Automating the Construction of Models based on Domain Views

César Cuevas Cuesta, Patricia López Martínez and José M. Drake
Group of Software Engineering and Real-Time, University of Cantabria, Santander, Spain

Keywords: MDSE, Meta-model, View, Meta-tool, HOT.

Abstract: This work addresses the automatic generation of the resources required for the assisted creation of domain models according to specialized views of their meta-model. The task of a designer who builds models compliant to a complex domain meta-model is eased if the model editor requests the information according to a specific view of the meta-model based on the conceptualization or the specific construction strategy that the designer uses. With that aim, this work presents 1) the meta-model with which a domain expert formulates the model creation strategy that he envisions, 2) the tool that, from that strategy information, generates the meta-model that drives data introduction and 3) the M2M transformation that generates the final model compliant to the original domain meta-model and that contains the newly introduced data.

1 INTRODUCTION

Formalizing very general application domains by means of meta-modelling requires specifying complex meta-models, with a great deal of details and options. Simplifying those domain meta-models by decomposing them in other partial and simpler ones may not be recommended or allowed if their goal is to cover the modelling of heterogeneous systems with different specific features, so that common processing techniques and tools can be applied on all of them. This is the case, for example, of some OMG meta-models, like SysML (OMG, 2012) or MARTE (OMG, 2011), which cover a so large set of options and specific features that hardly any designer uses in its entirety. For a designer working in this context, the domain meta-model complexity can be an unnecessary problem, since:

- He may not be an expert in the whole domain, but only in some partial aspects.
- He may use a limited set of design patterns or execution platforms.
- He may develop tools for processing models corresponding only to systems with restricted features among the domain variability.
- He may be interested only in a partial vision of the information represented in the models.

According to these premises, the task of the designer is eased if he is provided with adapted tools that offer a view of the information limited and focused on his interests. A first approach may be working with simpler subdomain meta-models that only address those aspects of interest. This kind of solution must be complemented in order for the created models to be compliant to the original meta-model and compatible with legacy tools. In some trivial cases, the meta-models supporting those specific features can be a mere subset of the domain meta-model, in which case the compliance is granted. However, in general, the subdomain meta-model may have structural differences with respect to the original one, so that the created models will not be directly compliant to the latter.

Another option for addressing the delimitation of a domain is formalizing it through a view specification for each specific case. A model whose structure satisfies a view specification is referred to as according to that view. Fig. 1 depicts this idea by showing in the upper part a domain meta-model on which a domain expert has specified two views and in the lower part three compliant models.

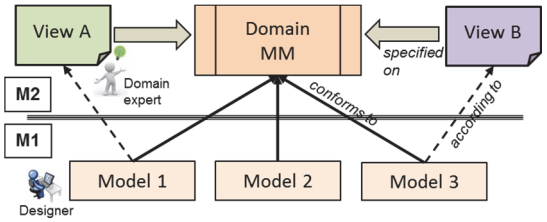

Figure 1: View specification on a domain meta-model.

Cuesta, C., Martínez, P. and Drake, J.
Automating the Construction of Models based on Domain Views.
DOI: 10.5220/0005688202410249
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 241-249
ISBN: 978-989-758-168-7

The compliant models can be according to View A (Model 1) or to View B (Model 3) or to none of them (like Model 2, that is simply compliant to the meta-model). If any kind of intersection is feasible, models according to both views would be perfectly possible.

The aim of this work has been to contribute to the development of tools for supporting models compliant to meta-models on which views have been specified. Specifically, it presents a strategy for easing the construction of models according to a view specified on a meta-model, as well as a generic tool that implements that strategy. The tool is generic, in the sense that it is applicable to any domain meta-model and to any view specified on it (following the view characterization exposed in section 3.3). Its generic nature is achieved by means of its conception as a meta-tool, i.e., it operates generating on-demand the specific tool implementing the building strategy for every specific meta-model/view couple. Fig. 2 outlines this approach of meta-tool. Given a domain meta-model, a view specified on it represents the meta-tool input, which generates the specific construction tool suitable for the view.

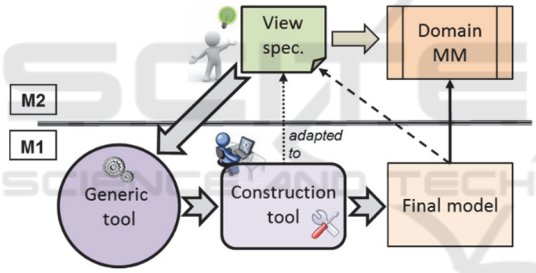

Figure 2: Generic tool as a meta-tool.

The initial motivation of this work was to support views in the MAST environment (González et al.,). MAST addresses the design and analysis of Distributed Real-Time Embedded Systems (DRES) with a very broad generality (heterogeneous distributed platforms with different operating systems, complex communication networks and support for software designed using different architectures and paradigms). For example, when a team is working with a certain RT-Linux platform and writes the code in the Ada language, the designers can use a subset of only 17 classes among the 143 classes of the MAST meta-model.

The paper is structured as follows. Section 2 addresses some related work in the literature. Section 3 characterizes the view concept adopted in this work, and presents the strategy designed for easing the construction of models according to a given view. Moreover, the section includes a proposal of a meta-model for view modelling. Section 4 exposes the generic tool implementing the strategy. Last, Section 5 draws some conclusions and future work directions.

2 RELATED WORK

The view concept has a long tradition in the databases area (Wiederhold, 1991). A number of contributions can be found in the MDSE literature proposing developments based on adopting and adapting this concept to the Modelling area. First of all, works that exhibit as motivation the problem of MDD processes that use several meta-models and, consequently, have to handle information disseminated in heterogeneous interrelated models. Recent proposals in this direction are the EMF-Views (Bruneliere et al., 2015) and Flexible Views (Burger, 2013) methodologies, which, in order to reduce the complexity, propose to manage the models by means of partial specialized views that select and/or aggregate the information of such heterogeneous models. The main difference with our proposal is that, while those approaches are oriented to the creation of views to be applied for visualization based on filtering and/or combining heterogeneous model elements already existent, our methodology is oriented to the construction of the models according to a strategy driven by the view itself. More close to this approach are the proposals in (Cicchetti et al., 2012) and (Bork et al., 2014), although in them the views are merely portions of the domain meta-model, so that the strategy for model construction is not specifically defined by the view specification.

3 MODEL MANAGEMENT IN THE PRESENCE OF VIEWS

3.1 Alternatives for the Creation of Models According to a View

Given a view specified on a certain domain meta-model, there are two alternatives that can be adopted for creating models according to it.

3.1.1 Use of Tools Driven by the Domain Meta-model

The models according to a view are, by definition, compliant to the domain meta-model on which the view is specified, so that they can be managed using

tools based on it. This strategy is shown in Fig. 3, where, as indicated, the model construction process is driven by the domain meta-model, being the designer the responsible of following the guidelines stipulated by the view specification. A model built this way is directly compliant to the meta-model and according to the view. The use of this strategy has the drawback of forcing the designer to work against the complexity of the whole meta-model, without any kind of specific assistance relative to the view, so the process is error-prone. In addition, there are no available tools for verifying that the models under construction are according to the view.

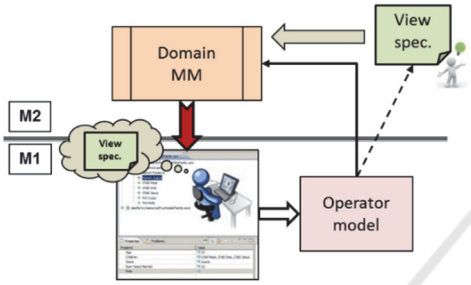

Figure 3: Construction driven by the domain meta-model.

3.1.2 Use of Tools Driven by the View

This alternative works on a meta-model that describes the information specific to the view, and, based on it, the designer builds the models according to it. The main advantage of this strategy is that, during model construction, the designer only confronts the view complexity. As a drawback, the resultant model in not necessarily compliant to the domain meta-model and it requires to be transformed to a compliant one. This is the alternative chosen as the basis of the strategy proposed in this paper. It addresses the automatic generation of the aforementioned meta-model specific to the view and the transformation tool that produces the final model, compliant to the domain meta-model (and according to the view). The details of the strategy are explained in the rest of the section.

3.2 Automating the Creation of Models According to a View

The proposed strategy prioritizes easing the creation of models according to a view. The supporting meta-model focuses on the view and on the expert's conceptualization. It has been called View Required Data (VRD) meta-model, because its goal is to assist the designer during the creation of models according to the view and not to formalize the constraints that the view introduces in the domain meta-model. Therefore, the designer builds models compliant to the VRD meta-model corresponding to the view at hand and a subsequent M2M transformation generates the final model, compliant to the initial meta-model and according to the view. Fig. 4 outlines the strategy.

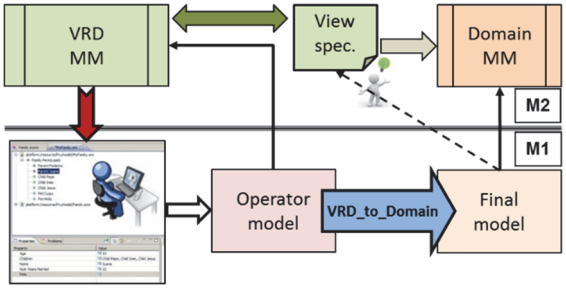

Figure 4: Construction driven by a VRD meta-model.

According to the contents exposed so far, given a meta-model on which a view has been specified, the assistance to the designer consists in generating two complementary components:

- The VRD meta-model on which the data introduction is based.
- The M2M transformation that generates the final model from the designer's model.

The aim of this work is the automatic generation of both components from the view specification and from the domain meta-model itself. For achieving it, an MDE strategy is used, where both components are generated as outputs of M2M transformations that have as inputs the view specification in model form plus the domain meta-model. Before the in-depth study of those transformations, which will be the goal of Section 4, the following subsections describe, respectively, the scope of the view concept in the context of this work and the meta-model supporting the formulation of views as models.

3.3 Scope of the View Concept

According to the aim of this work, a view can establish the following aspects on the data structure described by a meta-model:

- The specification of the meta-model classes allowed in the models according to the view.
- The description, partial or complete, of how the instances of those permitted classes must be (regarding attribute values, null references, etc.). This characterization of an allowed class is what we have called a *category* relative to it.
- The instances of the allowed classes and according to the defined categories that must

appear in the models.

- The definition of assemblies, i.e. groups of types (among the allowed ones) that are to be instantiated together, each of them in a specific number and with pre-established references between such instances.
- The specification of which instances of the assemblies must appear in the models.

3.3.1 The LinuxClassicRMA View

As an example, let's consider a company developing real-time software targeting equipment that uses a certain RT-Linux platform. In addition, given the nature of the required software, the Classic RMA schedulability analysis technique (Lehoczky et al., 1989) is applied. In this case, the domain meta-model is MAST 2.0, which allows to model the temporal behaviour of a very broad range of DRES. MAST 2.0 encompasses 143 classes and its complexity is totally justified, as it has to cover the models on which the set of tools of the MAST environment are to be applied (tools for schedulability analysis, priority assignment, slack calculation, etc.). The LinuxClassicRMA (LCRMA) view is defined on this meta-model, delimiting it to the case of the software developed by the company, due to the used platform and analysis technique.

Constraints Related to the Execution Platform. The fact that the processor is a certain RT-Linux platform drastically delimits the model of the system execution platform, which only contains:

- One processor element, with all its attributes taking a fixed value.
- One scheduler, associated to the unique processor of the platform. Its scheduling policy is fixed priority (FP) with a priority range of [0, 100]. Hence, every thread defined in a model must have scheduling parameters of FP kind and is scheduled by this unique scheduler.
- One clock, associated to the platform processor.
- One type of synchronization elements that can be used by the threads: shared resources with immediate ceiling protocol.

Constraints Related to the Nature of the Developed Software. The reactivity of the applications is defined as a set of tasks, where each of them:

- Has a periodic activation triggered by the system clock.
- Is executed in its own thread.
- Can lock shared resources when starting the execution, unlocking them when finishing it.
- Can have a hard deadline, associated to the end of its execution and relative to the activation.

According with the previous considerations, Fig. 5 shows a non-formal object diagram with a MAST 2.0 sample model according to the LCRMA view. Fig. 6 shows a high-level, reactive vision.

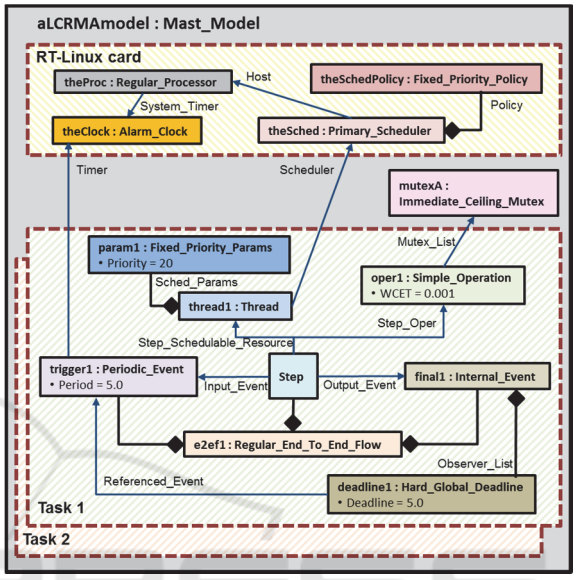

Figure 5: LCRMA sample model.

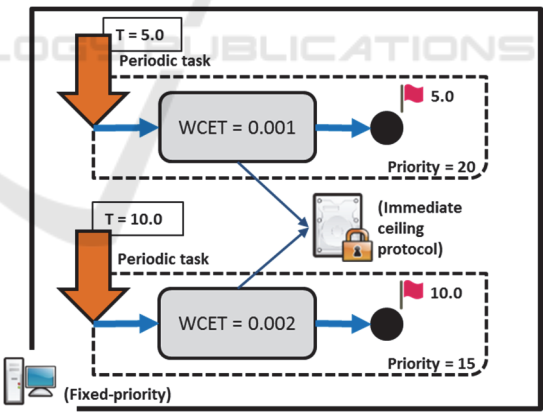

Figure 6: Reactive vision of the LCRMA sample model.

The aim of the LCRMA view is to alleviate the designer who wants to analyse the schedulability and to assign priorities to a new application. Like for any other view, it is based on the meta-model whose structure is to be constrained (MAST 2.0) and specifies those classes allowed to be instantiated. The specification also defines the categories use to formalize the restrictive conditions imposed on the instances of those allowed classes and declares the

elements that must exist in every model according to the view.

3.3.2 The LCRMA_VRD Meta-model

The class diagram in Fig. 7 represents the VRD meta-model corresponding to the LCRMA view.

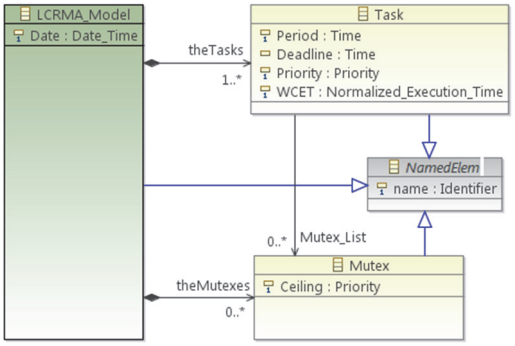

Figure 7: LCRMA_VRD meta-model.

The designer that confronts the task of creating LCRMA models has to build models simply compliant to this reduced meta-model. These models will be later transformed to MAST 2.0 models (according to LCRMA) by means of the LCRMA_VRD_to_MAST2 transformation (particularization of the VRD_to_Domain transformation shown in Fig. 4).

Fig. 8 shows the model that the designer has to build in order to obtain the model of Fig. 5. The difference of conceptual complexity and size is substantial.

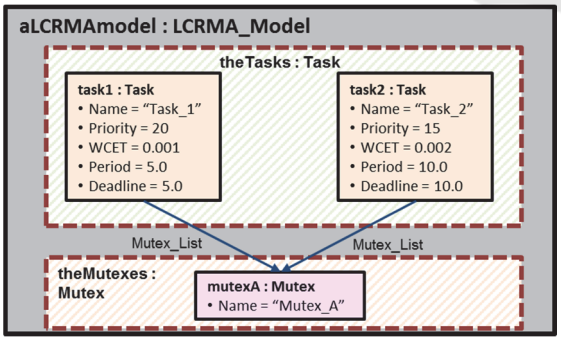

Figure 8: VRD model built by the designer.

3.4 Meta-model for Constraining View Specification

An overall vision of the meta-model for modelling view specifications, the Constraining View Specification (CVS) meta-model, is presented below, using Ecore as meta-modelling language. The class diagram in Fig. 9 shows the meta-model. It exhibits a conventional structure, with the `CVS_Model` class playing the role of main container class. It defines the `referencedMM` association through which the meta-model on which the view is specified is referenced.

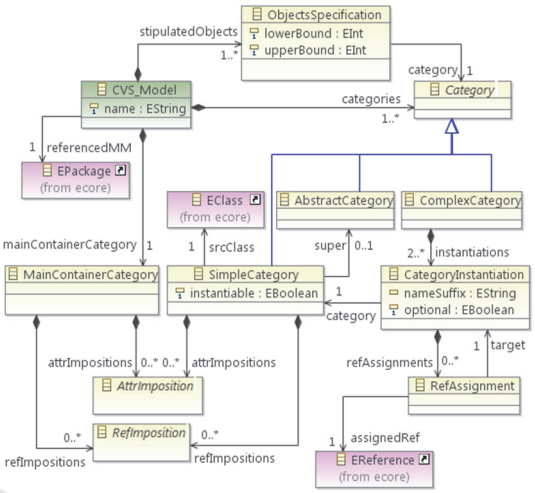

Figure 9: CVS meta-model.

The other basic classes are `Category` and `ObjectsSpecification`. The former is an abstract class that represents the concept of category defined by a view, on a class of the corresponding meta-model or in an assembly form. The latter represents the concept of element (individual or assembly) that must appear in every model according to the view. It defines the attributes `lowerBound` and `upperBound` that describe the multiplicity range of those elements.

The CVS_`Model` class defines two compositions: `categories` and `stipulatedObjects`. Through the first one, the main container of a CVS model contains the description of those categories defined by the view, while through the second one, the description of the mandatory elements. `ObjectsSpecification` defines the association `category` through which such mandatory objects indicate the category, among those ones specified by the view, they must be according to.

3.4.1 Categories

The CVS meta-model defines three subclasses of Category:

- `SimpleCategory`. It represents the concept of basic category defined by a view on a (permitted) class of the corresponding meta-model. This class defines the `srcClass` reference used to refer to the corresponding base class, as well as the `attrImpositions` and `refImpositions` compositions through which a simple category

defines the impositions that the view specifies on the properties of the base class. Moreover, through the `instantiable` attribute, an object declares if it represents an instantiable category, i.e. there can be instances according to it in a model. If false, they can appear only as part of assemblies.

- `AbstractCategory`. This class is introduced aiming to cover the case of simple categories defined on classes sharing superclass.
- `ComplexCategory`. It represents the concept of assembly defined by a view. This class defines the `instantiations` composition for specifying the elements that form the assembly.

The formulation of an assembly constitution involves the classes `CategoryInstantiation` and `RefAssignment`.

- `CategoryInstantiation`. It represents the concept of a particular instantiation of a certain simple category in an assembly. The class defines a `category` association through which its instances indicate the simple category at hand. It also defines the `optional` and `nameSuffix` attributes, for specifying if such an instantiation is optional inside an assembly and for declaring a possible literal suffix. Lastly, the class defines the `refAssignments` composition through which its instances can hold `RefAssignment` objects, in order to cover the fact that in an assembly can exist pre-established links between the components themselves or even between elements of different assemblies.
- `RefAssignment`. It represents a mechanism for setting a reference belonging to an element of an assembly towards another element of the same assembly or another one. The class defines two associations: `assignedRef` and `target`. Through the first one, its instances point to the reference that is to be set and through the second one to the target element on which the link is set.

A CVS model must not specify `SimpleCategory` instances on the domain meta-model main container class. For that purpose, the CVS meta-model presents an additional class, `MainContainerCategory`, that represents the description (partial or complete) about the way the main container of a model according to the view has to be configured. The compositions it defines are analogous to those ones with same name in the `SimpleCategory` class. The `CVS_Model` class defines one more composition, `mainContainer Category`, through which the main container of a CVS model contains the only `MainContainer Category` instance. This explicit and separated definition is a design decision in order to ease the development of the meta-tool exposed in Section 4.

3.4.2 Impositions on Attributes and References

The impositions that a view sets on the properties of a (permitted) class when defining a simple category on it are represented by instances of `AttrImposition` and `RefImposition`. There are two types of impositions on attributes (assignment of a fixed value and imposition of having the same value as other attribute) and three types of impositions on references (specification of a category to which its target must be according to, obligation of being null or imposition of having the same target as another reference). This variety is represented by the `ValueAssignment` and `EqualizationWithOtherAttribute` (subclasses of `AttrImposition`) and `TypeSpecification`, `Nullification` and `EqualizationWithOtherRef` (subclasses of `RefImposition`), respectively. Fig. 10 shows the aforementioned classes.

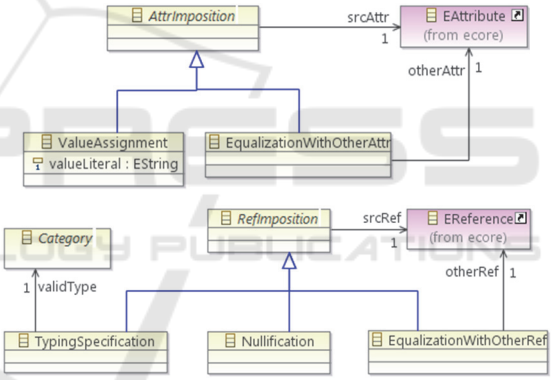

Figure 10: Classes related to impositions on attributes and references.

3.4.3 The LCRMA View as a CVS Model

Fig. 11 shows a part of the CVS model representing the LCRMA view. Due to space limitation the visualization of the whole model is not exposed here. In contrast, the figure focuses on how to model that LCRMA models can only have one RT-Linux platform, based on an FP scheduling policy. The assembly `RT-Lin_Card` is defined by means of a complex category that encompasses an instantiation of each of the following simple categories: `Processor`, `Clock`, `Scheduler` and `SchedPolicy`. They are defined respectively on the classes `Regular_Processor`, `Alarm_Clock`, `Primary_Scheduler` and `Fixed_Priority_Policy` of the MAST 2.0 meta-model. Finally, the card is declared

as an instance of `ObjectsSpecification`, pointing to the defined complex category and that sets "theCard" as name and that the instance is unique.

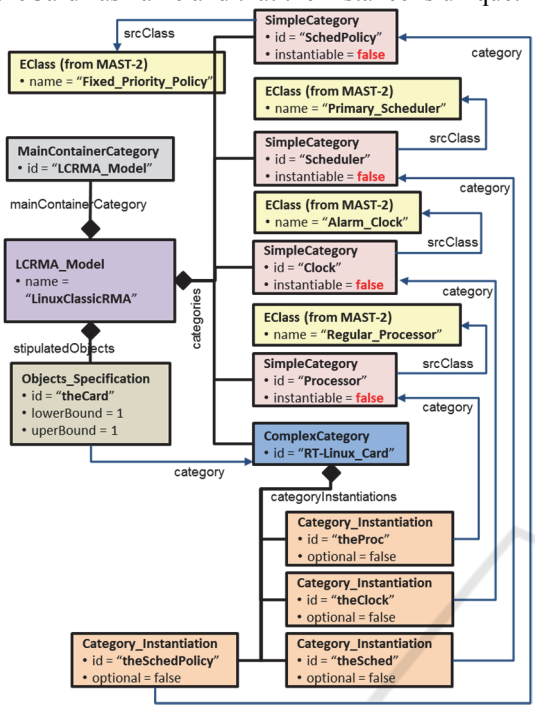

Figure 11: Part of the CVS model of the LCRMA view.

Fig. 12 shows in detail the configuration of the categories `SchedPolicy` and `Scheduler`. The former illustrates how to impose fixed values to attributes while the latter illustrates how to set links among the assembly components.

It is worth noting that, as well as a domain meta-model is formulated only once by the domain expert, the CVS model corresponding to a view is also formulated only once by the same agent. It will be used for the automatic generation of the VRD meta-model, the one used by the designer, as well as the M2M transformation for converting the reduced models built by the designer to domain models.

4 GENERIC TOOL FOR MODEL CONSTRUCTION

The previous section has explained the design of a tool for easing the construction of models according to a view, based on a strategy that requires developing two components (the VRD meta-model and the *VRD_to_Domain* transformation), relative to the view at hand (and hence to the domain meta-model).

This section exposes the design of a generic tool, applicable to any domain meta-model and to any view specified on it. Its generic nature is achieved as a meta-tool that generates on-demand the corresponding specific tool.

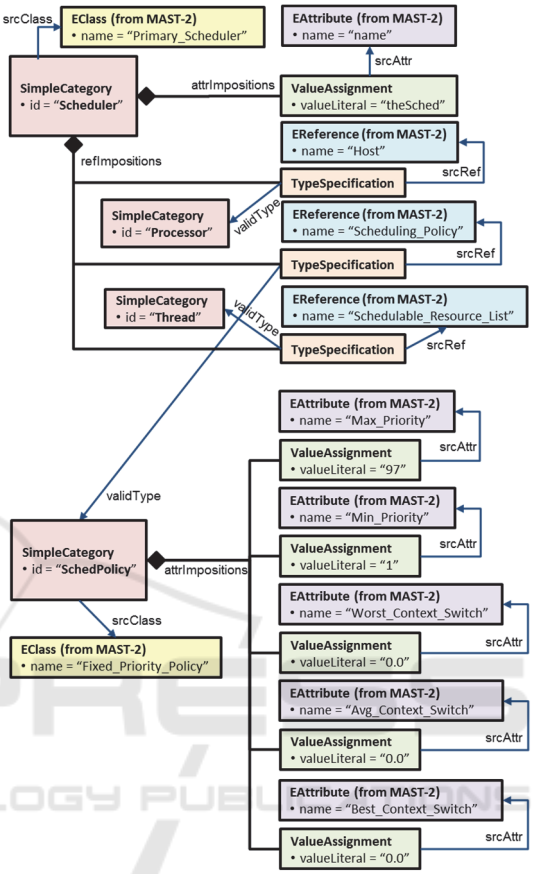

Figure 12: The simple categories `Scheduler` and `SchedPolicy` in detail.

The meta-tool operates in a two-step fashion, outlined in Fig. 13, subsequently generating the two components of every specific tool from the view formulated as a model compliant to the CVS meta-model shown in subsection 3.4:

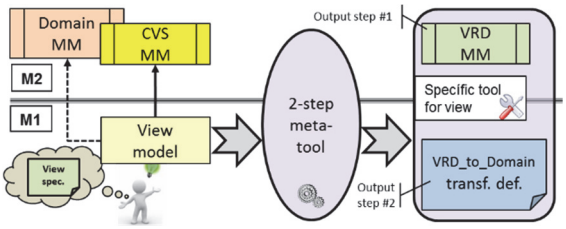

Figure 13: Two-step meta-tool.

1) The VRD meta-model that drives the restricted model construction. As depicted in Fig. 14, this component is obtained from the CVS model

through a promoting transformation (*CVS_to_VRD*), i.e. a transformation that takes an M1 layer artefact (model) and yields an M2 layer artefact (meta-model).

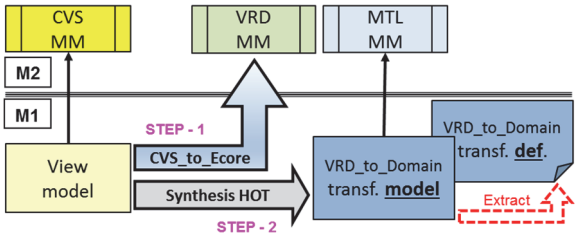

Figure 14: Automatic generation of the VRD meta-model and of the *VRD_to_Domain* transformation.

2) The M2M transformation that transforms the required data models in models compliant to the domain meta-model. As shown in Fig. 14, from the CVS model, the Higher Order Transformation (HOT) technique (Tisi et al., 2009) is used in order to get the *VRD_to_Domain* transformation as a model compliant to the meta-model of the used model transformation language (MTL). This work has used ATL (Jouault et al., 2008), which allows the application of the HOT technique, since its abstract syntax is formalized as a meta-model. In this case the HOT is of the synthesis kind and the generated transformation model is later extracted to the textual notation of ATL. The obtained transformation must correspond to the specific structure of the VRD meta-model generated before.

The developed strategy involves several M2M transformations. Given a specific view on a meta-model, the reduced models built by the designer are transformed to the final models through the corresponding *VRD_to_Domain* transformation. Regarding the generation of the components for each situation (the VRD meta-model and the *VRD_to_Domain* transformation itself), the M2MM transformation *CVS_to_VRD* and the HOT *CVS_to_MTL* are respectively used. Their implementation can be found, along with the rest of the material related to this work, in http://www.istr.unican.es/members/cesarcuevas/phd/constrainingViews.html.

5 CONCLUSIONS

The editors for creating models are usually assisted by the reflective information they get from the corresponding meta-models. This work improves those cases where this strategy is not friendly for the designer, due to the meta-model not following the expected logic for entering the data or because it is excessively complex. The proposed solution consists in using a specialized meta-model for the edition, along with the following model conversion.

In the current version, the method is applicable to any meta-model, but only considering the definition of edition views that are useful for the case of models compliant to a generic domain meta-model that is constrained by 1) reducing the number of used classes, 2) modifying the multiplicities or 3) incorporating new classes defining certain patterns of instances. Future work is to extend the methodology and the tools to views generated in other situations.

ACKNOWLEDGEMENTS

This work has been partially funded by the Spanish Government and FEDER funds, with references TIN2011-28567-C03-02 (HI-PARTES) and TIN2014-56158-C4-2-P (M2C2).

REFERENCES

OMG Formal/2012-06-01: *Systems Modeling Language (SYSML), V1.3.* 2012.

OMG Formal/2011-06-02: *UML Profile for Marte: Modeling and Analysis of Real-time Embedded Systems, V1.1.* 2011.

Bork, D., Karagiannis, D. I. And Fill, H. I., 2014. Model-Driven Development of Multi-view Modelling Tools: The Muviemot Approach.

Bruneliere, H., Perez, J.G., Wimmer, M. And Cabot, J., 2015. EMF Views: A View Mechanism for Integrating Heterogeneous Models, *34th International Conference on Conceptual Modeling (ER 2015)* 2015.

Burger, E., 2013. Flexible Views for Rapid Model-Driven Development, *Proceedings of the 1st Workshop on View-based, Aspect-oriented and Orthographic Software Modelling* 2013, ACM, Pp. 1.

Cicchetti, A., Ciccozzi, F. And Leveque, T., 2012. A Hybrid Approach for Multi-View Modeling. *Electronic Communications of the EASST,* 50.

González Harbour, M., Gutiérrez, J. J., Medina, J. L., Palencia, J. C., Drake, J. M., Rivas, J. M., López Martínez, P. And Cuevas, C., Mast: Bringing Response-time Analysis into Real-time Systems Engineering,

Jouault, F., Allilaire, F., Bézivin, J. And Kurtev, I., 2008. ATL: A Model Transformation Tool. *Science of Computer Programming,* 72(1), Pp. 31-39.

Lehoczky, J., Sha, L. And Ding, Y., 1989. The Rate Monotonic Scheduling Algorithm: Exact

Characterization and Average Case Behavior, *Real Time Systems Symposium, 1989. Proceedings.* 1989, Ieee.

Tisi, M., Jouault, F., Fraternali, P., Ceri, S. And Bézivin, J., 2009. On the Use of Higher-order Model Transformations, *Model Driven Architecture-Foundations and Applications* 2009, Springer.

Wiederhold, G., 1991. *Views, Objects, and Databases.* Springer.