Noname manuscript No.
(will be inserted by the editor)

CEViNEdit: improving the process of creating cognitively effective graphical editors with GMF

David Granada · Juan M. Vara · Mercedes Merayo · Esperanza Marcos

Received: date / Accepted: date

Abstract The rise of Domain Specific (Visual) Languages and the inherent complexity of developing graphical editors for these languages have led to the emergence of proposals that provide support for this task. Most of these proposals are principally based on EMF and GMF, which effectively help to simplify and increase the level of automation of the development process of the editors, but it is important to recall that these proposals have some important disadvantages, mainly related to the learning curve of these technologies, poor documentation or the complexity of providing all the customisation possibilities to the user. In addition, in the process of developing a Domain Specific Language, issues related to graphical conventions have historically been undervalued, while most of the effort has been focused on semantic aspects. In fact, definitions of the concrete (visual) syntax of modeling languages in Software Engineering are usually based on common sense, intuition, the reuse of existing notations or emulation of common practices. In order to alleviate the inherent complexity of the EMF/GMF approach for the development of graphical editors and to support the evaluation of the quality of visual notations of modeling languages, this article presents `CEViNEdit`, an intuitive tool that simultaneously supports the semi-automatic generation of graphical editors and the assessment of the cognitive effectiveness of the visual notation implemented by the editor.

Keywords Model Driven Engineering (MDE) · Domain Specific Language (DSL) · Visual Notation · Cognitive Effectiveness

D. Granada, J.M. Vara, E. Marcos
Kybele Research Group, Rey Juan Carlos University,
Calle Tulipán S/N, 28933, Móstoles, Madrid, Spain

D. Granada
E-mail: david.granada@urjc.es

J.M. Vara
E-mail: juanmanuel.vara@urjc.es

M. Merayo
Faculty of Computer Science, Complutense University of Madrid
Ciudad Universitaria, 28040, Madrid, Spain
E-mail: mgmerayo@fdi.ucm.es

E. Marcos
E-mail: esperanza.marcos@urjc.es

1 Introduction

Creating a Domain Specific Language (DSL) [1] is one of the best options as regards defining new modeling languages and implementing the principles of Model Driven Engineering (MDE) [2]. The fact that the models can mitigate the differences between the various stakeholders involved in the development of digital products and the DSLs are targeted towards a particular domain contributes to ease of use, increases their expressiveness and ultimately shortens the distance between the different types of users of DSLs [3,4].

Given that the two basic principles of the MDE are those of enhancing the role of models and raising the level of automation [5], there is a need for not only new modeling languages, but also tools that facilitate the processing and definition of models. Thus, just as programmers have Integrated Development Environments (IDEs), which help them with coding tasks, developers need modeling environments that can assist them when working with a DSL.

The rise of MDE has consequently led to the emergence of tools for the production of this type of environments. Most adopt an approach based on the metamodel, which first defines the abstract syntax of the language, which is then used as the basis for the generation of concrete syntax, editors, validators, transformation engines, etc., until a complete environment in which to work on one or more DSLs is available [6,7].

Unfortunately, the adoption of a metamodel-based approach helps to increase one of the common problems of current MDE tools: all principles of Human-Computer Interaction (HCI) [8] have been virtually ignored to date in building new solutions. In particular, the definition of visual notations has basically consisted of the arbitrary assignment of graphic symbols to the concepts that made up the abstract syntax of the new language [9].

This scenario is explained by the same fact that has previously influenced other emerging disciplines: at the beginning, the objective of providing solutions that worked in order to prove that the new discipline deserved attention and recognition prevailed over the consideration of aspects related to the quality of those solutions. This is actually another example of the classic dichotomy between time-to-market and quality. However, since the MDE has already reached certain levels of maturity [10], the time has now come to begin to consider aspects of quality in the development of new model-driven tools and proposals [11].

In order to contribute in this line, this paper presents `CEViNEdit`, a tool based on GMF (Graphical Modeling Framework) [7] that uses the principles of Moody's Physics of Notations [12] to support the development of graphic editors for visual DSLs, taking into account the cognitive effectiveness of a notation, i.e. the speed, ease, and precision with which visual notations can be processed by the human mind [13]. To do this, `CEViNEdit` also adopts a metamodel-based approach, but it helps developers in the process of assigning graphical representations to the concepts that compose it, according to the Physics of Notation theory [12]. The underlying idea is that the design of visual notations and the selection of graphic elements should be based (as far as possible) on some theoretical basis and on some empirical evidence of their cognitive effectiveness, rather than merely following certain good practices and conventionalities [14]. In addition, two experiments have been run in order to evaluate the proposal. The first experiment serves to compare the use of the proposal with regard to existing tools in order to show its main advantages in terms of speed, automation level and usability. The second serves to analyse the possibilities of the proposal in terms of providing guidelines for the creation of visual notations more closely aligned with the principles of the Physics of Notation theory.

The remainder of the paper is organised as follows: Section 2 presents a brief introduction to the research context, in which the main concepts used throughout the paper are clarified. Section 3 reviews the main characteristics of `CEViNEdit`, whose use is illustrated in Section 4 by means of a use case and the results of of an empirical study, based on this use case, conducted to compare `CEViNEdit` with two other tools: GMF and EuGENia; Section 5 discusses the main limitations of the current version of the tool and provides some directions for further work; Section 6 presents some works related to this article, and finally, Section 7 summarises the main conclusions derived from this work and highlights the major contributions of this paper.

2 Research context

Before exploring the contents of this paper, we consider it necessary to provide accurate definitions of some of the terms used throughout this work. As shown in Figure 1, every DSL is defined upon a metamodel. The metamodel collects the abstract syntax of the language that specifies the vocabulary of concepts or language elements provided by the language and how they may be combined to create models, whereas the meaning of those concepts and their connections is referred to as the semantics of the language [15].

Furthermore, the concrete syntax provides a notation that facilitates the presentation and construction of models or programs in the language. Two main types of concrete syntax are typically used by languages: textual syntax and visual syntax. The latter is commonly referred to as visual notation in the literature related to cognitive effectiveness [16]. The visual notation of a DSL consists of a set of graphical symbols and a set of composition rules. Graphical symbols are used to represent the concepts collected in the metamodel.

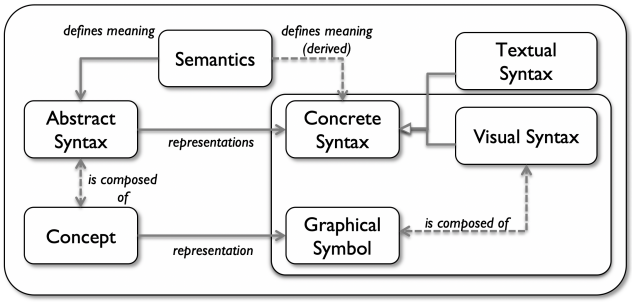

Fig. 1 Syntaxes and semantics of a DSL (adapted from [6])

The tool presented in this work, `CEViNEdit`, is intended, among other things, to define the concrete syntax of a DSL, and more specifically to define the visual syntax, which is composed of a series of graphic symbols. In summary, the tool focuses on the concepts shown in the inner box on the right-hand side of Figure 1.

As mentioned above, `CEViNEdit` also uses the Moody's Physics of Notations theory [12], which establishes nine principles with which to design, evaluate, compare and improve visual notations. These principles were defined from theory and empirical evidence gathered from different disciplines such as: cognitive and perceptual psychology, graphic design and cartography.

Each of the principles of the Physics of Notations contains: design strategies that may contribute to improving visual notations regarding that principle; a different evaluation procedure or metric that can be used to compare different notations; and examples of notations that satisfy or violate the principle.

The nine principles are summarised as follows:

1. Principle of **Semiotic Clarity**: there should be a one-to-one correspondence between elements of the language and graphical symbols.
2. Principle of **Perceptual Discriminability**: different symbols should be clearly distinguishable from each other.
3. Principle of **Visual Expressiveness**: the full range and capacities of visual variables should be used.
4. Principle of **Semantic Transparency**: the appearance of visual representations should suggest their meaning.
5. Principle of **Complexity Management**: explicit mechanisms to deal with complexity should be provided.
6. Principle of **Cognitive Integration**: explicit mechanisms to support the integration of information from different diagrams should be provided.
7. Principle of **Dual Coding**: text must be used to complement graphics.
8. Principle of **Graphic Economy**: the number of different graphical symbols should be cognitively manageable.
9. Principle of **Cognitive Fit**: different visual dialects for different tasks and audiences should be used when needed.

In this context, we believe that it is also important to introduce the concept of **visual variables**, a set of elementary building blocks that can be used to graphically encode information, which are often used and referenced in each of the principles of Moody's theory.

Studies conducted on the nature of graphical symbols have identified eight different visual variables (see Figure 2). The most important work in this regard is the seminal work of Bertin [17], which is considered to be to graphic design what the periodic table is to chemistry.

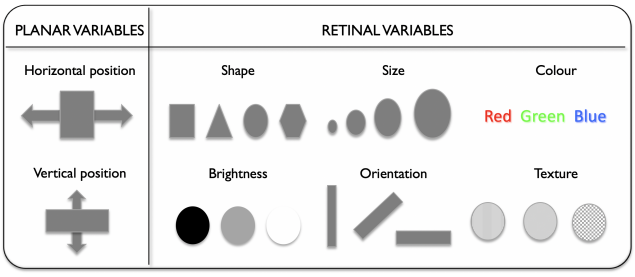

Fig. 2 Visual variables used to construct notations (adapted from [17])

Each of these visual variables has a set of properties that are used to encode certain types of information, and these properties must, therefore, be acknowledged if effective choices are to be made.

3 Automation and improvement of the process of developing a GMF-based graphical editor

The development of graphical editors based on GMF is an arduous and complex task, mainly owing to the learning curve of this technology, signifying that it takes a lot of time to use it successfully [18].

`CEViNEdit`, the tool presented in this paper, has been designed with the objective of increasing the abstraction level at which the designer works, with the consequent impact in terms of productivity, ease of use and shorter development times. This tool proposes an improvement that is related to the findings of a literature review of tools that provide the possibility of generating graphical editors from a domain model [9]. This review enabled us to discover that none of the current proposals consider aspects of quality/usability in the model-driven development of graphical editors. Furthermore, `CEViNEdit` aims to help improve the development process supported by GMF as regards building graphical editors. This improvement is oriented in two main directions: facilitating the production of visual notations that are cognitively effective according to some of the theoretical principles and empirical evidence proposed by Bertin [17] and Moody [12] and increasing the level of automation of the development process of the graphical editors.

3.1 Technological basis

This section presents the most relevant technologies employed during the development of the technological solution presented in this paper. In this respect, `CEViNEdit`, like many of the proposals that enable the generation of graphical editors from a metamodel, is a tool based on EMF (Eclipse Modelling Framework) [19] and GMF (Graphical Modelling Framework) [7]. A graphic summary of the technological dependencies of `CEViNEdit` is shown in Figure 3.

The main technological basis of `CEViNEdit` is obviously Eclipse, since the tool itself is a plug-in of this development environment. In fact, the graphic interface of `CEViNEdit` was constructed using libraries such as the SWT (Standard Widget Toolkit) and JFace, which are internal libraries of Eclipse. `CEViNEdit` also uses the EMF as a metamodelling framework, which is a requirement for GMF, since the product that it generates is an editor based on this framework with a technological dependence on Epsilon [20], a family of languages and tools that supports different means of managing or processing models.

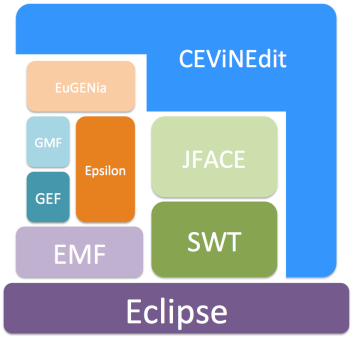

Fig. 3 Development environments on which CEViNEdit is based.

3.2 Technical design of CEViNEdit

CEViNEdit has been constructed in order to support a process focused on automating the creation of a graphic editor and the production of cognitively efficient visual notations, thanks to the framework proposed by Bertin [17] and Moody [12].

This has been achieved by using a model-driven development approach, with the objective of increasing the level of abstraction and reducing development times.

A summary of the design technique of CEViNEdit is shown in Figure 4, in which the components of CEViNEdit are represented with a blue rectangle.

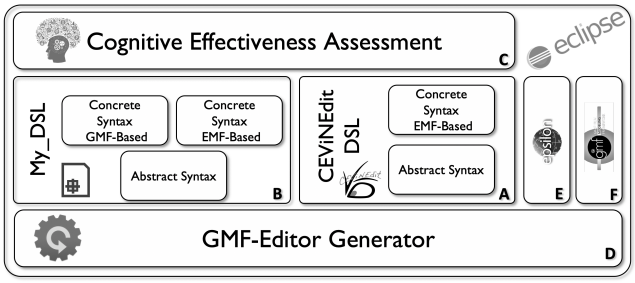

Fig. 4 Technical design of CEViNEdit

We first developed a DSL, which is the basis of the tool (Figure 4.A). The concrete syntax of this DSL was defined using the facilities provided by EMF in order to semi automatically generate tree-view editors. This DSL, in essence, abstracts two concepts: the graphic elements that can be used as building blocks when designing the GMF-based editors (nodes and links) and the visual variables defined in a renowned study on graphic semiology [17].

The models created with the CEViNEdit DSL make it possible to store the information related to the graphic representation of the elements from the domain model (Figure 4.B) for which we wish to generate a graphic editor based on GMF.

Once the DSL had been designed, a generic plug-in of Eclipse was generated and used as the basis on which to develop the entire tool. Moreover, a series of Eclipse libraries were used in order to generate a dynamic interface for the tool. The specific libraries used were SWT, since it enabled the construction of graphic interfaces in Java, which can be used on any platform, and JFace, a set of widgets with which to create user interfaces built on SWT.

Furthermore, in order to evaluate the cognitive effectiveness of the visual notation the user designs by assigning representations of the concepts of the metamodel or the domain model, CEViNEdit has a component denominated as *Cognitive Effectiveness Assessment* (Figure 4.C), which is based on a series of its own libraries created to evaluate some of the principles of Moody's Physics of Notations [12].

Finally, CEViNEdit has a component called *GMF-Editor Generator* (Figure 4.D) for the automatic generation of the editor. This uses the technology provided by Epsilon, which employs a domain model with EuGENia annotations and EOL code as a basis on which to automatically generate a GMF editor (Figure 4.E and Figure 4.F).

The main components of CEViNEdit are presented in the following subsections: we first present the development environment with which the user is provided, after which the development process supported by the tool is described. The DSL metamodel, which is the backbone of this process, is then presented, and finally some technical aspects are detailed

3.3 Development environment of CEViNEdit

One of the objectives of CEViNEdit was to provide developers with a tool that did not require previous knowledge in order to be capable of developing GMF-based editors.

We, therefore, designed a dynamic user interface based on panels developed using SWT and JFace, which show or hide controls according to the user's actions at each moment, and whose intention is to provide a simple and intuitive user experience.

Moreover, the graphic representation of the various elements in the domain model was defined so as to avoid, or at least minimise to the greatest possible extent, the possibility of users making mistakes, since at no time should they handle any of the models or transformations involved in the editor generation process. The level of abstraction is raised to the point at which all the design decisions that the user should make are articulated in various predefined pullouts. In order to illustrate these ideas, Figure 5 shows an extract of the CEViNEdit user interface when employed to develop a DSL in order to model file systems, a simple case study

commonly used in the documentation that accompanies EuGENia[1].

- The first panel (5.A) shows the domain model created for the case study.
- The second panel (5.B) shows the `CEViNEdit` model, which stores the relationships among the abstract syntax (domain model) and the concrete syntax of the DSL for which the editor is being developed.
- The third panel (5.C) is a dynamic panel that shows the various personalisation options on the basis of the type of graphic element selected.
- The fourth panel (5.D) shows contextual help related to the influence that the visual variable selected has on the cognitive effectiveness of the notation that is supporting the editor at that moment.
- Finally, the top tool bar (5.E) includes the controls that allow the generation process of the graphic editor to be invoked. This bar also contains the controls that make it possible to obtain information regarding the cognitive effectiveness of the visual notation defined by the user until that moment, in accordance with the principles defined in Moody's theory of the Physics of Notations [12].

3.4 Development process of graphical editors with `CEViNEdit`

Before presenting the development process of graphical editors supported by `CEViNEdit`, this section briefly describes those supported by GMF and EuGENia in order to ease the comparison. Visual summaries of these processes are provided in Figures 6 and 7.

GMF: Starting from the metamodel of the DSL (an *.ecore* model), EMF generates a Java API with which to handle conforming models, a tree-view editor for them, and some templates in order to write tests. An intermediate *.genmodel* model collects the options for this code generation step, which can be modified by the user.

Next, 3 models are needed to define: the elements of the visual syntax of the DSL (*gmfgraph*), the properties of the palette of the graphical editor (*gmftool*) and the correspondences between the elements of the metamodel and the two aforementioned models.

Again, an intermediate model (*gmfgen*) is then generated to set and collect the properties for the subsequent code generation step. All these models are then taken as input by GMF in order to generate the code that implements the graphical editor, which can be packaged as an Eclipse plugin (my_dsl.diagram) that can be imported and run into Eclipse.

EuGENia: This tool serves to reduce the learning curve when generating GMF-based graphical editors. This signifies that, unlike GMF, EuGENia merely needs to receive an annotated version of the Ecore metamodel as input in order to generate a fully functional editor. It is worth noting that the intermediate GMF models (*gmfgraph*, *gmftool* and *gmfmap*, plus *gmfgen*) are automatically generated in the background by the tool.

For more complex customisations, EuGENia allows the user to modify the automatically generated models, while some configuration options (such as the design of a compartment) can be established only in the *gmfgen* model.

Creation and use of the `CEViNEdit` model

The development process supported by `CEViNEdit` is illustrated in 8. Before using `CEViNEdit` it is necessary to have previously created the domain model (Ecore metamodel) using some kind of appropriate tool. As a first step, this tool is then used to create a `CEViNEdit` model within a general Eclipse Project. The first user interface is a simple panel that should be used to specify the path on which the domain model will be located. This domain model can be located in the Eclipse workspace itself or anywhere in the file system of the computer. The user then employs contextual menus to define, at a very high level, which of the graphic symbols that support GMF (diagrams, nodes, links, containers, etc.) will be used to represent the elements defined in the domain model for the user who wishes to define a visual notation and its respective graphic editor. The tool assistant takes the restrictions of GMF into consideration and, therefore, shows only possible values based on the element of the domain selected.

To return to the case study, the domain model in Figure 5.A shows how the class dominated as file system will correspond with the drawing panel (`gmfdiagram`); the *File*, *Drive*, *Folder* and *Shortcut* classes will be represented by means of a node (`gmfnode`), and finally, the Shortcut.target references and the Sync class will be shown as links (`gmflinks`).

This information is contained in the `CEViNEdit` model presented in the second panel of the tool (see Figure 5.B), and its metamodel is presented in the following section.

According to the type of graphic element selected in the `CEViNEdit` model (Node, Link or Compartment), the third panel of the tool (see Figure 5.C) shows a list of visual variables with pullouts containing the possible values for each variable. The user can, therefore, define the main characteristics of the concrete syntax using annotations with a high level of abstraction and in a simple and intuitive manner. In order to facilitate this task further still, the fourth panel of the tool (see Figure 5.D) shows contextual help each time the user selects a visual variable. This help contains information related to the work of Moody [12] and Bertin [17], with the objective of enabling the user to assign a value to each variable, which will maximise the cognitive

[1] EuGENia GMF Tutorial: `eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/`

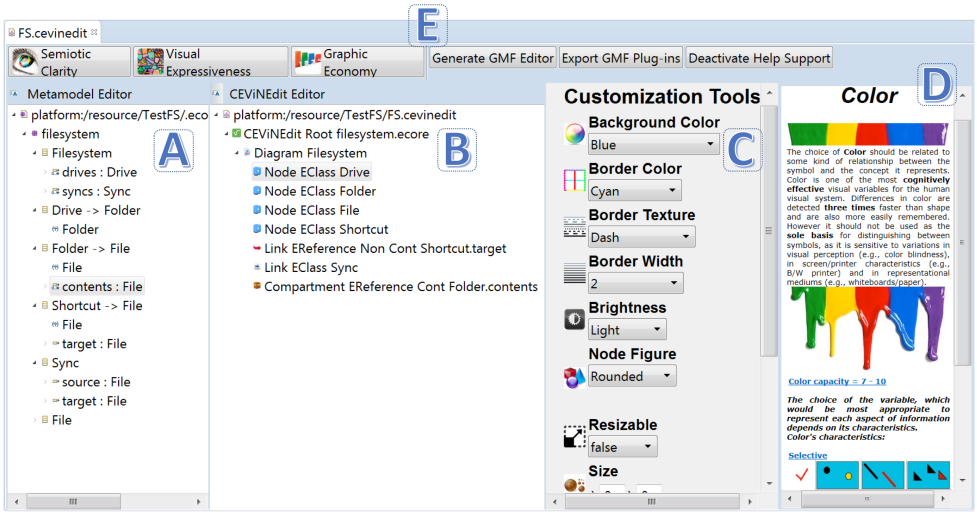

Fig. 5 Extract from the user interface of `CEViNEdit`

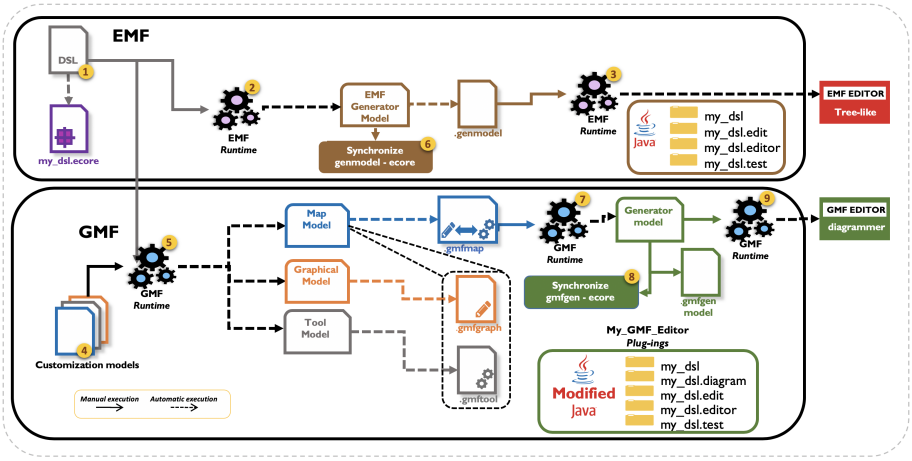

Fig. 6 Overview of the development process implemented by GMF

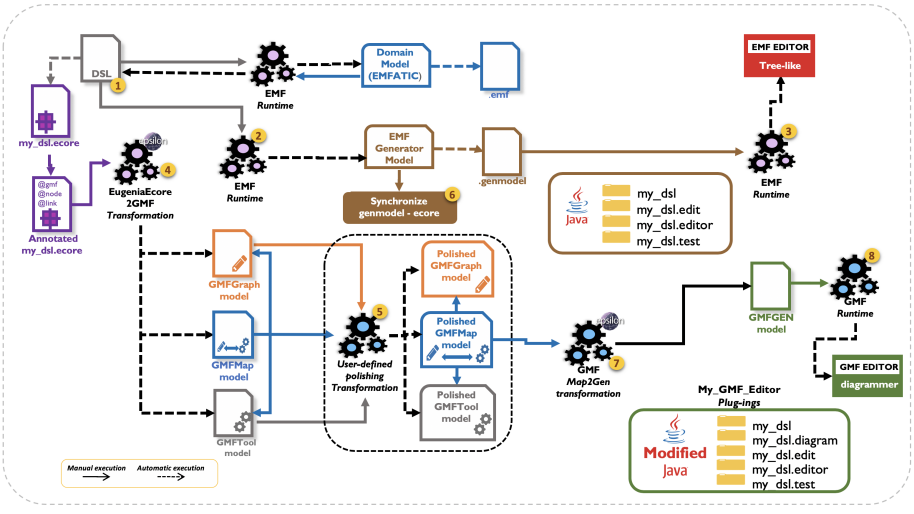

Fig. 7 Overview of the development process implemented by EuGENia

effectiveness of the visual notation.

Evaluation of cognitive effectiveness

Prior to generating the code that implements the editor, users can automatically evaluate the cognitive effectiveness of the visual notation resulting from the combination of their different design decisions. To that end, the tool produces several evaluation reports on how the notation aligns with three principles of the Physics of Notation theory. The details related to the evaluation of each of these principles are shown in section 3.7.

Generation of the code that implements the editor

Once the designer/user has defined the visual notation of the DSL, the remainder of the process is completely automatic, since all the information required to generate the code used by the editor is stored in the `CEViNEdit` model.

`CEViNEdit` generates an annotated copy of the domain model internally. These annotations (*EAnnotations*) of the GMF contain the design decisions regarding the visual aspect of the graphic elements comprising the notation. The annotated copy of the domain model is one of the inputs with which EuGENia is provided in order to obtain a completely functional editor. For a better visualisation of the entire process, we have created a screencast, which can be consulted at: `kybele.es/cevinedit`

In the process of generating the code that implements the editor, an instance of the object *EAnnotation* is first created, after which it is necessary to verify whether that instance should be converted into an annotation for a link (*gmf.link*) or for a node (*gmf.node*) with its respective key-value pair (*Entry<key, value>*), which will be transformed into GMF annotations. These annotations are added to their respective classes in the copy of the metamodel, as shown in Figure 9 and Figure 10. The former shows the original metamodel and the annotated metamodel in an tree-view editor, while the latter shows the same metamodels but in textual form. By way of illustration, the *Drive* elements have been highlighted in both cases in order to show the differences when the GMF annotations are added to the elements from the metamodel.

The use of an annotated copy of the metamodel makes it possible to avoid contaminating the original metamodel with information that does not correspond to the problem domain, thus maintaining a conceptual distinction between the abstract syntax and the concrete syntax of the DSL.

In order to support personalisation problems that cannot be contained in the annotations regarding the metamodel, `CEViNEdit` also employs EOL code [20] to automatically generate a file through the use of the user's design decisions that could not be transformed into annotations, and which are stored in the `CEViNEdit` model. A more detailed description of the generation process of this file is shown in section 3.6.

The `CEViNEdit` model can, therefore, use the facilities provided by EMF to create the annotated metamodel and the EOL file that will serve as the input with which to internally invoke EuGENia [18].

Advanced modification of the code that implements the editor

The final aspect to consider in the entire process is the modification of the Java code generated by EuGENia. This occurs when users decide to use their own (jpg, png or gif) images to graphically represent the instances of an element in their domain model. This personalisation cannot be expressed by means of annotations in the metamodel or by using EOL code and it is, therefore, necessary to later modify the Java code generated. The information required to carry out this type of modifications can also be found in the `CEViNEdit` model. In this context, it is worth mentioning that EuGENia supports only SVG images.

`CEViNEdit`, therefore, enables the user to define a visual notation for a domain model through the use of dynamic panels, drop-down lists and simple buttons and then automatically evaluates this notation which a selection of three principles from the Physics of Notation theory. This may contribute to the evaluation of the cognitive effectiveness of that visual notation. Finally, `CEViNEdit` also enables the automatic generation of a graphical editor by implementing the notation without the need to master GMF models or its code generation process.

Given all of the above, and having detailed the development process for graphical editors supported by `CEViNEdit`, the main advantages of our proposal can be summarised as follows:

- The complete automation of the development process.
- The ease of use of the tool.
- The ease of customising the visual notation.
- The support of a methodological basis on which to assess the visual quality of the notation.

3.5 The `CEViNEdit` metamodel

As mentioned in section 3.4, `CEViNEdit` makes it possible to create models in which to store information regarding the graphic representation of the elements in the domain model for which the user wishes to create a graphic editor. In order to do this, and as part of the development of `CEViNEdit`, we have defined a DSL whose metamodel is presented in Figure 11 and which, in essence, defines the graphic elements that can be used as building blocks during the design of GMF-based editors.

This metamodel was defined and implemented using EMF. When used with a metamodel, EMF makes it possi-

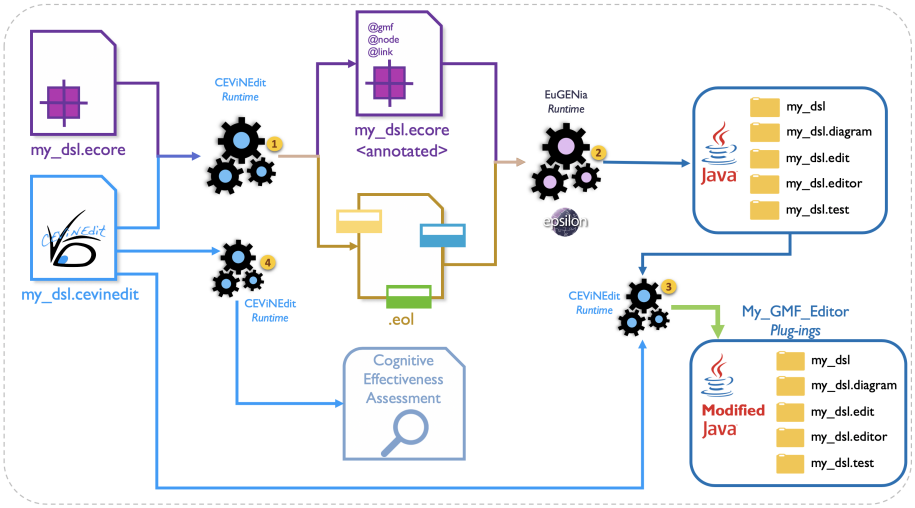

Fig. 8 Overview of the development process implemented by `CEViNEdit`

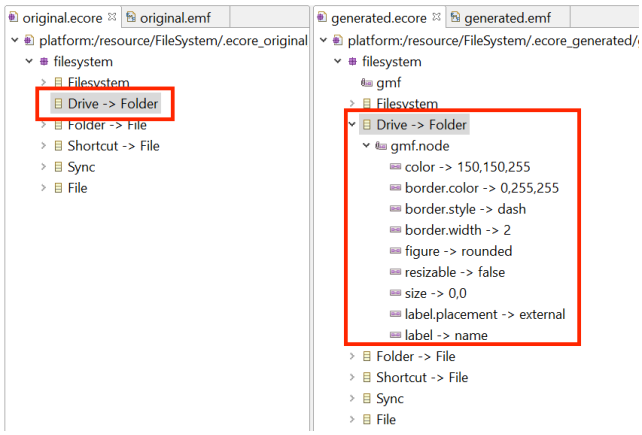

Fig. 9 Original domain model vs. Annotated domain model - Tree view

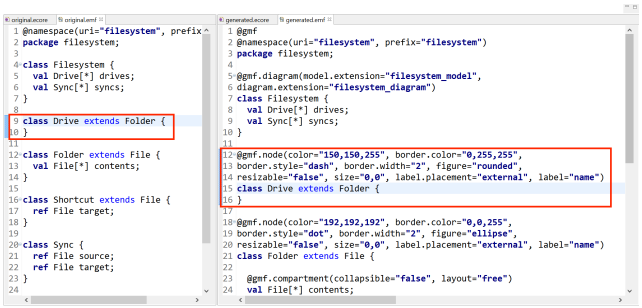

Fig. 10 Original domain model vs. Annotated domain model - Textual view

ble to generate different software artefacts for the editing and management of models in line with the metamodel.

In order to support the interconnection between the abstract syntax and the concrete syntax, the `CEViNEdit` models contain a root class (*CEViNEditRoot*) which has the only attribute in which the root of the file containing the Ecore domain model is stored, thus enabling it to define a visual notation. This element will contain a Diagram type object that will in turn contain the elements used to model which of the GMF elements will be used to represent each of the concepts of the domain model:

- `NodeEClass`: node inside a GMF diagram
- `Link`: link or connection between nodes. This element can be specialised in two sub-classes:
  - `LinkEClass`: link generated from an `EClass`
  - `LinkEReferenceNonCont`: link generated from an `EReference`
- `CompartmentEReferenceCont`: reference to the nodes of the metamodel that will act as containers of other nodes
- `AffixedEReferenceCont`: reference to those nodes that can be located adjacent to other nodes (on the perimeter or the edge of other nodes).
- `LabelEAttribute`: labels used on the elements of the diagram

These elements can also be graphically personalised through the use of various enumerated types, since each category (nodes or links) admits different values for its visual variables.

The basis employed to choose the different enumerated types of lists and the range of values that they admit was the method proposed in Bertin's theory regarding Graphic Semiology [17]. The information obtained from these values allows the cognitive effectiveness of the developers' design decisions to be evaluated, thanks to the use of some of the theoretical principles from the Physics of Notations proposed by Moody [12]. The lists selected, and their respective values, are the following:

- `Colour`: black, blue, cyan, grey, green, orange, red, white and yellow

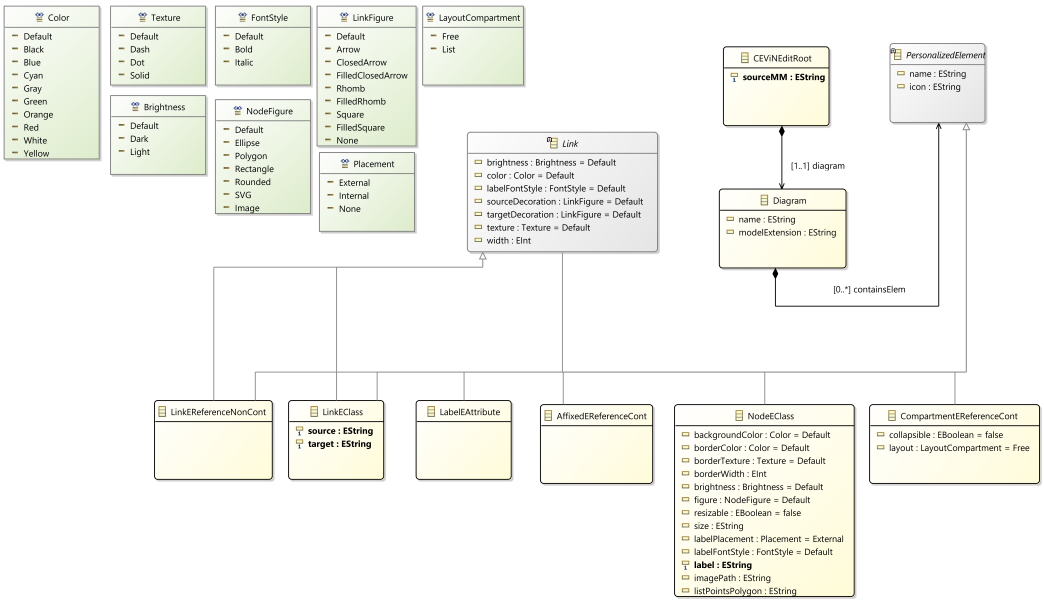

Fig. 11 `CEViNEdit` metamodel

- `Texture`: dashes, dots and solid line
- `Brightness`: dark and light
- `Font style`: bold and italics
- `Link shape`: open arrow, closed arrow, filled closed arrow, rhombus, filled rhombus, square, filled square and none
- `Node shape`: ellipse, polygon, rectangle, rounded rectangle, SVG and user's own image
- `Position`: external and internal
- `Container arrangement`: free and list style

The values in these lists allow the definition of the visual characteristics of the notation that can be personalised using `CEViNEdit`. In order to implement these characteristics at a technical level, `CEViNEdit` uses the values from the lists with the objective of generating annotations supported by EuGENia [18], which are later aggregated with the annotated metamodel. Moreover, as mentioned in the previous section, `CEViNEdit` supports other visual characteristics that are implemented thanks to the automatic generation of EOL [20] and Java code, such as the font style in boldface and italics and the use of the developer's own images to represent a node. These visual characteristics are grouped into three categories depending on the graphic element to which they belong: nodes, links or containers (nodes that can contain other nodes).

3.6 Advanced personalisation options

The first developed version of `CEViNEdit` was created with the intention of automatically generating Ecore metamodels annotated with EuGENia annotations. It would, therefore, be possible to employ all the advantages of this tool without the user having to understand the internal mechanism of the manual annotations by using a graphic panel to raise the level of abstraction.

Once this milestone had been attained, we then decided to support further graphic personalisation options that could not be carried out by means of annotations in the metamodel.

For example, EuGENia does not provide annotations with which to personalise the graphic appearance of *Label* objects, such as the text style, be it bold or italics. Since it is not possible to make these visual modifications through the use of metamodel annotations, an auxiliary file called Ecore2GMF.eol [21] coded with EOL is needed to introduce these design decisions. We, therefore, developed a mechanism used to automatically generate the EOL code which had previously been used for the polishing of intermediate GMF models (*gmfgraph*, *gmftool* and *gmfmap*).

Moreover, for those personalisations in which the user's own images (jpg, png or gif) are used to graphically represent the instances of an element in the domain model, we have developed another mechanism that enables modifications of the Java code implemented by the graphic editor.

3.7 Cognitive effectiveness evaluation

In addition to the improvements made by `CEViNEdit` with regard to the role of the models and the level of automation, one of the other objectives pursued when developing this tool was to support the evaluation of certain aspects related to the quality of the visual notations generated, a need that has long been recognised by the MDE community [11].

CEViNEdit, therefore, integrates mechanisms that allow the evaluation of the design decisions made by the user, in accordance with three of the nine principles from the theory of Physics of Notations [12]. A brief description of these mechanisms is provided below. It should be noted that our proposal does not support the assessment of all the principles. For example, Cognitive Integration applies only when multiple diagrams are used to represent a single system. This type of problem goes beyond the scope of our application, since this principle evaluates the cognitive effectiveness of systems in which different types of diagrams are used. However, we are currently working on the evolution of the tool in order to expand the number of principles supported.

3.7.1 Evaluation of Semiotic Clarity

The first of the principles that can be automatically evaluated is Semiotic Clarity, which, according to Goodman's theory of symbols [22], establishes that four types of anomalies regarding the correspondence between the elements of a language and their respective graphic symbols can exist:

- *Redundancy*: various graphic symbols are used to represent the same element of the language (*sinographs*)
- *Overloading*: the same graphic symbol is used to represent different elements of the language (*homographs*)
- *Excess*: this occurs when a graphic symbol does not represent any of the elements of the language
- *Deficit*: an element is not represented by any graphic symbol

The current development process of the tool prevents, a priori, anomalies related to redundancy and excessive appearing. On the one hand, redundancy is avoided because CEViNEdit does not allow more than one graphic symbol to be assigned to the same element in the domain model while, on the other, excess is avoided because it is not possible to define a graphic symbol without previously having assigned an element to the domain model. The report generated by CEViNEdit will, therefore, show the results of possible anomalies related exclusively to overloading and deficit. This report indicates the name of the elements that have these anomalies, thus allowing the user to quickly address the problems detected. In the case of overloading, the tool analyses all the graphic symbols and identifies as homographs those that are exactly the same. That is, those whose visual variables take the same values. Finally, for the detection of the deficit anomaly, all the elements of the domain model are analyzed, detecting those with no graphic representation assigned to them.

As an example, Figure 12 shows the report generated when evaluating the Semiotic Clarity of the visual notation of the DSL used to model the file systems, a simple case study commonly used in the documentation that accompanies EuGENia.

This report identifies the graphic elements of the domain model to which no graphic representation has been assigned. In this particular example, '*drives*' and '*syncs*' (Figure 12.A) lead to an anomaly with regard to deficit. This report also shows the homographs that are present, i.e. those elements that have the same graphic representation. Note also that the links '*Sync*' and '*target*' have this anomaly (see Figure 12.B).

3.7.2 Evaluation of Visual Expressiveness

The second principle that can be automatically evaluated is Visual Expressiveness, which can be briefly defined as the number of visual variables efficiently used by a visual notation.

In general, the notations used in Software Engineering tend to employ a limited range of these variables. For example, of the possible values of the shape variable, the rectangle is by far that most used in the majority of modelling languages. Nevertheless, rectangles are not very efficient if our objective is to facilitate the visual recognition process. In these cases, it would be more convenient to use other types of shapes, such as curves, 3D, or to directly use icons. The choice of visual variable used and its respective value should not, therefore, be an arbitrary decision, but rather a decision related to the type of information that one wishes to codify [23].

For the evaluation of this principle, it is necessary to take the different visual variables into account, along with their respective capacities; that is, the number of different levels that the human mind can distinguish for each variable (see Table 1).

Table 1 Capacity of visual variables

Variable	Capacity
Horizontal pos.	10-15
Vertical pos	10-15
Size	20
Brightness	6-7
Colour	7-10
Texture	2-5
Shape	Unlimited
Orientation	4

The Visual Expressiveness report produced by CEViNEdit in this context provides information related to the number of values used by each visual variable, its capacity and its saturation, i.e. the correlation between the number of values used and the capacity of each variable. This relationship illustrates to what extent the visual variable is efficiently used with the aim of avoiding, amongst other things, a saturated diagram as regards the number of colours.

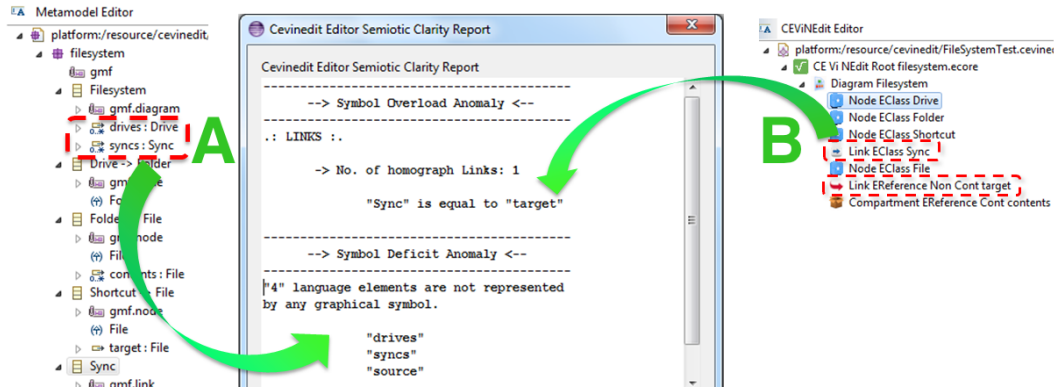

Fig. 12 Extract from the report on the evaluation of the Semiotic Clarity

Figure 13 shows a report on the model editor for the file system mentioned earlier. This report includes the metrics calculated for certain visual variables, such as colour, texture and brightness. These data show the quantity of values used for each visual variable, their respective capacity according to studies on graphic semiology [17], and the current saturation of each of these variables. It will, for example, be noted that eight different colours are used in the visual notation being evaluated and, bearing in mind that this variable has a capacity of between 7 and 10 values, a saturation level of between 80% and 114% percent is obtained. These figures indicate that it would be counterproductive to include many more values for this visual variable. It will also be noted that only one value is used for brilliance, which supposes that it has a saturation level of between 14% and 16%. These percentages indicate the possibility of using 5 or 6 other different values for this visual variable without exceeding the optimum saturation levels.

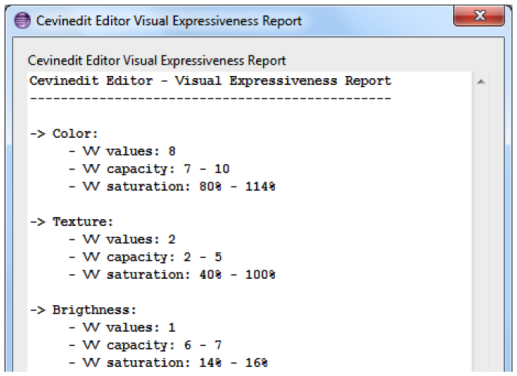

Fig. 13 Extract from the evaluation report on Visual Expressiveness

3.7.3 Evaluation of Graphic Economy

`CEViNEdit` also enables the evaluation of the Graphic Economy of the notation. The principle of Graphic Economy establishes that the strategy of providing a certain number of graphic symbols in a visual language is effective until the cognitive recognition process becomes an excessively complex task [24]. In fact, the human capacity to discriminate among the different alternatives perceived by the working memory consists of approximately six categories [24]. This number is, therefore, an upper limit for graphic complexity. However, modelling languages tend to increase their graphic complexity in an attempt to improve their semantic expressiveness. In order to reduce this problem, `CEViNEdit` informs the user of the graphic complexity of the visual notation being designed.

Figure 14, therefore, shows the report related to the graphic complexity of the visual notation of the file systems mentioned above.

The value obtained from the evaluation of this principle depends directly on the number of elements in the domain model to which a graphic representation is assigned. This can be observed in Figure 14, which shows that seven elements in the domain model correspond to an assigned graphic element.

3.7.4 On the evaluation of cognitive effectiveness

One of the objectives of this paper is not only to present a tool to support the evaluation of the cognitive effectiveness of visual notations, but also to promote interest in using a scientific basis to design, evaluate, improve and compare visual notations. In this respect, implementing a part of Moody's proposal does not ensure cognitively effective languages, but provides certain levels of confidence without compromising the balance between effort and reward when carrying out the type of assessment presented in this paper.

In this context, it should be noted that evaluating the accuracy and completeness of the Physics of Notation theory is not an objective of this work. In contrast, we aim to provide a methodological and technological framework with which to support a theory that has already been used to evaluate and

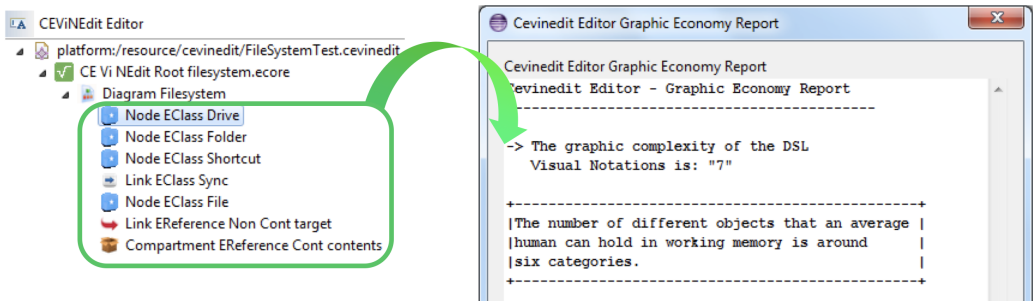

Fig. 14 Extract from the evaluation report on the Graphic Economy

propose improvements for existing visual languages and visual notations, such as ArchiMate [25], i* [26], BPMN [27], UML [28], UCM [29], Business Decision Modeling [30], SEAM [31] and WebML [32].

However, it is worth mentioning that there are certain interactions between the principles proposed by Moody. It is, therefore, important to master each of the principles and their possible interactions in order to avoid certain conflicts and exploit some advantages when designing modeling languages. Which principle should prevail in each case is a matter of decision making for which no generic statement can be made. Our tool performs an individual analysis of each of the three principles supported, signifying that designers should consider the data gathered in the context of their DSL. As an example, and regarding the principles evaluated by our tool, Semiotic Clarity can affect the Graphic Economy positively or negatively, since excess and redundancy anomalies can increase the graphical complexity, while overload and deficit anomalies could reduce it. In addition, increasing Visual Expressiveness reduces the effects of graphic complexity and Graphic Economy can define certain limits of Visual Expressiveness [12]. In fact, the issue of dealing with the trade-offs between principles, good practices, rules or whichever is the building-block of the framework, is inherently associated with the application of assessment frameworks.

One of the main flaws of Moody's Physics of Notations theory is indeed the fact that it does not consider the visual language as a whole but rather as a set of notation elements that are analyzed in an isolated manner. This point of view could lead to cognitively ineffective notations, despite the fact that all of Moody's principles were considered when designing the notation. Nevertheless, we still believe that this theory constitutes a useful tool with which to evaluate visual notations. This study has shown that it can be used to implement a set of metrics that are automatically computed and to generate reports on the assessment of visual notation. This information can then be used to identify possible flaws and to guide actions regarding those elements that have room for improvement. The resulting notation should, however, later be exposed to some kind of qualitative assessment involving domain experts and/or final users. The fact that the notation implemented by the editor aligns better with the Physics of Notations theory does not ensure that it yields a better editor.

4 Evaluation of the proposal

In the sphere of Software Engineering, as in many other disciplines, empirical methods are required to validate the theories proposed, evaluate the functionality delivered, identify the possible limitations and discover aspects for improvement [33]. The three empirical methods most used to that end are surveys, case studies and experiments.

This section, therefore, discusses the evaluation of the technological proposal presented in this work. As mentioned previously, the proposal is based on GMF and EuGENia, signifying that we have considered it appropriate to compare the functionality delivered by the three proposals as part of the evaluation.

In order to avoid bias related to the skills of the participants or the type of DSL used, these types of evaluations should be carried out with large-scale experiments. However, gathering a considerable number of participants would have gone beyond our capacity, since the objective of our proposal was a reduced niche of users (developers of graphical editors for DSLs). This is, indeed, a common problem in the field of Evidence-based Software Engineering [34], as has been stated in previous works including similar evaluations [18, 35].

In order to partly overcome these problems, we shall discuss the main findings of two experiments run with two small groups of participants in the following paragraphs (61 students and 12 researchers).

In the first, a group of students developed several graphical editors for the Entity-Relationship model using GMF, EuGENia and `CEViNEdit`. In this case, questionnaires and direct observation were employed to collect data that would enable the comparison of these tools. In other words, this experiment was focused on evaluating the functionality provided by the tool for the development of graphical editors.

In the second experiment, a group of researchers were provided with the reports generated by CEViNEdit on the cognitive effectiveness of the visual notation implemented by the graphical editor for the Entity-Relationship model. The researches, then, used this information to develop new versions of the editor in line with the guidelines provided by those reports. This experiment was, therefore, focused on the assessment of the capabilities of CEViNEdit regarding the provision of metrics to assess notations according to the Physics of Notations theory and the utility of those metrics to improve the notation.

We limited both experiments to a single modeling language (the Entity-Relationship model) in order to minimize the addition of accidental complexity to the evaluation process, since all the subjects involved mastered the model and the particular notation implemented in the experiments. As a matter of fact, KyBDele[2], a GMF-based diagrammer for conceptual modeling that implements a very similar notation has been used for the last five years on the introductory course concerning relational DBs on the CS degrees at the URJC. This signified that both the students involved in the first experiment, who had already passed the course; and the researchers involved in the second one, who taught the different editions of the course, were used to the notation. It should be noted, however, that we have recently developed GMF-based editors for more complex languages, such as PCN, Service Blueprint or BPMN [36,37]. All of them have also been used as test cases for the development of CEViNEdit.

4.1 Definition of the experiments

The empirical study designed to evaluate the technological solution proposed in this document is based on the use of a simplified version of the Entity-Relationship model proposed in [38]. This DSL was chosen by considering the widespread dissemination and acceptance of the Entity-Relationship model. In spite of the existence of many versions of this language, such as the Crow's Foot model [39] or the EER (Enhanced Entity-Relationship) model [40], we opted to use a simplified version of Chen's original version presented in 1976.

Figure 15 shows the metamodel of the simplified version of the Entity-Relationship model used, which comprises the abstract syntax of the DSL to be developed.

This metamodel is composed of Entities (Strong and Weak), Relationships (Relationship) and Attributes (Simple, Optional, PrimaryKey and Composite).

In view of the above, the study designed to evaluate the technological solution proposed in this paper was structured as follows:

- **Experiment 1** - The visual notation and the graphical editor of the Entity-Relationship model would be created with GMF, EuGENia and CEViNEdit.
- **Experiment 2** - Metrics regarding the cognitive effectiveness of that visual notation according to certain PoN principles would be computed automatically. Improved versions would be produced according to the recommendations of such metrics.

In order to answer the questions posed before carrying out the evaluation, as much information as possible had to be gathered from these experiments. In the specific case of this paper, the objective of the evaluation was to verify the correct performance of the functionalities provided by CEViNEdit, which would allow the definition of a visual notation for a DSL on the basis of its metamodel, the evaluation of the cognitive effectiveness of this notation and the generation of a GMF-based graphical editor that would implement it.

A set of evaluation questions (EQ) aligned with these objectives is consequently presented. Specifically, the first subset of five questions represent an attempt to discover out whether CEViNEdit is, in general terms, more productive than other similar tools, while a second subset of two questions focus on whether it is possible to achieve better visual notations following the indications provided by CEViNEdit. These questions are answered in section 4.4, in which this experiment is analysed, and are presented as follows:

EQ-1.1 Is it possible to use a previously generated domain metamodel or model with another tool as the input for the development of the editor?

EQ-1.2 Is it possible to use all the GMF-type graphic elements (*gmf.diagram*, *gmf.node*, *gmf.link*, etc.) as a graphic representation of the elements in the domain model?

EQ-1.3 Is it possible to decide the graphic appearance of each of the elements that would form part of the visual notation?

EQ-1.4 During the development process, does the user have to manually modify any of the intermediate models generated?

EQ-1.5 When the user automatically generates the graphic editor, is an executable product obtained? If not, what types of errors and shortcomings are identified?

EQ-2.1 Is it possible to evaluate the cognitive effectiveness of the visual notation defined? If so, to what extent is it possible to use that information to improve this aspect of the notation?

EQ-2.2 Does the tool provide assistance regarding the properties of each of the variables employed? To what extent is this kind of information useful?

[2] http://www.kybele.es/kybdele/

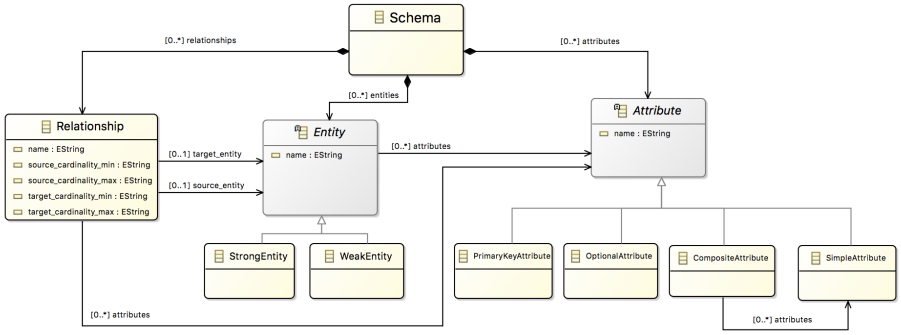

Fig. 15 Simplified metamodel of Entity-Relationship model

With regard to the data collection process used to obtain evidence with which to answer the questions above, we decided to use the following sources of information:

- Direct observation of the execution of the experiments.
- The models and code generated throughout the editor development process.
- The *log* generated by the tool on the basis of the user's activity.

4.2 **Experiment 1**: GMF vs EuGENia vs CEViNEdit

Section 3 showed a detailed description of the main technical characteristics of the tool developed in the context of this paper in order to generate graphical editors from a domain model and evaluate the cognitive effectiveness of the visual notations defined in accordance with Moody's Physics of Notations [12]. However, there are many tools with which to carry out the first task, i.e. to generate graphical editors using a metamodel. Some of these tools are GMF [7] and EuGENia [18].

While developing `CEViNEdit` we realised that the development of a graphical editor with the tool seemed to be simpler and faster that its development with other existing tools. In order to back up these impressions and gather data that could help to identify possible advantages or limitations of the proposal, an experiment was run.

The experiment consisted of developing, a simple graphical editor with GMF, EuGENia and `CEViNEdit`, like that shown in Figure 16.

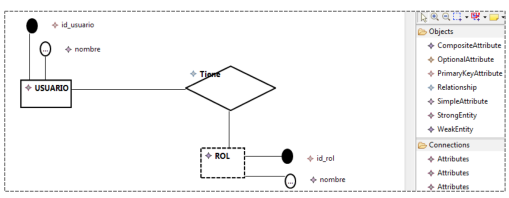

Fig. 16 ER graphical editor

This experiment was carried out in the context of an assessed practical class of a Bachelor's degree subject at the Universidad Rey Juan Carlos, Madrid (Spain).

As with any other experiment in this context, we followed certain guidelines which established the need to describe the approach, the material used, the method and the analysis of the results [41].

It should be noted that, in this experiment, we shall compare three tools that can be used for the development of editors for graphical DSLs: GMF, EuGENia and `CEViNEdit`. However, it is worth noting that they have considerable differences. The objective of GMF was to enable the model-based development of editors. EuGENia was built atop of GMF in order to maximize the level of automation in the development process. Finally, `CEViNEdit` goes further in the level of automation by providing the possibility of automating the introduction of detailed graphical customizing. Furthermore, `CEViNEdit` fosters the development of cognitive effective visual notations according to Moody's principles.

Despite these differences, the comparison made in the following experiment focuses on evaluating the three tools from a single point of view: their usability when developing graphical editors. In this respect, we have not targeted a modeling language that fits particularly well with the nature of the improvements provided by `CEViNEdit`; that is, the use of detailed customisations or the analysis of the cognitive effectiveness of visual notations. Instead, we have chosen a very simple use case, which should not, in theory, favour the characteristics of `CEViNEdit`.

4.2.1 Planning - Experiment 1

The experiment was carried out by following the recommendations and guidelines of [42]. Figure 17 shows a general view in which three principal components can be found: the subjects, the material and the analysis.

- The subjects who participated in this study were 61 Software Engineering students in the third year of their Bachelor's degree in Computer Engineering at the University of Rey Juan Carlos.

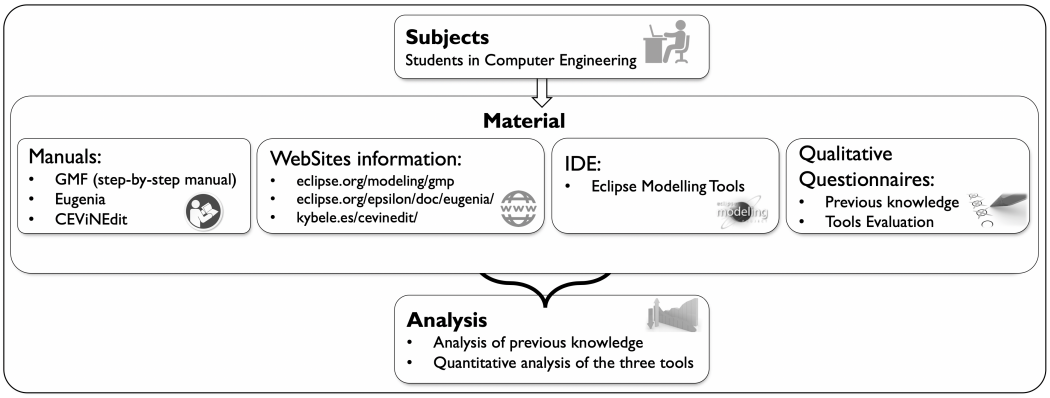

Fig. 17 Summary of the experiment

- The material used included the manuals for each tool, tutorials in the form of a video and a detailed guide to GMF containing all the information required to develop a graphical editor using that tool. We also provided a version of Eclipse into which it was necessary to install three tools and the metamodel used as a base input for the development of the editor. Finally, the tools were evaluated using two quantitative questionnaires.
- Two analyses were eventually carried out: one whose objective was to identify the participants' prior knowledge and another quantitative analysis used to process their answers to the questionnaire.

Each of the three components of the experiment is described in detail below.

4.2.2 Subjects - Experiment 1

As mentioned earlier, the subjects in this experiment were 61 Software Engineering students (number of students enrolled in that subject) in the third year of their Bachelor's degree in Computer Engineering at the University of Rey Juan Carlos.

The homogeneity of the participants meant that it was not necessary to make any kind of distinction among them or to place them in separate groups, as is fairly common in this type of experiment.

The requirements to participate were limited to being enrolled in the subject in question and having a basic knowledge of the use of Integrated Development Environments in general and Eclipse in particular. Knowledge of MDE was not a requirement, since this is not taught until Master's degree level at the University of Rey Juan Carlos.

4.2.3 Material - Experiment 1

The experiment was carried out in one of the computing laboratories at the University of Rey Juan Carlos. This laboratory has 75 Dell Optiplex GX280 computers, and both the Eclipse Luna Modelling Tools development environment and the 1.7 32 bit version of the Java Development Kit were installed in each computer beforehand.

The first material handed out to the subjects in the study was a document[3] containing a description of the practical work that they had to carry out.

The subjects were, in essence, requested to develop a graphical editor for the Entity-Relationship model using three different tools: GMF, EuGENia and `CEViNEdit`. In order to do this, they were provided with a graphic description and a brief explanation of the metamodel that they would use as a basis to develop the graphical editor, along with an illustration of the result expected after using each of the three tools by means of screen captures of completed editors. The students were also provided with the Ecore file containing the metamodel.

Before carrying out this practical work, the participants filled in a questionnaire[4] consisting of eight questions related to their prior knowledge on the use of modelling languages, software design, model designers and certain personal preferences in this context.

The participants were then given basic documentation taken from the official websites of each of the tools:

- **GMF**: `eclipse.org/modeling/gmp`
- **EuGENia**: `eclipse.org/epsilon/doc/eugenia`
- **CEViNEdit**: `kybele.es/cevinedit`

Owing to some problems related to the complexity of working with GMF which arose during some of the preliminary tests carried out for the experiment, we opted to provide each student with detailed guidelines of the steps required to develop a graphical editor with GMF, since the time available to carry each experiment out was limited, and this did not fit in with the GMF learning curve. The GMF manual

[3] Generation of GMF Graphical Editors for the Entity - Relationship Model: `kybele.es/cevinedit/EditorsPractice.pdf`

[4] First questionnaire - Knowledge and Preferences: `kybele.es/cevinedit/Q1.pdf`

given to the students is not included in this paper for reasons of space, since it is a document of some considerable size (49 pages)[5].

Having developed the editors, the participants then had to hand in the source code used by each of the three graphical editors which had been developed and respond to another questionnaire consisting of 18 questions that would enable them to give their opinions on different aspects of each of the tools, such as ease of learning and use, its graphical interface, development times, advantages and disadvantages and the problems encountered.

4.2.4 Results and analysis - Experiment 1

A brief analysis of the data obtained from the responses to the questionnaire provided by the students who participated in the study is presented in the following sections, in which particular emphasis is placed on the evaluation of the three tools used. The dataset containing all the subjects' responses is available online[6].

As mentioned previously, one of the first steps in this experiment consisted of giving a first questionnaire to each of the participants in order to identify their prior knowledge and some of their preferences in the context of the languages and software modelled.

The questions were basically formulated with the intention of identifying which software modelling languages the participants already knew, discovering some of their preferences when using a model designer and ascertaining whether they preferred working with graphical models or, on the contrary, directly with source code.

A summary of the responses provided by the participants for each of the eight questions in the questionnaire is shown in Table 2.

The data in Table 2 reveal that the modelling languages best known by the participants in this experiment are UML and E/R, whilst only a few of them knew DFD or SysML and none of them were familiar with or had worked with the i* language.

These data confirm that the participants are students with a basic knowledge of modelling languages and that the few languages they know of are fundamentally those commonly used in some of the subjects on their Computer Engineering degree. It was thus possible to ensure that there was a certain amount of homogeneity as regards the subjects' prior knowledge.

With regard to tools, the data revealed that all the subjects had prior experience of working with Gimp, a basic graphic editing tool, while only some of them had already used certain advanced tools such as FreeHand, Illustrator or Corel Draw. Furthermore, many of the participants had used some of the tools created for conceptual and/or logical models. Many of them had worked with Oracle Data Modeller and KyBDele, since these tools had been used in some of the subjects on their Computer Engineering degree.

Finally, almost all the participants stated that their preference for the use of designers was owing to their ease of use, and that they would prefer to struggle with models, diagrams and other abstractions than to do so with source code, although they were not clear as to whether this option was more efficient.

The second questionnaire[7], was used to attain the subjects' impressions after using GMF, EuGENia and `CEViNEdit` to develop three graphical editors for the Entity-Relationship model.

In the first five questions in the questionnaire, the participants had to provide a numerical value of between 1 (negative) and 5 (positive) regarding certain aspects of the three tools such as ease of learning, usability, interface functionality, benefits over the learning curve and the intuitiveness of the interface. From the results obtained after processing the evaluations provided by the 61 participants, an analysis of variance was carried out. This analysis has allowed us to obtain, among other things, the average rating for each of the 5 questions asked, as well as identifying the standard deviation (SD), which enabled us to discover how dispersed the data were with respect to the mean, as is shown in Table 3.

These results enabled us to verify that `CEViNEdit` generally stood out as regards usability when compared to GMF and EuGENia, and always obtained an average evaluation of over 3.5/5.

GMF obtained a lower average evaluation in the 5 sections, probably because of its complexity. Moreover, as mentioned previously, the participants in this experiment were provided with a detailed step-by-step manual in order to construct the graphical editor with GMF. The results of the evaluation of GMF would probably have been even worse if they had not been given this manual.

Since the evaluation was carried out using a quantitative questionnaire, it was appropriate to prove that the evaluations made by the participants are reliable from a statistical point of view. This was done by calculating an Intraclass Correlation Coefficient (ICC), which evaluates the reliability of the evaluations, comparing the variability of one single participant's evaluations with the total variability among all evaluations and all participants. The basic concept underlying the ICC was originally introduced in [43] as a special formulation of the Pearson's R when the measures and variances of the distributions implied are equal.

[5] Graphical editor development with GMF (*Spanish*): `kybele.es/cevinedit/GMFTutorial.pdf`

[6] Dataset DOI: `http://dx.doi.org/10.17632/ddcrpmrhbb.1#file-bdc524f5-e3b6-4f53-aca8-0c9ef69f2f0e`

[7] Second questionnaire – Tools Evaluation: `kybele.es/cevinedit/Q2.pdf`

Table 2 Results of the first questionnaire

Question 1 *Mark which of the following modeling languages you know*	UML 50	E/R 47	DFD 3	BPMN 9	i* 0	SysML 3	Other 0
Question 2 *Indicate which of the following modeling languages you have worked with*	UML 48	E/R 41	DFD 3	BPMN 6	i* 0	SysML 1	Other 0
Question 3 *Of the following design programmes, indicate which you have ever used*	Freehand 1	Illustrator 9	CorelDraw 3	Paint.NET 15	Gimp 61	None 8	Other 9
Question 4 *Have you ever used a data diagrammer (tool to create graphical models) to model software and/or databases?*						Yes 49	No 12

Question 5 *Specifically, which of the following tools have you ever used?*	OR-MYSQL 6	OR-SQL 39	ER-win 15	MS-SQL 6	KyB-Dele 45	Visio 10	Magic-Draw 9	R-Rose 3	None 6	Other 3

Question 6 *Which of the following characteristics do you consider most important for a data diagrammer?*	Easy to use 58	Efficient 25	No prior knowledge 21	Multi-purpose 19	Other 0

Question 7 *When developing a piece of Software, which of the following options do you prefer?*	Graphical 44	Source code 17

Question 8 *Do you think that the use of graphical tools is more efficient than directly editing the source code?*	Yes 35	No 12	I don't know 14

Table 3 Summary of the results of the quantitative questions

	GMF		**EuGENia**		**CEViNEdit**	
	Avg	SD	Avg	SD	Avg	SD
Ease of learning	2.49	1.043	2.79	0.897	3.54	0.959
Usability	2.77	1.216	2.97	0.983	3.73	1.094
Functionality of the GUI	2.87	1.087	3.03	0.930	3.72	0.951
Benefit learning curve	2.57	0.991	2.82	0.992	3.52	0.942
Intuitiveness of the GUI	2.73	1.031	2.98	0.885	3.86	0.826

The general idea is that the total variability observed in the evaluations of the three tools can be divided into three components: variability owing to the differences among the questions in the questionnaire ($var\beta$), variability owing to the differences among the evaluators ($var\epsilon$), and a residual (random) variability associated with the error inherent in any measurement ($var\alpha$). In the case of this experiment, the ICC is defined as the proportion between the variability of the questions and the total variability:

$$c = \frac{var\beta}{var\beta + var\epsilon + var\alpha} \tag{1}$$

The guidelines provided in [44] were followed in order to calculate ICC estimates and their 95% confident intervals using the analysis of variance (ANOVA) procedure from MS Excel Analysis ToolPak (Microsoft Corporation, Redmond, WA) based on single rater/measurement, absolute-agreement, 2-way random-effects model, which is equivalent to using the following formula:

$$\frac{MS_R - MS_E}{MS_R + (k-1) + \frac{k}{n}(MS_C - MS_E)} \tag{2}$$

This formula for estimating the ICC is one of the ten forms proposed by McGraw and Wong [45], where MS_R = mean square for rows; MS_E = mean square for error; MS_C = mean square for columns; *n* = number of subjects and *k* = number of raters/measurements. The choice of the appropriate form was made by taking into account the guidelines proposed in [44] regarding the selection of the most appropriate model, type and definition according to the characteristics of our experiment.

Once the analysis of variance (ANOVA) had been executed on the set of 61 responses to each of the 5 quantitative questions of our experiment, we obtained the values required in order to calculate the ICC corresponding to each of the three tools evaluated, as summarized in Table 4.

The values of this coefficient may vary between 0 and 1, where 0 indicates an absence of any type of agreement and 1 indicates complete consistence and an absolute agreement

Table 4 Summary of the Intraclass Correlation Coefficient values

GMF - ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critic*
Rows	298,2098361	60	4,970163934	24,10015898	1,97077E-73	1,374555394
Columns	5,704918033	4	1,426229508	6,915739269	2,76647E-05	2,409256601
Error	49,49508197	240	0,206229508			
Total	353,4098361	304				
					ICC	**0,8081201**

EuGENia - ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critic*
Rows	224,5508197	60	3,742513661	22,71198806	6,71331E-71	1,374555394
Columns	2,852459016	4	0,713114754	4,327640524	0,002129051	2,409256601
Error	39,54754098	240	0,164781421			
Total	266,9508197	304				
					ICC	**0,8046023**

CEViNEdit - ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F critic*
Rows	232,9114754	60	3,881857923	21,93885114	1,95874E-69	1,374555394
Columns	5,13442623	4	1,283606557	7,254478073	1,57033E-05	2,409256601
Error	42,46557377	240	0,176939891			
Total	280,5114754	304				
					ICC	**0,7915937**

of the results. ICC values lower than 0.4 generally represent low reliability, while those between 0.4 and 0.75 represent good reliability and those over 0.75 represent excellent reliability [46]. It will, therefore, be noted that all the values obtained for the three tools evaluated are close to 0.8, which indicates that the subjects' responses have a high degree of reliability.

These ICC values allow us to reject the theory that the responses to the questionnaire were made in a random manner. The coincidence in the variability of the evaluations can probably be explained to a certain extent by the fact that the participants had a very similar profile and the same prior knowledge. It is also important to state that the questionnaires were not anonymous and that the development of the experiment had repercussions on the evaluation of the subject in question, and it is, therefore, unlikely that the participants gave random responses.

Furthermore, the answer to another set of questions in the questionnaire allowed us to identify the average time spent developing the graphical editor with each of the three tools. Likewise, in order to identify the dispersion of the data, we obtained the standard deviation of the time spent using each of the three tools and we have carried out an analysis of the quartiles of the data obtained in this regard, which has allowed us to obtain the interquartile range (IQR): the distance between the first quartile (Q1) and the third quartile (Q3). In this respect, it should be noted that the IQRs are quite high, reflecting notable differences in the time spent by some of the subjects in the experiment. These data are summarised in Table 5.

Table 5 Summary of the analysis of the development time of graphical editors

(*hours*)	**GMF**	**EuGENia**	**CEViNEdit**
Average	14.78	14.10	11.44
SD	20.75	13.72	12.02
IQR	27	23.66	23

It is again important to state that the participants had a detailed manual that provided a step-by-step explanation of all the tasks to be carried out when using GMF. Without the help from this manual, they would have taken much longer when using this tool. In fact, in one of the tests carried out by a doctorate student, approximately 28 hours were required to develop a GMF editor for the Entity-Relationship model.

It is obviously worth highlighting that, in this case study, the generation of the graphical editor when using `CEViNEdit` required quite a lot less time on average (11.44 hours).

The next block of questions similarly enabled us to identify the main problems encountered by the subjects. The main problems regarding GMF were related to difficulty of use, lack of intuitiveness and the time required to develop the graphical editor. The problems related to EuGENia were similar, but to a lesser extent, while those related to `CEViNEdit` were principally associated with the complexity of installation and the errors found when executing it. These problems can be explained by the fact that `CEViNEdit` is a prototype and, therefore, has some incompatibilities when executed on certain versions of certain operative systems or different versions of the Java virtual machine.

With regard to the advantages that each tool has over the others, GMF stands out because of its wide range of personalisation options (31 responses) and the greater level of control that the user has over the different development phases of the editor (25 responses), whilst EuGENia stands out because of the development time required and its ease of use when compared to GMF (33 responses). Finally, `CEViNEdit` stands out because of its ease of use (43 responses), its intuitiveness (30 responses) and the speed at which it is possible to generate a graphical editor (27 responses), which makes it the subjects' preferred option (35 votes, as opposed to 10 for EuGENia and 16 for GMF).

Finally, the participants were asked whether, after carrying out the experiment, they believed that the development of software using a model-based approach was better than when using a manual programming approach. 30 of them opted for the use of models, 16 stated that they preferred the programming approach and 15 had no clear opinion in this regard.

4.3 **Experiment 2**: visual quality improvements

As mentioned throughout this paper, one of the main functionalities of our proposal is the possibility of evaluating the visual quality of the notations defined by the users according to some of the principles of the Physics of Notations theory.

The tool specifically provides the user with information on the graphic economy, the semiotic clarity and the visual expressiveness of the notation. In other words, it analyses the overall number of elements of the notation, the number of identical elements, the number of metamodel elements without a graphic representation and the amount of colours, textures and brightness of the graphic elements with their respective saturation.

In order to assess these functionalities, we carried out another small-scale experiment with 12 members of the Kybele research group[8], to which some of the authors belong).

The experiment consisted of the subjects analysing the 3 reports generated by the tool for the case study of the previous experiment (the graphical editor for the Entity-Relationship model) shown in Figure 18. We asked the subjects to modify the visual notation as required in order to improve it according to the information collected in those reports and to later generate a new graphical editor.

As with the previous experiment, we shall now briefly describe the approach, the materials used and the analysis of the main results obtained.

4.3.1 Planning - Experiment 2

The experiment was carried out following the recommendations and guidelines of [42]. It is, therefore, necessary to clearly specify the three main components of this experiment:

- The subjects who participated in this experiment were 12 researchers who are members of the Kybele research group (University Rey Juan Carlos).
- The material used included the tool manual, a brief presentation of the cognitive effectiveness of visual notations and a graphical editor for the Entity Relationship model generated before the experiment.
- Finally, an analysis of the results was carried out. Its objective was to identify whether the subjects had been able to generate new graphical editors that included improvements as regards resolving the anomalies identified in the reports generated by the tool.

Each of these components are described in detail below.

4.3.2 Subjects - Experiment 2

As mentioned earlier, the subjects in this experiment were 12 researchers, who are members of the Kybele research group. Although the group has 29 members, the reduced number of subjects in the experiment was owing exclusively due to the number of people available to perform this activity.

Given the homogeneity of the subjects (all of them have a PhD in Computer Science in the field of Software Engineering), no distinction or subgroups were made, as is fairly usual in this type of experiments.

4.3.3 Material - Experiment 2

The experiment was carried out in one of the group laboratories, containing 75 Dell Optiplex GX280 computers. Both the Eclipse IDE (Luna Modelling Tools package) and the Java Development Kit (1.7 32 bit version) were installed on each computer beforehand.

The participants received basic documentation taken from the official website of the tool `kybele.es/cevinedit`, a graphical editor of the Entity-Relationship model previosuly developed with the tool and a brief presentation[9] on the cognitive efficiency of visual notations, in which each of the 9 principles of the Physics of Notations theory were explained, with special emphasis on the 3 principles supported by the tool. This information is also available in the contextual help of `CEViNEdit`.

4.3.4 Results and analysis - Experiment 2

As mentioned previously, a simplified version of the Entity-Relationship model was used in both experiments to ensure that its complexity would not hinder the understanding and analysis of the results.

Bearing all of the above in mind, the initial reports of the graphical editor used in this experiment (see Figure 18) mainly reflected a series of anomalies related to 2 homographs (the same graphic symbol is used to represent different elements of the language) and a limited use of the amount of visual variables employed. In particular, only 2 different values were being used for colours, 4 for textures and 1 for brightness.

With regard to the Visual Expressiveness of the notation, the subjects relied on the help provided by the tool panels to modify different visual aspects, which resulted in an increase in the amount of values of visual variables used. The saturation was consequently within the limits proposed in the Physics of Notation theory.

[8] Members of the Kybele research group: `kybele.es/es/miembros/`

[9] Cognitively Effective Visual Notations: `kybele.es/cevinedit/VNPrinciples.pdf`

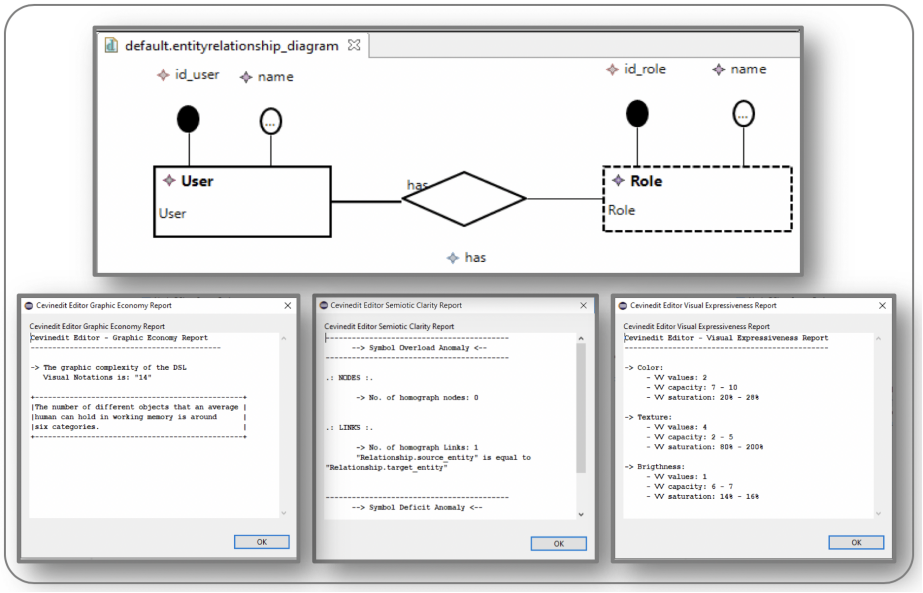

Fig. 18 Reports and graphical editor for the Entity-Relationship model

As an example of the results obtained, Figure 19 shows some of the graphical editors developed by the participants in the experiment. Figure 20 shows the new reports that were generated. They reflect the changes and improvements mentioned above. Note that the anomaly related to the presence of homographs has been eliminated. Note also the increase in the number of values of the visual variables used with their respective saturation.

With this experiment, we do not intend to claim that the visual notations for the Entity-Relationship model defined by the participants in the experiment are better than the original notation proposed by Chen [38], just as we cannot state that a new UML notation addressing some of the anomalies detected in [28] would be better than the current UML notation. It is clear that the level of adoption of the original notation is of greater importance when considering the goodness of a given notation. Our objective was, therefore, merely to show that the tool helps with the definition of notations more in accordance with the guidelines proposed in the Physics of Notations theory.

4.4 Discussion

Having described the experiments and their results, we can now provide an answer to each of the questions posed during the definition of the experiments. To that end, we rely on the analysis of the information obtained through the use of the questionnaires and our direct observation.

On the one hand, experiment 1 has allowed us to answer the first 5 questions posed, as shown below:

EQ-1.1 Is it possible to use a previously generated domain metamodel or model with another tool as the input for the development of the editor?
Yes. The initial assistance allows the domain model, created beforehand in an Ecore format, to be loaded. However, no prior verification is carried out, which may lead to errors in the selection phase of the domain model elements that will have a graphic presentation.

EQ-1.2 Is it possible to use all the GMF-type graphic elements (*gmf.diagram*, *gmf.node*, *gmf.link*, etc.) as a graphic representation of the elements in the domain model?
Yes. Moreover, the available options are restricted according to the nature of the elements in the domain model for which their representation is being defined.

EQ-1.3 Is it possible to decide the graphic appearance of each of the elements that would form part of the visual notation?
Partially. Although the tool provides a series of drop-down lists that allow the graphic appearance of the elements of which the visual notation is composed to be modified, `CEViNEdit` does not, for example, support the generation and/or modification of figures composed of various layers of graphics, thus implying that only flat figures can be defined. The range of possible values for the different variables is also limited to a far fewer number than those supported by the GMF annotations.

EQ-1.4 During the development process, does the user have to manually modify any of the intermediate models generated?
No. At no time does the user have to handle or modify the models or software artefacts generated throughout

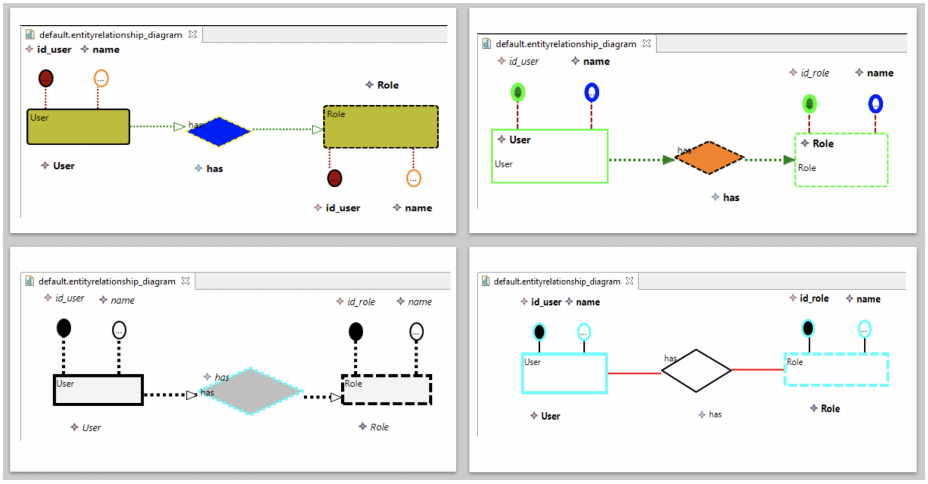

Fig. 19 Examples of modified graphical editors

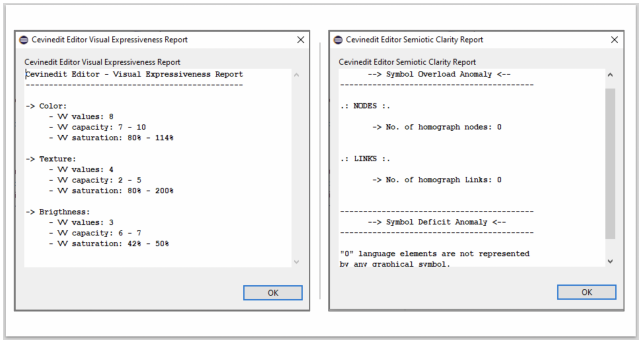

Fig. 20 Examples of reports from new graphical editors

the process. The user must only make a series of decisions at a high level of abstraction or invoke the evaluation of the cognitive effectiveness or the generation of the editor.

EQ-1.5 When the user automatically generates the graphical editor, is an executable product obtained? If not, what types of errors and shortcomings are identified?
Yes. The editor generated is fully functional and supports the creation of diagrams with the DSL in question. The plugins are generated automatically, signifying that `CEViNEdit` serves to reduce the probability of the user introducing errors when packaging the code that implements the editor. The former manual steps are now automated by the tool. Nevertheless, it was found that `CEViNEdit` does not help to personalise the tool palette of the graphical editor generated. The artifacts related to this personalisation depend on the intermediate GMF model (the so called gmftool), which `CEViNEdit` does not handle at any time.

On the other hand, experiment 2, which was related to the evaluation of the visual quality of the notations, allowed us to answer the last 2 questions posed during the definition of the experiments:

EQ-2.1 Is it possible to evaluate the cognitive effectiveness of the visual notation defined? If so, to what extent is it possible to use that information to improve this aspect of the notation?
Partially. The tool makes it possible to evaluate the cognitive effectiveness of the resulting visual notation on the basis of the metrics of three of the nine principles of Moody's Physics of Notation [12]. Moreover, the information obtained is simply descriptive and provides information that can be used to improve quality, but does not support any mechanism that automates the application of the improvement.

EQ-2.2 Does the tool provide assistance with regard to the properties of each of the variables employed? To what extent is this kind of information useful?
Yes. The tool provides, in a dynamic manner, information about each of the visual variables being modified at each moment. This help contains information regarding the correct use and capacity of the visual variable. Nevertheless, once this information and the principles related to cognitive effectiveness are known in depth, it may be unnecessary to consult it. There is, therefore, the possibility of showing or occluding the help panel.

The execution of the experiments has enabled us to provide satisfactory responses to the majority of the questions posed and to identify some of the principal limitations of `CEViNEdit`, which has provided us with some lines of future work.

It should be noted that a set of tests with different and more complex DSLs was carried out. For instance, a simplified version of the Process Chain Network (PCN) metamodel (23 elements between nodes, links and compartments) [47] was used. These internal tests served to verify the absence of failures in the generation of graphical editors, along with the ability to generate the assessment reports regarding the cognitive efficiency of the notation.

4.5 Threats to validity

In the following, we discuss the main threats to the validity of this study, according to the classification shown in [48] and following the order proposed in [49], which can be understood as advancing from the experiment details to the more general landscape.

Construct validity refers to the relationship between the theoretical aspects taken into account in the experiment and the observations obtained from it. In fact, the first version of the paper included just one experiment, which was used to compare `CEViNEdit` with existing tools for the model-driven development of graphical editors. It was, however, suggested, that the experiment in question could not be used to draw conclusions related to the ability of the tool to help in the definition of cognitively effective notations. A second experiment with MDE researchers was consequently run. This was specifically intended to prove that `CEViNEdit` can be used to automatically measure the cognitive efficiency of the notation implemented by a GMF editor and to improve the refinement of the editor with these purposes in mind.

Indeed, the focus of the second experiment might somehow constitute another threat to construct validity. In fact, our aim here was mainly to show that the Physics of Notations theory could be implemented thus enabling the provision of tool-support with which to assess existing notations and to guide the improvement to the notation regarding the principles implemented. This is inherently different from stating that the resulting notation is better. Complementary experiments should be run to check whether the improvements made result in better editors in terms of ease of use or model comprehension. At the end of the day, one experiment cannot be used to test all the ramifications of a given idea.

Internal validity is related to causality; that is, how sure we can be that the data processing led to the result obtained. This is traditionally one of the greatest threats to validity. The first experiment consisted of creating a graphic editor with three different tools (GMF, EuGENia and `CEViNEdit`), which undoubtedly implies that the subjects accumulated some experience as the experiment progressed. In order to mitigate this learning effect, the subjects were provided with a detailed step-by-step guide for the development of the editor with GMF, whereas they lacked any help with the rest of tools. This GMF guide was, in some respects, an attempt to make up for the experience the students had when confronted with the development of the editor with `CEViNEdit`.

The subjects were not separated into different groups in the experiments, since we considered the expertise of the students and researchers involved to be homogeneous. None of the students had previous experience with any of the tools or concepts related to the experiment. The previous lectures on MDE were, in fact, given to ensure that all the students shared a basic understanding of the main concepts of MDE. Furthermore, all the researchers involved in the second experiment shared a common understanding of the main purposes of `CEViNEdit`, but had no practical experience with the tool since it was the first prototype developed. Indeed, any researchers directly related to `CEViNEdit` were excluded from the experiment in order to avoid the participation of subjects familiarised with the tool. Nevertheless, we should acknowledge that the separation of subjects into different groups would have helped, for instance, to minimize the learning effect mentioned above (for example, assigning different groups to each tool) and to mitigate some of the threats reported here.

The fact that a very simple editor was chosen as the target of both experiments might also be considered a threat to internal validity. As mentioned in the paper, we have also used `CEViNEdit` for the development of graphical editors for more complex DSLs, but we aimed to minimise the amount of accidental complexity. This also, allowed us to mitigate the fatigue threat for students since MDE concepts are not immediate for students. With regard to the second experiment, the development of graphical editors for complex DSLs is not a trivial task, even for experienced researchers, and would have distracted us from our main objective: assessing whether `CEViNEdit` could help to produce cognitively effective notations. However, we do acknowledge that complex case studies should be developed and documented for future versions of the tool.

Last, the experiments were controlled in order to minimise confounding factors: all the subjects were in the same room during each experiment. They used very similar equipment and the same software tools. Moreover, the fact that the subjects were SE students from our course and researchers from our group meant that we were acting against demotivation issues that may arise when SE practitioners are involved (if it is possible to involve them). In addition, all of the students enrolled on the course took part in the experiment (for instance, we did not involve those interested in the topic or those with better marks), and all the members of our research group were also involved (except those discarded because of their familiarity with `CEViNEdit`).

Conclusion validity is related to the security of having obtained a specific result based on the treatment of the data that we have used in the experiment. We are fairly convinced that the statistical analysis performed in this study is correct. In fact, it was greatly extended in the second version of the paper. These amendments make us more confident of the fairness of the analysis. Likewise, the questionnaires used in both experiments served to measure exactly what we aimed to measure. The results of the questionnaires, along with the editors produced by the subjects during the experiments, actually showed that the tasks had been well understood. We can, therefore, conclude that misunderstanding issues were

not a threat to the experiments. Nevertheless, to favour reproducibility and transparency, the dataset containing all the subjects' responses is available online, thus allowing the data analysis to be replicated by any interested reader.

Finally, **external validity** refers to how generalisable the results obtained are when researching other settings. Unfortunately, as already discussed, the size and the nature of both groups of subjects were limited, and it is not, therefore, possible to state that our findings can be generalised to other contexts. In particular, SE practitioners will probably lack previous knowledge on Model-Driven Engineering, in contrast with the two groups of subjects involved in our experiments. This implies that we cannot conclude that the use of `CEViNEdit` improves the user experience for any type of user, nor state that its use will always result in better editors. In fact, with regard to the scope of the experiments, the applicability of the proposal is currently constrained to the research contexts in which this type of tooling is currently being applied. We cannot derive conclusions regarding industrial practice without running new experiments with different set-ups.

5 Current limitations and further work

The previous section provided a description of the process carried out to evaluate the proposed technology presented in this paper. This has been done by means of a case study, which has allowed us to verify that all the components in the proposal function correctly and to identify the improvements made in comparison with similar proposals through the use of empirical evidence and statistical data.

In addition to the improvements made, this evaluation has also allowed us to identify a series of limitations, which are described as follows:

- One of the main limitations of `CEViNEdit` is that many of the graphic personalisation options supported are implemented by means of the annotations supported by EuGENia and it, therefore, depends to a great extent on the underlying framework, Epsilon. Nevertheless, and as mentioned previously, `CEViNEdit` implements some personalisation options by means of the later modification of the Java code used by the editor. This approach could be adopted for all the personalisation options, thus reducing this dependency.
- As occurs with the EuGENia annotations, `CEViNEdit` does not support the generation and/or modification of graphic figures composed of various graphic layers, which implies that it is only possible to define flat figures. This problem could be solved by using SVG images or figures, but their use may not be sufficient to cover all needs of all users.
- It does not currently support the personalisation of the tool palette of the generated graphic editor. Nevertheless, it is feasible to adopt the same approach and provide the users with some intuitive controls that will allow them to express design decisions on the tool palette of the editor to be generated.
- The current version of `CEViNEdit` does not allow an evaluation of the cognitive effectiveness of those GMF-based graphic editors that have not been generated with `CEViNEdit`. Another addition, which is currently in its development stage, is, therefore, being carried out in order to include a mechanism that will enable a `CEViNEdit` model to be generated from a *gmfgrapf* model obtained from one of the existing editors.
- As discussed in general in section 4.5, as the development of the tool progresses in future versions, new experiments could be carried out that would include some aspects not been taken into account in the experiments presented in this paper. In this respect, and specifically with regard to experiment 2, it would be interesting to carry out some additional types of tests in which a comparison is made between the results obtained by each subject when implementing some of the suggestions provided by our tool, given that in the current experiment each subject made the improvements they considered appropriate based on the evaluation of an cognitive effectiveness of the visual notation generated, but no comparison was made between the results obtained by the different subjects of the experiment.
- Moreover, with regard to the evaluation of the cognitive effectiveness of the visual notation, `CEViNEdit` obtains metrics for only three of the nine principles in the Physics of Notations. Moreover, the information obtained is simply descriptive. We are, therefore, working to develop an extension to the mechanism that will evaluate the cognitive effectiveness, with the objective of not only detecting the problems related to the visual notation, but also providing a possible solution to those problems, which can be automatically expressed in the visual notation.
- Finally, this work introduces a prototype that will be refined in subsequent versions in order to deal with the technical issues that have been detected to date. For instance, in order to avoid polluting the source metamodel, an annotated copy is produced during the development of the graphical editor. This duplication might lead to some problems, signifying that the way in which this potential pollution is avoided must be reconsidered. Some errors with the JVM might also appear when the tool is run over some particular versions of some OSs. We are now working to provide a more stable version that will address this type of issues.

6 Related works

In this section, we briefly present some related works. After conducting a systematic mapping study of the literature on the model-based tools used to generate editors for visual DSLs [9], we identified that the major tools are: DiaGen [50], EuGENia [21], DSL tao [51], GMF [7], Graphiti [52], MetaEdit+ [53], Obeo Designer [54], Sirius [55] and Tiger [56]. Although some other tools were identified during the development of the study, a number of them have been deprecated or integrated into other frameworks, such as TopCased [57] or Poseidon [58]. Note also that this is indeed a never-ending task, since new proposals appear every so often, such as metaBUP [59] or EMF-Stencil [60].

Furthermore, there are several studies in which one or several tools are analysed, such as [61–67]. Most of these studies are focused on presenting a proposal that can be used to generate editors that support a DSL. More specifically, some works, such as [61,62], 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 decision to present a state of the art divided into these two categories was 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 [63], the authors present a comparative study of tools for model-driven development and, more specifically, a comparison between 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, [64] presents a comparative study of five 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, analysing each of the five tools.

In addition, in [65] the authors present, as previously mentioned in other works, a tool that can be used to generate editors for DSLs, but in this case, the tool generates textual and not graphical editors. However, in this work, a comparison is made with other tools for the generation of textual editors, such as xText or TEF.

Likewise, [66] 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, [67] presents a comparison of a set of metamodeling languages such as ARIS, Ecore, GOPPRR, GME or MS DSL Tools, which employ the heavyweight metamodelling approach [68] and are available as tools.

In conclusion, these works are mainly focused on presenting one or several tools in detail, and showing only some differences among those and other existing proposals. According to the analysis of these works, none of the tools presented in these papers consider quality aspects during the development of visual editors, unlike the tool presented in this work. This might be related to the fact that most of these tools were built by experienced developers not used to dealing with Human-Computer Interaction issues.

As a final consideration, it is worth mentioning that there are a number of existing theories and frameworks for notation analysis in literature, apart from the Physics of Notations theory, such as the paper by Krogstie, Sindre and Jorgensen on semiotic quality (SEQUAL) [69]; the set of guidelines of modeling (GoM) by Schuette and Rotthowe [70]; or the cognitive dimensions (CDs) proposal, which was first introduced by Green [71] and whose application to the analysis of the usability of visual programming environments presented by Green and Petre [72] yielded the Cognitive Dimensions framework. There are even some works highlighting and addressing some shortcomings in the Physics of Notations theory [73]. However, please note that it was not our intention to evaluate those frameworks here. We merely aimed to provide a tool that would enable the application of the Physics of Notations principles to the development of graphical editors, taking for granted that those principles were correct. We point the reader to [32] for a discussion on existing frameworks for notation analysis.

7 Conclusions

In this paper, we have presented a tool called CEViNEdit, which supports the model-driven development of graphical editors, thus enabling the definition of cognitively effective visual notations. This tool allows users to generate these editors in a semi-automatic and intuitive manner. In addition, CEViNEdit automatically provides users with metrics that can be employed to evaluate the speed, ease and accuracy with which the visual notation that is being generated may be processed by the human mind [13].

Furthermore, in this paper we have presented an experiment with 61 students, who were asked to build a graphical editor for the Entity-Relationship model, using three different tools: GMF, EuGENia and `CEViNEdit`. This experiment has helped us to contrast the fact that, in comparison with GMF and EuGENia, the use of `CEViNEdit` results in shorter development times, a shorter learning curve and, in general, better usability levels of the tool. However, the results of this experiment have also allowed us to identify that there is room for improvement as regards some minor aspects of the tool, which are mainly related to compatibility with different installed versions of Java Virtual Machines and the appearance of the GUI in different operating systems.

Likewise, we have presented a second experiment in which 12 researchers analyzed the reports generated by our tool on the graphic economy, semiotic clarity and visual expressiveness of a visual notation. This experiment showed that the reports were found useful to identify some anomalies and to introduce a number of possible improvements related to the cognitive effectiveness of the visual notation.

We consider that the experiments conducted serve to illustrate and support the proposal by showing that it improves the state of the art and that it is feasible, at least to some extent, to exploit Moody's Physics of Notations when building visual editors. However, more experiments would help to broad the scope of the proposal and to deal with some of the threats discussed in the paper. These experiments are already planned to take place as soon as the pandemic situation in which we currently find ourselves as we write this paper comes to an end.

Beyond these results, to the best of our knowledge, `CEViNEdit` is the first tool to consider aspects of quality/usability in the model-driven development of graphical editors. This may be because most of these tools have been built by developers with a technical background but without in-depth experience of Human-Computer Interaction or cognitive issues [8]. However, it is worth noting that the use of `CEViNEdit` is by no means sufficient to assert that the visual notation of the graphical editor produced poses no problems, but it does provide certain levels of confidence.

In this respect, this work has served to show the considerable improvement that can be made to the process of generating a graphical editor that supports a domain specific language in terms of automation and quality assessment of the editor. However, we believe that there is much room for improvement, on which we are currently working.

Acknowledgements This research has been partially funded by the Regional Government of Madrid under the SICOMORo-CM (S2013/ICE-3006) project and the ELASTIC (TIN2014-52938-C2-1-R), MADRID (TIN2017-88557-R) and MINECO/FEDER FAME (RTI2018-093608-B-C31) projects, financed by the Spanish Ministry of Economy, Industry and Competitiveness.

References

1. Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
2. Jean Bézivin. In search of a basic principle for Model Driven Engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.
3. Debasish Ghosh. DSL for the uninitiated. *Communications of the ACM*, 54(7):44–50, 2011.
4. Jeff Gray and Bernhard Rumpe. Models for the digital transformation. *Software and Systems Modeling*, 16(2):307–308, 2017.
5. Douglas C Schmidt. Model-Driven Engineering. *Computer-IEEE Computer Society*, 39(2):25, 2006.
6. Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-Driven Software Engineering in Practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.
7. Richard C Gronback. *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009.
8. Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. Industrial adoption of Model-Driven Engineering: Are the tools really the problem? In *International Conference on Model Driven Engineering Languages and Systems*, pages 1–17. Springer, 2013.
9. David Granada, Juan M Vara, Francisco Pérez Blanco, and Esperanza Marcos. Model-based Tool Support for the Development of Visual Editors-A Systematic Mapping Study. In *Proceedings of the 12th International Conference on Software Technologies, ICSOFT 2017*, pages 330–337, 2017.
10. Markus Volter. From programming to modeling-and back again. *IEEE software*, 28(6):20–25, 2011.
11. Bran Selic. What will it take? A view on adoption of model-based methods in practice. *Software & Systems Modeling*, 11(4):513–526, 2012.
12. Daniel L Moody. The Physics of Notations: a scientific approach to designing visual notations in software engineering. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 2, pages 485–486. IEEE, 2010.
13. Jiaje Zhang and Donald A Norman. Representations in distributed cognitive tasks. *Cognitive science*, 18(1):87–122, 1994.
14. C. Wheildon, D. Ogilvy, and G. Heard. *Type and Layout: Are You Communicating Or Just Making Pretty Shapes?* Worsley Press, 2005.
15. Tony Clark, Paul Sammut, and James Willans. Applied metamodelling: a foundation for language driven development. *arXiv preprint arXiv:1505.00149*, 2015.
16. Stephen J Morris and OCZ Gotel. Flow diagrams: rise and fall of the first software engineering notation. In *International Conference on Theory and Application of Diagrams*, pages 130–144. Springer, 2006.
17. Jacques Bertin. Semiology of graphics: diagrams, networks, maps. *University of Wisconsin press*, 1983.
18. Dimitrios S Kolovos, Antonio García-Domínguez, Louis M Rose, and Richard F Paige. EuGENia: towards disciplined and automated development of GMF-based graphical model editors. *Software & Systems Modeling*, 16(1):229–255, 2017.
19. Frank Budinsky, David Steinberg, Raymond Ellersick, Timothy J Grose, and Ed Merks. *Eclipse Modeling Framework: a developer's guide*. Addison-Wesley Professional, 2004.
20. Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. The Epsilon Object Language (EOL). In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 128–142. Springer, 2006.
21. Dimitrios S Kolovos, Louis M Rose, Saad Bin Abid, Richard F Paige, Fiona AC Polack, and Goetz Botterweck. Taming EMF and GMF using model transformation. In *International Conference on Model Driven Engineering Languages and Systems*, pages 211–225. Springer, 2010.

22. Nelson Goodman. *Languages of art: An approach to a theory of symbols*. Hackett publishing, 1976.
23. Gerald Lee Lohse. A cognitive model for understanding graphical perception. *Human-Computer Interaction*, 8(4):353–388, 1993.
24. George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.
25. DL Moody. Review of ArchiMate: The road to international standardisation. *Report commissioned by the ArchiMate Foundation and BiZZDesign BV, Enschede, The Netherlands*, 77, 2007.
26. Daniel Laurence Moody, Patrick Heymans, and Raimundas Matulevicius. Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax. In *2009 17th IEEE International Requirements Engineering Conference*, pages 171–180. IEEE, 2009.
27. Nicolas Genon, Patrick Heymans, and Daniel Amyot. Analysing the cognitive effectiveness of the BPMN 2.0 visual notation. In *International Conference on Software Language Engineering*, pages 377–396. Springer, 2010.
28. Daniel Moody and Jos van Hillegersberg. Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams. In *International Conference on Software Language Engineering*, pages 16–34. Springer, 2008.
29. Nicolas Genon, Daniel Amyot, and Patrick Heymans. Analysing the cognitive effectiveness of the UCM visual notation. In *International Workshop on System Analysis and Modeling*, pages 221–240. Springer, 2010.
30. John C Thomas, Judah Diament, Jacquelyn Martino, and Rachel KE Bellamy. Using the "Physics" of Notations to analyze a visual representation of Business Decision Modeling. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 41–44. IEEE, 2012.
31. George Popescu and Alain Wegmann. Using the Physics of Notations theory to evaluate the visual notation of seam. In *2014 IEEE 16th Conference on Business Informatics*, volume 2, pages 166–173. IEEE, 2014.
32. David Granada, Juan Manuel Vara, Marco Brambilla, Verónica Bollati, and Esperanza Marcos. Analysing the cognitive effectiveness of the WebML visual notation. *Software & Systems Modeling*, 16(1):195–227, 2017.
33. Leslie Kish. Some statistical problems in research design. *American Sociological Review*, pages 328–338, 1959.
34. Barbara A Kitchenham, Tore Dyba, and Magne Jorgensen. Evidence-based Software Engineering. In *Proceedings. 26th International Conference on Software Engineering*, pages 273–281. IEEE, 2004.
35. Amine El Kouhen, Cedric Dumoulin, Sébastien Gerard, and Pierre Boulet. Evaluation of modeling tools adaptation. Technical Report. *Laboratoire d'Intégration des Systèmes et des Technologies. [Online]. Avaliable: https://hal.archives-ouvertes.fr/hal-00706701v1*, 2012.
36. Francisco J Pérez-Blanco, Juan M Vara, Cristian Gómez, Valeria De Castro, and Esperanza Marcos. Model-Based Tool Support for Service Design. In *International Conference on Fundamental Approaches to Software Engineering*, pages 266–272. Springer, 2020.
37. Montserrat Estañol, Esperanza Marcos, Xavier Oriol, Francisco J Pérez, Ernest Teniente, and Juan M Vara. Validation of service blueprint models by means of formal simulation techniques. In *International Conference on Service-Oriented Computing*, pages 80–95. Springer, 2017.
38. Peter Pin-Shan Chen. The Entity-Relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)*, 1(1):9–36, 1976.
39. Keith Moss. The Entity-Relationship model. In *Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON)*, pages 1–6. IEEE, 2012.
40. Thomas M Connolly and Carolyn E Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.
41. Forrest J Shull, Jeffrey C Carver, Sira Vegas, and Natalia Juristo. The role of replications in empirical software engineering. *Empirical software engineering*, 13(2):211–218, 2008.
42. Beatriz Mora, Félix García, Francisco Ruiz, and Mario Piattini. Graphical versus textual software measurement modelling: an empirical study. *Software Quality Journal*, 19(1):201–233, 2011.
43. Ronald A Fisher. On the probable error of a coefficient of correlation deduced from a small sample. *Metron*, 1:1–32, 1921.
44. Terry K Koo and Mae Y Li. A guideline of selecting and reporting Intraclass Correlation Coefficients for reliability research. *Journal of chiropractic medicine*, 15(2):155–163, 2016.
45. Kenneth O McGraw and Seok P Wong. Forming inferences about some Intraclass Correlation Coefficients. *Psychological methods*, 1(1):30, 1996.
46. Joseph L Fleiss. Design and Analysis of Clinical Experiments (Wiley Classics Library). *Journal of the American Statistical Association*, 94(448):1384–1384, 1999.
47. Yahya Kazemzadeh, Simon K Milton, Lester W Johnson, et al. Process Chain Network (PCN) and Business Process Modeling Notation (BPMN): a comparison of concepts. *Journal of Management and Strategy*, 6(1):88–99, 2015.
48. Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.
49. Paul Cairns, M Soegaard, and RF Dam. Experimental methods in Human-Computer Interaction. *Encyclopedia of Human-Computer Interaction*, 2016.
50. Mark Minas and Oliver Köth. Generating diagram editors with DiaGen. In *International Workshop on Applications of Graph Transformations with Industrial Relevance*, pages 433–440. Springer, 1999.
51. Ana Pescador, Antonio Garmendia, Esther Guerra, Jesús Sánchez Cuadrado, and Juan de Lara. Pattern-based development of domain-specific modelling languages. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 166–175. IEEE, 2015.
52. Christian Brand, Matthias Gorning, Tim Kaiser, Jürgen Pasch, and Michael Wenz. Development of high-quality graphical model editors. *Eclipse Magazine*, 2011.
53. Juha-Pekka Tolvanen and Steven Kelly. MetaEdit+ defining and using integrated domain-specific modeling languages. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 819–820. ACM, 2009.
54. Etienne Juliot and Jérôme Benois. Viewpoints creation using Obeo Designer or how to build Eclipse DSM without being an expert developer. *Obeo Designer Whitepaper. [Online]. Avaliable: http://archive.is/bnAUd*, 2010.
55. Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. Sirius: A rapid development of DSM graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, pages 233–238. IEEE, 2014.
56. Karsten Ehrig, Claudia Ermel, Stefan Hänsgen, and Gabriele Taentzer. Towards graph transformation based generation of visual editors using eclipse. *Electronic Notes in Theoretical Computer Science*, 127(4):127–143, 2005.
57. Nadege Pontisso and David Chemouil. Topcased combining formal methods with Model-Driven Engineering. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pages 359–360. IEEE, 2006.
58. Carsten Gutwenger, Joachim Kupke, Karsten Klein, and Sebastian Leipert. GoVisual for CASE Tools Borland Together Control Center and Gentleware Poseidon–System Demonstration. In *International Symposium on Graph Drawing*, pages 123–128. Springer, 2003.

59. Jesús J López-Fernández, Antonio Garmendia, Esther Guerra, and Juan de Lara. An example is worth a thousand words: Creating graphical modelling environments by example. *Software & Systems Modeling*, 18(2):961–993, 2019.
60. Antonio Garmendia, Esther Guerra, Juan de Lara, Antonio García-Domínguez, and Dimitris Kolovos. Scaling-up domain-specific modelling languages through modularity services. *Information and Software Technology*, 115:97–118, 2019.
61. Gabriele Taentzer. Towards generating domain-specific model editors with complex editing commands. In *Proc. International Workshop Eclipse Technology eXchange (eTX), Satellite Event of ECOOP*, 2006.
62. Karsten Ehrig, Claudia Ermel, Stefan Hänsgen, and Gabriele Taentzer. Generation of visual editors as eclipse plug-ins. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 134–143, 2005.
63. Vicente Pelechano, Manoli Albert, Javier Muñoz, and Carlos Cetina. Building Tools for Model Driven Development. Comparing Microsoft DSL Tools and Eclipse Modeling Plug-ins. In *Actas del Taller sobre Desarrollo de Software Dirigido por Modelos. MDA y Aplicaciones. Sitges, Spain, October 3, 2006*, volume 227 of *CEUR Workshop Proceedings*, 2006.
64. Daniel Amyot, Hanna Farah, and Jean-François Roy. Evaluation of development tools for domain-specific modeling languages. In *International Workshop on System Analysis and Modeling*, pages 183–197. Springer, 2006.
65. Paulus Sentosa, Ralf Möller, Dieter Gollmann, and Miguel Garcia. Generation of Text Editors for Custom Domain Specific Language on the Eclipse Platform. Master's thesis, Hamburg University of Technology, 2007.
66. Steven Kelly. Comparison of Eclipse EMF/GEF and MetaEdit+ for DSM. In *19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, Portland. [Online]. Avaliable: https://s23m.com/oopsla2004/kelly.pdf*, pages 87–96, 2004.
67. Heiko Kern, Axel Hummel, and Stefan Kühne. Towards a comparative analysis of meta-metamodels. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11*, pages 7–12, 2011.
68. David S Frankel. *Model Driven Architecture applying MDA to Enterprise Computing*. OMG Press. Wiley Publishing, Inc., New York, 2003.
69. John Krogstie, Guttorm Sindre, and Håvard Jørgensen. Process models representing knowledge for action: a revised quality framework. *European Journal of Information Systems*, 15(1):91–102, 2006.
70. Reinhard Schuette and Thomas Rotthowe. The guidelines of modeling–an approach to enhance the quality in information models. In *International Conference on Conceptual Modeling*, pages 240–254. Springer, 1998.
71. Thomas RG Green. Cognitive dimensions of notations. In *A. Sutcliffe and L. Macaulay (Eds.), People and computers V*, pages 443–460. Cambridge University Press, 1989.
72. Thomas R. G. Green and Marian Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.
73. Harald Störrle and Andrew Fish. Towards an Operationalization of the "Physics of Notations" for the Analysis of Visual Languages. In *International Conference on Model Driven Engineering Languages and Systems*, pages 104–120. Springer, 2013.