# Model-based Tool Support for the Development of Visual Editors
## *A Systematic Mapping Study*

David Granada, Juan M. Vara, Francisco Pérez Blanco and Esperanza Marcos
*Kybele Research Group, Rey Juan Carlos University, Móstoles, Madrid, Spain*

Keywords: Model Driven Engineering, DSL, IDE, Systematic Mapping Study.

Abstract: Visual Domain Specific Languages play a fundamental role in the development of model-driven software. The increase in this type of visual languages and the inherent complexity as regards the development of graphical editors for them has, in recent years, led to the emergence of several tools that provide technical support for this task. Most of these tools are based on the use of models and increase the level of automation of software development, which are the basic principles of Model Driven Engineering. This paper therefore reviews the main features, potential advantages and current limitations of the main tools that exist for the development of graphical editors for visual DSLs.

## 1 INTRODUCTION

Model-Driven Engineering (MDE) (Schmidt, 2006) is a natural step in the historical tendency of software engineering as regards raising the level of abstraction at which software is developed. In this context, Domain Specific Languages (DSLs) (Mernik et al., 2005) play a cornerstone role in almost any proposal that applies the principles of MDE. The fact that DSLs are targeted toward a particular domain contributes to ease of use and greater expressiveness allowing the distance between business users and developers to be shortened (Ghosh, 2011). DSLs also allow us to find solutions by using appropriate terms and abstraction level for the problem domain, which improves the quality, productivity and usability of software.

During the last few years, provided that MDE is approaching its slope of enlightenment in the traditional hype cycle experienced by any technological proposal (Linden and Fenn, 2003), we have undertaken several works oriented towards the adoption of more rigorous and systematic approaches for the development of MDE tools. They were first focused on identifying best practices for the development of DSL toolkits (Vara and Marcos, 2012), while the focus latter shifted to the development of model transformations (Bollati et al., 2013) as the main assets with which model-based proposals can be automated. We have been working as well in the application of MDE principles in other fields (Vara et al., 2012).

One of the main lessons learned during this time is that probably the most attractive feature of MDE for researchers from other fields is the ability to use visual models to depict their ideas, represent plans or processes, etc. In order for this feature to become really useful those DSLs should come accompanied by the corresponding tooling, basically in the shape of visual editors or diagrammers (Selic, 2012).

Therefore, in order to know what the possibilities are when building visual editors for DSLs and to obtain a clear understanding of the state of the art as regards tools for the development of these editors, in this paper we present the main results obtained after carrying out a systematic mapping study according to the guidelines proposed in (Biolchini et al., 2005) for the development of this type of studies in the context of Software Engineering, which is composed of three main phases: *Planning*, *Execution* and *Analysis of Results*.

Note also that only model-based tools are considered. That is, tools that make some use of models for the development of visual editors. We do so under the premises that: we firmly believe in the advantages brought by MDE and we acknowledge that it is time to start *eating our own dog food* (use models to develop solutions) if we want MDE to be effective and definitively adopted by the industry.

The results obtained from this study were then used in an attempt to gain a general idea about the current state of the different technologies used in the tools analysed. We also wished to check for possible improvements based on certain criteria evaluated. For



Granada, D., Vara, J., Blanco, F. and Marcos, E.
Model-based Tool Support for the Development of Visual Editors - A Systematic Mapping Study.
DOI: 10.5220/0006430503300337
In *Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017)*, pages 330-337
ISBN: 978-989-758-262-2

the sake of space, we do not include here all the data gathered during the development of the study, which can be found in (Granada, 2016).

The remainder of this paper is structured as follows: Section 2 describes the main phases of the method followed to carry out the systematic mapping study; Section 3 presents its main results; Section 4 is devoted to answer the research questions of the study and Section 5 to compare this work with existing literature. Finally, Section 6 summarizes the main conclusions derived from this work.

# 2 METHOD

In this section, we shall focus on the planning of the study (Biolchini et al., 2005), in which the research objectives and how the study was conducted are introduced. More specifically, Section 2.1 presents the research questions defined for the study; Section 2.2 presents the digital libraries and query strings used; and finally, Section 2.3 enumerates the inclusion and exclusion criteria.

## 2.1 Research Questions

The first step conducted in the *planning* phase was that of defining the main objective of the systematic mapping study, i.e., to identify and analyse the state of the art as regards the generation of editors that support DSLs in the context of Model-Driven Engineering. To achieve this goal, we defined the two research questions (***RQ***) that gave rise to this study.

- ***RQ1***: How many model-based tools with which to generate editors for visual DSLs exist?
- ***RQ2***: What are the main features and functionalities of these tools?

## 2.2 Digital Libraries and Query Strings

The next phase includes the identification of the digital libraries in which the search for studies will take place (Biolchini et al., 2005). It is also necessary to define a set of query strings that will be used in these digital libraries.

Table 1 shows the digital libraries that were selected for this study. It is worth noting that, although Table 1 shows only six data sources, we initially considered another four digital libraries that were discarded during the search process, given the lack of relevant information and data obtained from them.

The box below shows the query string used to search for the studies. Note that given that each of the digital libraries supports a slightly different syntax, the query string had to be adapted to each search engine.

Table 1: Digital libraries.

| Name (Acronym) | Website |
|---|---|
| ACM Digital Library (ACM) | http://portal.acm.org/ |
| Google Scholar (GS) | http://scholar.google.com/ |
| IEEEXplore (IEEE) | http://ieeexplore.ieee.org/ |
| ISI Web of Knowledege (ISI) | http://www.webofknowledge.com/ |
| Science Direct (SD) | http://www.sciencedirect.com/ |
| SpringerLink (SL) | http://www.springerlink.com/ |

("development" OR "generation") AND ("visual editor" OR "diagrammer" OR "graphical editor" OR "IDE") AND ("DSL" OR "DOMAIN SPECIFIC LANGUAGE")

## 2.3 Inclusion and Exclusion Criteria

In any systematic mapping study, most of the results first retrieved from the digital libraries are not related to the research questions posed. To specify inclusion and exclusion criteria directly derived from the research questions are therefore needed in order to filter those studies (Biolchini et al., 2005). The inclusion criterion used in this study was formulated as follows: *the abstract indicates that it is a study related to modelling tools that can be used to develop editors for visual DSLs*. We therefore checked the abstract (as well as the title and keywords of each study) in order to confirm whether they satisfied the inclusion criterion.

Resulting studies were then subject to a detailed reading in order to exclude non relevant ones according to the following exclusion criteria: studies whose main purpose is to classify other articles or are themselves systematic literature reviews; studies whose main purpose is not related to graphical editors or visual DSLs, although these concepts are mentioned in the content of the study; studies related to other types of graphical editors, tools and/or technologies that are not relevant to the purpose of this analysis.

The remaining set of studies were then considered as potential candidates to become primary studies for this review.

# 3 RESULTS

This section presents the results of this systematic mapping study. First, section 3.1 provides some data related to the number of works retrieved from the digital libraries. Section 3.2 then presents the main findings and conclusions gathered after analysing the set of primary studies considered here.

## 3.1 Studies Selection

Running the search in each digital library (using the adapted version of the query string) yielded the results shown in (Table 2) which computed 10,116 works. First two columns on the left show the number of results for each digital library. Third column shows the number of relevant studies, i.e., those that fulfilled the inclusion criterion. Fourth column shows the number of primary studies: those remaining that were not excluded by the exclusion criteria. Last column shows the percentage of primary studies found in each data source. Note that most of the primary studies were found in Google Scholar (27.82%) and ISI Web of Knowledge (21.8%).

Table 2: Distribution of selected studies.

| Source | Studies | | | Percentage |
|---|---|---|---|---|
| | *Founded* | *Relevant* | *Primaries* | (/primaries) |
| **ACM** | 301 | 11 | 8 | 6,01% |
| **GS** | 5856 | 64 | 37 | 27,82% |
| **IEEE** | 35 | 28 | 16 | 12,03% |
| **ISI** | 2573 | 45 | 29 | 21,80% |
| **SD** | 684 | 26 | 18 | 13,53% |
| **SL** | 667 | 38 | 25 | 18,80% |
| **TOTAL** | **10.116** | **212** | **133** | **100%** |

However, many of the relevant, and therefore, the primary studies were available in more than one digital library. Table 3 shows the eventual number of primary studies found after removing all the repeated studies. These data show that more than 36% of the primary studies found in this review were duplicated in different digital libraries. We therefore removed the copies and consequently obtained a list of 85 non-duplicated primary studies.

Table 3: Duplicated primary studies.

| | Studies | Percentage |
|---|---|---|
| Primary studies | 133 | 100% |
| Duplicated primary studies | 48 | 36,1% |
| Non-duplicated primary studies | 85 | 63,9% |

## 3.2 Data Extraction

The data extraction phase of the systematic mapping study allowed us to collect the most relevant information from the primary studies. Regarding ***RQ1***, the main model-based tools for the generation of editors for visual DSLs resulted to be: `DiaGen` (Minas and Köth, 2000), `EuGENia` (Kolovos et al., 2009), `GMF` (Gronback, 2009), `Graphiti` (Brand et al., 2011), `MetaEdit+` (Tolvanen and Kelly, 2009), `Obeo Designer` (Juliot and Benois, 2010), `Sirius` (Viyovic et al., 2014) and `Tiger` (Ehrig et al., 2005). Note also that this is indeed a never-ending task, since new proposals arise every so often.

In order to analyse each of these tools in a systematic manner and gather the data needed to answer ***RQ2***, the following criteria were used, derived from the features analyzed by the primary studies considered:

**Scope.** One of the main issues to consider regarding software engineering tools is whether the tool is commercial or open source. The type of license is also considered here.

**Framework.** Frameworks are technological structures that can provide the basis for the development of software. In the context of MDE, Eclipse is the most widely used framework (Vara and Marcos, 2012). Given this, it is interesting to identify whether the tool analysed is isolated or runs on an existing framework.

**Distinction between Abstract and Concrete Syntax.** The abstract syntax defines the set of abstractions provided by a language and how they may be combined to create models, while the concrete syntax provides a notation that facilitates the representation of models expressed with such language.

**Abstract Syntax.** The first step toward the definition of a DSL is the specification of its metamodel. It is therefore relevant to identify the way abstract syntaxes must be specified for the reviewed tool to generate the visual editor.

**Concrete Syntax.** One of the main distinctive features between these tools is the mechanisms provided for the definition of the concrete syntax of the DSL for which the editor is been developed.

**Editing capabilities.** Once the editor has been developed, the ability to manipulate and modify the elements of the models that can be created with such editor in a simple, complete and functional manner becomes the most relevant need. Editing capabilities of the editors produced with the reviewed tool is something that is worth attention as well.

**Use of Models.** Given that we focus on model-based tools, the extent to which models are used in the development process implemented by the reviewed tool need attention.

**Automation.** The level of automation is one of the most relevant features of MDE proposals. Moreover, one of the main factors for a software engineering tool to success is the ability to perform its job without adding external complexity to the

process. All this given, the level of automation with which these tools are able to produce visual editors is also considered in this study,

**Usability.** This criterion was included to check the ease of use of each tool. In other words, the simplicity with which users could perform all the steps required to complete the development of a visual editor for their DSLs. We check for instance whether the tools provide official documentation, wizards, guided tours, examples and/or any other form of assistance.

**Methodological Basis.** Finally, when designing a visual notation, the issues related to the quality of that notation should be considered. Several mechanisms with which to assess the quality of the metamodel that collects the abstract syntax of a DSL can be employed for this purpose (López-Fernández et al., 2014). There are as well different metrics with which to assess whether the graphical symbols composing the concrete syntax could be correctly processed by the human mind. We analyse consequently whether the tools follows or applies any kind of scientific theory, empirical evidence or any given method to guide, derive or define visual notations.

# 4 DISCUSSION

In this section, we present the detailed answers to the research questions posed in Section 2.1. For each of the criteria defined in the Data extraction results section, a brief analysis will be presented.

## 4.1 Scope

We discovered that most of the tools analyzed are open source with different license types. Only two of these proposals have a commercial license: `MetaEdit+` and `Obeo Designer`.

## 4.2 Framework

Frameworks are characterized by the fact that they provide various types of support that facilitate the reduction of development time. In this context, most of the proposals under discussion are designed to be used in the Eclipse development environment. This framework is widely accepted by the development community owing to the ease with which its functionalities can be expanded through the creation of new extensions that can be added to its core. It is noteworthy that `MetaEdit+` is the only tool that provides its own framework independent of Eclipse.

## 4.3 Distinction between Abstract and Concrete Syntax

With regard to the distinction between abstract syntax and concrete syntax, most of the tools analyzed allow users to perform this process. For example, `GMF` uses the component called GMF Notation. This component has an EMF-standard mechanism to provide annotations in an Ecore metamodel, which allows information on graphical decisions and the definition of elements in the domain model to be maintained as two different concepts.

Another interesting proposal in this regard is `Eugenia`, which provides not only the ability to define visual annotations during the definition of the metamodel (Kolovos et al., 2009), but also the definition of these visual annotations by means of external libraries written in EOL, signifying that the two syntaxes can be kept conceptually separated. Of the tools analyzed, the only proposals that do not provide such a separation are `Diagen` and `Tiger`.

## 4.4 Abstract Syntax

Most of the proposals analyzed use Ecore to define the abstract syntax of DSLs, and of these proposals, some may also use other mechanisms to define the abstract syntax. For example, `Graphiti` can use objects defined in Java, while `Diagen` can do so via UML diagrams. The tools that do not use Ecore to define the abstract syntax are: `MetaEdit+` and `Tiger`. `Tiger` uses the graphic transformation integrated engine AGG (Attributed Graph Grammar) to define the abstract syntax. Finally, `MetaEdit+` uses the GOPPRR (Graph Object Property Port Role Relationship) language, which also provides a support tool with which to import and use other modeling languages.

## 4.5 Concrete Syntax

After specifying the abstract syntax it is necessary to associate a visual representation with each of its elements. `GMF` and `Graphiti` have in common the use of Draw2D tool, which is based on the use of Java objects to define the graphical representation of the domain elements. In `Obeo Designer` and `Sirius`, the graphical aspects of the domain model elements are defined by means of templates called *Odesign*. These templates allow us to define some visual properties through the use of intuitive contextual menus. In `Eugenia`, the definition of the concrete syntax is conducted by means of textual annotations on the metamodel, but it also provides the ability to define these annotations using the EOL language in an external

file. MetaEdit+ uses an internal library and a symbol editor that allow the user to assign a graphical representation to each of the elements of the metamodel. Finally, Tiger uses a component called *ShapeFigures*, which allows the user to assign the shape, color and size to each of the elements of the domain model.

## 4.6 Editing Capabilities

Most of the proposals provide the common operations needed to edit the elements of the models that can be created with the editor developed. However, only a few proposals have sophisticated and advanced options with which to edit these elements.

In this context, the tools that provide the greatest editing capabilities are: GMF, Graphiti, MetaEdit+, Obeo Designer and Sirius. For example, GMF provides some advanced editing features such as: a design palette, a browser, decorators, an assistant diagram, editing tips, filters, drag and drop, animated zoom and exportation to other formats.

Likewise, Obeo Designer and Sirius, in addition to providing a wide variety of styles by default, make it possible to extend these styles. Most of the editing options can be easily performed through the use of intuitive contextual menus and the properties table of the elements related to the generated editor.

## 4.7 Use of Models

With regard to the extent to which models are used for the production of graphical editors, Eugenia, GMF, Graphiti, Obeo Designer and Sirius are those that use different types of models in the development of editors. GMF is specifically a proposal that provides a model-driven approach with which to generate graphical editors. Its core is the Graphical Definition Model, which contains information related to the graphical elements that will appear in a GEF-based runtime (editor), while the tool palette are defined by the Tooling Definition Model. The latter two models are related by means of the Mapping Definition Model, which links the graphical and tooling definitions to the domain model selected. The other tools that have a high level of use of models are based on GMF, and therefore inherit its approach.

## 4.8 Automation

It is important to obtain the editor, or at least the intermediate stages, automatically in order to provide the user with a clear and efficient process. In some of the tools analyzed it is possible to achieve this goal by means of a few initial instructions and the selection of some default options. The proposals that offer a higher level of automation during the process of generating an editor are Eugenia, Graphiti, MetaEdit+, Obeo Designer and Sirius. More specifically, in the case of Eugenia, the definition of the abstract syntax and visual annotations for each element of the domain makes it possible to generate all the models involved in the editor-generation process automatically. Similarly, Graphiti is another tool that has an editor-generation process in which refinements and modifications made to the models created during the process are not required. Obeo Designer and Sirius go one step further, because these tools provide the user with the ability to generate the editor without having to either manually launch any intermediate process or use a second instance of Eclipse.

## 4.9 Usability

In the case of usability, it is interesting to identify the facilities provided by the proposals employed to create and modify new editors. In this context, DiaGen, EuGENia, Graphiti, MetaEdit+, Obeo Designer, Sirius and Tiger provide a better usability. For example, Diagen provides a set of interfaces that guide users when defining the abstract syntax and show them how to use the editor generated. Moreover, one of the main goals of Graphiti is to provide the user with easy and intuitive methods when generating graphical editors. This goal is achieved by means of an interface based on an API that allows the user to build an editor without having prior knowledge of GEF or Draw2D.

On the contrary, as mentioned previously, GMF is one of the most important projects for the generation of graphical editors, although one of its weaknesses is the complexity of the learning curve when compared to other tools.

## 4.10 Methodological Basis

In this case we can conclude that none of the tools studied follows a methodological basis in the process of construction of graphical editors, nor use a theoretical basis for assessing the quality of the visual notations generated. In this respect, none of the tools analyzed takes into account the basic principles of Human Computer Interaction (HCI) in order to obtain concrete (visual) syntaxes that can be understood and correctly analyzed by the human mind. The definition of the concrete (visual) syntaxes therefore usually consists of the arbitrary assignment of graphical symbols to the concepts defined, of which the abstract

syntax of the language is composed.

In this context, only a few tools such as `MetaEdit+`, `Obeo Designer` and `Sirius` provide certain facilities related to the use of visual variables (*location*, *shape*, *color*, *brightness*, *size*, *orientation* and *texture*) (Bertin, 1983). These three tools have a properties panel from which it is possible to select, in a way that is intuitive for the user, all the possible values of the visual variables that can be used in the elements of the concrete syntax generated.

### 4.11 Summary Results

Having analyzed the different proposals, and in order to provide a rapid overview of them, Table 4 shows a summary of the comparative study according to the characteristics defined in the previous subsections.

As will be observed in the summary of the comparative study presented in Table 4, most of the proposals solve the problem of the conceptual separation between the metamodel (abstract syntax) and the design features that are attached to each of its elements (concrete syntax), without adding information related to the domain of the solution in the context of the problem domain. There are also proposals that automatically generate many of the intermediate phases during the process of generating an editor. However, most of these tools have to refine each intermediate phase through the use of mechanisms that are complex and unintuitive for the user.

As regards the use of models, only a few of the tools use models in each of the editor-generation phases, which considerably facilitates the usability of the tool and allows the user to focus on the important aspects of the domain. Furthermore, in this work we have found that most of the proposals have been designed to be used in Eclipse, which is one of the most popular frameworks in the software development community. This feature facilitates integration and interoperability with other software systems developed under Eclipse. Regarding the use of a scientific basis to design visual notations, we have identified that virtually no tool provides mechanisms with which to achieve this goal, and only a few of them provide the possible values of visual variables in a way that is orderly and intuitive for the user.

Furthermore, the values of the last row in Table 4 represent the percentage of the maximum possible score for the qualitative criteria evaluated (ticks obtained / maximum possible ticks). As this data shows, the highest values are obtained for the usability (91.6%) and the use of models (87.5%) criteria, while it is worth noting that a very small value (12.5%) is obtained for the criterion related to the use of a methodological basis so as to take into account the quality of the visual notations generated. Given that, we can state that of the commercial tools, `MetaEdit+` and `Obeo Designer` are the most complete proposals according to the characteristics evaluated in this work. Of the open source tools, `Eugenia`, `GMF`, `Graphiti` and `Sirius` can, meanwhile, be considered as the most complete and functional for the purpose of generating editors for DSL. And this may be understandable, considering that these four tools have technological bases in common, such as the fact that they use EMF.

Many of the proposals discussed in this paper therefore have interesting features as regards generating graphical editors for DSL, although several of their features have room for improvement and refinement. In order to obtain tools that are able to create editors with a minimal difficulty and transparency for the user, it is important that these tools take into account aspects such as the use of frameworks that follow the standards of the software development community, the use of models in each phase of editor generation, the separation between abstract and concrete syntax, automation in the development process and the usability of the tool itself.

Finally, we have concluded that none of the tools follows a methodological basis in the process of construction of editors that support DSLs, nor use a theoretical basis for assessing the quality of the visual notations generated, in order to guide to the user in the generation of visual notations that are easier to process by the human mind, without being limited to the classic geometric shapes like boxes and arrows.

## 5 RELATED WORKS

In this section, we briefly present some related works (Ehrig et al., 2005; Pelechano et al., 2006; Amyot et al., 2006; Kelly, 2004; Kern et al., 2011). More specifically, some works, such as (Ehrig et al., 2005), present the `Tiger` tool, but they also present a state-of-the-art generation of a graphical editor divided into two generation categories: the Eclipse-based editor generation and the Graph-transformation based editor generation. The choice of presenting a state of the art divided into these two categories is made because the tool presented combines the advantages of formal visual language specifications using graph transformations with the facilities offered by Eclipse/GEF in order to generate graphical editors.

In (Pelechano et al., 2006) the authors present a comparative study of tools for model-driven development, and more specifically a comparison be-

Table 4: Overview of tools for graphical editors development.

| *Tools* | *Scope* | *Frame-work* | *Abs. syntax* | *Conc. syntax* | *Syntax distinct.* | *Editing* | *Models* | *Auto-mation* | *Usabi-lity* | *Meth. basis* |
|---|---|---|---|---|---|---|---|---|---|---|
| **Diagen** | OS (GPL) | Eclipse | Ecore/ UML | DiaMeta Design | No | ✔✔ | ✔✔ | ✔✔ | ✔✔✔ | - |
| **Eugenia** | OS (EPL) | Eclipse | Ecore | EOL | Yes | ✔✔ | ✔✔✔ | ✔✔✔ | ✔✔✔ | - |
| **GMF** | OS (EPL) | Eclipse | Ecore | Draw2D | Yes | ✔✔✔ | ✔✔✔ | ✔✔ | ✔✔ | - |
| **Graphiti** | OS (BSD) | Eclipse | Ecore/ Java | Draw2D | Yes | ✔✔✔ | ✔✔✔ | ✔✔✔ | ✔✔ | - |
| **MetaEdit+** | Com | Own | GOP-PRR | Internal API | Yes | ✔✔✔ | ✔✔ | ✔✔✔ | ✔✔✔ | ✔ |
| **Obeo Designer** | Com | Eclipse | Ecore | Odesign | Yes | ✔✔✔ | ✔✔✔ | ✔✔✔ | ✔✔✔ | ✔ |
| **Sirius** | OS | Eclipse | Ecore | Odesign | Yes | ✔✔✔ | ✔✔✔ | ✔✔✔ | ✔✔✔ | ✔ |
| **Tiger** | OS (GPL) | Eclipse | AGG | Shape-Fig | No | ✔ | ✔✔ | ✔ | ✔✔✔ | - |
| % of compliance of the qualitative criteria: | | | | | | 83.3% | 87.5% | 83.3% | 91.6% | 12.5% |

Legend (for weightable fields): *None* (-), *Poor* (✔), *Good* (✔✔), *Excellent* (✔✔✔)

tween Microsoft DSL tools and Eclipse Modeling plugins. This study presents the main differences between these two categories of tools according to certain criteria, such as how to collect the abstract syntax, the format of its models, the type of concrete syntax used and different types of transformations between models. It also presents an experiment with real users of the tools, which allowed some significant values to be obtained as regards user preferences and the effectiveness of using each of the different approaches.

Furthermore, (Amyot et al., 2006) present a comparative study of 5 tools that support the development of DSML environments: `GME`, `Telelogic Tau G2`, `Rational Software Architect`, `XMF-Mosaic` and `Eclipse EMF/GEF`. In this study, the following evaluation criteria are used: graphical completeness, editor usability, effortlessness, language evolution, integration and transformation. The authors of this paper assessed these criteria by developing a graphical editor for a case study with each of the five tools analyzed.

Likewise, (Kelly, 2004) shows a comparison between Eclipse `EMF/GEF` and `MetaEdit+` for Domain-Specific Modeling. This paper presents a brief description of these types of tools and a comparison between them, focused on comparing only those aspects related to the development time required and the number of lines of code generated.

Finally, (Kern et al., 2011) present a comparison of a set of metamodeling languages like `ARIS`, `Ecore`, `GOPPRR`, `GME` or `MS DSL Tools`, which are using the heavyweight metamodelling approach (Frankel, 2003) and are available as tools.

In conclusion, the main differences between these articles and the study presented here is that these articles are mainly focused on presenting one (or several) tools in detail, and showing only some differences among these tools. Furthermore, those works that show a wider comparison do not follow a rigorous literature review process, and the research criteria are different and fewer than those evaluated here.

# 6 CONCLUSIONS

Visual DSLs are becoming increasingly more important and this implies that there is clearly a need for tools with which to create supporting tools for these DSLs as quickly and effectively as possible (Völter, 2009). In order to address this need, the findings of this systematic mapping study have enabled us to gather some data and ideas that provide a complete picture of existing tool support for the development of visual editors for DSLs. This study has also served to identify some of the main challenges that the generation of these editors should address in the forthcoming years. Some of them are summarized as follows:

Giving more relevance to the two basic principles of MDE when developing visual editors, i.e., leveraging the role of models throughout the entire development process and enhancing the level of automation in the development process.

Usability plays a cornerstone role for this type of tools. Note that domain experts have not to be necessarily familiarized with software engineering tools at all. Ease of use is then key to enable users to develop visual editors for the domain of interest without even noticing that they are applying model-based principles or developing a DSL.

According to the findings of this study, none of the analysed tools consider quality aspects during the development of visual editors. This might be related with the fact that most of these tools were built by experienced developers not used to deal with HCI issues.

It is worth noting that this type of study is a never-ending task since new tools appear every some often. One of our future goals is to periodically update this study in order to identify new tools that enable the generation of visual editors from a domain model.

## ACKNOWLEDGEMENTS

This research has been funded by the Government of Madrid under the SICOMORo-CM project (S2013/ICE-3006), the ELASTIC project (TIN2014-52938-C2-1-R), financed by the Spanish Ministry of Science and Innovation, and by the GES2ME Research Excellence Group, co-funded by URJC and Banco Santander

## REFERENCES

Amyot, D., Farah, H., and Roy, J.-F. (2006). Evaluation of development tools for domain-specific modeling languages. In *International Workshop on System Analysis and Modeling*, pages 183–197. Springer.

Bertin, J. (1983). Semiology of graphics: diagrams, networks, maps. *University of Wisconsin press*.

Biolchini, J., Mian, P. G., Natali, A. C. C., and Travassos, G. H. (2005). Systematic review in software engineering. *COPPE/UFRJ, Technical Report ES*, 679(05):45.

Bollati, V. A., Vara, J. M., Jiménez, A., and Marcos, E. (2013). Applying MDE to the (semi-)automatic development of model transformations. *Information and Software Technology*, 55(4):699 – 718.

Brand, C., Gorning, M., Kaiser, T., Pasch, J., and Wenz, M. (2011). Graphiti - development of high-quality graphical model editors. *Eclipse Magazine*.

Ehrig, K., Ermel, C., Hänsgen, S., and Taentzer, G. (2005). Generation of visual editors as eclipse plug-ins. In *Proceedings of the 20th ASE International Conference*, pages 134–143. ACM.

Frankel, D. S. (2003). *Model Driven Architecture Applying Mda*. John Wiley & Sons.

Ghosh, D. (2011). Dsl for the uninitiated. *Communications of the ACM*, 54(7):44–50.

Granada, D. (2016). *Desarrollo dirigido por modelos de editores grficos cognitivamente eficaces para Lenguajes Especficos de Dominio*. PhD thesis, Rey Juan Carlos University.

Gronback, R. (2009). *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Eclipse Series. Pearson Education.

Juliot, E. and Benois, J. (2010). How to build eclipse dsm witwith being an expert developer? *Obeo Designer whitepaper*.

Kelly, S. (2004). Comparison of eclipse emf/gef and metaedit+ for dsm. In *19th ACM SIGPLAN conference*.

Kern, H., Hummel, A., and Kühne, S. (2011). Towards a comparative analysis of meta-metamodels. In *Proceedings of DSM'11*, pages 7–12. ACM.

Kolovos, D. S., Rose, L. M., Paige, R. F., and Polack, F. A. (2009). Raising the level of abstraction in the development of gmf-based graphical model editors. In *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*, pages 13–19.

Linden, A. and Fenn, J. (2003). Understanding gartners hype cycles. *Strategic Analysis Report N$^o$ R-20-1971. Gartner, Inc*.

López-Fernández, J. J., Guerra, E., and de Lara, J. (2014). Meta-model validation and verification with metabest. In *Proceedings of the 29th ASE international conference*, pages 831–834. ACM.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

Minas, M. and Köth, O. (2000). Generating diagram editors with diagen. *Applications of Graph Transformations with Industrial Relevance*, pages 539–541.

Pelechano, V., Albert, M., Muñoz, J., and Cetina, C. (2006). Building tools for model driven development. In *Comparing Microsoft DSL Tools and Eclipse Modeling Plug-ins*, volume 227, pages 1613–0073.

Schmidt, D. C. (2006). Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25.

Selic, B. (2012). What will it take? a view on adoption of model-based methods in practice. *Software and System Modeling*, 11(4):513–526.

Tolvanen, J.-P. and Kelly, S. (2009). Metaedit+: defining and using integrated domain-specific modeling languages. In *Proceedings of the 24th ACM SIGPLAN conference*, pages 819–820. ACM.

Vara, J. M., Andrikopoulos, V., Papazoglou, M. P., and Marcos, E. (2012). Towards model-driven engineering support for service evolution. *Journal of Universal Computer Science*, 18(17):2364–2382.

Vara, J. M. and Marcos, E. (2012). A framework for model-driven development of information systems: technical decisions and lessons learned. *Journal of Systems and Software*, 85(10):2368–2384.

Viyovic, V., Maksimovic, M., and Perisic, B. (2014). Sirius: A rapid development of dsm graphical editor. In *Intelligent Engineering Systems (INES)*, pages 233–238.

Völter, M. (2009). Md* best practices. *Journal of Object Technology*, 8(6):79–102.