Towards a Metamodel Design Methodology

Experiences from a model transformation metamodel design

Magalhaães, A.P.; Maciel, R.S.P.; Andrade, A.
Science Computer Department
Federal University of Bahia
Salvador, Brazil
anapatriciamgalhaes@gmail.com
{ritasuzana, aline}@dcc.ufba.br

Abstract— **Software engineering makes extensive use of models to provide a better understanding of artifacts produced during system development. Models are specified in modeling languages such as UML or using Domain Specific Languages. In this paradigm of development, metamodeling is essential because it is usually used to specify the abstract syntax of these languages. However, the design of metamodels is not a trivial task, it requires expertise in specific domains, language definition and abstraction capabilities. This paper provides a guide for metamodel design towards a metamodel development methodology based on some lessons learned from metamodel design experiences.**

Keywords- metamodel guide; metamodeling design; metamodel methodology

I. Introduction

In software engineering models have been extensively used to provide a better understanding of the artifacts used in system development. A model can be seen as a set of elements that describes a system in a specific purpose [1]. Models are specified conform to modeling languages such as UML or Domain Specific Languages (DSL) [17] and usually the abstract syntax of modeling languages are specified as metamodels. The design of a metamodel requires expertise in metamodeling techniques and knowledge in the domain of the language under construction as well as a good capacity of abstraction [3].

Our research group had been working on many projects that require the definition of metamodels [5][6][7]. In all of these projects we have felt the need for a method to guide us in some issues such as: how to define a metamodel concepts, how to guarantee that a metamodel covers all the desired concepts of the target domain; how to structurally organize the concepts; and how to validate a metamodel.

Some work has been done in Domain Specific Language creation [14][18], about strategies to specify structural aspects of a metamodel [4], and metamodels pattern identification [3][20]. However, most of them do not focus on aspects such as concepts identification and metamodel validation. These aspects are important to guarantee the coverage level of the metamodel when instantiating models. Furthermore, the existing works do not guide developers through the entire development of the metamodel.

This paper presents a proposal to guide developers in metamodel design based on our experiences in developing metamodels. This guide puts together the tasks that our group performed during the development of some metamodels (e.g. how we selected metamodels concepts) and the lessons learned. As these tasks started to be performed in a systemically manner we organized them, step by step, towards a design metamodel methodology. We aim to systematize the tasks involved in metamodel development leveraging the quality of the produced metamodels in terms of coverage of the concept definition, metamodel detailing (e.g. definitions of concept attributes) and organization of these concepts (e.g. use of specializations).

As we have recently designed a metamodel for transformation domain [7], called MMT (MetaModel for Transformation), we used this to explain the proposed guide.

The rest of this paper is organized as follows: section 2 presents the related works; section 3 presents the proposed guide using the design of a transformation metamodel as an example; section 4 presents the validation of the proposed guide; and section 5 presents our conclusions.

II. Related Works

Nowadays, there are several approaches to help in metamodel design. These approaches can be divided into structural approaches and validation approaches.

In [4] the author gives guidelines for designing metamodels focusing on structural modeling aspects. These guidelines comprise rules to better organize the domain concepts (e.g. how to specialize concepts with similar attributes and associations). In the same vein [3] [20] propose design patterns for metamodels. The authors analyze many different metamodels and identify recurrent problems in the metamodel structures, for example different concepts with the same attributes or relationships. Patterns are suggested to solve these problems e.g. the use of concept generalizations or specializations. When developing a metamodel, developers may use patterns to structure the domain concepts.

In [15] the authors propose a methodology for developing metamodels focusing on simulation based on mathematical statistics techniques. Therefore, this work has a different field of study than ours whose principal objective is the definition of metamodel constructors.

(DOI reference number:10.18293/SEKE2015-054)

The work proposed in [26] uses Test-Driven Development (TDD) to define and validate metamodels. It represents the requirements of a metamodel as models and uses these models as test cases to perform validations. From the outcome of these validations it incrementally defines the metamodel. Differently, we capture metamodel concepts through examples of models in the referred domain and from comparison of metamodel concepts to existing theories (e.g. taxonomy). Our validation assesses metamodel expressiveness through instantiation of models.

In [27] the authors use elements of generic programming to give solutions for the specification of metamodels concerned to reuse and modularization (e.g. it uses *templates* to define patterns and libraries). In a different direction, our work focuses on the definition of a guide do develop metamodels based on traditional software development life cycle.

There are works focusing on the creation of Domain Specific Language. The book [14] lists many definitions of language, grammar, syntax and semantics, how to implement a parser, what a semantic model is and other aspects related to language creation. Similarly, [18] proposes guidelines for DSLs creation related to concrete syntax (e.g. language representation using textual or graphic notation, redundancies control, and so on). In [16] the authors criticize the use of languages such as MOF on metamodel creation due to the time consumed on development and propose a DSL to design metamodels; and [19] proposes the systematic use of examples to increase quality in domain knowledge definition.

Therefore, these works usually focus on specific aspects of metamodel design and do not provide an integrated solution that covers the definition of metamodel concepts, structural design and validation. Besides this, none of these works provide a guide for developers on metamodel design tasks. Our work aims to cover the development of metamodels from concepts definition to validation. Furthermore, some of these works can be integrated to our proposal as part of some tasks (e.g. we used the guidelines proposed by [4] to better organize the metamodel structural aspects).

III. Metamodel Design Guide

In the absence of a methodology that focuses on metamodel design we began to define metamodels in our laboratory in an ad hoc way generating releases incrementally. However, after some development iterations we observed that the tasks performed during the metamodel definition were almost the same. As a result, we started executing them systematically. We organized these tasks as a guide (specified using SPEM 2.0 metamodel) to help in metamodel development. An overview of this guide is shown in Fig. 1, it comprises three phases: (i) *Conceptual Modeling*; (ii) *Design*; and (iii) *Validation*. Each phase can be executed in many cycles of iteration performing a set of tasks.

Fig. 2 shows a work flow with the tasks of the *Conceptual Modeling* phase: initially the *Domain Knowledge* and *Concepts Identification* tasks are executed to select the relevant concepts in order to initiate the metamodel definition (*Create Metamodel* task). Then, this metamodel can be compared to an existing theory (*Theory Comparison* task) and might be reviewed many times (*Metamodel Review* task) until the definition of its first release. When necessary it is also possible to return to *Domain Knowledge* task to get some more examples.

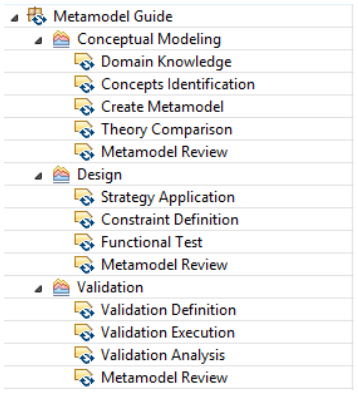
- Metamodel Guide
  - Conceptual Modeling
    - Domain Knowledge
    - Concepts Identification
    - Create Metamodel
    - Theory Comparison
    - Metamodel Review
  - Design
    - Strategy Application
    - Constraint Definition
    - Functional Test
    - Metamodel Review
  - Validation
    - Validation Definition
    - Validation Execution
    - Validation Analysis
    - Metamodel Review

Figure 1. Phases and tasks of the Metamodel development guide.

According to SPEM, a task can be performed in a set of steps and may consume / produce work products. Besides this, roles are responsible for the tasks. For each task of the guide we specified all of its elements (steps, input and output work products and roles). For example, the *Domain Knowledge* task comprises two steps that are performed by the *Domain Specialist.* This task generates a *list of sources of knowledge* (e.g. languages and examples of diagrams from the application domain) as output that will be used in the next task.

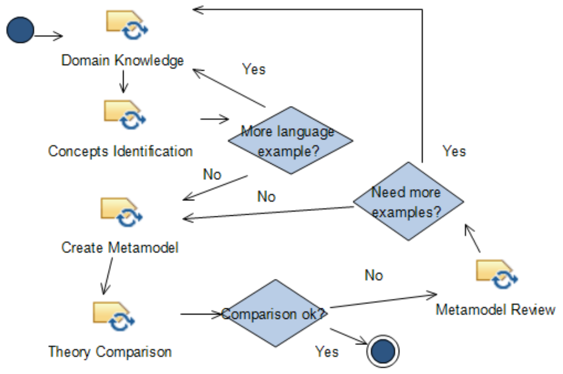

Figure 2. Conceptual Modeling workflow

This guide may be used in several domains. As our laboratory works with Model Driven Development [2] and model transformations, we used the design of a metamodel for the transformation domain as an example to guide explanation. In section 3(A) we briefly introduce the transformation domain and then in the following subsections we detail each one of the guide phases and tasks.

A. *Designing a Metamodel for Transformations Domain*

Model Driven Development (MDD) is a software development approach that makes intensive use of models instead of code. In MDD models are developed at a high abstraction level and transformed through a transformation chain until code. At the core of MDD is the transformation chain which encapsulates the mapping strategies to transform input into output models. The transformation chain comprises a set of transformations responsible for automating/semi automating the MDD software development process [2].

Transformations receive models as input and generate models or texts as output [8]. Input and output models should conform to metamodels. The design of a transformation requires the definition of the relationships among elements of the source metamodel to elements of the target metamodel. A transformation itself may be specified as a model, called a model transformation model [21], which also should conform to a metamodel. In this scenario metamodels are necessary to: model the input and output models; develop the transformation chain (the relationships between source and target metamodel elements); and to design the model transformation metamodel.

In this paper we show the design of the Metamodel for Model Transformation (MMT) to illustrate our guide tasks. MMT is defined to support the development of model transformations at a high abstraction level. It comprises the necessary concepts for transformation specification and design independent of platform through a MDD approach to develop model transformations. So transformations code can be generated from the specification of transformation models.

B. *Conceptual Modeling Phase*

The main goal of the first phase of the guide, *Conceptual Modeling*, is to identify the relevant concepts of the domain. The result of this phase is the preliminary release of the metamodel. It consists of five tasks: *Domain Knowledge*; *Concepts Identification*; *Create Metamodel*; *Theory Comparison*; and *Metamodel Review*.

The first task (*Domain Knowledge*) consists of learning about the domain. Similar to the strategy used in [3] to identify metamodeling candidates for patterns, the most popular languages or some examples of applications designed in the domain should be selected.

Considering our example, the design of the transformation metamodel MMT, the languages initially selected were QVT (query / view / transformation) [10], because this is the OMG standard to design model transformations and ATL (Atlas Transformation Language) [11] due to its wide use in MDD projects to develop transformations.

In the second task, *Concepts Identification*, we should analyze the selected languages / application examples to identify the commonalities and specificities of the domain. The common concepts are then selected to be used in the construction of the metamodel. In the design of the MMT metamodel we had analyzed the constructors of the ATL and QVT (Relation) languages to find their commonalities and specificities. For example, in ATL a transformation is a *Module* comprised of *Rules*. There is one kind of rule, named *Matched Rule*, which is automatically executed when a source element matches a target element. Similarly, in QVT a *Transformation* comprises *Rules* that are specialized in *Relational Rules* for declarative definitions. The *Relational Rule* can be defined as a *Top Relation* to indicate that it must hold in order to be executed. Comparing the concept of transformation in these two languages, in MMT we defined both the *Transformation* and the *Relation* concepts and for the *Relation* we added an attribute (*isRequired*) that indicates when the *Relation* must hold in a transformation execution.

In the third task, *Create Metamodel*, the previously selected concepts were organized as classes and their associations, generating the initial release of the metamodel. Attributes are also identified for the concepts.

The following task, *Theory Comparison*, consists of analyzing transformation theoretical concepts and comparing them to the concepts used in the initial release of the metamodel. Different theoretical approaches can be used in comparison, such as taxonomies and ontologies.

In the design of MMT we used the taxonomy presented in [9] as a reference to perform the comparison. This taxonomy classifies the concepts of transformation domain and its purpose is to address the essential characteristics of model transformations and existing languages and tools. Table 1 illustrates part of the comparison done.

TABLE I. PART OF THE TAXONOMY COMPARISON

Taxonomy [9]	Representation in MMT
Transformation type (Model transformation or Program transformation)	*Transformation* was specialized in: *M2M Transformation* for the model transformation type; *M2T Transformation* for the program transformation type
Endogenous x Exogenous transformation	Endogenous transformation: use the same metamodel on *SourceModel* and *TargetModel* associations; Exogenous transformation: *SourceModel* and *TargetModel* are different metamodels
Reuse (generic reuse, HOT, grouping, composition, decomposition)	Transformations are composed of other transformations (auto association on *M2M transformation*). It is possible to reuse existing transformation (combining them) to build new ones. It also allows the use of high order transformation (HOT)

The first column lists the taxonomy concepts and the second lists how MMT interprets these concepts. Cells in gray emphasize the concepts of MMT that were modified in order to suit the taxonomy. For example, in MMT the *Transformation* concept was specialized as *M2M Transformation* and *M2T Transformation* to support the two kinds of transformation modeling present in the taxonomy.

After the comparison, the *Metamodel Review task* was performed and some modifications were done in the metamodel. In our example, MMT, an association was added to the *Transformation* concept allowing transformation composition and reuse and the *Transformation* concept was specialized in *M2M Transformation* and *M2T Transformation* as partially shown in Fig. 3.

C. *Metamodel Design Phase*

The main goal of this phase is to organize the concepts defined for the initial structure of the metamodel. This phase comprises four tasks: *Structural Design*; *Constraint Definition*; *Functional Test*; and *metamodel Review*. A detailed release of the metamodel should be generated at the end of this phase.

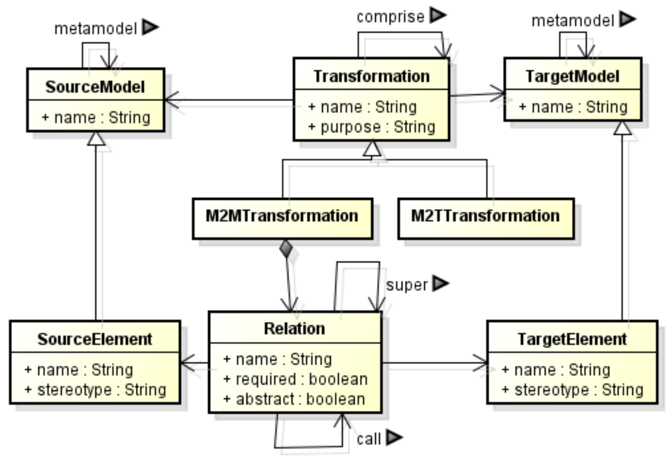

Figure 3. Part of MMT after conceptual modeling phase

The first task consists of the structural organization of the metamodel. Many kinds of strategies can be applied in order to structure the metamodel concepts, such as the use of packages to aggregate reusable concepts, the definition of general concepts to represent common attributes, etc. We recommend the adoption of the strategies proposed by [4]. These strategies guide metamodel developers in terms of: definition of packages to enable reuse of concepts; specification of association, e.g. how to define association member end features; specification of common attributes; how to create generalizations; definition of default values; when to use enumeration; and so on.

For the MMT metamodel many strategies from [4] were used. For example, we first used the strategy *Adding Abstraction Package* to group the constructors into two packages separating them in abstraction levels *MMTSpec* and *MMTDesign*. Then we used the strategies *Abstracting Common Attributes*, *Abstracting Common Associations* and *Generalizing Common Attributes* to identify constructors with the same attributes and/or associations and create a generalization for these common definitions (e.g. the *Model* concept was created to generalize *sourceModel* and *targetModel*), we defined the association end names and defined enumerations (Fig.4).

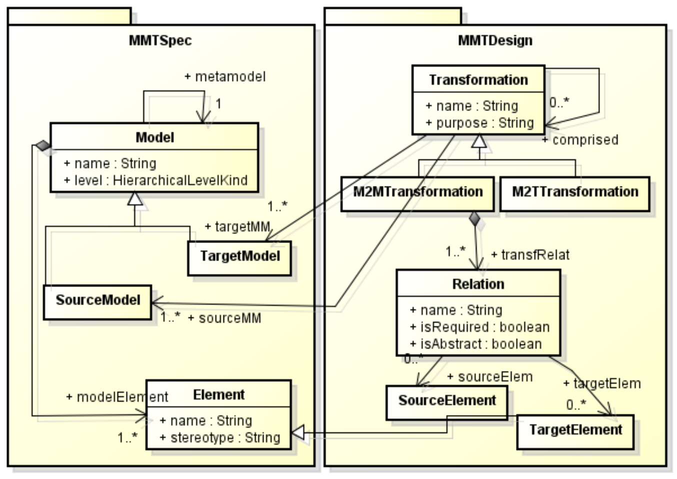

Figure 4. Part of MMT metamodel after Metamodel Design Phase

With the metamodel concepts defined and well-structured the next task, *Constraints Definition*, consists of the specification of the metamodel constraints using OCL (object Constraint Language). For the MMT we defined constraints to specify model / metamodel conformance.

Although we had defined the concepts based on the available theory and have organized these concepts applying structural techniques, we had never used MMT to model a transformation yet. So, the last task of this phase, named *Functional Test*, consist in specify an instance of the metamodel and evaluate the effective use of the defined concepts and relationships in the instanced metamodel.

For the MMT *Functional Test* we instantiated the OO2RDBMS transformation which receives a class model as input and generates a logical data base model as output. Besides this we use UML diagrams stereotyped by MMT to visually model the transformation. The adoption of UML was a decision based on some premises: UML diagrams are well known by system developers; UML has a large number of tools to support the development tasks. The complete specification can be seen in [7].

After the functional test we observed that MMT concepts were almost sufficient to model the transformation. However, we observed a deficiency in the low level design of this transformation because we specified which elements of the source metamodel were transformed into elements of the target metamodel but we did not specify how this transformation should be done. As a result the metamodel was modified again to introduce the necessary elements for the lower level specification (*Metamodel Review* task).

D. Metamodel Validation Phase

The main goal of this phase is to evaluate the metamodel expressiveness in terms of coverage of the defined concepts. It comprises four tasks: *Validation Definition*; *Validation Execution*; *Validation Analysis* and *Metamodel Review*. The first task of this phase, *Validation Definition*, starts with the goal and the definition of the research questions. Any guideline related to software engineering experimentation can be used in this task, such as the guidelines presented in [23]. In the second task, the validation is performed (*Validation Execution task*) and according to the results (*Validation Analysis*) the metamodel can be modified (*Metamodel Review task*).

Regarding the design of the MMT metamodel we used a GQM template [22] (Fig. 5) to summarize goal definition and defined the following questions: (Q1) Do the MMT constructors sufficiently specify transformations written in ATL/QVT? (Q2) Is it necessary to add new constructors in MMT to enable the transformation specification written in ATL/QVT?

Analyze the MMT constructors
For the purpose of evaluating the metamodel expressiveness
With respect to coverage level
From the perspective of transformation developers
In the context of existing transformation developed in ATL/QVT languages

Figure 5. Experimental goal according to GQM template

The validation of MMT was executed over five months. During this period seven transformations that had already been developed in ATL / QVT language were specified using MMT. The transformations were selected from web repositories such as [24]. We performed the experiment in two stages (an initial test and the main experiment) where some dependent variables were measured, such as *specification completeness* and the *amount of used / new constructors*. After the validation we were able to conclude that MMT concepts covered most of the transformation specification, although some points for improvement were identified. For example, we observed that MMT could implement the concept of *transformation design pattern* proposed by [13] in order to simplify the specification.

Considering the results obtained, we performed the third task of the *Validation* phase, *Metamodel Review* and modified the metamodel (e.g, we added the *Pattern* concept in the MMTLowDesign).

During *Validation* phase, developers might observe the necessity of new attributes, associations or even new concepts that should be added to the metamodel. Therefore, the *Validatio*n phase can be done iteratively, instantiating the metamodel in different models, until developers observe that the amount of modifications decreases considerably. At this point the metamodel is considered stable enough for use.

IV. Metamodel Design Guide Validation

Methods and processes for validation which involve humans are very challenging and they should be carried out in phases. Each phase should be an evolution from the previous one. So we first decided to evaluate the feasibility of the guide in driving developers in metamodels design. We followed the guidelines for software engineering experimentation presented in [23] and used GQM template [22] to define the goal of the experiment (Fig. 6).

Analyze feasibility of using the design metamodel guide
For the purpose of improving metamodel development
With respect to metamodel coverage and completeness
From the perspective of metamodel developers
In the context of model driven development

Figure 6. Goal definition according to GQM template

To reach our goal the following questions were defined: Q1: Does the guide help developers in metamodel definition? Q2: Does the guide improve the quality of the produced metamodel? In relation to the quality, we checked the coverage level of the defined concepts and the metamodel completeness. Accordingly, the following *null*/**alternative** hypotheses were formulated.

$H1_0$/**H1**: The use of the guide [*does not impact*]$_0$/[**decreases**] the participants difficulty in producing metamodels.

$H2_0$/**H2**: The use of the guide [*does not impact*]$_0$/[**improves**] the coverage of the concepts of the metamodels created by developers.

$H3_0$/**H3**: The use of the guide [*does not impact*]$_0$/[**improves**] the completeness of the class diagram created to represent the metamodel.

To evaluate the proposed guide we performed a controlled experiment to verify whether the guide improves metamodel development in building metamodels. Therefore we defined one independent variable, the *modeling method*, used to create the metamodel which varies across two groups: the *control group*, which developed metamodel in ad hoc way; and the *guide group*, which developed metamodel using our guide. The dependent variables are the *metamodel coverage*, the *metamodel completeness* and the *perceived difficulty* in metamodel development.

The experiment was performed over the period of 4 months divided in two phases: Initially a pilot study was performed and then the main experiment. The participants were undergraduate students (four students in the pilot study) and master degree students (twelve students in the main experiment). All of them had knowledge in MDD and process specification using languages such as SPEM. None of them had knowledge in metamodel design. Students were arranged in two groups according to the design method: the *control group* and the *guide group*. The data presented in the rest of this section relates to the main experiment.

As we had already developed a metamodel for the definition of MDD processes in our laboratory [12] we used this work as the problem domain in this experiment. To perform the experiment students received some examples of MDD processes and the SPEM specification. For the guide group we also gave the proposed guide and asked them to follow it. The experiment comprises three activities (related to our guide phases): *define metamodel concepts*, *specify metamodel structure*, *validate metamodel*. Students spent three days (one day for each activity) to design the metamodel, and each activity started for all students at the same time. We gave a maximum of two hours a day to perform each activity and measure how much time each student spent completing each task. At the end of the third day the students were supposed to have a metamodel representing MDD Process domain and an instantiated model conformed to this metamodel. Finally, participants answered a questionnaire that collected their perceptions about the difficulty of performing the tasks.

The produced metamodels were analyzed by our research group and compared to the reference metamodel, the metamodel that we had already developed for this domain. This comparison was based on a previously defined template and aims to evaluate metamodel coverage and completeness. The metamodel coverage was defined considering the amount of concepts identified by developers for the domain and attributes of each concept. The metamodel completeness was defined considering the amount of structural aspects specified in metamodels. We analyze the use of specializations, associations between concepts, the use of patterns to name variables, the use of enumerations and so on.

For the first activity, *define metamodel concepts*, we observed that the *control group* (who designed the metamodel in an ad hoc way) identified 66% of the concepts and the *guide group* (who used our guide) identified 87%. Nevertheless, the second group did it in greater detail (e.g. they also defined the attributes of each concept) so that the coverage rate was higher for the group that used our guide. For the second activity, *specify metamodel structure*, we observed a big difference between the metamodels produced by the two groups. In the first group we noticed the absence of *specializations*, *enumerations* and *associations end role* definitions which make these metamodels more verbose and difficult to understand. As a result the completeness rate for the *control group* was 45% while for the *guide group* it was 81%. The third activity, *validate metamodel*, was not so helpful for our evaluation because all the participants could instantiate a model conforms to their metamodel (it differs on the level of detail of each instance according to the coverage/completeness of the metamodel). Analyzing the perceiving difficulty reported by the participants we noticed that metamodel design is still a challenge due to the high level of abstraction needed in definitions. We did not observed any big difference between the two groups related to the time spent by each participant on performing the three tasks in the experiment. However we observed that the metamodel guide led to metamodels with a high level of coverage and completeness.

To decrease any threat to validity some strategies were adopted. With regard to participant knowledge we checked that they were familiar with the MDD approach (and metamodeling) as they were students doing an MDD course or they had already participated in MDD projects. We used randomization to assign participants to each group and prevented communication between them. As far as the tasks were concerned we gave all the participants the same amount of time to perform them. Besides this the domain might be another threat so we tried to choose a domain familiar to software engineers.

Empirical assessment usually takes into account the amount of data collected from the subjects. However, in the case of a guide validation it is difficult to involve a great number of people in case studies. A case study rather than a rigorous experiment was the most suitable choice. We know that the study results are limited and do not provide statistical evidence to support general conclusions. However, we believe that it can be considered an initial step in future case studies to be performed in order to observe other aspects.

V. CONCLUSION

This paper reported our experience in metamodel design through the definition of the MMT metamodel. From this experience and some expertise in the domain of MDD and software engineering we defined a guide for metamodel developers that we believe can be used as a base for a metamodel development methodology.

Two main difficulties were encountered in this work: the metamodel conceptual modeling and the validation. To address the first one we used a taxonomy and specific languages (e.g. ATL) to identify the relevant concepts of the transformation domain. Validating a metamodel is quite different from validating a piece of software. Metamodel validation requires instantiation examples. As a result alternatives were used in validation (e.g. reverse engineering techniques) until we considered the metamodel was stable enough.

We believe that the guide to develop metamodels generated by our experience can be adapted and evolved to be used in the design of other kinds of metamodels other than transformation domains. We are now working on this generalization and on case study scenarios to achieve this goal.

REFERENCES

[1] S. Mellor, A. Clark, T. Futagami "Model Driven Development" IEEE Software,2003

[2] T. Stahl, M. Volter, J. Bettin, A. Haase, S. Helsen foreword by K. Czarnecki "Model-Driven Software Development" Wiley, 2006.

[3] H. Cho, J. Gray "Design Patterns for Metamodels" SPLASH´11 workshops, Portland, Oregon, USA, october, 2011.

[4] A. Vieira, F. Ramalho "Identifying Guidelines for Constructing Metamodels" In: III Brazilian Workshop on Model-Driven Software Development, Natal, 2012.

[5] R.S.P. Maciel, R.A. Gomes, A.P. Magalhaes, B. Silva "Supporting model-driven development using a process-centered software engineering environment." ASE, v1, p1, 2013.

[6] A.P. Magalhaes, J. David, R.S.P. Maciel "Modden: an integrated approach for Model Driven Development and Software Product Line Processes" in 5th SBCAR, São Paulo, 2011.

[7] A.P. Magalhaes, A. Andrade, R.S.P Maciel "MTP : Model Transformation Profile" In : 7th SBCARS, Brasilia, 2013.

[8] M. Brambilla, J. Cabot, M. Wimmer "Model-Driven Software Engineering in Practice" Morgan & Claypool Publishers, 2012.

[9] T. Mens, K. Czarnecki, P. Van Gorp "A Taxonomy of Model Transformation" Dagstuhl Seminar Proceedings 04101, 2005.

[10] QVT specification - http://www.omg.org/spec/QVT/1.0/PDF/

[11] ATL Project - http://www.eclipse.org/m2m/atl/

[12] Maciel, R. S. P. ; Magalhães, A. P. ; "An Integrated Approach for Model Driven Process Modeling and Enactment." In: SBES , Fortaleza, Brazil 2009

[13] Iacob, M.; Steen, M.;Heerink, L. "Reusable Model Transformation Pattern." In 3M4EC´08, pages 1-10, 2008.

[14] Fowler, M. "Domain Specific Languages." Addisson Wesley, 2011.

[15] Kleijnen, J.; Sargent,R. "A methodology for fitting and validating metamodels in simulation."European Journal of Operational Research, pp. 14-29, 2000.

[16] Cuadrado, J.;Molina, J. "Building Domain-Specific Languages for Model-Driven Development", Software IEEE 24.5, pp.48-55, 2007.

[17] Luoma, J.;Kelly, S.;Tolvanen, J. "Defining Domain-specific modeling languages: collect experiences." 4th Workshop on DSM, 2004.

[18] Karsai, G.Krahn, H.; Pinkernell, C.; Rumpe, B.;Shindler, M.; Volkel, S. "Design Guidelines for Domain Specific Languages."In proceedings of Domain-Specific Modeling, 2009

[19] Bak, K.; Zayan, D.;Czarnecki, K.; Wasowski, A.; Rayside, D. "Example-Driven Modeling. Model=Abstraction+Example." ICSE, 2013, San Francisco, USA.

[20] Mernik, M.Sloane, A. "When and how to develop Domain Specific Languages." ACM Computing Surveys, Vol 37, Nr.4, pp.316-344, 2005.

[21] Bézivin, Jean et al. "Model Transformations? Transformation Models?" Springer-Verlag Berlin Heidelberg, 2006

[22] Solingen, R. Basili, V.;Caldiera,G.; Rombach, H.D. Goal Question Metric (GQM) Approach. John Wiley & Sons. Inc., 2002.

[23] Wohlin, C. Aurum, A. Towards a decision-making structure for selecting a research design in empirical software Engineering. Empir Software Eng DOI 10.1007/s 10664-014-9319-7. Springer, 2014

[24] SimpleGT, available in "http://soft.vub.ac.be/viewvc/SimpleGT/

[25] Mellor,S.; Clark, A.; Futagami, T. "Model Driven Development" IEEE Software,2003

[26] Cicchetti A.; Ruscio, D.;Kolovos, D.S.; Pierantonio, A. A Test-driven approach for metamodel development. Chapter of the book Emerging Technologies for Evolution and Maintenance of Software Models, p. 319-342, IGI Global, 2012.

[27] Lara, J.; Guerra, E. Generic meta-modelling concepts, templates and mixin layers. Models, 2010.