Building Scalable Graphical Modelling Environments with EMFSplitter (tool demo)

Antonio Garmendia, Esther Guerra, and Juan de Lara

Universidad Autónoma de Madrid (Spain)

Abstract. In Model-Driven Engineering the creation of Domain-Specific Modelling Languages (DSMLs) is a recurrent demanding task. Usually DSMLs are built in an ad-hoc manner and the generated environments do not scale well to face scenarios with complex systems. To improve this situation, we propose an approach to facilitate the engineering of DSMLs through a catalogue of patterns and a set of wizards to reduce the implementation time of such environments. Our approach is supported by a tool called EMFSplitter, which proposes a *Modularity* pattern to fragment the models and a *Graphical Representation* pattern, for the definition of graphical and tabular syntax.

Keywords: Domain-Specific Modelling Languages, Graphical Modelling Environments, Scalable Modelling, Meta-modelling, Modularity.

1 Introduction

Model-Driven Engineering (MDE) promotes models as the main artefact to support software construction [2]. To build those models, developers usually design Domain-Specific Modelling Languages (DSMLs) [12]. DSMLs are languages of limited expressiveness, because they focus on the description of a narrow domain.

The models created with DSMLs are usually monolithic, making them difficult to manage and visualize when their size grows. In order to make MDE effective for industrial use, scalability is an important matter of concern, considering that is usual to find scenarios with large systems [8]. To solve this issue, EMF [11] is an often used framework in the MDE community.

Although some companies report the use of EMF in their applications, scalability is still a challenge. Taking inspiration on the modularity concepts of programming languages, we propose fragmentation strategies for DSMLs. This way, models are no longer monolithic, but organized into projects, structured into (hierarchical) packages and fragmented into units [7].

There are some proposals aiming at improving scalability in MDE. Some works propose storing models in NoSQL databases. Among them, we can find tools like Neo4EMF [1] and Morsa [4]. The first tool employs Neo4j as a NoSql database and also implements an on-demand loading mechanism that reduces memory footprint. The latter one, uses MongoDB as a persistence layer for a quick access to a large model. We can also highlight CDO [3] as one of the most mature tools in this field. CDO uses a relational database to store EMF models.

Eclipse has promoted several tools for the creation of graphical modelling environments, such as GMF [5], Graphiti [6] and Sirius [10]. Graphiti provides a Java API for coding, while the other two tools are model-based. None of these frameworks natively support scalability mechanisms, such as fragmentation, and would require considerable programming effort to integrate them.

Our proposal tool called EMFSplitter improves the current state of the art by providing both a simple way to define graphical DSMLs, and scalability by model fragmentation. In this demo paper, we will present the main features of EMFSplitter, using the cloud system described in [13] as running example.

2 Architecture

The first step to implement a DSML is designing an abstract syntax, a meta-model containing the primitives of the domain. To help in the construction of the meta-model, and to define functionalities for the modelling environment, our approach is based on patterns.

Our pattern-based approach is realized within a tool called DSL-tao [9]. This tool has an extensible catalogue of patterns, each one can define services to contribute functionality of the generated environment. In this paper, we will focus on two of them: *Modularity* and *Graphical Representation*. These are contributed by a plug-in called EMFSplitter, which can also be used stand-alone [7].

In the *Modularity Pattern* we propose three key concepts to organise the model in a hierarchical structure which are: Project, Package and Unit. These elements have to be attached to the classes in the meta-model. The class tagged as Project is typically the root of the meta-model. Inside classes tagged as Package, objects of type units as well other packages can be added. Finally, the Unit classes can be defined within packages or projects. With the definition of these concepts at the meta-model level, we generate modelling environments where, the Project instances are mapped to Eclipse projects, objects of type package are mapped to folders and units to files. In this way, models are no longer monolithic.

To define the graphical syntax we implement the *Graphical Representation* pattern. Using this pattern we can define the different diagram layers and the appearance of nodes and edges. Nodes can be represented using basic shapes like, circle and polygon, or an image. In case of edges, we have decorators for the style (e.g. dotted, dashed) and also for the source and target arrows.

3 Tool Support and Running Example

This section illustrates the combined use of EMFSplitter and DSL-tao through a case study based on the DSML presented in [13]. The DSML, called CRALA, is a language for architecture-centric cloud robotics systems. Fig. 1 shows DSL-tao with our excerpt of the DSML, a small extract from the one described in the paper. DSL-tao permits the addition of new patterns, wizards for their application, and code generation services to contribute the final environment. We have

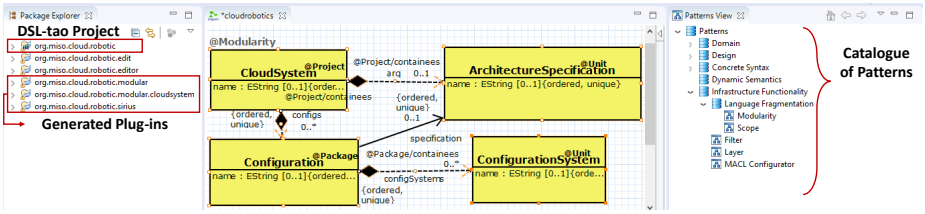

Fig. 1: Excerpt of the meta-model in DSL-tao environment.

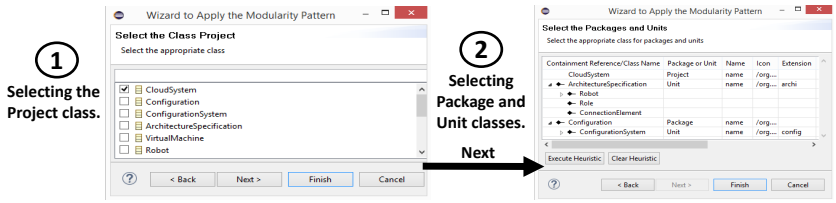

Fig. 2: Modularity Pattern wizard contributed by EMFSplitter.

applied two of the patterns contributed by EMFSplitter: Modularity and Graphical Representation. For these, EMFSplitter provides the pattern structure, the application wizard and the code generation service.

We applied a fragmentation strategy to the language, using the wizard shown in Fig. 2. The class CloudSubsystem is tagged as a Project and can contain a unit of type ArchitectureSpecification. This specification is possible due to the containment relationship that exists between these two classes. The class Configuration is tagged as Package and can contain several units of type ConfigurationSystem. Fig. 1 shows these elements mapped to Project, Package and Unit.

The instantiation of a *Graphical Representation* pattern is through a dedicated wizard. Fig. 3 shows the wizard for specifying the graphical syntax for unit ArchitectureSpecification. The first step in the wizard (label 1) allows selecting the elements to be visualized, and heuristically proposes their shape (node/edge). The second step (label 2) permits customizing elements appearance.

Fig. 1 shows the meta-model of the DSML annotated with the patterns. Once the patterns have been specified, the code generation services provided by EMFSplitter can be invoked to generate an environment where the models are fragmented and visualized according to the defined patterns. The functional environment based on Sirius is shown in Fig. 4.

4 Conclusions and future work

We have presented EMFSplitter, a tool that supports the automatic generation of modelling environments using patterns. As now we can fragment models into folders and files, our next step is to add further functionality, like defining element visibility and a scoping for references.

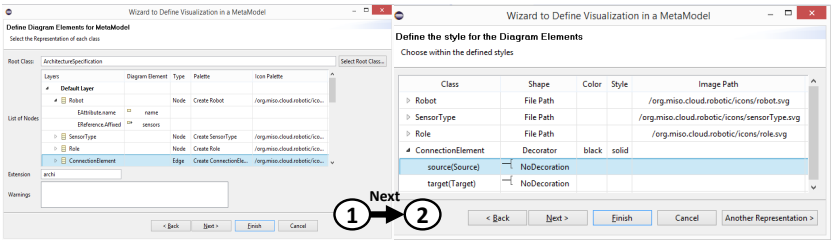

Fig. 3: Graphical Representation Wizard contributed by EMFSplitter.

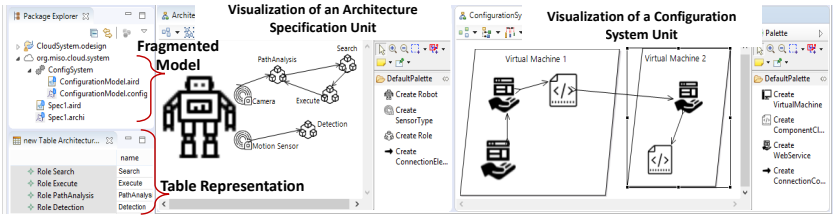

Fig. 4: Graphical Modelling Environment generated by EMFSplitter.

Acknowledgements. Work funded by the Spanish Ministry of Economy and Competitivity (TIN2014-52129-R), and the R&D programme of the Madrid Region (S2013/ICE-3006).

References

1. A. Benelallam, A. Gómez, G. Sunyé, M. Tisi, and D. Launay. Neo4emf, a scalable persistence layer for emf models. In *ECMFA*, pages 230–241. Springer, 2014.
2. M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, USA, 2012.
3. Connected Data Objects (CDO). `http://www.eclipse.org/cdo/`.
4. J. Espinazo-Pagán, J. S. Cuadrado, and J. G. Molina. Morsa: A scalable approach for persisting and accessing large models. In *MODELS 2011*, pages 77–92, 2011.
5. GMF. `http://www.eclipse.org/modeling/gmp/`.
6. Graphiti. `https://eclipse.org/graphiti/`.
7. A. Jiménez-Pastor, A. Garmendia, and J. de Lara. Scalable model exploration for model-driven engineering. *Journal of Systems and Software*, 132:204–225, 2017.
8. D. S. Kolovos et al. A research roadmap towards achieving scalability in model driven engineering. BigMDE '13, pages 2:1–2:10, New York, NY, USA, 2013. ACM.
9. A. Pescador, A. Garmendia, E. Guerra, J. S. Cuadrado, and J. de Lara. Pattern-based development of domain-specific modelling languages. In *MoDELS*, pages 166–175. IEEE, 2015.
10. Sirius. `https://eclipse.org/sirius/`.
11. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley Professional, 2008.
12. M. Voelter. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org, 2013.
13. L. Zhang et al. Towards an architecture-centric approach to manage variability of cloud robotics. In *DSLROB*, 2015.