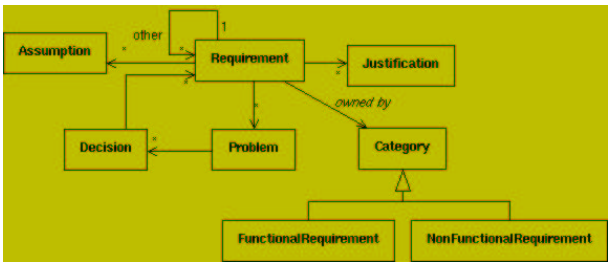

Figure 1: Requirement's meta model

4 traceability and MDA

With the MDA approach new problematics are coming into view in the context of modeling engineering :

- model transformations which demonstrate that we must have different models at different levels of abstraction and we can base the passage between different levels on a transformation which can be computed automatically. In this scope, the OMG RFP on Query, View and Transformations shows that a standardization is required on this new topics.
- the diversity of the models in an MDA process increases the modularity and the specialization of the models. A specialization on a domain can

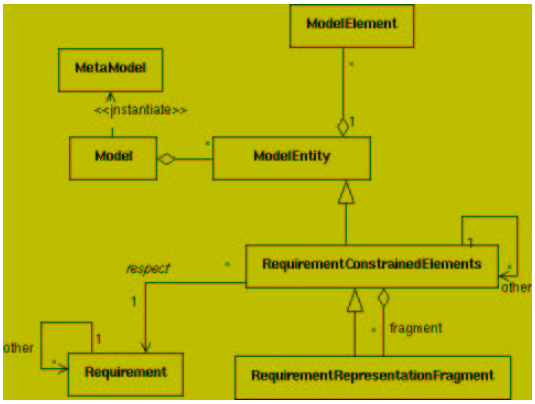

Figure 2: Requirement in a model

provide a dedicated meta model for this domain. This meta model can be managed in a UML profile or not, but in these different cases, we can see that we have different models with different meta models. In the scope of embedded system, a PIM model can include some time specifications using the SPT profile and a platform model with time specifications and some embedded constraints which are absent of the SPT profile. These constraints can be described in a specific meta model. In this case we have 2 models with some differences between the 2 meta models.

The transformations in MDA context can be classified in different types due to the transformation semantic applied during the process. Based on the literature [3], we can share the transformations into 2 main classes if we use the same meta model or if each model is defined with its own meta model. With the same meta model, we can have :

- refinement. Addition of new model elements to precise the description of an existing model element (adding subclasses to a class, or adding sub-statechart in a state).
- refactoring. Locally, optimizing model description to prepare the next modeling phase. (Minimizing the number of association in a class diagram or the number of transitions in a statechart)
- merging. If a design pattern is modeled, we can merge this model with a PIM model in a design step.
- building a view. If we want to extract temporally model elements to create a view. We must select model elements associated to a dedicated point of view.

With several meta models :

- merging. A PIM model can be merged with a PDM, and this PDM can have its own meta model dedicated to a specific platform.
- go and back in another language with another semantic. In the case of performance evaluation [11], or behavior validation [9], we can translate UML models in another language for evaluation purpose and integrate the results in UML models. In this case, we must establish a correspondence between model elements of the 2 languages.

During a design process, each transformation can be classified in the set of types and the used model element identified. The traceability process must store all these informations. The traceability must include a precise definition of the stored data via a meta model definition, presented in the section 5.

In a process based on transformations, the analyze of the traces can be very difficult due to a great quantity of traces. Visualization of a model element, from the abstract PIM model to the implementation model, should take into account many transformations and many traces. The analyze process must be based on the management of :

- transformation classes. We will suggest an analyze of the traces by a selection of the transformation by types. We can inspect all the transformations applied on a model element or a group.
- meta models of the different models. If we use different meta models, performance evaluation purpose for example, we must know which model elements are touched by the translation, and which model elements are created after the evaluation process. We must take into account the meta model information and it can represent an entry point for the analyze trace system.

For the analyze of the traces, we can see in the MDA context that the traceability process can not be based only on the visualization of all the links from a selected model element. The analyze process must be more flexible and for example we must be able to create views based on a specific transformation that can extract informations from different models (for different abstraction level or from different meta models) without any updates on the models. This process must be dynamic and based on precise definitions of points of views which must be defined the entry point, the wanted model elements and their meta model.

5 Traceability: a conceptual model

In the previous sections, we have seen the different elements and stakeholders concerned by the traceability. We will present now our traceability conceptual model.

The first elements of our model, the requirement meta model seen in the figure 1, is used to introduce the requirements in the models. It is now possible to link any elements of an UML model to a requirement which is here an

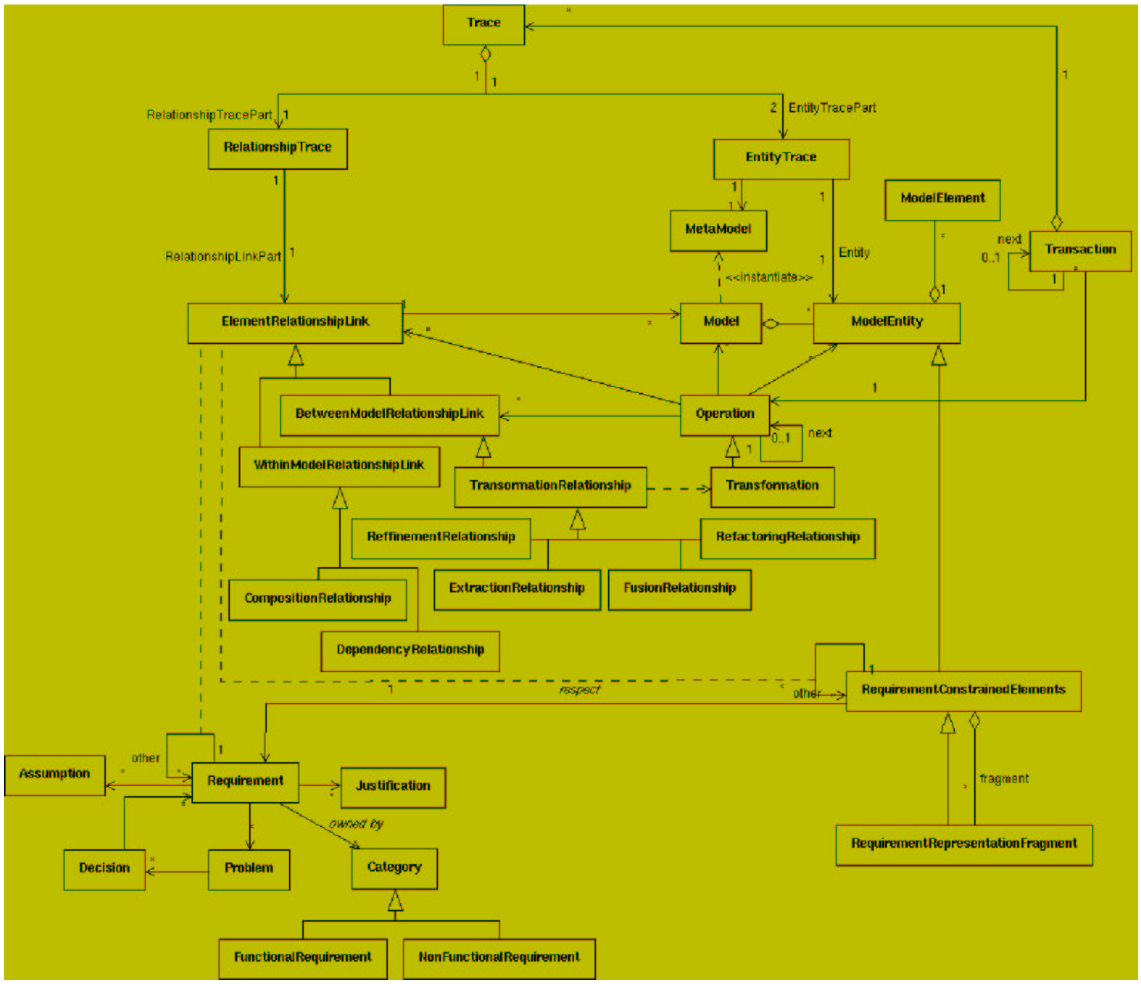

Figure 3: Traceability conceptual model

stereotyped element. The requirements are present is the models and they could be manipulated as every other UML elements. We can use them as reference elements or describe them in models.

And a model is viewed as a collection of "entities", which are a set of *modelElement*. Some of these "entities" could be linked to requirements. But in each entity, it is not possible to know which element is concerned by the requirement. For example, a car could be linked to a requirement: the car must be able to carry a person from a location to another. A car is composed with a motor, 4 wheels, a body, and many others elements. Each mentioned elements are concerned with the requirement, but they can satisfies this requirement only together. In this meaning, each element satisfy a fragment of the requirement.

In the traceability context, an important information is the relationship between two elements: models or *entities*. The models could be modified, or as seen before, could be transformed. Therefore there is a transformation relationship between these models, like refinement, or refactoring. And the model's elements are link together by composition or dependency relationship. In this meaning, we can distinguish two relationship classes: within model and between model. These elements of the conceptual model must obtain automatically if possible, or manually.

In the section 4, we have seen the importance of the meta models. Many meta models could be used to design the different system's models, and they are susceptible of changing along the iterative development process. By example, The PIM meta model could be different to the PDM and PSM meta model.

We have seen the different elements and informations concerned by the traceability, but we haven't talk about trace, the traceability element. A trace is the record of the change between two elements. She assemble the "entity" and relationship information on these two elements. And each trace is store and managed by a transaction.

Now, the conceptual model is completely defined. It is possible to create a tool, and a UML profile to use requirement in the models. Another way to view the traceability is to focus the on the relation between stakeholders, specification and rationale [10]. But we have oriented the traceability on the transformation aspect and on the relation between the models elements.

6 Conclusions

We have seen that the traceability is the ability to describe and follow the life of a requirement. And that there is no tool to trace the requirement during the whole development process.

In a MDA process, the models take an important place. They are often modified or transformed. We must be able to trace the models and the elements related to the requirements.

In this paper we proposed a traceability model and an UML profile to introduce the requirement notion in the UML models. It is now possible to clearly write which models and model element are related to requirements. And the

result is that we can trace the requirements and their impact during the whole development process.

References

[1] Incose requirements management tool survey. Online http://www.incose.org.

[2] S.A Bohner and R.S. Arnold. Software change impact analysis. *IEEE Computer Society Press*, 1996.

[3] J. Bézivin and X. Blanc. Mda - promesses et interrogations. *Développeur Référence*, V2.17, September 2002.

[4] Rational Software Corp. Requisite pro. Online http://www.rationale.com.

[5] J. Foisseau and M. Lemoine. Gestion des exigences pour la maîtrise de la pérennité. In *2éme Conférence Annuelle d'Ingénierie Système*, 2001.

[6] O. Gotel. *Contribution Structures for Requirements Traceability.* Thèse de doctorat, Imperial College of Science, Technology and Medicine, 1995.

[7] I. Groher and S. SSchulze. Generating aspect code from uml model. In *AOSD 2003 - 3e International Conference on Aspect-Oriented Software Development*, 2003.

[8] R. Harwell. What is a requirement? In *Third International Symposium of the INCOSE*, 1993.

[9] A. Knapp, S. Merz, and C Rauh. Model checking, time uml state machines and collaboration. Technical report, University of Munchen, 2003.

[10] P. Letelier. A framework for requirements traceability in uml-based projects. In *1st International Workshop on Traceability in Emerging Forms of Software Engineering*, September 2002.

[11] D. Petriu. Applying spt profile for performance analysis. In *Ecole d'été "MDA for Embedded System"*, September 2002.

[12] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions On software Engineering*, 27(1), January 2001.

[13] TNI-Valiosys. Reqtify.

[14] A. von Knethen and B. Paech. A survey on tracing approaches in practice. Technical report, Fraunhofer Institut Experimentelles Software Engineering, January 2002.