



A domain specific language notation for a language learning activity generation tool

Gabriel Sebastián¹ · Ricardo Tesoriero² · Jose A. Gallud²

Received: 6 July 2020 / Revised: 30 May 2021 / Accepted: 9 July 2021 /
Published online: 3 September 2021
© The Author(s) 2021

Abstract

Globalization has increased the need for society to master new languages. This need has encouraged the launch of many applications dedicated to language learning. This paper presents a graphical notation for a domain specific language to represent language learning activities. It describes how this notation enables developers to represent language learning activity characteristics using workflow, presentation, content, media and activity model conforming a metamodel that defines the abstract syntax of the domain specific language. This notation is implemented as part of an integrated development environment to build model-based applications. Finally, this proposal is evaluated with a framework that uses the cognitive dimensions of notations for notational systems. The proposed graphic diagram editor exceeds the experience that the user has with the reflexive model editor. In relation to the creation and editing of workflow models and presentation/activity models, the proposed graphical notation its more intuitive and easy to maintain visually than the traditional reflexive tree notation used by many model-based development frameworks.

Keywords Model-driven architecture · Software Engineering · Domain Specific Language · Language-learning Applications

1 Introduction

Globalization has increased the need for society to master new languages. This need has encouraged the launch of many applications dedicated to language learning. These applications are usually developed for the various work environments that are the product of this globalization. The most popular platform to develop applications of these characteristics is

✉ Gabriel Sebastián
gabriel.sebastian@uclm.es

Ricardo Tesoriero
ricardo.tesoriero@uclm.es

Jose A. Gallud
jose.gallud@uclm.es

¹ Albacete Research Institute of Informatics, Universidad de Castilla-La Mancha, Albacete, Spain

² Computing Systems Department, Universidad de Castilla-La Mancha, Albacete, Spain

undoubtedly the Web. The Web not only allows easy access to resources, but also allows interaction through learning communities.

There are different learning methods. Formal learning is the strictly academic one that was traditionally applied in classrooms, although there are currently on-line alternatives (i.e. e-learning and b-learning). On the other hand, informal learning has the main characteristics of not being structured and support different learning rhythms. Finally, unlike informal learning, non-formal learning is structured; and unlike the formal one, it is not strictly academic. As in formal learning, learning resources are produced professionally [41]. These types of non-formal materials are often designed for educational purposes with a structured progression. While this form of learning can be offered as in-class, it is more common to find it on-line (eg Busuu¹, Duolingo², Babbel³, etc.).

The development of such applications is complex due to the diversity of language learning methodologies, the variety of execution environments (desktop, mobile and web) and the amount of different technologies that can be used. The preparation of learning exercises to implement a language learning application is a complex, tedious and repetitive process, which involves the repetition of resource management tasks, the generation of duplicate code that is difficult to maintain and prone to errors, etc. These problems are further aggravated by the large number of deployment platforms in which applications must be available (e.g. iOS, Linux, Windows, Web, etc.).

A survey of the published literature [1] revealed that much of it justified the cognitive advantages of visual languages in terms of concepts from popular psychology such as the left-brain right-brain dichotomy. This was in spite of published research into the usability implications of diagrammatic notations, dating back to the 1970s [14]. Empirical investigations of notation usability have tended to find that while diagrams are useful in some situations, they are not efficient in others (e.g. [19]).

Given these circumstances, languages, tools and techniques have emerged in recent years to assist designers to support instructional design⁴, either in Spanish or English, of learning scenarios [23] that include the configuration of Learning Management Systems (LMS) or Technology Enhanced Learning (TEL) systems.

As we will see later, there are many approaches that support models and visual languages through editors that allow designers to express the mental and conceptual models to communicate the work to be developed [24]. Thus, the complexity of creating learning exercises is increased by the complexity of creating tools that allow manipulating visual artefacts through specific notations associated with specific domain languages capable of representing them, also known as Visual Instructional Design Languages (VIDL).

These tools should be designed to help professionals specify learning scenarios with specific terminology and graphic formalism. In addition, these specifications must be interpretable by computers.

Therefore, the objective of this proposal is the definition and implementation of a graphic language capable of representing activities to learn languages. In particular, the goal is the development of a language learning methodology editor capable of constructing representations of language learning activities that be interpretable by computers regardless of the execution platform.

¹ Busuu home page. URL = <https://www.busuu.com/> (last access 05/04/2020)

² Duolingo home page. URL = <https://www.duolingo.com/> (last access 05/04/2020)

³ Babbel home page. URL = <https://www.babbel.com/> (last access 05/04/2020)

⁴ Instructional design. URL = https://es.wikipedia.org/wiki/Instructional_Design (last access 05/04/2020)

The potential end users of this editor would be both, language teaching academics and people who are familiar with TEL systems. The editor will assist them in modelling and specifying methodologies to increase the reuse of learning resources (e.g. workflow, multimedia resources, exercises, etc.).

To ensure that the specifications are interpretable by computers supporting different execution platforms, we will apply an approach based on Domain Specific Modelling (DSM).

The implementation of the DSM will be based on the specification of an abstract syntax based on the metamodel presented in [36] and explained in detail in [37], which supports the development of language learning applications that separates the definition of language learning methodologies in 5 concerns. These 5 concerns structure the learning methodologies in terms of concepts, multimedia resources, user interface, activities and their workflow.

Thus, the specific syntax associated with this metamodel is implemented based on the graphic editor proposed in this work. Models built with the editor are used to generate the source code of the language learning application modeled with the editor using transformation rules expressed in the ATL and ACCELEO languages, maximizing code reuse and minimizing maintenance costs.

The structure of this article is as follows. In Sect. 2 we will see how this work relates to MDA/MDE technologies, and we will discuss about work related to our proposal. The aim of Sect. 2.3 is introducing the reader to the domain of language learning activities which are implemented as part of the language learning methodologies deployed in the Internet. In Sect. 3 we will present the graphical notation for the abstract syntax defined in the metamodel shown in Fig. 1 and presented in [37].

Section 4 presents the evaluation of the notation described in Sect. 3. In Sect. 5 we define a set of metrics to evaluate the proposed editor usability using traditional approaches in order to improve the validation process of the evaluation based on notational dimensions. In Sect. 6 we discuss our proposal, and the results of the evaluation and the relationships between some of the notational dimensions are analysed. Finally, in Sect. 7 we will show the conclusions of this work and future work.

2 Related work

To address the solution to the aforementioned problem, in this article we will use DSM [21] as a development methodology, applying the principles and techniques defined in the standard of Model Driven Architectures (MDA) within the context of Model Driven Engineering (MDE) [35].

In this section we will explain how this work is related to these technologies (Sect. 2.1) and we will discuss some works that are related to our proposal (Sect. 2.2). Besides, in Sect. 2.3 we will introduce to the domain of language learning activities which are implemented as part of the language learning methodologies.

2.1 How this work relates to MDA/MDE

The most important principle of the MDE is the creation of non-contemplative productive models (i.e. interpretable by computers); therefore, they must be well defined, that is, they

Fig. 1 The metamodel (abstract syntax) ▶

have to correspond to a specific metamodel. In this way, productive models can be managed and interpreted with MDE [5] tools.

The DSM is a flexible model-based approach that is based on the definition of different Domain Specific Modelling Languages (DSML) defined by experts in the different domains covered by the system; instead of a single modelling language like Unified Modelling Language (UML) [33].

A DSML is defined in terms of three basic components: *abstract syntax*, *concrete syntax* and *semantics*.

The *abstract syntax* is specified by a metamodel that describes the concepts of language, the relationships between them and the structuring rules that restrict the elements of the model and their combinations to represent the domain rules. In our case, the metamodel is the one shown in Fig. 1, and presented in detail in [36] and [37]. Fig. 2 shows different packages representing the common elements. The central package of this model architecture is Methodology and is divided into five packages: workflow, content, media, activity and presentation. A sixth package called commons, that provides other meta-classes with the identification (entity meta-class) and extension (property meta-class) capabilities

The *concrete syntax* of a DSML is defined as a mapping between the concepts of the metamodel and its textual or graphic representation providing users with a more intuitive notation for the representation of models. In this paper we present the definition of a specific syntax to represent language learning methodologies.

Finally, the *semantics* of a DSML describes the precise meaning of its models through model transformations that support the update *in situ* [8] in terms of the allowed actions. UML profiles are the mechanism that UML provides for the definition of DSMLs. Some proposals that use this mechanism for the definition of learning systems are presented in [25] and [26].

However, the use of UML profiles implies that DSML users must know UML, which determines the number of end users of the language. This limitation can be saved by using a specific editor for the DSML, instead of defining an extension to UML.

Several technical approaches coexist to develop DSMLs [20]: MetaCase/MetaEdit +⁵, Microsoft Visual Studio DSL Tools⁶, EMF and GMF⁷.

All these tools allow to define both the abstract and the concrete syntax of the DSMLs as well as facilities for the persistence of the built models. In addition, some of them provide support for the execution of transformations, model-to-model, and model-to-text.

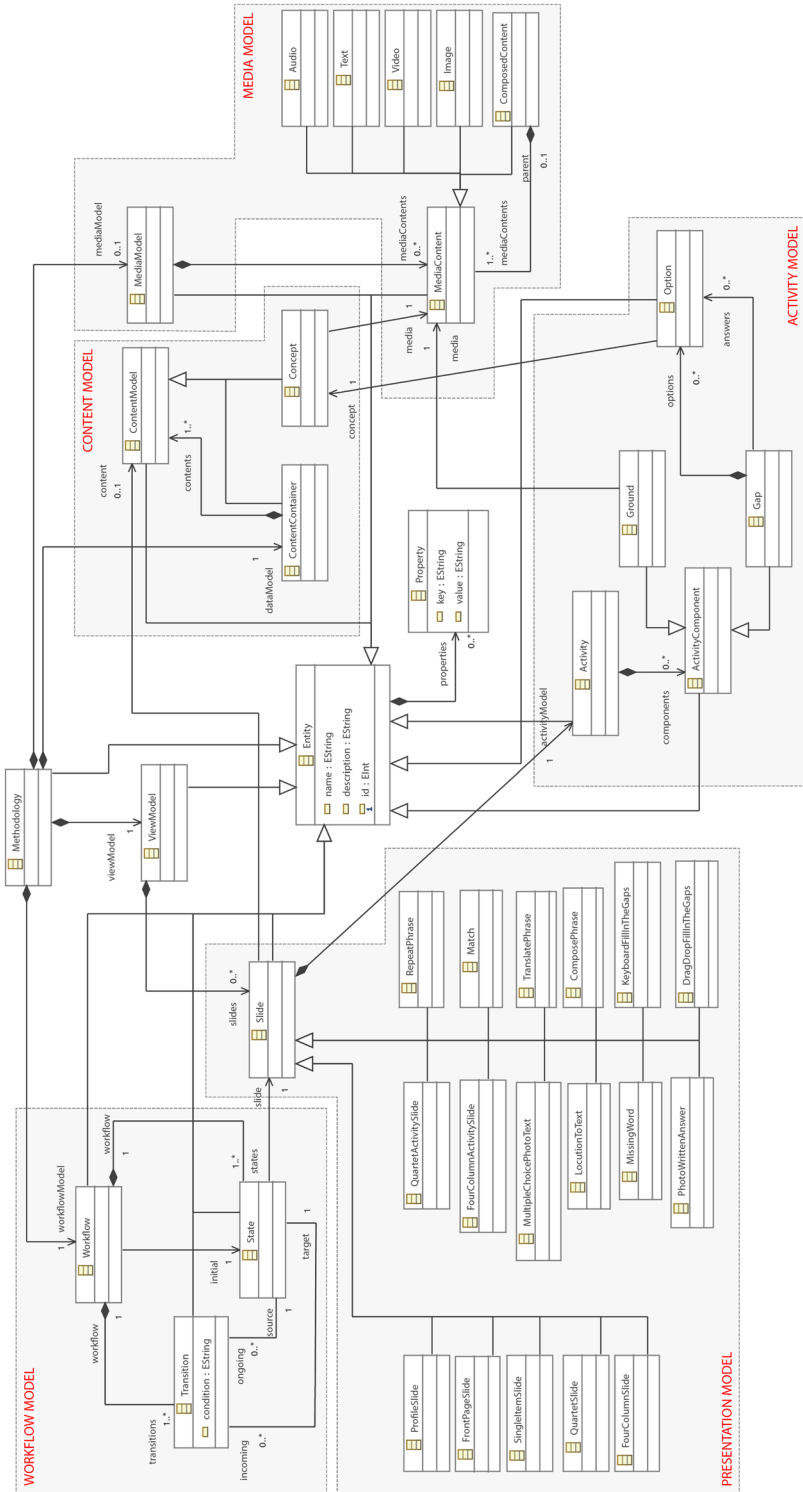
In our case we have decided to work with the tools provided by the Eclipse Modeling Project (EMP)⁸ with the objective of creating plug-ins for Eclipse IDE that implement the DSML. The EMP includes the Eclipse Modeling Framework (EMF) that allows the generation of modifiable code to specify metamodels, edit instances of models through a basic reflective editor (in the form of a tree, table or properties). It also provides runtime support that allows XMI notification, persistence and serialization of the models. However, it is

⁵ MetaCase/MetaEdit +. URL = <https://www.metacase.com/products.html> (last access 05/04/2020)

⁶ Microsoft Visual Studio DSL tools. URL = <https://docs.microsoft.com/en-us/visualstudio/modeling/overview-of-domain-specific-language-tools?view=vs-2019> (last access 05/04/2020)

⁷ GMF. Eclipse Graphical Framework. URL = <https://www.eclipse.org/gmf-tooling/> (last access 05/04/2020)

⁸ EMP. Eclipse Modeling Project. URL = <https://www.eclipse.org/modeling/> (last access 05/04/2020)



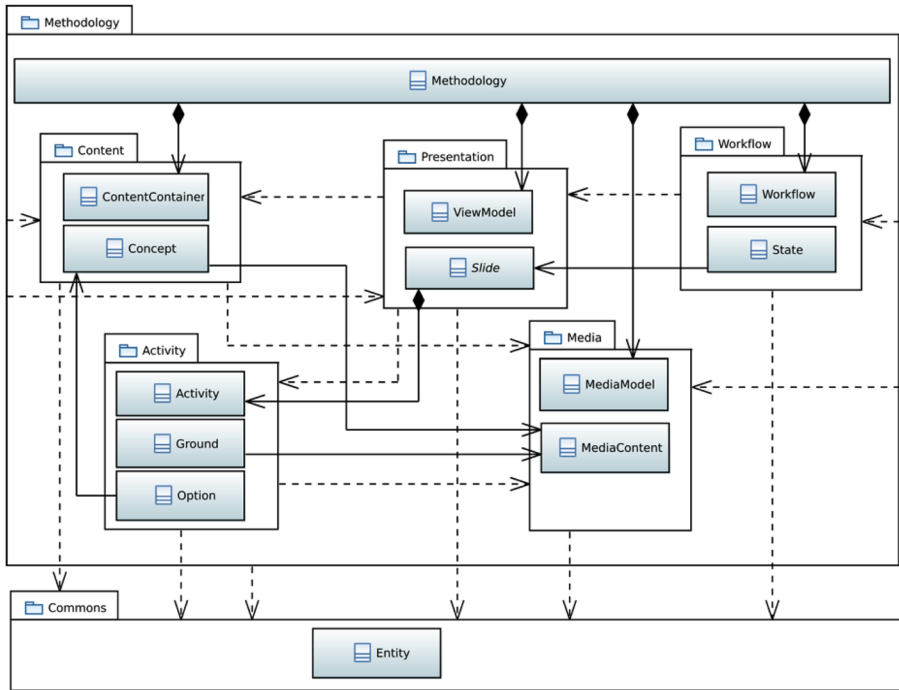


Fig. 2 The metamodel package model: sub-packages defined as part of Methodology package

noted that the basic reflective editor is insufficient in terms of usability, at least for complex metamodels, such as those used in the creation of language-learning methodologies.

The EMP also includes the Graphical Editing Framework (GEF) that allows developers to create a graphic editor from a metamodel generated with EMF. Another framework included in the EMP is the ATLAS Transformation Language (ATL)⁹ that allows you to define model-to-model transformations.

From the point of view of VIDL, the Graphical Modelling Framework (GMF)¹⁰ provides a generative component and a runtime infrastructure to develop graphic editors of EMF-based diagrams and GEF using an MDE approach. In addition, GMF allows us to define restrictions in OCL¹¹ to define rules to build well-formed models. Fig. 3 illustrates the workflow and domain models (ECORE), used during GMF-based development: the reflective editor code generator (*Domain Gen Model*), the Graphic Definition model (*Graphical Def Model*), the Tool Definition Model (Tooling Def Model), the Mapping Model (Mapping Model) and the code generator of the diagram graphic editor (*Diagram Editor Gen Model*).

⁹ ATLAS Transformation Language. URL = <https://www.eclipse.org/at/> (last access 05/04/2020)

¹⁰ GMP. Graphical Modeling Project. URL = <https://www.eclipse.org/modeling/gmp/> (last access 05/04/2020)

¹¹ Object Constraint Language (OCL) specification. URL = <https://www.omg.org/spec/OCL/> (last access 05/04/2020)

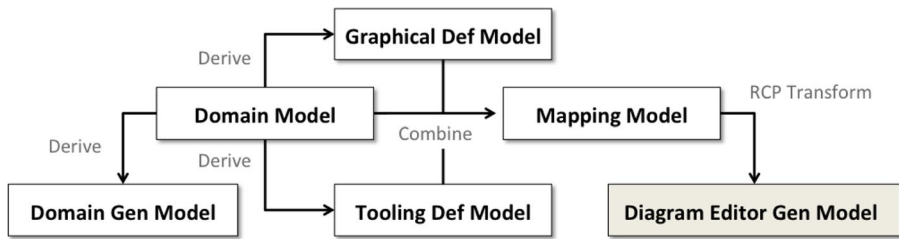


Fig. 3 Model editor development workflow

The *Domain Model* is specified when defining an Ecore model (meta-model name for EMF/GMF, in reference to the Eclipse meta-meta-model, called Ecore, to define meta-models) with the editor from EMF or importing the model from other Ecore providers. The Graphic Definition Model is formalized through the use of a formula-based wizard, which guides and generates a graphic definition model of the previously specified domain model. The Graphic Definition Model contains information related to graphic elements (figures, nodes, compartments, links, etc.) without direct connection to the elements of the domain model for which they will provide representation and editing. The Tool Definition Model is specified using another wizard; It is used to design the palette and other visual controls, such as menus, actions and toolbars. The Mapping Model is specified, with the help of another assistant, linking the three previous models: the domain, the graphic definition and the definition of tools. In this way, GMF allows you to reuse the graphic definition for several domains. A separate Mapping Model is used to link the graphic and tool definitions to the selected domain models. Once the appropriate assignments are developed, a Generator Model is created to allow the implementation details or the code generation phase to be defined. Finally, the Diagram Plug-in is generated and then tested in a new Eclipse Application at runtime. When a user is working with a diagram, at runtime, the notation and domain models are joined providing both persistence and synchronization. Instead of generating workbench plug-ins, GMF can be used to generate diagram editors as Rich Client Platform (RCP)¹², that is, independent applications.

2.2 Related work on visual learning designs

E-learning is one of the mostly address fields of application of the MDA papers [38]. Complex Learning Processes (CLPs) are represented using Educational Modeling Languages (EMLs). IMS Learning Design (IMS-LD)¹³ is a commonly used EML for which some visual editors are being created that help the authoring process of learning scenarios (learning designs).

In [11] authors describe the FLEXO language, a Domain Specific Language (DSL) based upon the definition of a generic IMS-LD. A visual representation of FLEXO can be provided using an editing tool that abstracts the text-based specification of courses. The

¹² Rich Client Platform applications. URL = https://wiki.eclipse.org/Rich_Client_Platform (last access 05/04/2020)

¹³ Learning Design Specification. URL=<http://www.imsglobal.org/learningdesign> (last access 05/04/2020)

FLEXO language delivers the course in a variety of EMLs, as required by the execution environment or LMS. It hides most of the technical details from the designer, and can be easily extended to deliver other formats. The designer must only be aware of the possibilities of the target language, but not of how to use them. If the target platform in which the learning design has to be run is changed, the course can be generated from the FLEXO specification with scarce modifications.

COLLAGE [28] is too a high-level creation tool compatible with IMS-LD specialized in Computer-Supported Collaborative Learning (CSCL). COLLAGE helps teachers in the process of Create your own potentially effective collaborative learning designs by reusing and customizing patterns, according to the requirements of a particular learning situation. These patterns, called Collaborative Learning Flow Patterns (CLFP), represent best practices that are reused by professionals when structuring the flow of learning activities (collaborative).

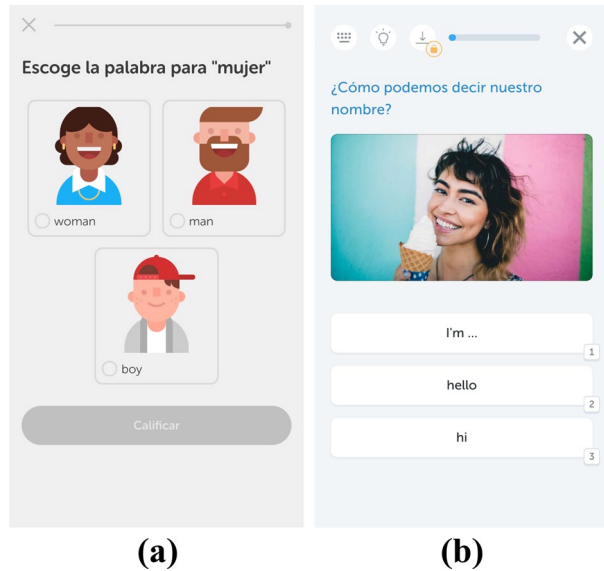
In [42], the authors also present a visual authoring tool that complies in this case with the LPCEL (Learning Process Composition and Execution Language) specification proposed in [43]. The objective of this tool is also to facilitate the process of creating learning scenarios, with the advantage that the level of expressiveness of LD-LPCEL (as objective EML) is broader than IMS-LD. So, the LPCEL Editor provides a broad level of expressiveness and facilitates the authoring process with an editor that includes: (1) Visual Elements, (2) Intermediate Representation, (3) Learning Patterns, (4) Collaboration tools and (5) Web Services

In [34] authors proposes an authoring system, referred to as ASK Learning Design Tool (ASK-LDT), that utilizes the LD principles to provide the means for designing activity-based learning services and systems. The ASK-LDT relies on user-defined visual representations of IMS-LD level A formalisms from which an XML-based description of the target Unit-of-Learning (UoL) can be generated. Visual shapes, icons, connectors and all source elements are kept close to the core IMS-LD level A model.

The Model-Driven Learning Design (MDLD) [12] proposes a set of visual abstractions for the authoring process and the mechanisms to generate XML files compliant with an EML. In [30], the core of the approach is the structural and operational specification of a DSL used to describe key aspects of the final learning application. On the other hand, in [40] a language is proposed for the high-level design of interactive applications created in accordance with the model-view-controller pattern. This approach is especially suitable for applications that incorporate content with sophisticated structures and whose interactive behavior is driven by these structures. In both [30] and in [40], the resulting designs are compatible with rapid prototyping, exploration and early discovery of application features, and systematic implementation using technologies web based standard. Also, these two approaches facilitate active collaboration between instructors and developers throughout the process of developing and maintaining e-learning applications.

Gamification has been proven to increase engagement and motivation in multiple and different non-game contexts such as education. In [7] authors describe a graphical modeling editor for gamification domain definition and a graphical modeling editor for gamification strategy design, monitoring and code generation in event-based systems. The solution makes use of Model-Driven Engineering (MDE) and Complex Event Processing (CEP) technology. This work also shows how the proposal can be used to design and automate the implementation and monitoring of a gamification strategy in an educational domain supported by a well-known Learning Management System (LMS) such as Moodle.

Fig. 4 The multiple-choice learning activity in Duolingo (a) and Busuu (b)



Finally, the following research works [6, 29, 44], and [22], although they do not apply to the field of e-learning, are also related to VIDL and are related to our work by their approach or by the tools they use for their development.

2.3 Analyzing language-learning methodologies

The aim of this section is introducing the reader to the domain of language learning activities which are implemented as part of the language learning methodologies deployed in the Internet.

Regarding the definition of language learning activities, the authors of [4] describe the language learning experience using mobile devices in formal and informal learning scenarios. From the technological perspective, different approaches to develop e-learning solutions are explored in [13] and a summary of the advances in the development of these systems based on experiences and methodologies is presented in [18]. Finally, the eLSE methodology [27] describes a systematic approach to evaluate them ensuring their quality. The importance of interactive multimedia packages for language learning is explored in [45]. This work exposes the impact of proving high flexibility to designers in the definition of such resources in order to focus the attention on a learner-based orientation instead of technology-driven learning methodologies. The analysis performed in [9], which studies the situation of language learning software currently available, shows that more stimulating activities are being proposed. Consequently, it suggests a positive evolution of language learning software in the near future.

Although the implementation of most of these methodologies has different look and feel; they share common characteristics such as the Concepts to Learn (CL), the structural organization of contents (i.e. levels, blocks, lessons, activities, etc.), multimedia resources (i.e. audios, texts, images, etc.), Learning Activity Mechanisms (LAMs), and so on.

For instance, the Fig. 4 depicts the multiple-choice LAM in two different learning methodologies. While Fig. 4a depicts the Duolingo¹⁴ version of the mechanism using radio buttons associated to images to display available options, and a button to confirm the option selection; the Fig. 4b depicts the Bussu¹⁵) version the employs a set of buttons labelled with the available options instead of the radio buttons. Moreover, both versions share common characteristics in the presentation too (e.g. providing student lesson progress at the top of the language learning activity UI).

This is just one example of the similarity among language learning methodologies, since it is not the aim of this paper to perform a review of language learning methodologies.

3 Graphical notation description

The graphical notation for the abstract syntax defined in [37] was implemented as an Eclipse IDE plugin using the Eclipse Graphical Framework (GMF)¹⁶ that is part of the Eclipse Modeling Project (GMP)¹⁷. These tools are multi-platform because Eclipse IDE runs on MacOS, Linux, and Microsoft Windows.

The purpose of the specific syntax of a DSL is to provide users with an intuitive and friendly notation similar to others they normally use. In the case of this work, this notation makes easier for users to specify the models of a language learning methodology. The concrete syntax is usually defined as a mapping between the concepts of the meta-model and its textual or graphic representation. Thus, to define these visual representations, it is necessary to establish links between these concepts and the visual symbols that represent them. For all this, we have chosen visual symbols for our graphic editor so that they are as intuitive as possible. A complete example illustrating the concrete syntax adopted can be seen in Fig. 6. The description of each of the icons used can be found in the Table 3.

The description of the graphical notation will be carried out following the description tables presented in [16], which are described in the Tables 1 and 2.

The Graphical User Interface (GUI) of our editor consists of the following components: a title bar, a properties panel, a tool palette and the workspace. Figure 5 shows the distribution of these elements in the GUI.

Title bar In this bar, both the name of the application and the name of the current job file are displayed.

Properties panel In this panel, the user can view and edit the properties of the selected elements in the workspace.

Tools palette The elements of the tool palette are organized into five categories: Contents, Resources, Presentation, Activities and Control Flow. In the Table 3, we provide the description of each of the icons.

¹⁴ Duolingo. URL = <https://www.duolingo.com> (last access 05/04/2020)

¹⁵ Busuu. URL = <https://www.busuu.com> (last access 05/04/2020)

¹⁶ GMF. Eclipse Graphical Framework. URL=<https://www.eclipse.org/gmf-tooling/> (last access 09/23/19)

¹⁷ GMP. Graphical Modelling Project. URL=<https://www.eclipse.org/modeling/gmp/> (last access 09/23/19)

Table 1 Description of the graphic notation (Content model and resource model)

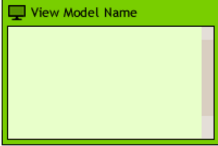


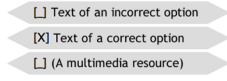
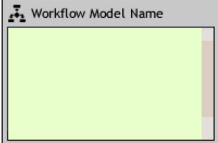
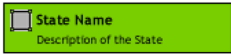
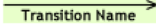
Concept & Graphical representation	Rationale (shape/icon & color)
Content model	<p>Shape: Rectangular window that will contain each of the levels and sub-levels of tree-shaped content, as well as the concepts or words (content tree sheets). In the header of the window we show a representative icon and the model name (normally it corresponds to the methodology or level of the methodology we are modelling; for example, Level 11 of Duolingo). See an example in Fig. 6 (a). In Fig. 7 (b) we visually show the association of contents of the previous example, and in Fig. 7 (a) another example (in this case of Busuu methodology).</p> <p>Color: The head of this window is painted red (and the canvas in faint red).</p>
Content levels	<p>Shape: We represent them with the metaphor of a folder, because they contain concepts (or sub-levels of content).</p> <p>Color: Red icon</p>
Concepts or words	<p>Shape: We represent them with an icon of a concept, because they represent the new concepts or words that are studied in a lesson.</p> <p>Color: Red icon</p>
Media model	<p>Shape: Rectangular window that will contain each of the levels and sub-levels of average resources. In the header we show a representative icon and the model name. See an example in Fig. 6 (b). On the right side of Fig. 8 the resource association of the previous example is shown.</p> <p>Color: This window is painted violet (and the canvas in a faint violet).</p>
Media levels	<p>Shape: We represent them with the metaphor of a folder, because they contain resources made up of several resources (or resource sub-levels). In the one in Fig. 8 we show the media resources of a multiple-choice activity in the Duolingo methodology, to illustrate what we mean by “compound resources”.</p> <p>Color: Violet icon.</p>
Media	<p>Shape: We represent them with a representative icon of the type of resource in question: text, audio, video or image.</p> <p>Color: Violet icon.</p>

The **Workspace** is the canvas in which we place all the elements that will define the methodology that we want to model with the graphic editor. In the Fig. 6, an example of the content of the “Workspace” is shown.

As we have already mentioned, the diagram of any methodology consists of 4 sub-diagrams within it: the sub-diagram of the **Contents**, the sub-diagram of the **Resources**, the sub-diagram of the **Slides**, and the sub-diagram of the **Control flow**. In the 4 cases we will use 4 containers of graphic elements, each of which is distinguished by an identifying icon, a description in its header bar, and in general by a color associated with the mentioned sub-diagram. To illustrate this, in Fig. 6, the 4 sub-definitions of a small part of Duolingo’s methodology and activities are shown.

For the Contents we will use the paradigm of a tree-like structure to express the different levels and sub-levels of the contents of a level of the language learning method



























Table 2 Description of the graphical notation (Presentation and workflow models)

Concept & Graphical representation	Rationale (shape/icon & color)
Presentation and activity model	<p>Shape: Rectangular window that will contain each of the slides and activities. In the header we show a representative icon and the model name. See an example in Fig. 6 (c). In Fig. 10 we illustrate the modelling of a Busuu FillInTheGaps activity.</p> <p>Color: We paint this window green (and the canvas with a very faint green).</p>
	
Statements and passive contents	<p>Shape: Icon of a generic media resource.</p> <p>Color: Green icon.</p>
	
Gap	<p>Shape: We represent them with an icon of a gap.</p> <p>Color: Green icon.</p>
	
Option	<p>Shape: We represent them with a rhomboid rectangle with a checkbox (checked if it is the correct option), and with the text of the option. If the option is of type text, the corresponding text is partially shown; and if the option is a multimedia resource, the name of the multimedia resource is shown in parentheses.</p> <p>Color: Gray rhomboid rectangle.</p>
	
Workflow Model	<p>Shape: Rectangular window that will contain the states, transitions and sub-workflows. In its header we show a representative icon and the name specified for the model. An example can be seen in Fig. 6 (d). In Fig. 9 another workflow that also has a sub-workflow is shown.</p> <p>Color: This window is painted gray with a faint green background.</p>
	
State	<p>Shape: Rectangle with a representative icon, the name of the state and the description of the state.</p> <p>Color: Green rectangle. The states (and the background of the Workflow Model) are painted green because they are closely related to the slides and activities of the Activity Model.</p>
	
Transition	<p>Shape: We represent them with an arrow (which connects the source state with the destination state) together with a descriptive label.</p> <p>Color: Black arrow.</p>
	

that we are defining. For example, the Duolingo methodology is structured in Levels, which in turn are configured by Units, which consist of Lessons, and in which a series of Concepts or Words are used to learn. See this illustrated in Fig. 7b.

For Resources we will also use the paradigm of a tree-like structure to define the library of multimedia resources that will be used in interactive activities. Basically, folders or resource containers are used, which may contain sub-folders. These resources or media can be of 4 types (audio, text, video and image), which can be combined to

Table 3 Descriptions of the icons

Icon	Description
	Content model icon
	Content container icon
	Concept icon (concept or word studied in the parent-lesson)
	Media model icon
	Composed content (multi-resource labelled set)
	Represents an Audio resource
	Represents an Image resource
	Represents a Text resource
	Represents a Video resource
	Workflow model icon
	Workflow/Sub-workflow icon
	State icon
	Transition icon
	View model icon
	ProfileSlide icon
	Lesson FrontPage Slide icon
	Single Item Slide icon
	Spelling Slide icon
	Keyboard Fill In The Gaps icon
	Compose Phrase icon
	Drag & Drop Fill In The Gaps icon
	Locution To Text icon
	Repeat Phrase icon
	Match icon
	Missing Word icon
	Multiple Choice Photo Text icon
	Photo Written Answer icon
	Repeat Phrase icon
	Translate Phrase icon
	Activity icon
	Ground icon
	Gap icon
	Option icon

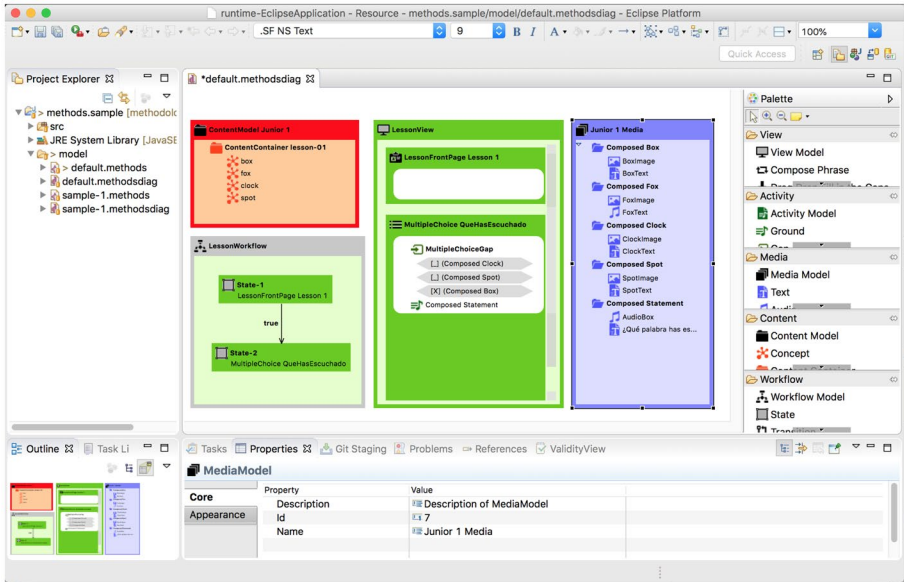


Fig. 5 Editor User Interface

represent complex media (eg text and speech or eg image and text, etc.). See this illustrated in Fig. 8.

To model the control flow with the graphic editor, we have defined a state diagram. This can be seen in Fig. 9.

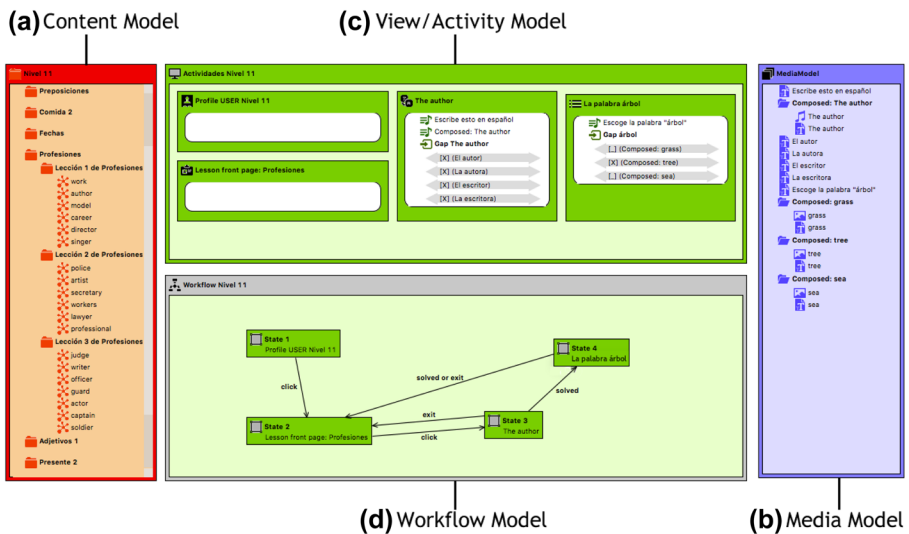


Fig. 6 The 4 sub-diagrams that make up the diagram of a methodology (as Duolingo)

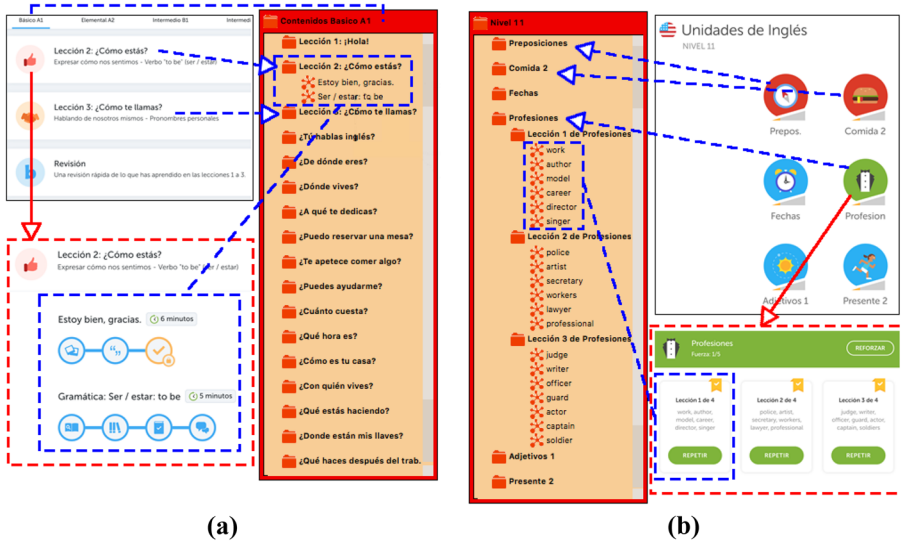


Fig. 7 Tree structure to represent content in Busuu (a) and Duolingo (b)

Finally, to model the activities, like all of them are modeled according to the paradigm of the exercises to fill gaps, they are always modeled with resources for the statement (Ground) and with gaps with possible answers, valid or not (see Fig. 10).

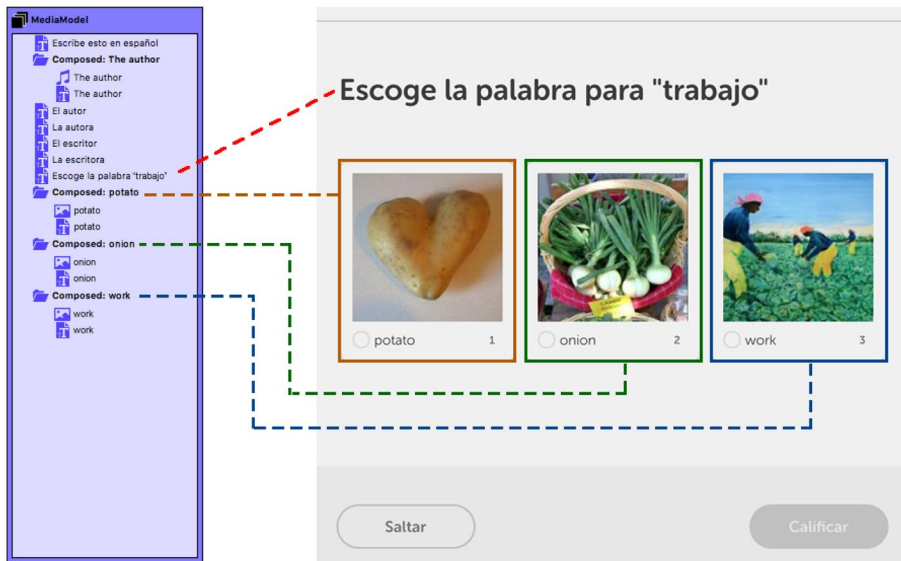


Fig. 8 The media resources of a multiple-choice activity in the Duolingo methodology

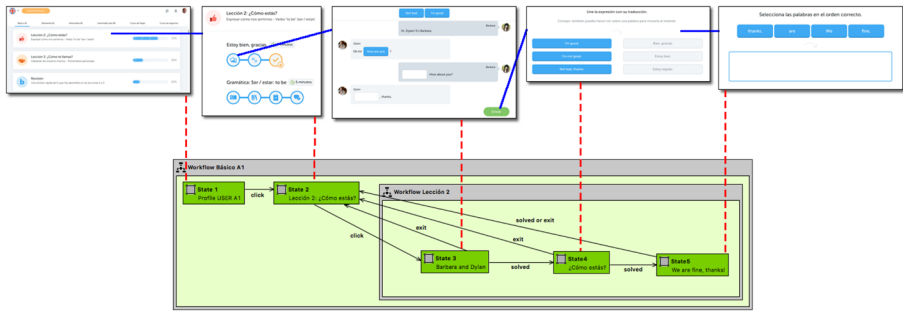


Fig. 9 The workflow of 5 slides in the Busuu methodology

4 Notation evaluation

This section presents the evaluation of the notation described in Sect. 3. This evaluation uses the Cognitive Dimensions of Notations framework for Notational systems presented in [2] and extended in [3] to provide a common vocabulary for use by user interface designers to conduct comprehensible evaluations. It is solely based on structural properties; it does not representational issues (i.e. effectiveness and aesthetics are outside its purview). Therefore, the question addressed by the Cognitive Dimensions (CDs) framework is: *are the users' intended activities adequately supported by the structure of the information artifact?* and if not, *what design maneuver would fix it, and what trade-offs would be entailed?*

This evaluation, and the conclusions of this work (Sect. 7), also offers the comparison of the reflexive model editor with the diagram model editor supporting the proposed notation. The reflexive model editor is the *de facto* default model editor used in the Eclipse Modeling Tools (EMT)¹⁸ (Eclipse IDE distribution to edit models). It supports a tree-based notation that employs reflexive properties of metamodels to build models conforming these metamodels.

The procedure to carry out this evaluation starts with the classification of intended activities; it continues with the analysis of the cognitive dimensions; and it decides if activities' requirements are met. Note that the term “activity” refers to a notational activity in cognitive dimensions as is a manner of interacting with a notation and it is not related to Activity Theory, or Active User Theory.

4.1 Activities

The CDs framework contrasts the following 6 generic types of notation use activity on artifacts based on contributions from a variety of researchers [2]: **incrementation** (e.g. add a media resource to a repository), **transcription** (e.g. set a question text using a media resource), **modification** (e.g. change the media resource to render a multiple-choice option from text to image), **exploratory design** (e.g. define a multiple-choice using audio resources as options instead of text), **searching** (e.g. look for the set of learning activities

¹⁸ EMT. Eclipse Modeling Tools. URL=<https://www.eclipse.org/downloads/packages/release/juno/sr1/eclipse-modeling-tools> (last access 02/06/2020)

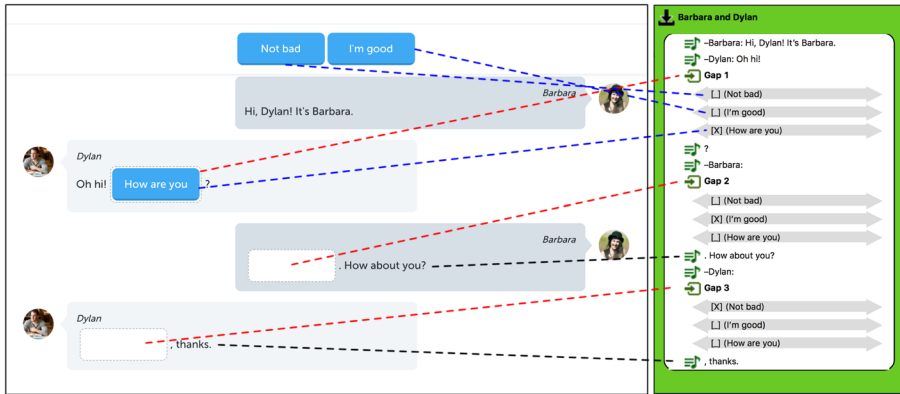


Fig. 10 Graphical model of a Busuu FillInTheGaps activity

related to a CtL), and **exploratory understanding** (e.g. discover next learning activity given a starting activity and the result of a user interaction).

In the Table 4, we list a series of generic activities and show to what extent they satisfy or not each of the 6 types of usage notations mentioned above and exemplified.

Each activity imposes different demands on the system; thus, nothing marked in Table 4 is properly good or bad, unless an activity does not operate as is expected to do.

Some comments regarding the Table 4 of Activities and types of notation use, are the following:

- All activities, except those of pure editing of properties, involve the use of “Incrementation”.
- All activities offer and require a use of “Transcription”.
- The 4 general submodels are designed to be created only once, and do not require any “Modification”. The ones that will have a type of “Modification” notation will be the Management activities of the mentioned submodels.
- “Exploratory design”: Only the states of a Workflow cannot be predicted directly (visually) and must be ‘discovered’ using the properties window.
- “Searching”: Only the Media, the Concepts, are easily searchable in the corresponding Resources or Content trees. Also the creation and editing of Activities, because their elements are always visible.
- The activity of Editing a View is the only one that does not properly have an “Exploratory understanding”.

So, we can affirm that the proposed graphic editor meets all the requirements of the activities.

4.2 Components

The CDs framework also defines 4 types of components: an **interaction language** or **notation**, an **environment** for editing the notation, a **medium** of interaction, and possibly **sub-devices**.

4.2.1 Interaction language or notation

The **interaction language** or **notation** is defined as what the user sees and edits (i.e. the graphical notation presented in Sect. 3).

Everything related to the definition of a methodology, in our graphic editor is visually represented on the canvas. The distribution and sizes of the graphic notation, is editable by the user. Based on this general graphic representation, the user can configure each of the elements thanks to properties windows and pop-up windows. Some of these options or configuration properties will be visible on the canvas, but others will not.

Additionally, there are many interaction resources related to what the user sees and edits; Some of those resources are:

- In all buttons, the user is shown a tooltip as a feedback that tells him what that button does.
- When a window or container, due to its current size, does not reveal its contents, the interface offers the user the possibility of interacting on a horizontal and vertical scroll-view, which allows him to visualize any part of the sub-canvas that initially appeared hidden.
- The editor interface allows the user to configure many aspects of the environment.

4.2.2 Editing environment

The **editing environment** and other interaction resources (e.g. for digram editors: drag and drop support, inline text edition support, zoom support, and so on) that facilitate editing activities. As it has been mentioned before, users visually manage certain visual diagrams with freedom and great versatility. We now list some interface aspects related to the editing environment:

- Drag&drop interaction to add slides, activities, states, and so on.
- Depending on which object is selected on the canvas, one or another context-sensitive sub-palette will appear, showing the user what operations he can do on that object at that time.
- To help the user, when moving the mouse on the canvas or on the objects contained in it, the cursor changes its aspect to let you know what the user can do.
- Once a transition is created, the arrow can be edited freely, moving it, changing the start and end position, and even creating new vertices.
- There are user operations that might trigger a feedback of violation of the integrity of the model.
- During the specification of an activity, it is easy to manage options, correct answers, and so on, through property windows, drop-down, parallel lists with elements bidirectionally transferable from one to another to specify, among the possible answers, which we establish as correct answers, and so on.
- When creating activities, states, or other types of entities, the interface facilitates the user's work by making certain auto-fill of default values (e.g. automatic generation of object names).

4.2.3 Medium of interaction

The **medium of interaction** is the medium used to interact with the notation language (e.g. touch screens, displays, and so on).

In our case, being a graphic editor managed with a desktop computer, the means of interaction are monitors (as an output device) and the mouse and keyboard (as input devices). It is a persistent medium preceded by many fully editable transitional actions. Although the order of the actions is free, to guarantee the integrity of the model, in some cases the editor restricts the actions that can be performed at any time or according to the context.

4.2.4 Sub-devices

We now mention the two **sub-devices** supported by the main device. While the **helper** sub-device offers cross-references (e.g. references from *Workflow model* to *Presentation model*, diagram outline view, and so on); the **redefinition** sub-device enables the main notation changes (e.g. collapsing model views to get a high level view).

4.3 Notational dimensions

In this evaluation we have also considered the 14 notational dimensions defined by the CDs framework presented in Table 5, where each dimension describes an aspect of an information structure that is reasonably general.

4.3.1 Viscosity

Cross references are automatically redrawn in the diagram. For example, if we change the name of a resource, it is automatically modified in all the places in the diagram where it is referenced.

4.3.2 Visibility

As we have already mentioned in Sect. 4.2, everything related to the definition of a methodology, in our graphic editor, is visually represented in a diagram on the canvas. Logically, there are properties that remain hidden, but these are easily accessible, as appropriate, in the properties window.

4.3.3 Premature commitment

The editor, depending on the context, provides restrictions on the order of the operations. For example, depending on which object is selected on the canvas, one or another context-sensitive sub-palette will appear, depending on the operations I can do on that object at that time; In addition, operations that could be performed in other contexts are disabled.

4.3.4 Hidden dependencies

Initially the diagram did not graphically show the dependencies of the elements of the Content Model (concepts/words) with the slides/activities in which these concepts/words

are worked on or studied. Later, we implemented that these dependencies are shown graphically, but, as many lines are generated on the diagram and the canvas, we have left the default option for these dependencies to remain hidden.

4.3.5 Role-expressiveness

In general, the purpose of all the entities that make up the diagram of a methodology is easily deduced. Certainly, in the case of the graphic expression of the operation (contents) of some activities, it is not easy to discover what the author wants to model. When all the activities are based on the fill-in the gaps activities paradigm, all of them are seen in the diagram in a very similar way. It is therefore necessary for the user to understand the dynamics - for example - of the creation of an activity of *Match* based on the paradigm of the exercises of fill-in the gaps. Once you understand this dynamic, it is easier and more obvious to understand what another author has modelled.

4.3.6 Error-proneness

The diagram can be validated at any time. In addition, when the user tries to add an entity inside a container, thanks to the dynamic change of the appearance of the cursor (while mouse over the different models, activities, etc.), the user knows whether or not such operation is allowed. If it is not allowed, the user will not be able to make the mistake of doing it.

4.3.7 Abstraction

As we saw in [37], the “*fill-in the gaps*” abstraction is at the base of the modelling of all types of activities supported by the editor. This is the main abstraction mechanism available in the editor, but other new abstraction mechanisms cannot be managed or created.

4.3.8 Secondary notation

In our editor 3 types of secondary annotations that users can freely use are supported. We refer to: (1) the text boxes for making annotations on the canvas of the diagram, (2) the adhesive notes, as well as (3) the relationship between an adhesive note and an entity of the diagram.

4.3.9 Closeness of mapping

In general the diagram of a methodology, that is, the *Content model*, the *Media model* and the *Workflow model*, is closely related to what it describes. It is not so obvious in the case of *View model (Activity model)*. As we have already mentioned, being the “*fill-in the gaps*” abstraction at the base of the modelling of all types of activities that the editor supports, the modelling is not always visually evident and is closely related to the result that it describes. However, it is in the case of most activities, such as those of type *Image Audio Text Options*, *Image Text Options*, *Question Answer*, *Gap*, *Listening*, and *Selection*.

4.3.10 Consistency

In relation to the *Consistency*, two aspects stand out:

1. We identify the containers with the icon of a folder, of one colour or another, as appropriate, both in the context of a *Content Model* or a *Media Model*.
2. The housing of the diagrams of all the Models (packages) are represented in a similar way (with a window, an identifying icon, and a descriptive name), but with different colours for better identification and differentiation.

4.3.11 Diffuseness

There are no long annotations, nor they occupy valuable spaces within a viewing area. In any case, if this happened, the user can at any time resize and relocate any entity of the diagram that is generating, if he considers it appropriate to facilitate the readability of the diagram. In addition, you can minimize some parts of the diagram that may no longer need to be displayed, to make room for other more valuable parts of the diagram at any given time.

4.3.12 Hard mental operations

As already mentioned in the Sects. 4.3.7 and 4.3.9, the readability of modelling of some types of activities requires more cognitive resources from the user. This is the case of activities of type *Compose Phrase*, *Match*, and so on.

4.3.13 Provisionality

Users at any times (thanks to the restrictions that avoid the most obvious errors of inconsistency of the model) can take provisional actions or *playing “what-if” games*. In addition, at all times, after such provisional or trial actions, the user, by validating the diagram, can know whether or not it is consistent.

4.3.14 Progressive evaluation

Users, as just mentioned in the Sect. 4.3.13, can verify their work at any time. Our editor allows you to validate any part of the diagram (as well as its entirety), at any time, progressively verifying the stage of the work you are doing or where you are up to that moment. In the context menu, which can be accessed by having selected any entity in the diagram, the user always has the option *Validate* available, which provides the necessary feedback to locate and resolve errors.

5 Preliminary usability heuristic evaluation results

In order to evaluate the validity of the evaluation based on notational dimensions 4, we have performed a set of preliminary usability heuristic evaluations based on [10] recommendations defined by Shneiderman rules [39], Nielsen heuristics [31], and Norman principles [32].

These evaluations were performed by 4 Human-Computer Interaction (HCI) experts who have experience in the definition of learning activity models using the reflexive editor. The score used to measure each usability issue was defined in a scale from 0 to 10 resulting in the following average scores in parenthesis.

5.1 Shneiderman rules

The following list shows the average score of each rule between parenthesis:

- Consistency: Strive for consistency in action sequences, layout, terminology, command use and so on (**8.15**).
- Shortcuts: Enable frequent users to use shortcuts, such as abbreviations, special key sequences and macros, to perform regular, familiar actions more quickly (**7.25**).
- Feedback: Offer informative feedback for every user action, at a level appropriate to the magnitude of the action (**7.35**).
- Dialogs: Design dialogs to yield closure so that the user knows when they have completed a task (**6.15**).
- Error prevention and handling: Offer error prevention and simple error handling so that, ideally, users are prevented from making mistakes and, if they do, they are offered clear and informative instructions to enable them to recover (**9.05**).
- Easy reversal of actions: Permit easy reversal of actions in order to relieve anxiety and encourage exploration, since the user knows that he can always return to the previous state (**8.05**).
- Control: Support internal locus of control so that the user is in control of the system, which responds to his actions (**8.35**).
- Short-term memory load: Reduce short-term memory load by keeping displays simple, consolidating multiple page displays and providing time for learning action sequences (**7.25**).

5.2 Norman principles

The following list shows the average score of each principle between parenthesis:

- Memory use: Use both knowledge in the world and knowledge in the head (**8.25**).
- Task structure simplicity: Simplify the structure of tasks (**8.25**).
- Affordance: Make things visible; bridge the gulfs of execution and evaluation (**7.05**).
- Metaphor mapping: Get the mappings right; user intentions should map clearly onto system controls (**6.05**).
- Constraints: Exploit the power of constraints (**8.95**).
- Error handling: Design for error (**8.15**).
- Standardization: When all else fails, standardize (**9.25**).

5.3 Nielsen heuristics

The following list shows the average score of each heuristic between parenthesis:

- System visibility: Visibility of system state (**8.05**).
- Affordance: System and real world correspondence (**7.05**).
- Control: User sense of control and freedom (**8.35**).
- Consistency and Standards: Use of standards and look and feel consistency (**9.25**).
- Error prevention: Avoiding users to make mistakes (**8.25**).
- Metaphors: Mappings to reduce user memory use (**5.95**).

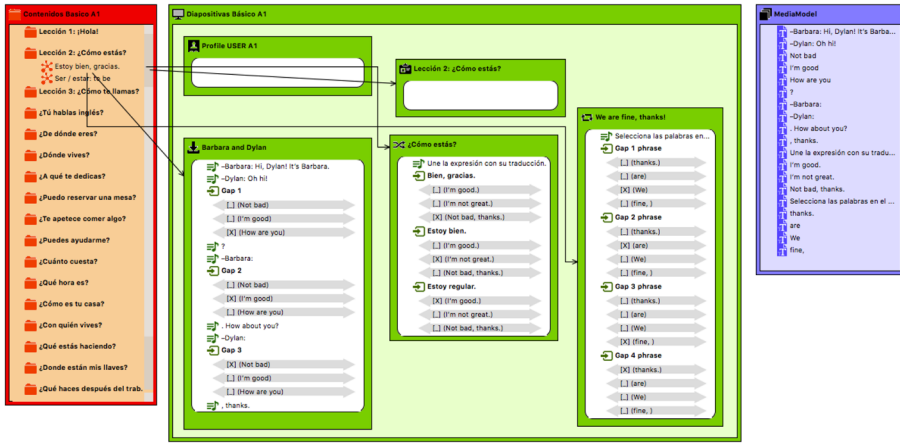


Fig. 11 Content model nodes are connected to learning activity model nodes defined within presentation model nodes using diagram lines

- Flexibility: Effectivity and Efficiency in use (6.75).
- Aesthetics: Graphic design (6.25).
- Feedback: Help users to recognize actions, diagnose errors, and fix them(9.45).
- Help: Help documentation support(8.05).

The Table 6 summarizes preliminary usability evaluation results showing that the average score in all heuristic evaluation approaches are a above 7.70 which is a reasonable good result.

6 Discussion

One of the most difficult issues to address in this proposal is the selection of the graphical notation supporting the concrete syntax for the language learning methodology meta-model. This selection involves the definition of the types of diagram structures to represent different models (i.e. content, media, activity, presentation, and workflow).

The Content model learning units usually follow a tree-based structure where units are described in terms of concepts to be developed by students. It is also possible to build complex units describing general concepts that are divided into smaller units to deal with their complexity by the means of concepts that are more concrete and simpler to understand.

The Media model represents the set of multimedia resources (not language learning concepts) that can be used to represent language learning concepts in learning activities using a tree-based structure too. This model also supports the definition of complex resources to represent concepts using different media resources (i.e audio, text, video, or images).

As both models employ a tree-based structure, we applied the Composite design pattern [17]. The main drawback of using a reflexive model editor (which also employs a tree-based structure) instead of the proposed editors is the lack of relationship between reflexive editors (e.g. lines connecting tree nodes).

Table 4 Activities and types of notation use

Activity	Incrementation	Transcription	Modification	Exploratory design	Searching	Exploratory understanding
Create View Model	X	X		X		X
Add a View or Activity	X	X	X	X		X
Add a Ground	X	X	X	X		X
Add a Gap	X	X	X	X		X
Add a Option	X	X	X	X		X
Create Media Model	X	X		X		X
Add a Media resource	X	X	X	X	X	X
Create Content Model	X	X		X		X
Add a Concept	X	X	X	X	X	X
Add a Content Container	X	X	X	X		X
Create Workflow Model	X	X				X
Add a State	X	X	X			X
Add a Transition	X	X	X	X		X
Add a Workflow Model	X	X	X			X
Manage View Model	X	X	X	X		X
Edit a View		X	X	X		
Edit a Ground		X	X	X	X	X
Edit a Gap	X	X	X	X	X	X
Edit a Option		X	X	X	X	X
Manage Media Model	X	X	X	X	X	X
Edit a Media resource		X	X	X	X	X
Manage Content Model	X	X	X	X	X	X
Edit a Concept		X	X	X	X	X
Edit a Content Container		X	X	X		X
Manage a Workflow Model	X	X	X	X		X
Edit a State		X	X			X
Edit a Transition		X	X	X		X

For instance, content model nodes (Concept in Content Model) can be linked to learning activities (Activity in Activity Model) through presentations (Presentation in Presentation Model) using a diagram line, instead of using a combo box, which remains hidden in property the palette (e.g. see Fig. 11). Moreover, our solution is particularly helpful in Workflow and Presentation/Activity models because it provides a visually intuitive mechanism that is clearly easier to maintain (see Sect. 4.3).

Table 5 Summary of notational dimensions

#	Dimension	Aspect
1	Viscosity	Resistance to change
2	Visibility	Ability to view components easily
3	Premature commitment	Constraints on the order of doing things
4	Hidden dependencies	Important links between entities are not visible
5	Role-expressiveness	The purpose of an entity is readily inferred
6	Error-proneness	The notation invites mistakes and the system gives little protection
7	Abstraction (redefinitions)	Types and availability of abstraction mechanisms
8	Secondary notation	Extra information in means other than formal syntax
9	Closeness of mapping	Closeness of representation to domain
10	Consistency	Similar semantics are expressed in similar syntactic forms
11	Diffuseness	Verbosity of language
12	Hard mental operations	High demand on cognitive resources
13	Provisionality	Degree of commitment to actions or marks
14	Progressive evaluation	Work-to-date can be checked at any time

The Workflow model diagram structure is based on a hierarchical graphs because it enables developers to build hierarchical state machines to represent the learning development state. This hierarchical graph structure is particularly important because it relates learning concepts to user interface presentation by the means of definition of abstract learning activities following the “fill in the gaps” paradigm. Consequently, although reflexive model editors are suitable to define hierarchies; they are not good enough to define graphs. Also note that this relationship could not be set if content and media models were not defined as part of the diagram modeling editor.

The Presentation model diagram structure enables developers nest activity model diagrams in order to define the learning activity to be developed by students. Reflexive editors could be employed to define these models; however, the lack of support for direct connection between presentation/diagram models and the rest of methodology models decreases model editor usability (i.e. readability, learnability, etc.).

Another important advantage of this approach compared to other approaches is the use of MDA OMG standards to define the abstract and concrete syntax of the set of domain specific languages to model language learning methodologies and activities. In fact, it enables developers to generate WebApp source code by the means of M2M and M2T transformations which maximize application development abstraction and reuse as well design time interoperability. Thus, developers are capable of modeling and generating technology independent applications. Moreover, proposed concrete syntax notation, this MDA approach enables developers to model language learning activity concerns at different levels which are connected using a graphical notation.

Table 6 Average score of heuristic evaluation results

Heuristics	Average
Shneiderman rules	7.70
Norman principles	7.99
Nielsen heuristics	7.74

Table 7 Score of notational dimensions for the proposed editor (PE) and for the reflective editor (RE)

#	Dimension	PE score	RE score
1	Viscosity	X X X X X	X X X X
2	Visibility	X X X X	X X X
3	Premature commitment	X X X X	X X X
4	Hidden dependencies	X X X X	X X X
5	Role-expressiveness	X X X	X
6	Error-proneness	X X X X	X X X
7	Abstraction	X X X	X
8	Secondary notation	X X X X X	X
9	Closeness of mapping	X X X X	X X
10	Consistency	X X X X	X X
11	Diffuseness	X X X X X	X X
12	Hard mental operations	X X X	X
13	Provisionality	X X X X	X X X
14	Progressive evaluation	X X X X	X X X

Finally, due to this approach follows OMG standards, software artifacts generated during the application development process of this architecture can be easily reused with third party tools encouraging the interoperability at design time.

6.1 Analysis of notational dimension results

This section explores the results and consequences of the evaluation of this proposal in terms of the notational dimensions (see Sect. 4).

Table 7 shows the comparison of the scores of the proposed editor and the reflective editor obtained during the evaluation of each notational dimension that was exposed in Sect. 4.3.

Although is an internal assessment of this work, it illustrates the graphic and usability power of the graphic notation of the developed editor. We have scored with the criterion that 5 means the maximum level of satisfaction by the user in relation to the notational dimension evaluated, and 1 the minimum.

Thus, Table 7 shows a significant number of aspects where the proposed editor outstands the reflexive editor. For instance, an adequate management and resistance to change (*Viscosity*), a complete offer of mechanisms non-formal to capture extra information (*Secondary notation*), and the absence of language verbosity (*Diffuseness*).

In addition, Table 7 shows that the proposed editor also stands out for the right solutions (graphical and interaction) achieved in relation to other 8 important notational dimensions: *Visibility*, *Premature commitment*, *Hidden dependencies*, *Error-proneness*, *Closeness of mapping*, *Consistency*, *Provisionality* and *Progressive evaluation*.

Naturally, some of these aspects are related to each other. For example, the good score in *Premature commitment* and *Error-proneness* is due to the fact that the editor effectively offers restrictions in the order of making things, which in turn provides the user with adequate protection against possible errors. Likewise, the good score in the aspects of the *Provisionality* and the *Progressive evaluation*, indicate us that the editor has an adequate commitment to the actions, allowing the users to carry out provisional actions, and facilitating them to at the same time validating -at any time- what they are doing.

Finally, the aspects *Role-expressiveness*, *Abstraction* and *Hard mental operations*, which, although adequate, they received less score, are also related to each other (but significantly better than in the reflective editor). The adopted score responds to the fact that, as we have already mentioned in the corresponding subsections: (1) in the diagrams generated with the editor, the purpose of some entities is not always easily deduced, (2) the editor does not allow the definition of new mechanisms of abstraction, and (3) sometimes the user may not find it obvious what another author has modelled.

6.2 Limitations

The main limitation of this proposal compared to UML is the lack of capability to model “low level” software details.

From the software architecture perspective, this approach does not introduce deployment and component models because the generation system employs a fixed architecture which is not easy to extend or distribute.

From the user interface point of view, although this proposal enables users to model media resources; it does not provide an easy way to use different front-end Web frameworks because the code generation process use a fixed framework.

The reason behind these decisions lies on hiding “low level” details from developers/designers to encourage model abstraction. Consequently, this limitation enables the simplification of model transformations because the proposed DSL is considerably “smaller” than UML.

This limitation also improves software maintenance in terms of model extension because new learning concepts can be easily introduced into the language avoiding the definition software details which delegated to model transformation specification.

This proposal generates code that does not support a service oriented architecture (i.e. REST [15] API) which limits the interoperability with third party systems.

7 Conclusions

In this paper we have presented a graphical notation for a domain specific language to represent language learning activities. We have described how this notation enables developers to represent language learning activity characteristics using workflow, presentation, content, media and activity model conforming a metamodel that defines the abstract syntax of the domain specific language. This notation is implemented as part of an integrated development environment to build model-based applications. In addition, we have evaluated this proposal with a framework that uses the cognitive dimensions of notations for notational systems. Finally, in this work we have compared the proposed graphical notation to the traditional reflexive tree notation used by many model-based development frameworks.

We can conclude that the proposed graphic diagram editor exceeds the experience that the user has with the reflexive model editor. Although there is not much difference in the modeling of content models and resource models of methodologies, in relation to the creation and editing of workflow models and presentation/activity models, the solution provided -unlike the one offered by the reflective editor- it is intuitive and easy to maintain visually. We list some other more prominent conclusions:

1. The maintenance of the workflow and activity submodels are very tedious with the reflective editor, among other things because many elements and properties remain hidden... With the proposed graphic editor you can easily and visually create and maintain the states, transitions between them and sub-workflows.
2. In the reflexive editor, the editing operations using *copy&paste*, are delicate, and sometimes, to find errors you have to go to the text editor, with all the difficulty that entails.
3. The proposed editor correctly generates the models that we then use as input in a process of transformations that generate source code (fully functional) in HTML and JavaScript.

As future work, we think we have to improve certain aspects of the “notation dimension” that we have commented and scored in Sects. 4.3 and 6 respectively. In particular we will look at these aspects:

1. “Role-expressiveness”: Show more graphically what the author wants to model when designing activities that are not as obvious as an exercise in filling in gaps. This is the case of the activities of type *Word Ordering* and *Join*, because its design is not evident using the paradigm on which we always rely, “fill-in the gaps”. Actually this dimension (and the improvement outlined) is related to these other two “Abstraction” and “Hard mental operations”.
2. “Visibility”: We think it may be interesting to make it easier for users to preview multimedia resources. We refer to the Media subdiagram files, such as image, audio and video.
3. “Hidden dependencies”: To facilitate the maintenance of workflow diagrams, we find it interesting to show or make more evident the relationship-dependence between states and views.

In addition, some additional work likes could be the following:

1. Incorporate in the editor the possibility of specifying and validating certain restrictions of each method (Duolingo, Busuu, etc.) in relation to for example: types of activities that it supports, restrictions of its content structure, types of media resources that it supports, etc.
2. Continuing with the previous aspect, although the different methods use the same types of activities, in some cases, they do not work visually exactly in the same way, as they may unequally require Ground resources to compose and paint the statements. In this sense, the current graphic editor is limited and should guide the user more when it comes to, for example, modeling activities of type *Dialog*, *Word Ordering*, or *Join*.
3. Improve the graphic aspect in general, eliminating aspects of the interface that distract the user with functionality provided by GMF, but which are not necessary in our case.

Author contributions Gabriel Sebastián and Ricardo Tesoriero elaborated the graphical notation description, Gabriel Sebastián and Ricardo Tesoriero designed the notation evaluation, Gabriel Sebastián and Jose A. Gallud prepared the related work and the conclusions. All authors participated in the elaboration of the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work has been partially supported by the national project granted by the Ministry of Science, Innovation and Universities (Spain) with reference RTI2018-099942-B-I00 and by the project TecnoCRA (ref:

SBPLY/17/180501/ 000495) granted by the regional government (JCCM) and the European Regional Development Funds (FEDER).

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Blackwell AF (1996) Metacognitive theories of visual programming: what do we think we are doing?, in Proceedings 1996 IEEE Symposium on Visual Languages, pp. 240–246
2. Blackwell AF, Britton C, Cox A, Green TRG, Gurr C, Kadoda G, Kutar MS, Loomes M, Nehaniv CL, Petre M, Roast C, Roe C, Wong A, Young RM (2001) Cognitive dimensions of notations: Design tools for cognitive technology. In Beynon M, Nehaniv CL, Dautenhahn K (eds) *Cognitive Technology: Instruments of Mind*, (Berlin, Heidelberg), Springer Berlin Heidelberg pp. 325–341
3. Blackwell AF, Green TR (2003) Notational systems - the cognitive dimensions of notations framework. In Carroll JM, (ed) *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*, San Francisco: Morgan Kaufmann pp. 103–134
4. Bárcena E (2015) State of the art of language learning design using mobile technology: sample apps and some critical reflection. In Proceedings of the 2015 EUROCALL Conference, Padova, Italy, pp. 36–43
5. Bézivin J, Gérard S, Muller P-A, Rioux L (2003) MDA components: Challenges and Opportunities, in Workshop on Metamodelling for MDA, (York, England, United Kingdom), pp 23–41
6. Caro MF, Josyula DP, Jiménez JA, Kennedy CM, Cox MT (2015) A domain-specific visual language for modeling metacognition in intelligent systems. *Biologically Inspired Cognitive Architectures* 13:75–90
7. Calderón A, Boubeta-Puig J, Ruiz M (2018) Medit4cep-gam: A model-driven approach for user-friendly gamification design, monitoring and code generation in cep-based systems. *Inf Softw Technol* 95:238–264
8. Czarnecki K, Helsen S (2003) Classification of model transformation approaches, in OOPSLA03 Workshop on Generative Techniques in the Context of MDA, (California, USA)
9. Dettori G, Lupi V (2010) Ict and new methodologies in language learning. *Procedia - Social and Behavioral Sciences* vol. 2, no. 2, pp. 2712–2716. *Innovation and Creativity in Education*
10. Dix A, Finlay JE, Abowd GD, Beale R (2003) *Human-Computer Interaction* (3rd Edition). Prentice-Hall Inc, USA
11. Dodero JM, del Val ÁM, Torres J (2010) An extensible approach to visually editing adaptive learning activities and designs based on services. *J Vis Lang Comput* 21(6):332–346
12. Dodero JM, Ruiz-Rube I, Palomo-Duarte M, Cabot J (2012) Model-driven learning design. *J Res Pract Inf Technol* 44:267–288
13. Dodero JM (2014) Development of e-learning solutions: Different approaches, a common mission. *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje* 9(2):72–80
14. Fitter M, Green T (1979) When do diagrams make good computer languages? *Int J Man Mach Stud* 11(2):235–261
15. Fielding R (2000) *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, USA
16. Fondement F, Baar T (2005) Making metamodels aware of concrete syntax. In Hartman A, Kreische D (eds) *Model Driven Architecture – Foundations and Applications*, Springer Berlin Heidelberg pp. 190–204

17. Gamma E, Helm R, Johnson R, Vlissides J (1995) *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA
18. García F (2008) *Advances in E-Learning: experiences and methodologies*. Information Science Reference
19. Green T, Petre M (1992) When visual programs are harder to read than textual programs. pp. 167–180
20. Jouault F, Bézivin J, Consel C, Kurtev I, Latry F (2006) Building dsls with amma/atl, a case study on spl and cpl telephony languages, in ECOOP Workshop on Domain-Specific Program Development. Nantes, France
21. Kelly S, Tolvanen J (2008) *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press
22. Kim CH, Grundy J, Hosking J (2015) A suite of visual languages for model-driven development of statistical surveys and services. *J Vis Lang Comput* 26:99–125
23. Koper R (2006) Editorial: Current research in learning design. *J Educ Technol Soc* 9(1):13–22
24. Kopp G (2009) *Handbook of visual languages for instructional design: Theories and practices*. Can J Learn Technol 34:03
25. Laforcade P (2005) Towards a uml-based educational modeling language, In Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05), pp. 855–859
26. Laforcade P (2007) Visualization of learning scenarios with uml4ld. *J Learn Des* 2(2):31–42
27. Lanzilotti R, Ardito C, Costabile MF, Angeli AD (2006) eLSE methodology: A systematic approach to the e-learning systems evaluation. *Educ Technol Soc* 9(4):42–53
28. Leo DH, Villascargas-Fernández ED, Asensio-Pérez JI, Dimitriadis YA, Jorrín-Abellán IM, Ruiz-Requies I, Rubia-Avi B (2006) Collage: A collaborative learning design editor based on patterns. *Educ Technol Soc* 9:58–71
29. Marand EA, Marand EA, Challenger M (2015) Dsm14cp: A domain-specific modeling language for concurrent programming. *Comput Lang Syst Struct* 44:319–341
30. Martínez-Ortiz I, Sierra J-L, Fernández-Manjón B, Fernández-Valmayor A (2009) Language engineering techniques for the development of e-learning applications. *J Netw Comput Appl* 32(5):1092–1105
31. Nielsen J, Mack RL (eds) (1994) *Usability Inspection Methods*. John Wiley & Sons Inc, USA
32. Norman DA (2002) *The Design of Everyday Things*. Basic Books Inc, USA
33. Ráth I, Schmidt A, Vago D (2005) *Automated model transformations in domain specific visual languages*, Scientific Students' Associations Report
34. Sampson D, Karampiperis P, Zervas P (2005) Ask-ldt: A web-based learning scenarios authoring environment based on ims learning design. *Adv Technol Learn* 2:207–215
35. Schmidt DC (2006) Guest editor's introduction: Model-driven engineering. *IEEE Computer* 39(2):25–31
36. Sebastián G, Tesoriero R, Gallud JA (2017) Modeling language-learning applications. *IEEE Lat Am Trans.* 15(9):1771–1776
37. Sebastián G, Tesoriero R, Gallud JA (2018) Model-based approach to develop learning exercises in language-learning applications. *IET Software* vol. 18, pp. 206–2014
38. Sebastián G, Gallud JA, Tesoriero R (2020) Code generation using model driven architecture: A systematic mapping study. *Journal of Computer Languages* 56
39. Shneiderman B, Plaisant C, Cohen M, Jacobs S (2009) *Designing the user interface: strategies for effective human-computer interaction*. USA: Addison-Wesley Publishing Company, 5th edition
40. Sierra JL, Fernández-Manjón B, Fernández-Valmayor A (2008) A language-driven approach for the design of interactive applications. *Interacting with Computers* 20(1):112–127
41. Sockett G (2014) *The online informal learning of English*, p. 11. Palgrave Macmillan UK
42. Torres J, Resendiz J, Aedo I, Dodero JM (2014) A model-driven development approach for learning design using the lpcel editor. *J King Saud Univ - Comput Inf Sci* vol. 26, no. 1, Supplement, pp. 17–27
43. Torres J, Dodero JM, Aedo I, Diaz P (2006) Designing the execution of learning activities in complex learning processes using lpcel, In Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06), pp. 415–419
44. Troya J, Vallecillo A (2014) Specification and simulation of queuing network models using domain-specific languages. *Comput Stand Interfaces* 36(5):863–879
45. Watts N (1997) A learner-based design model for interactive multimedia language learning packages. *System* 25(1):1–8