Automated Migration of EuGENia Graphical Editors to the Web

Fatima Rani
UGROUND GLOBAL, S.L
Madrid, Spain
frani@uground.com

Pablo Diez
UGROUND GLOBAL, S.L
Madrid, Spain
pdiez@uground.com

Enrique Chavarriaga
UGROUND GLOBAL, S.L
Madrid, Spain
echavarriaga@uground.com

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain
esther.guerra@uam.es

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
Juan.deLara@uam.es

ABSTRACT

Domain-specific languages (DSLs) are languages tailored for particular domains. Many frameworks and tools have been proposed to develop editors for DSLs, especially for desktop IDEs, like Eclipse.

We are witnessing the advent of low-code development platforms, which are cloud-based environments supporting rapid application development by using graphical languages and forms. While this approach is very promising, the creation of new low-code platforms may require the migration of existing desktop-based editors to the web. However, this is a technically challenging task.

To fill this gap, we present *ROCCO*, a tool that migrates Eclipse-based graphical modelling editors to the web, to facilitate their integration with low-code platforms. The tool reads a meta-model annotated with `EuGENia` annotations, and generates a web editor using the `DPG` web framework used by the UGROUND company. In this paper, we present the approach, including tool support and an evaluation based on migrating nine editors created by third parties, which shows the usefulness of the tool.

CCS CONCEPTS

• **Software and its engineering → Domain specific languages**.

KEYWORDS

Model-Driven Engineering, Graphical DSLs, Low-code platforms

ACM Reference Format:
Fatima Rani, Pablo Diez, Enrique Chavarriaga, Esther Guerra, and Juan de Lara. 2020. Automated Migration of EuGENia Graphical Editors to the Web. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3417990.3420205

1 INTRODUCTION

Model-Driven Engineering (MDE) [29] is a software paradigm that prescribes an active use of models during the construction process. Thus, models are used to describe, validate, simulate, generate code and maintain the software, among many other activities. These models can be built using general-purpose modelling languages, such as the Unified Modelling Language (UML), but the use of Domain-Specific Languages (DSLs) is also common [16].

DSLs are languages tailored to a specific problem, containing customized concepts representing the abstractions within the domain [10, 21, 22]. This way, the definition of (graphical) DSLs, and their modelling environments is recurrent when building MDE solutions [15]. The benefit of using a DSL is that it can lead to more effective solutions within a domain, while allowing end-users with little technical background to perform programming tasks [17]. In the last years, many tools have been developed to automate the construction of DSL editors [5, 8, 11–14, 19, 25, 30, 34], especially since the popularization of extensible IDEs like Eclipse. Still, the development of DSL editors, in particular for graphical DSLs is a challenging task.

We have recently witnessed the advent of low-code development platforms, which are cloud-based environments supporting the rapid development of applications by using graphical languages and forms. While this approach is very promising, the creation of new low-code platforms may profit from the reuse of existing MDE solutions in a cloud environment. This implies the migration of existing desktop-based editors to the web. However, without automation, this is a technically challenging task.

In order to fill this gap, we propose a tool to automate the migration of Eclipse-based graphical DSL editors to the web. In particular, the tool supports editors built using `EuGENia` [8] and creates an editor for `DPG-PsiEngine` [2, 3]. This is a web-based framework used within UGROUND as a basis for its ROSE low-code development platform [7]. This way, our solution enables the automated reuse of DSL editors within low-code environments, eliminating the need for re-developing such editors. We have evaluated our tool by migrating nine editors built by third parties, obtaining good results, and demonstrating the applicability of this tool in practice.

The rest of this paper is organized as follows. Section 2 provides background on technologies used in our work, and introduces a running example. Section 3 describes our approach to migration, and Section 4 tool support. Section 5 presents an evaluation. Section 6 compares with related work and Section 7 presents conclusions and future work.

2 BACKGROUND AND MOTIVATION

In this section we introduce the different technologies involved in our approach, on the basis of a running example.

2.1 Eclipse Modeling Framework and EuGENia

The Eclipse Modeling Framework (EMF) [9, 31] is a widely used (partial) implementation of the meta-object facility (MOF) standard of the OMG [23]. EMF is a meta-modelling framework integrated within Eclipse. It supports a notation (Ecore) for creating meta-models, and a code generation facility that produces Java code (enabling the construction of models and model transformations programmatically) and tree editors to create models interactively. EMF supports model serialization using the XML Metadata Interchange (XMI), an OMG standard for meta-data exchange [35].

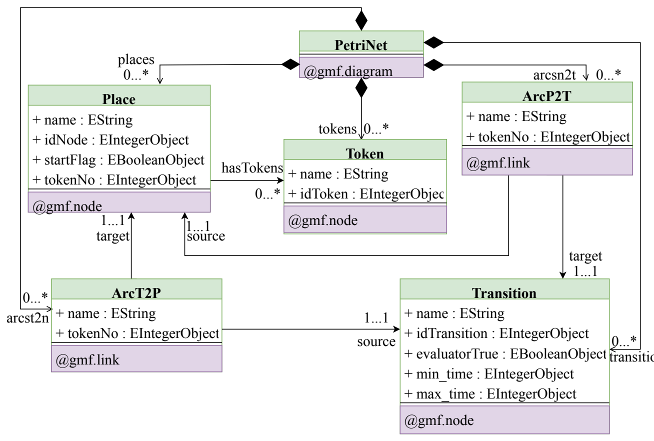


Figure 1: Ecore meta-model for Petri nets with EuGENia annotations

As a running example, assume we would like to create a modelling editor for Petri nets [24] using EMF. Petri nets are a popular formalism to model concurrent system, made of places, which may hold zero or more tokens, and that can be connected to transitions via input and output arcs. Figure 1 shows the ecore meta-model of Petri nets with EuGENia annotations. The meta-model supports Petri nets with weighted arcs (attribute *tokenNo* in *ArcPT* and *ArcTP*) and time in transitions to model delays. By convention, most Ecore meta-models have a root class, which contains directly or indirectly all other classes via composition relations. This way, each class – except the root class – is contained into another one. This is especially useful to edit models using the generated tree editors, since the instances of the contained classes appear as children of the container ones.

While EMF generates a tree editor by default, more user-friendly editors are typically required. They are normally either textual or graphical. In this paper we focus on graphical DSLs. There are several technologies to help in the construction of graphical editors within Eclipse, like Graphiti [12], Sirius [30], GMF [11] or EuGENia

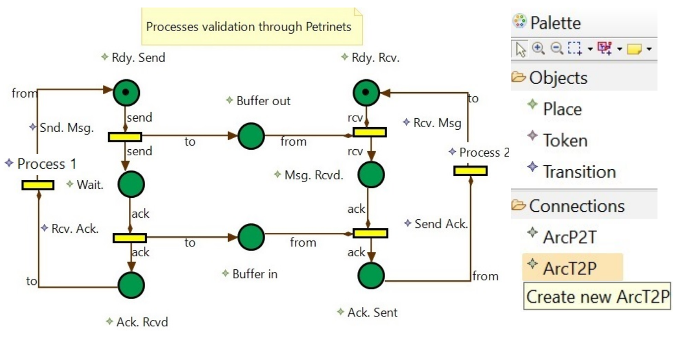


Figure 2: EuGENia graphical editor for Petri nets

[8] (which we analyse in Section 6). We decided to support the latter technology due to its popularity, since it is extensively used by both researchers and practitioners.

`EuGENia` [18] simplifies the development of GMF-based graphical model editors by automatically generating the required models needed by the GMF editor construction framework from a single annotated Ecore meta-model.

As an example, Figure 2 shows some annotations indicating that *Places*, *Transitions* and *Tokens* are to be displayed as nodes (annotation *gmf.node*), while arcs are to be represented as links (annotation *gmf.link*), and all elements are to be placed in the diagram represented by the *PetriNet* (annotation *gmf.diagram*). Figure 5 shows the resulting editor. It contains a canvas to graphically edit the model, and a palette to create the objects. The object attributes can be changed from an Eclipse properties view (not shown in the Figure).

2.2 DPG (Diagram Programming Generator)

The web technology that we target in this work is based on the Programmable Solutions Interpreter Engine, (`PsiEngine`) [2, 3]. `PsiEngine` implements, evaluates, interprets and executes DSLs described in XML within the web client. It uses HTML5, CSS3, JavaScript and DOM together with technologies, services and tools from Web 2.0 and the specification of XML-DSL grammars in order to build web components, widgets, and dynamic web sites to give the solution to specific web application problems or parts of them.

`PsiEngine` is a generic lightweight JavaScript framework (it a cross browser platform) that processes and evaluates programs written in Psi Language [2]. On top of `PsiEngine`, a set of Psi languages (Graph Library Psi GLPsi, Diagram Psi DPsi, Visual Tool Psi TPsi and Data Form Psi DFPsi) were created to develop the functionality of a programmable diagram.

Formally, a programmable diagram is a set of graphical elements that define an SVG-based diagram, and each graphical element can have associated visual tools (like dialogue boxes, toolbars, pop-ups, floating menus, menus, and drag and drop), programming utilities (classes, scripts, functions, and variables) and heterogeneous information data sources (XML/JSON) that determine their appearance and content.

The Diagram Programming Generator (DPG) is a layer on top of the `PsiEngine` that encodes DSLs as a JSON-based grammar, which specifies the elements necessary to create a programmable diagram

Table 1: Mappings between EuGENia and DPG for figures and decorations

figure		source/target.decoration		border.styles	
EuGENia	**DPG-PsiEngine**	**EuGENia**	**DPG-PsiEngine**	**EuGENia**	**DPG-PsiEngine**
`rectangle ->`	Box	`arrow ->`	rightarrow	`solid ->`	".line-CT-CT ",
`ellipse ->`	Ball	`closedarrow ->`	rightarrow	`dash ->`	".line-CT-CT
`rounded(default) ->`	Box	`filledclosedarrow ->`	rightarrow		{stroke-dasharray:5,5;}",
`polygon ->`	Rhombus	`rhomb ->`	circlecross	`dot ->`	".line-CT-A
`svg (svg.uri)->`	"Url":"..." , "Doc": "..."	`filledrhomb ->`	circlecross		{stroke-dasharray:2,2;}",
`java class name ->`	"Shape":"Image"	`square ->`	circle		
	"Field":"IMG_URL"	`filledsquare ->`	circle		
	"Symbol":"name_symbol"	`java class ->`	java class		

in PsiEngine. The execution engine that interprets a DPG specification, is called DPG-PsiEngine. Its objective is to generate the code in the different Psi languages and to start the programmable diagram. Additionally, in DPG-PsiEngine a template engine was incorporated for the administration and generation of graphical elements and forms. It also includes components for connection via REST API, to obtain JSON information. These frameworks are used within UGROUND as a basis for the low-code solutions, and some examples are available at http://devrho.com/.

3 AUTOMATING EDITOR MIGRATION

This section presents our migration approach, we first describe how the abstract syntax is migrated (Section 3.1), and then the graphical concrete syntax (Section 3.2).

3.1 Migrating the abstract syntax

Our approach starts reading the domain meta-model and transforming the classes into the DSL-JSON format of DPG. All the (non-abstract) classes in the EMF meta-model are placed inside a DPG table called *"Elements":* {...}. Then, inside such table, they are placed into a key *"Type":* {...}. Later, classes need to be classified as nodes or connectors, as we will explain in next section. All the attributes and references of the classes are stored into a key *"Fields":* {...}, and then attributes are associated with different types of DPG controls depending on their eType, as shown in Table 2. The source and target of references are specified using the JSON key *"Start":* and *"End":*.

Table 2: Mapping EMF types to DPG Controls

EMF	DPG-PsiEngine
boolean	checkbox(disabled=0)
int, long, short, double, float	number
char	text(disabled=no)
String	textarea(disabled=no)
Date	date (configDate= 'format=dd/mm/yyyy'; configTime='autoclose=true')

In EMF meta-models, we may have abstract classes, and inheritance hierarchies. However, neither abstract classes or inheritance are supported by DPG. To overcome this limitation, our mapping *flattens* the inheritance hierarchy, collecting the attributes and references of upper classes in the hierarchy, and replicating them in lower classes where they are required.

3.2 Migrating the concrete syntax

Our approach also translates EuGENia annotations to the DPG format, which distinguishes *nodes* and *connectors*. Hence, we have to decide on the basis of EuGENia annotations which classes or references of EMF meta-model are going to become nodes, and which ones connectors. In addition, we need to consider the different graphical styles configured in these annotations.

Listing 1: Node and Link annotations in EuGENia

```
@gmf.node(
    label="name",figure="ellipse",border.styles="dash",
     color ="0,153,76",label.color = "0,0,0",border.color
     ="0,0,0",border.width="3"
    )
@gmf.link(
    label="name",source="source",target="target",style="
     dash",width="2",color="102,51,0",source.decoration="
     none",target.decoration="filledrhomb"
    )
```

An example of EuGENia annotations is shown in Listing 1. For the case of nodes, using key-value pairs, the DSL designer can specify the attribute to be shown as label of the node (key *label*), the figure representing the node (*ellipse*), the style of the figure border (*border.styles*), the colours of the figure background, label and border (*color*, *label.color*, *border.color*) and the border width (*border.width*).

Listing 2: DPG Code for nodes and connectors

```
Elements Classes
GraphLibraryNODE:
 {
 "GraphLibrary":
 {
    "Config": {
        "Shape": "Ball",
        "Styles":
            "boxFill=rgb(0,153,76);
            nameColorFont=rgb(0,0,0);
            boxStroke=rgb(0,0,0);
            boxStrokeWidth=3px"
    }
 }
GraphLibraryCONNECTOR:
 {
    "EndMarker": "rightarrow",
    "GraphLibrary":
        "class=line-CT-CT;
         stroke-width=2;
         stroke=rgb(102,51,0);
         minimum=none"
 }
```

For links attached to nodes, we need to define their source and target references as (keys *source* and *target*), and we can also specify a label for the link (*label*) and graphical styles including colour, width, line style, and decorations for the source and targets of the link. Details on additional EuGENia styles can be found at [8].

In DPG the graphical information is stored in JSON, where we need to define libraries of nodes and connectors, as Listing 2 shows. Please note that DPG uses CSS styles, e.g., for colour values.

Table 3: Mapping EuGENia and DPG styles for nodes/links

EuGENia	DPG-PsiEngine
@gmf.node	**NODE**
border.color	boxStroke or stroke
border.styles	as in Table 1
border.width	boxStrokeWidth or strokeWidth
color	boxFill or fill
figure	as in Table 1
label	name
label.color	nameColorFont or colorFont
tool...(fields)	icon (symbol & color)
@gmf.link	**CONNECTOR**
color	fill or stroke
incoming	"Starts":
label	name
source	"Starts":
source.decoration	as in Table 1
style	as in Table 1
target	"Ends":
target.decoration	as in Table 1
width	strokeWidth
tool...(fields)	icon (symbol & color)

The details of the mapping of different styles of nodes and links between EuGENia and DPG are described in Tables 1 and 3. The former table details the mappings between figures and decorations for links, and the latter describes the mapping between the other annotation options.

In DPG, when attributes of nodes or links are to be edited, a dialog box is presented. As explained in the previous section (cf., Table 2), each attribute can be associated with a different control, depending on its eType, to properly edit its value. Finally, the information about the palette, and the corresponding icons are also translated.

Regarding limitations of our translation, we do not currently support the EuGENia annotation @gmf.affixed to attach nodes to the border of another one. In addition, the @gmf.compartment annotation, which allows node nesting, is not natively supported either. However, we provide a workaround that allows inserting elements in the container objects (we will see an example in Section 5).

4 TOOL SUPPORT

We have created a tool called **ROCCO** (Mig***R***ati***O***n towards ***C***loud-based Graphi***C***al Edit***O***r) that implements the previous mapping. ROCCO is an Eclipse plugin that reads Ecore meta-models with EuGENia annotations and synthesises the necessary DPG files. For this purpose, it uses the Acceleo code generation language. The architecture of the tool is shown in Figure 3, which also shows the generation process.

A screenshot of the *ROCCO* IDE is shown in Figure 4 in which *labels (1,3)* indicate the main generic templates and the different modules and files templates. *Label 2* shows the meta-model of

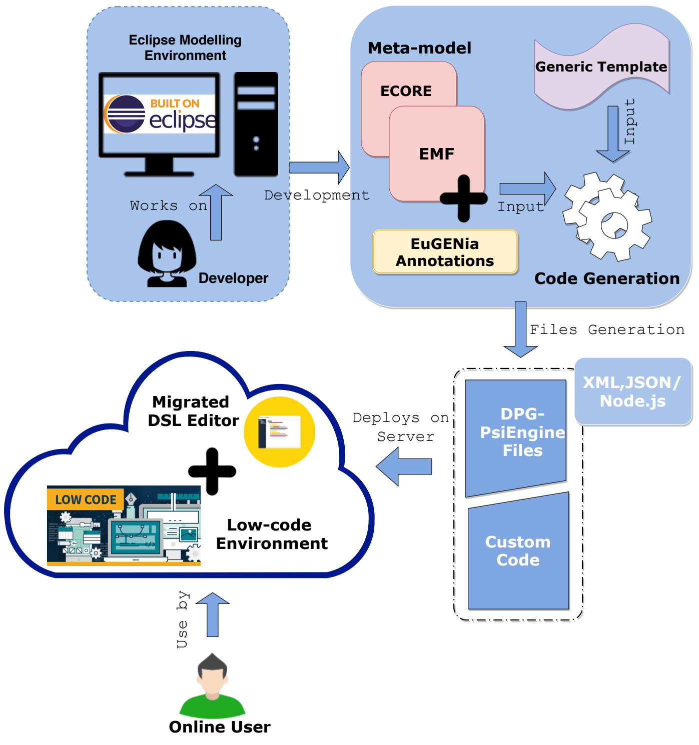


Figure 3: Code Generation Template Schema

the running example with EuGENia annotations. It is shown using the emfatic editor for Ecore models. *Label 4* are the files required to support the graphical editor of the DPG-PsiEngine Platform. *Label 5* signals the specification file generated for DPG-PsiEngine. *Label 6* is the generated Acceleo project. Finally *Label 7* contains the meta-models with EuGENia annotations to be migrated.

Figure 5 shows the resulting DPG editor for the running example. It can be seen how the synthesized editor mimics well the original one defined with EuGENia (cf. Figure 2).

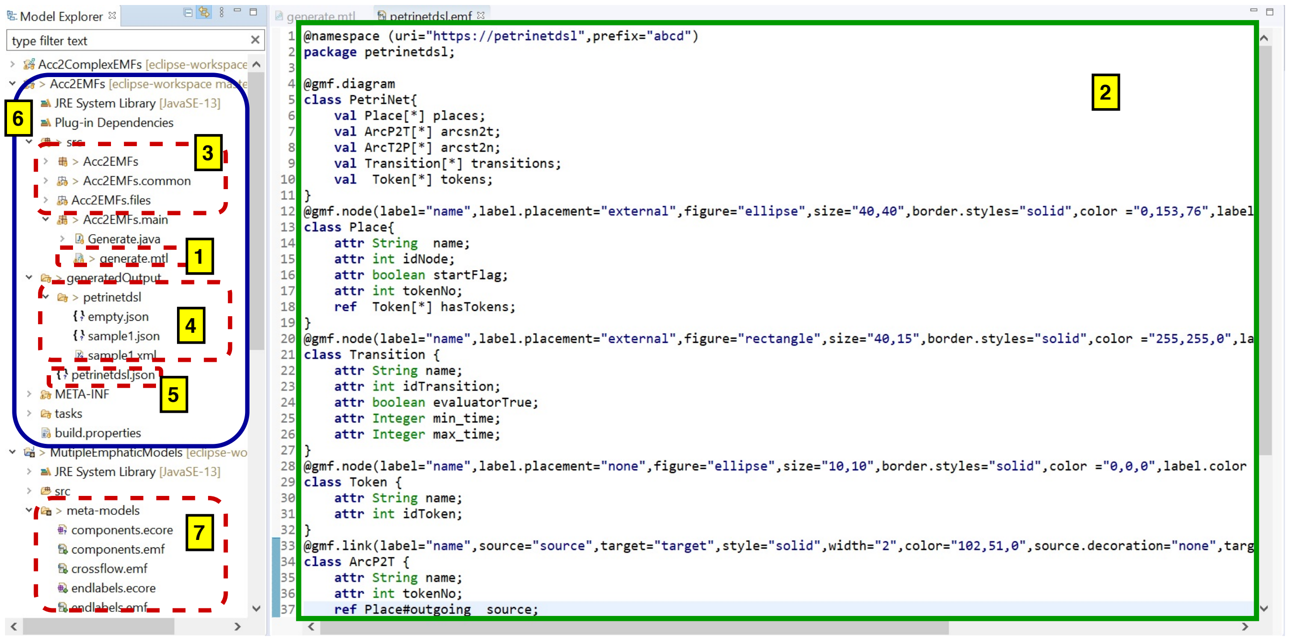
5 EVALUATION

In this section, our goal is to answer the following research question: *Can ROCCO migrate Eclipse-based graphical modelling editors (EuGENia) to the web, to facilitate their integration with low-code platforms?*

For this purpose, we have evaluated our tool by migrating existing EuGENia editors into DPG-PsiEngine. We have taken nine EuGENia editors from the Epsilon online repository [1] (taking all meta-models with EuGENia annotations), and checked whether a complete DPG editor is obtained, required manual changes or had lacking functionality.

The evaluation results are shown in Table 4. The table shows the meta-model size in classes, and the lines of code of the generated DPG specifications. Then it shows the number of node, link (applied to nodes and references), compartment and affixed annotations. We

[1]https://git.eclipse.org/c/epsilon/org.eclipse.epsilon.git/plain/examples/



```
@namespace (uri="https://petrinetdsl",prefix="abcd")
package petrinetdsl;

@gmf.diagram
class PetriNet{
    val Place[*] places;
    val ArcP2T[*] arcsn2t;
    val ArcT2P[*] arcst2n;
    val Transition[*] transitions;
    val  Token[*] tokens;
}
@gmf.node(label="name",label.placement="external",figure="ellipse",size="40,40",border.styles="solid",color ="0,153,76",label
class Place{
    attr String  name;
    attr int idNode;
    attr boolean startFlag;
    attr int tokenNo;
    ref  Token[*] hasTokens;
}
@gmf.node(label="name",label.placement="external",figure="rectangle",size="40,15",border.styles="solid",color ="255,255,0",la
class Transition {
    attr String name;
    attr int idTransition;
    attr boolean evaluatorTrue;
    attr Integer min_time;
    attr Integer max_time;
}
@gmf.node(label="name",label.placement="none",figure="ellipse",size="10,10",border.styles="solid",color ="0,0,0",label.color
class Token {
    attr String name;
    attr int idToken;
}
@gmf.link(label="name",source="source",target="target",style="solid",width="2",color="102,51,0",source.decoration="none",targ
class ArcP2T {
    attr String name;
    attr int tokenNo;
    ref Place#outgoing  source;
```

Figure 4: The ROCCO tool in action

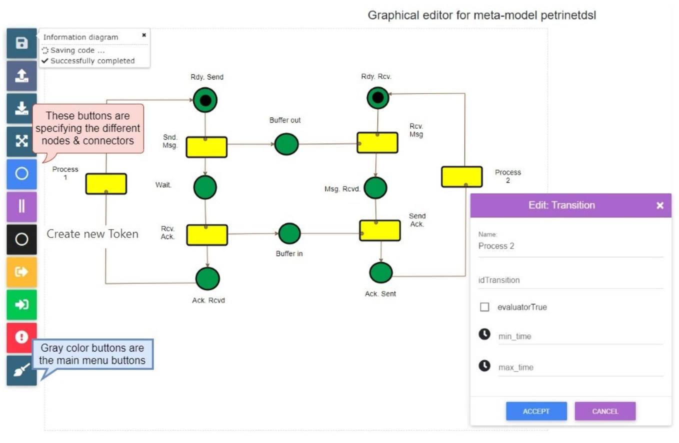


Figure 5: DPG-PsiEngine graphical editor for Petri nets

marked with Yes/No whether the different annotations were correctly translated between these two platforms. For containment, we used Yes*, since we emulate compartments as graphical areas with less functionalities than in EuGENia. Overall, we could fully migrate eight out of the nine editors. For the other one (*components*), we obtained a working editor, but with lacking functionality regarding affixed features. In the EuGENia editor ports can be affixed to components, to appear in their borders. Instead they are connected via links in DPG editor.

As an example, Figure 6 shows the BPMN editor in EuGENia and the resulting one in DPG. We could emulate lanes (containment compartments in EuGENia) as rectangles. However, these have less functionality than the ones in EuGENia for handling the children elements.

Overall, we can answer our research question in a positive way, since in all cases we obtained a working editor, albeit with lacking functionality for affixed, and less sophistication for compartments. These limitations will be addressed in future work.

6 RELATED WORK

Next, we review some of the main frameworks to define graphical DSLs both for desktop environments (especially Eclipse) and on the web.

The most widely used MDE environment nowadays is the Eclipse Modelling Project of the Eclipse Modeling Framework [9], which can integrate with different tools for specifying DSLs. First, we review some tools in the Eclipse ecosystem, whether they are desktop applications or cloud-based IDE/tool, then we talk about some other tools outside the Eclipse ecosystem in the same manner.

Graphiti [12] is a framework that allows building graphical editors programmatically in Java. The Graphical Modeling Framework (GMF) [11] is a model-based solution for developing graphical editors, which requires specifying three types of models: GMF-map (describing the structure), GMF-graph (describing the apperance) and GMF-tool (with the editor behaviour). GMF and Graphiti are large and powerful frameworks, and consequently their use requires considerable technical expertise.

EuGENia [8] seeks to reduce the amount of technical experience required to specify a DSL with GMF, by providing higher level annotations that are mapped into a subset of the underlying framework.

Sirius [30], is a more recent framework for building graphical editors within Eclipse. It is also model-based, but it is interpreted,

Table 4: Results of the evaluation.

			EClass @gmf.node		EClass @gmf.link		EReference (non-containment) @gmf.link		EReference (containment) @gmf.compartment		EReference (containment) @gmf.affixed	
Meta-Models	MM Size	DPG LOCs	EuGENia	DPG	EuGENia	DPG	EuGENia	DPG	EuGENia	DPG	EuGENia	DPG
scl	3	100	2	Yes	1	Yes	1	Yes	1	Yes*	0	0
petrinetdsl	6	149	3	Yes	2	Yes	0	0	0	0	0	0
components	5	86	2	Yes	1	Yes	0	0	0	0	1	No
bpmn	21	352	10	Yes	4	Yes	0	0	2	Yes*	0	0
fed	4	134	1	Yes	0	0	2	Yes	1	Yes*	0	0
filesystem	6	161	3	Yes	1	Yes	0	0	0	0	0	0
friends	2	62	1	Yes	0	0	2	Yes	0	0	0	0
rcpapp	2	65	1	Yes	0	0	0	0	0	0	0	0
widgets	2	65	1	Yes	0	0	0	0	0	0	0	0

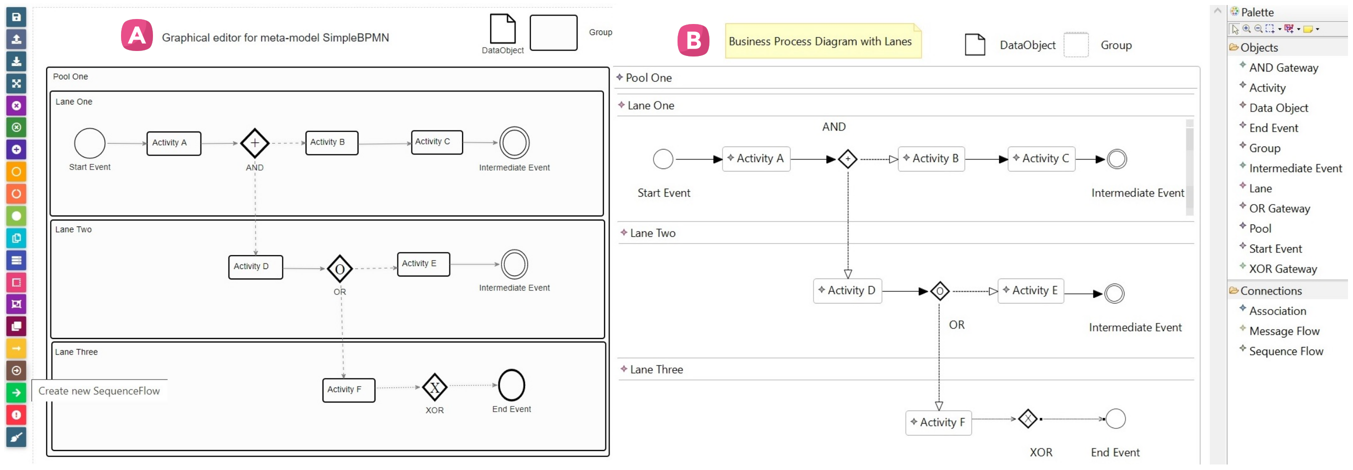


Figure 6: Business Process Graphical Editor: A) in DPG-PsiEngine B) in EuGENia

instead of compiled (like GMF). While there are plans to enable the deployment of Sirius editors on the web, this is still not supported.

Obeo Designer [6] is a commercial framework which allows user to create own graphical workbenches for specific domains like software applications, industrial systems and organization of major companies. It also integrates with Sirius and other MDE tools. It facilitates the collaborative work, allows user to store models and representations (diagrams, tables, matrices, trees) in a shared repository. Multiple users can work on the same data simultaneously without any conflict and technical skills. Users can also import/export the project from the standard data access point i.e CDO, the Eclipse open-source solution to store EMF models into a database.

Outside the Eclipse ecosystem, MetaEdit+ [1] is a graphical language workbench, which allows specifying domain-specific modelling languages by defining their abstract syntax, concrete visual syntax and semantics (via code generators). It supports graphical notations as well as textual, symbolic or tabular notations. MetaEdit+ uses a proprietary persistence format, therefore models and languages created with MetaEdit+ cannot be directly used with other modelling tools or language workbenches.

The previous tools do not provide a framework which can run on cloud for their integration with low-code development platform. However, some solutions exist to build web-based editors for graphical DSLs. For example, WebGME [20] is a tool to create graphical DSLs directly in the browser[2]. It uses a UML class diagram-based meta-model to specify the modelling concepts, relationships and attributes. It also supports model versioning and collaboration on the cloud.

AToMPM [32] is a web version of AToM[3] [5]. It allows defining graphical DSL editors that run on the web[3], and to specify DSL semantics using graph transformations. It supports two types of collaboration mechanisms distributed in real time. *Screenshare* allows two or more clients to share exactly the same drawing area: any modification made to a model (abstract or concrete syntax) is replicated on all observing clients. *Modelshare* only shares the abstract syntax of a model between clients.

EMF.cloud[4] is a project – still under development – aiming at making EMF-based technologies accessible via the cloud, including

[2]https://www.webgme.org/
[3]https://atompm.github.io/
[4]https://www.eclipse.org/emfcloud/

graphical editors. These are based on the Graphical Language Server Platform (GLSP) [26, 27][5]. GLSP transfers the advantages of the language server protocol (LSP) to graphical modelling languages. These editors, then can be deployed on web-based IDE such as Eclipse Theia.

Finally, EuGENia Live [28] is a web-based tool for designing graphical DSLs. It encourages the construction and collaboration of models and meta-models in iterative and incremental development.

Altogether, we have analysed several tools to create graphical editors either for desktop applications or for the web. However, we are not aware of solutions supporting editor migration to the web, to enable their integration with low-code development platforms.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an approach for the automated migration of `EuGENia` editors to the web. The target framework is DPG, a web framework developed and used in the UGROUND company as a basis for its ROSE low-code platform. We have evaluated the approach on editors created by third parties, showing whether a complete DPG editor is obtained, required manual changes or had lacking functionality.

In the future, we plan to improve the mapping with support for affixed nodes, and better support for compartments. We are working on creating a default web editor for meta-models with no `EuGENia` annotations. Moreover, we would like to profit from other Psi languages to map editing actions, like refactorings or model abstraction operations [4] to enable more scalable modelling and connect models of heterogeneous languages.

We would also like to improve our migration to consider not only editors, but also existing instance models. Technically, we would like to improve the tool to automate deployment, and to offer a better GUI. We also plan to support the migration of editors built with other frameworks, like Sirius.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement n° 813884, Lowcomote [33]. The work has also been supported by the Spanish Ministry of Science (project MASSIVE, RTI2018-095255-B-I00) and the R&D programme of Madrid (project FORTE, P2018/TCS-4314).

REFERENCES

[1] Meta Case. 1991. *MetaEdit+ Workbench*. Retrieved May 11 - June 26, 2020 from https://www.metacase.com/mwb/
[2] Enrique Chavarriaga, Francisco Jurado, and Fernando Díez. 2017. An approach to build XML-based domain specific languages solutions for client-side web applications. *Computer Languages, Systems & Structures* 49 (2017), 133–151.
[3] Enrique Chavarriaga, Francisco Jurado, and Fernando Díez. 2017. PsiLight: a Lightweight Programming Language to Explore Multiple Program Execution and Data-binding in a Web-Client DSL Evaluation Engine. *J. UCS* 23, 10 (2017), 953–968.
[4] Juan de Lara, Esther Guerra, and Jesús Sánchez Cuadrado. 2013. Reusable abstractions for modeling languages. *Inf. Syst.* 38, 8 (2013), 1128–1149.
[5] Juan De Lara and Hans Vangheluwe. 2002. AToM 3: A Tool for Multi-formalism and Meta-modelling. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 174–188.
[6] Obeo Designer. 2008. *Obeo Designer Workbench*. Retrieved May 11 - June 26, 2020 from https://www.obeodesigner.com/
[7] Alfonso Diez, Nga Nguyen, Fernando Díez, and Enrique Chavarriaga. 2013. MDE for Enterprise Application Systems. In *MODELSWARD*. SciTePress, 253–256.
[8] Eclipse EuGENia. 2004. *Graphical Model Editor development with EuGENia/GMF*. Retrieved May 11 - June 26, 2020 from https://www.eclipse.org/epsilon/doc/eugenia/
[9] Eclipse Foundation. 2004. *Eclipse Modeling Framework*. Retrieved May 11 - June 26, 2020 from https://www.eclipse.org/modeling/emf/
[10] M Fowler. 2010. Domain specific languages. Addison-Wesley Professional. *Boston, MA, USA* (2010).
[11] Eclipse GMF. 2004. *GMF Tooling*. Retrieved May 11 - June 26, 2020 from https://www.eclipse.org/gmf-tooling/
[12] Eclipse Graphiti. 2004. *Graphiti*. Retrieved May 11 - June 26, 2020 from https://eclipse.org/graphiti/
[13] Richard C Gronback. 2009. *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education.
[14] John C Grundy, John Hosking, Karen Na Li, Norhayati Mohd Ali, Jun Huh, and Richard Lei Li. 2012. Generating domain-specific visual language tools from abstract visual specifications. *IEEE Transactions on Software Engineering* 39, 4 (2012), 487–515.
[15] John Hutchinson, Jon Whittle, and Mark Rouncefield. 2014. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming* 89 (2014), 144–161.
[16] Steven Kelly and Juha-Pekka Tolvanen. 2008. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons.
[17] Andrew J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, et al. 2011. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)* 43, 3 (2011), 1–44.
[18] Dimitrios S Kolovos, Antonio García-Domínguez, Louis M Rose, and Richard F Paige. 2017. Eugenia: towards disciplined and automated development of GMF-based graphical model editors. *Software & Systems Modeling* 16, 1 (2017), 229–255.
[19] Dimitrios S Kolovos, Louis M Rose, Saad Bin Abid, Richard F Paige, Fiona AC Polack, and Goetz Botterweck. 2010. Taming EMF and GMF using model transformation. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 211–225.
[20] Miklós Maróti, Tamás Kecskés, Róbert Kereskényi, Brian Broll, Péter Völgyesi, László Jurácz, Tihamer Levendovszky, and Ákos Lédeczi. 2014. Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure. *MPM@ MoDELS* 1237 (2014), 41–60.
[21] Marjan Mernik. 2012. *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments: Recent Developments*. IGI Global.
[22] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.
[23] MOF. 2016. http://www.omg.org/spec/MOF.
[24] T. Murata. 1989. Petri nets: Properties, analysis and applications. *Proc. IEEE* 77, 4 (1989), 541–580.
[25] Eclipse Picto. 2004. *Visualising Models with Picto*. Retrieved May 11 - June 26, 2020 from https://www.eclipse.org/epsilon/doc/picto/
[26] Roberto Rodríguez-Echeverría, Javier Luis Cánovas Izquierdo, Manuel Wimmer, and Jordi Cabot. 2018. An LSP infrastructure to build EMF language servers for web-deployable model editors. In *Proceedings of MODELS 2018 Workshops (CEUR Workshop Proceedings, Vol. 2245)*. CEUR-WS.org, 326–335.
[27] Roberto Rodríguez-Echeverría, Javier Luis Cánovas Izquierdo, Manuel Wimmer, and Jordi Cabot. 2018. Towards a Language Server Protocol Infrastructure for Graphical Modeling. In *Proceedings of the 21th ACM/IEEE MODELS*. ACM, 370–380.
[28] Louis M Rose, Dimitrios S Kolovos, and Richard F Paige. 2012. Eugenia live: a flexible graphical modelling tool. In *Proceedings of the 2012 Extreme Modeling Workshop*. 15–20.
[29] Douglas C Schmidt. 2006. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-* 39, 2 (2006), 25.
[30] Eclipse Sirius. 2004. *Sirius*. Retrieved May 11 - June 26, 2020 from https://eclipse.org/sirius/
[31] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: eclipse modeling framework*. Pearson Education.
[32] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Hüseyin Ergin. 2013. AToMPM: A Web-based Modeling Environment. *Demos/Posters/StudentResearch@ MoDELS* 2013 (2013), 21–25.
[33] Massimo Tisi, Jean-Marie Mottu, Dimitrios S. Kolovos, and Juan De Lara. 2019. Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms. In *2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems (CEUR Workshop Proceedings (CEUR-WS.org))*. https://doi.org/10.5220/0006555700710082
[34] Juha-Pekka Tolvanen and Steven Kelly. 2008. Domain-specific modeling: Enabling full code generation. *Wiley-IEEE Computer Society* 444 (2008), 231.
[35] XMI. 2015. https://www.omg.org/spec/XMI/.

[5] https://www.eclipse.org/glsp/