An Overview of the Landscape and Trends of (Semi-)Automatically Generated Graphical Editors for Modeling

Andreas Huber

Technische Universität Dresden, Dresden, Germany
andreas.huber2@mailbox.tu-dresden.de

Abstract. Graphical representations and editing capabilities of models in model-driven software engineering facilitate reaching a broad range of users. There are several approaches and tools for generating graphical editors based on descriptions of a problem domain. 21 are analyzed describing the execution domain, general approach, and mapping from domain elements to their graphical representations. Based on this analysis, trends can be derived. In the most recent development, editors are exclusively executed in web applications or as plugins to IDEs. Furthermore, a trend towards abstraction layers for hiding complexity on top of existing solutions can be observed.

1 Introduction

As the term itself already implies, models are of central importance in model-driven software engineering. Metamodels or other formalisms like grammars are able to express the structure of virtually any problem domain but to empower a broad range of users besides developers to leverage this expressiveness, an uncomplicated way of designing models has to be employed. Graphical modeling is a way of facilitating this uncomplicated way. Nevertheless, graphical editors for models conforming to the formalisms describing a certain domain have to be designed and implemented. For not having to reinvent the wheel for each and every domain or evolution of its model, there are tools and approaches for (semi-)automatically generating editors out of these specifications. In this work, an overview of this landscape and developments and trends is to be presented. Therefore, two research questions are posed.

> ***Research Question 1****: What approaches and tools for (semi-)automatically generating graphical modeling editors exist?*
>
> ***Research Question 2****: What trends or developments can be observed?*

For answering the first research question, an overview of existing works is given in Section 2. Based on this overview, a visual representation arranging the works according to their years of publication, execution domain, and interdependence is drawn in Figure 1. This figure allows for deriving findings that answer the second research question in Section 3.

2 Tools and Approaches for Generating Graphical Modeling Editors

This section presents an overview of the discovered tools and approaches for (semi-)automatically generating graphical modeling editors. Therefore, it tries to answer the first research question. 21 solutions were identified. Each found tool or approach is described shortly with a focus on the general approach and artifact generation, how graphical representations are specified and composed, and how model elements described in formalisms like metamodels or grammars are mapped to their graphical representations.

2.1 GMF

GMF (Graphical Modeling Framework) is a toolkit for visualizing and editing models described in Ecore, a meta-meta model for specifying metamodels in the *Eclipse Modeling Framework*[1] (EMF). The automatically generated implementations of the editors are based on *Eclipse GEF*, the Graphical Editing Framework[2], a graphical tool suite following the Model-View-Controller-Pattern (MVC). All generated editor components are implemented and packaged as Eclipse plugins integrating well with the automatically generated EMF code based on the domain model. GMF employs this implementation for serializing the graphically created and edited models [Gro09].

For defining the editor, a model-driven approach is utilized. Besides the domain model, four other models have to be specified. GMF provides three complex metamodels for describing the graphical representation of model elements, the definition of toolings like the palette, toolbar, and menus, and a metamodel for mapping domain elements to their graphical representations. To facilitate reuse, all models besides the mapping model are independent of each other [Gro09].

The *graphical definition model* describes a set of shapes and styles of nodes and connections. Shapes are composed of basic geometric primitives like rectangles, ellipses, or lines, custom scalable vector graphics (SVG) markup, and dynamic and static labels. Shapes can be nested inside each other to express composite shapes. Styles are defined as well, providing the possibility to adjust e.g. colors or border styles. The desired tooling is expressed in the *tooling definition*. In this model, a palette for element creation via drag and drop, a toolbar and additional menus are specified [Gro09][Ecl19a][Ecl19b].

The *mapping model* maps elements from the domain model to both the graphical model and tooling definition. Domain elements are assigned to a graphical representation, e.g. mapping from a domain class to a rectangle with a dynamic label inside of it displaying a certain attribute of the class. A root model element, which then serves as the canvas of the editable model, has to be chosen. With the help of a *generator model*, basically identical with generator models in EMF, metadata for the models and tooling is described. With the help of

[1] `https://www.eclipse.org/modeling/emf/`, 1.12.2020
[2] `https://www.eclipse.org/gef/`, 2.12.2020

these two models, which connect the previously defined domain-, graphical and tooling models, the editor plugins can be generated by GMF. The editor plugins are plain Java code, which then can be further customized on a lower level. GMF also provides form-based wizards for a less cumbersome specification of instances of all models and their interconnection [Gro09].

As a result of the distribution of definitions of different parts of the editor to different models, several can be reused. Nevertheless, this introduces complexity due to having to maintain all of them. Changes in one of them may lead to the invalidation of others. Furthermore, user changes to the automatically generated code may not be reflected in re-generation due to the complex nature of the models and their interweaving at compile-time.

2.2 Graphiti

Graphiti, like GMF, produces generated editors as plugins to Eclipse, too. It is based on GEF as well and serves as an abstraction layer to hide its complexity. In Contrast to GMF, Graphiti is purely code-centric, therefore changes to the editor specification don't have to be transformed in a model to text transformation but are executed directly in the runtime environment [BGK$^+$].

The domain model is encoded in generated EMF code based on an Ecore metamodel. A programmatically specified *pictogram model* contains information about the graphical representation of a diagram. Shapes for nodes and connections are specified in Java with the help of said abstraction layer on top of GEF, which is tailored to fit graphical modeling use cases. Shapes are composed of basic geometric building blocks and labels. Complex figures can be specified with polylines or as shapes containing other shapes. The *link model*, which is programmatically specified, too, serves as the connection between model elements and their graphical representation. Interfaces provide *link* methods for connecting domain elements to graphical elements. Behaviour is specified with *feature* and *tool* providers. With their help, features like the creation, deletion, or editing of linked elements are implemented [Fou20][BGK$^+$].

A provided *interaction component* intercepts user interaction and forwards them to the glue of the user-specified programmatical models, the *diagram type agent*. The diagram type agent is the component responsible for modifying the model data and providing references to defined tools and features [Wen20][BGK$^+$].

2.3 Sirius

Sirius is a generator for graphical editors for domain-specific modeling. Editors run as a plugin in the environment of an Eclipse instance. Graphical models can be generated, visualized, and edited. As well as both previously described generators, metamodels describing the structure of models are specified in Ecore. Sirius encapsulates EMF, GEF, and GMF, building another abstraction layer on top of them. Editors are declared with a specification file, which is interpreted at runtime. Changes to the specification file are reflected directly, no regeneration is needed. The specification defines a so-called viewpoint specification model,

which describes the structure, appearance, and behavior of the generated editor. All aspects are declared in a single file [VMP14]. Viewpoints describe graphical representations of a model. Sirius supports diagram-, table-, and tree representations. In this work, the diagram representation is discussed. For the specification of a diagram representation, a root model element has to be chosen. Models can be then created for this type [VMP14][Fou19c].

Nodes in models generated by the editor represent domain classes. Displayed elements can be filtered with the Acceleo Query Language (AQL)[3]. Nodes can be styled with either predefined or user-specified SVG shapes. Dynamic labels for model attributes can be specified and customized in terms of size, position, and color. Nodes can act as containers for other nodes. On the one hand, edges can be relation-based, where a relationship from the domain model is mapped directly to a relationship between graphical model elements. On the other, they can be element-based, where a domain class represents them with relations to a source class and a target class. Tools for e.g. node creation or edition can be specified easily as well. Sirius provides a form- and tree-based editor for editing viewpoint specification-files [VMP14][Fou19c].

2.4 Sirius Web

Sirius Web is a recently presented novel way of executing viewpoint specification models in a web application. Previously defined viewpoint specifications can be used again without the need to adjust or adapt them. They are ported in a one-to-one translation in terms of features to browser environments. Sirius Web is based on a client-server-architecture with a backend running on Spring, Reactor, and EMF-Json and a frontend implemented in React. Besides the modeling capabilities, Sirius web provides a project manager and persistency support. A Sirius Web Application is configured via Spring Beans and, for basic functionality, just need paths to the domain model implementation by EMF and the viewpoint specification file [BB20]. Unfortunately, as of now, neither in-depth documentation nor presentations are available.

2.5 Cinco

Cinco is a tooling suite for creating domain-specific modeling tools. It generates Eclipse plugins based on Graphiti. The basic idea of Cinco is to let users define tools that are based on graph-based modeling languages, not on any Ecore model. It trades universality for an easy specification of model structures and functionality. Cinco provides three domain-specific languages for defining these graph-based modeling languages [Lyb19].

The graph language itself is defined in MGL (Meta Graph Language). Models in this language serve as the actual metamodel of the domain but are based on nodes and edges. Types of nodes and edges as well as enumerations with attributes each are specified. Nodes can serve as containers for other nodes.

[3] `https://www.eclipse.org/acceleo/documentation/`, 1.1.2021

User-specified custom data types can be specified as well, but they will not be displayed graphically [Lyb19][NLKS18].

Having modeled the abstract syntax of a language, the concrete syntax is modeled in MSL (Meta Style Language). A model in this language defines the shapes of the nodes and edges. Node styles employ a set of built-in basic shapes. Every node has at least one container shape and can include other shapes like labels or other containers. Labels can be defined with variable parts, where model data can be bound to. There are constraints for graphical layouting like how containers relate to each other and how they are arranged. Edge styles like color or shape are modeled according to predefined values. They specify labels and their positioning, too [Lyb19].

The mapping between domain nodes and edges and styles is conducted via annotations in the meta graph language model. Nodes are annotated with node styles, whereas edges are annotated with edge styles. Mapping from attributes to labels is facilitated via variables in annotations. For further processing, a model conforming to the Cinco Product Definition language needs to be expressed. It collects the other model definitions and defines project metadata like name, version, and branding. Based on these specifications, Eclipse plugins containing an implementation for Graphiti editors are automatically generated. Concerning the behavior, for each model element, basic features like addition and deletion are offered in the palette of an editor. Out of the box, resizing, layouting and reconnection features are implemented [Lyb19][NLKS18].

2.6 Pyro

Pyro is a generator for web-based modeling tools based on Cinco. Cinco generates a modeling tool based on Eclipse, whereas Pyro ports the Cinco ecosystem to the web. Pyro implementations are based on the *DyWa* (dynamic web application) framework[4], a web application framework for working on a user-defined metadata scheme implemented in Java. DyWa manages the creation and persistency management of instances for the defined types. Cinco's MGL model instances are mapped to DyWa representations, whereas styling information from MSL is directly mapped to SVG markup and attributes. These SVG elements are used as templates for JointJS[5], a web-based diagramming tool [ZNS19][Zwe15].

2.7 Eugenia

As already noted in Section 2.1, describing editors in GMF may be cumbersome due to the underlying complexity of the metamodels and erroneous re-generation. *Eugenia* is an abstraction layer on top of GMF. Its main approach is to single-source the specification of a modeling editor. While GMF scatters the specification over multiple model instances, Eugenia reduces this work to annotations in

[4] `http://ls5-www.cs.tu-dortmund.de/cms/en/projects/index.shtml`, 2.1.2021
[5] `https://www.jointjs.com/`, 2.1.2021

the metamodel, which has to be specified in EMFatic[6], a textual human-readable representation of Ecore models. Eugenia parses these annotations and performs model to model transformations to automatically generate all respective GMF models which are needed for the regular GMF workflow [KGDRP17].

While only a subset of commands from the GMF ecosystem is useable, Eugenia claims to support the most used command allowing to utilize a large fraction of GMF functionality. Tooling like node shapes, static labels, and those based on domain element features, class and reference-based links, compartments, and styling defining colors and borders are supported. Eugenia instructions are annotating the desired model elements in the metamodel directly, defining the mapping from model parts to graphical editor parts [KGDRP17].

2.8 Rocco

Rocco is a tool based on Eugenia, porting its execution domain to the web. The resulting web application is based on PsiEngine [CJD17], a framework for implementing, evaluating, and executing domain-specific languages specified in XML. It provides a graph library, diagramming, and visual tools and forms for data manipulation for user-specified languages. An Ecore model with Eugenia annotations is encoded as a DSL in a grammar readable by PsiEngine. This model to model transformation enables the execution of diagram editors based in Eugenia in web browsers [RDC+20].

2.9 MetaEdit+

MetaEdit+ is a meta computer-aided software engineering tool that provides functionality for the specification of graphical editors, graphical modeling, data management, and code generation. MetaEdit+ is executed as a standalone native application. Metamodeling is based on a graph-based meta-meta model called GOPPRR (Graph-Object-Property-Port-Relationship-Role). It allows for the specification of graphs, denoting multiple representations of underlying conceptual objects. A class of elements is called an object, which typically appear as shapes in the graphical representation and can have properties, typically appearing as labels. Objects optionally have ports, defining a specific graphical part of the object instance representation, to which a role can connect. Roles define how objects participate in relationships between objects, typically appearing as the start and the end of the graphical representation of a relationship. Relationships are connections between objects and can have properties, too. They typically appear as lines between shapes. Relationships can be of the types inheritance, association or decomposition [Poh05][KLR96].

MetaEdit+ provides graphical editors for metamodeling with a representation similar to UML as well as a symbol editor for objects, relationships, and roles. The symbol editor is a drawing tool for creating and editing graphical representation of an object, relationship, or role. Representations consist of shapes

[6] `https://www.eclipse.org/emfatic/`, 2.12.2020

and text fields. Predefined vector-based shapes exist, but user-provided SVG can be imported, too. Properties of instances of objects are data-bound to labels in the defined representation. Compounds are possible, too. The metamodel and its symbolic representation are persisted in a single file. This file is then interpreted when a new editor instance is generated according to this specification. Models can then be created, edited, and serialized in XML. Code generation is possible with user-defined templates, too [Met16][Poh05].

2.10 Atom^3

Atom^3 is a generator for customized computer-aided software engineering tools, which creates Python code for stand-alone editor implementations. In its metamodeling layer, metamodels are specified graphically conforming to an extended Entity-Relationship (ER) model consisting of types of entities with attributes of a basic data type, and relationships with attributes between them. Metamodels can be enriched with constraints defined in OCL (Object Constraint Language) or Python code. These models can be loaded by Atom^3 for modeling instances. The meta-meta model is bootstrapped in Atom^3 [LV02].

Concerning the graphical representation, Atom^3 introduces a special type of attribute for entities. Attributes can either be regular or generative. Regular attributes are of a basic data type like string or number, whereas generative attributes are used to generate new attributes at a lower meta-level. The graphical representation is specified in that way. Atom^3 provides very basic shapes like ovals or boxes to represent entities [LV02].

2.11 GME

GME (Generic Modeling Environment) is a graphical modeling tool suite for the generation of domain-specific modeling environments. Besides metamodeling and modeling, it can be used for code generation. The complex meta-meta model of GME is object-oriented and comparable with Ecore, but with a very strong focus on nesting objects in each other for composing them and the implementation of visibility control and definition of aspects, which serve as views on a given model filtering certain elements. Metamodeling in GME is conducted graphically with an editor which provides a UML-like graphical language for the metamodel of GME. Constraints can be specified in OCL [LMB$^+$01][MBL07].

The graphical representation of a metamodel or parts of it is specified through UML stereotypes, specifying very basic geometric shapes. The initial visualization is limited to a set of presentation idioms provided by GME. Based on the specification of the metamodel and its representation, a new editor can be instantiated. If a model element is defined as a type that contains other first-class objects, the graphical representation of its inner structure can be viewed and edited in a separate window, separating the inner structure from the outer. Though providing only basic graphical representation, GME offers an API for every graphical representation through the Component Object Model deferring the implementation of the visualization to further development [LMVL01][Dav03].

2.12 WebGME

WebGME is a derivation of GME, introducing changes to the metamodel and porting the execution environment to web browsers. The metamodel is enhanced with further reference and relation concepts and the inheritance system is extended to support prototypical inheritance. For visualization, WebGME provides graphical tooling for representations of domain elements. With the help of this tooling, shapes can be composed of graphical building blocks like lines and basic shapes [MKK+14][ISI21]. Unfortunately, no extensive documentation of the mapping and visualization mechanism is available.

2.13 GLSP

GLSP, the Graphical Language Server Protocol, is an EMF-based diagram editing tool, which is focused on being embedded in web integrated development environments (IDE). It lets the user execute graphical modeling tools directly in the browser. The basic concept is the application of the architectural pattern of the Language Server Protocol (LSP)[7] to graphical modeling. LSP is a standardized protocol for interaction between language servers and IDEs. Language servers provide features for programming language support like auto-completion, refactoring support, or type hints. The graphical editor communicates with the language server, which has knowledge about the used modeling language and can provide information about allowed diagram manipulations [LF19].

GLSP as a whole employs a client-server-architecture. Diagram rules and EMF integration for persistency run on the server. The existing model infrastructure of EMF with its generated code can be reused. Editing rules, commands, transactions, and validation are computed on the server as well. The client focuses only on rendering the diagrams and client interaction. The LSP4J[8] implementation is used for the LSP component. Sprotty[9], an open-source diagramming framework, is responsible for diagram rendering and interaction [LF19].

The web server component, which knows the underlying metamodel, specifies which nodes and edges are allowed to be created and which operations the palette offers. Operations for all model manipulation like the creation or deletion of nodes and how they are linked to the EMF model specified by the developer via an implementation of interfaces, which offers generic methods for the manipulation and persistency of nodes and models. The palette and tooling are specified in Java, too. With this implementation, the server knows all legal operations which can be performed on a model and pass this knowledge to the frontend via GLSP. The graphical representation of the model elements is declared in the frontend code. There, templates in SVG are declared in concrete implementations of provided generic handlers. The templates can contain dynamic parts, where data from model elements is bound to slots [LF19].

[7] `https://langserver.org/`, 1.1.2021
[8] `https://github.com/eclipse/lsp4j`, 2.1.2021
[9] `https://github.com/eclipse/sprotty`, 2.1.2021

2.14 GEMS

GEMS, the Generic Eclipse Modeling System, is a project in the Eclipse ecosystem for the automatic generation of the implementation of graphical modeling tools from a metamodel specified in Ecore. Out of this specification, GEMS generates the necessary EMF, GEF, and Eclipse plugin implementations. It claims to be simpler in use and more easily usable by domain experts due to its API-agnostic design. The GEMS metamodel consists of classes with attributes and basic data types. Relations can be either inheritance, a part-of-relation specifying a part-whole relation, and a connection type which is conceptually an association class with source and target objects. One class in the domain model has to be marked with a flag that indicated that it is the root element of the model, serving as the canvas of the generated editor [WSM06][WSNW07].

Classes and their inner structure are mapped to nodes in the diagramming environment. Parent-child-relations are modeled by the inclusion of children elements in expandable compartments in the shape which represents the parent. Connection types are mapped to lines between classes. The appearance of nodes and lines can be customized with a GEMS stylesheet with a language that resembles Cascading Style Sheets. The layout of nodes can be customized, e.g. specifying the layouting of its content in a vertical or horizontal layout. Properties of classes can be data-bound to dynamic labels. Connection types can be changed in their color, style (e.g. dashed), width, and predefined symbols for their source and target ends [Fou19b][Fou19a].

2.15 Spray

Spray is another model-driven approach for the generation of graphical editors. It is an abstraction layer on top of Graphiti. The metamodel is defined in Ecore. Spray offers three textual modeling languages specified in Xtext. Models expressed in these languages define, how the editor for models conforming to the metamodel appears and behaves. Based on the models and the metamodel, Java code implementing a Graphiti Editor as well as Eclipse plugins with EMF code, responsible for serialization, are automatically generated [GB16].

The spray core language defines the mapping from metamodel elements to graphical representations in the editor and the editor behavior. One root element, which serves as the canvas and is further modeled, has to be specified in the model. For every domain class, several aspects can be defined. Attributes can be marked with a *ask for*-flag. These attributes can be edited with a popup dialog on interaction. Classes can be defined as containers, which indicates, that other domain elements will be modeled in the graphical representation of instances of this class. By default, a rectangle is used, where contained elements are contained in this rectangle. If a class or relationship should appear in the palette, flags for this can be set. Furthermore, icons for the palette for element creation can be specified. Relationships can be customized with static and dynamic labels at the start, middle, and end of the graphical representation [GB16].

The other two languages define the shapes and styles. With the shapes language, complex shapes can be constructed from geometric primitives like rectangles and ellipses, which can serve as containers. The position, resizing, and nesting policy can be configured, as well as anchors for connections. Anchors can have fixed positions or positions relative to the containing shape. The style language deals with colors and font style [GB16].

2.16 JetBrains MPS

JetBrains MPS, the Meta Programming System, is a language workbench comparable to EMF and Xtext. Metamodels consist of concepts with properties of basic data types, children, and references, which both have the type of a concept. References denote links to other nodes and are not bidirectional, whereas children conceptualize existentially dependent part-whole relationships [Jet20b].

Besides the functionality to define metamodels and domain-specific languages, MPS provides rudimentary diagramming features. In this representation, diagrams typically consist of blocks, which represent nodes, and connectors, visualized by lines, which represent references. The diagram is specified in a textual language and saved in a diagram definition file. In there, one concept has to be specified as the root of the model and the canvas of the diagram. The palette for object creation just represents blocks, connections have to be drawn via drag and drop from one block to another. Shapes can be chosen out of a predefined set of shapes or specified programmatically in Java through implementing an interface provided by MPS [Jet20a].

In contrast to the other solutions, the provided diagramming system of MPS is rather basic. There exist another extension[10] for supporting diagrams in MPS, but unfortunately, there is no comprehensive documentation.

2.17 Tiger

Tiger is a proposal for a modeling mechanism and editor generation. Tiger stands for *transformation-based generation of modeling environments*. It sees itself as a general approach for defining visual languages and for the generation of tool environments for them. Tiger is based on metamodels, graph grammars, and layout definitions. These specifications are input for a generator for Eclipse plugins, which are based in GEF and manage the model persistency in EMF [EEHT05].

The abstract syntax of a visual language that is to be expressed and made editable with Tiger is specified with a graph. The user specifies the alphabet visually in a tree view, a rule set, and a start graph. The alphabet consists of symbol types and link types. Symbols may be enriched by attributes. The rule set describes all allowed transformations on graphs. Rules can be of the kinds create, delete, edit and move. Rules specify a pair of (sub)graphs, where a so-called left-hand side is replaced by a right-hand side and optionally conditions,

[10] `https://jetbrains.github.io/MPS-extensions/extensions/diagrams/`, 3.12.2020

under which the graph may be edited. All legal operations are specified in this way. The specified start graph is the starting point for all graph transformations based on the defined rules. Rules of the kind *create* result in symbol icons in the palette for element creation, whereas the application of rules for deletion and edition trigger dialogs for editing. The graphical layout is given by mapping all alphabet elements to shape figures like rectangles, circles, ellipses, or polygons, which can be customized in terms of color and border layout and layouting constraints. The connection between shapes can be customized in terms of color, width, style, and a marker at the end of a connection [EEHT05].

2.18 Pounamu

Pounamu is a tooling suite for the specification and generation of editors that provide multiple views on models. Metamodels are specified conforming to the ER model. Pounamu provides visual tools for modeling in ER, the definition of visual representation of model elements, and for views on the graphical representation of models and their manipulation. A programmatically extendable event handling system for editing behavior, model constraints, and user-defined events enables the event-driven extension and customization [ZGH04].

The shape designer allows for the definition of graphical representations independently of the domain model to facilitate reuse. Shapes can be basic geometric primitives like rectangles or ellipses. They can have embedded sub shapes such as labels, single- or multi-line editable text fields, or images. Labels that are to be exposed for mapping to a metamodel are specified using a property sheet tab. The connector designer lets the user design inter-shape connectors. The line format and styling, shapes at the start and the end of the line, and labels can be specified. Lastly, the view designer connects the metamodel and shapes and connections. Due to the simplicity of the ER model, entities can just be matched with shapes and their exposed labels and connections just with defined lines and their respective labels. In the resulting editor, entity instances can be defined. Models are persisted in XML [ZGH04].

2.19 Marama

Marama is an Eclipse plugin-based reimplementation of Pounamu, porting it into the Eclipse ecosystem. The graphical visualization and specification tools are reimplemented using GEF. Constraints can be specified directly in OCL, not via event handlers in Pounamu. The visual editors for shapes and connectors are reworked entirely and enhanced with mechanisms for alignment and improved usability. Furthermore, a mapping specification tool for model to model and model to text transformations called MaramaTorua is offered [GHHL08].

2.20 Microsoft DSL Tools

Microsoft DSL Tools is a plugin for Visual Studio, which again can generate plugins for Visual Studio. The plugins implement graphical editors based on

metamodels specified via DSL Tools. Several tools support model-driven development with the ability to generate code, commands for transformations on models, and the ability to interact with code [Par20].

For generating graphical editors, the workflow is as follows. At first, the abstract syntax of a visual language is defined with metamodels. After or during that, graphical representations and mappings from the domain model to the representations are specified. Microsoft DSL Tools offers a graphical metamodeling environment providing a basic meta-meta model. Domain models consist of classes and binary relationships. Classes have properties of basic data types. Both ends of the relationship are modeled as roles, which themselves have names. Relationships are defined as either references or containment, comparable with associations and compositions in UML. Containment is usually not displayed via a graphical connection, but with containment. Inheritance is possible, too. One class in the metamodel has to be marked as the root class. Every class except the root class then must be the target of at least one containment relationship or must inherit from one, that is contained in another [Par20].

After the definition of the metamodel or during that, graphical elements are defined. This definition is directly integrated into the graphical editor for domain definition, laying focus on the graphical approach. Shapes for domain classes can be rectangles, ellipses, images specified via a path, or compartments. Compartments are displayed as rectangles that contain one or more list of items. Text decorators for properties can be added, positioned freely, and later be databound. Connectors can be enriched with text decorators to display labels at a specified position. Ports can be added to the boundary of other shapes, which will then serve as an anchor point for connections. Ports appear as rectangles at the border of other shapes and can slide along the borders of their parents [Par20].

The mapping from domain elements to their graphical representation occurs in the same view in the plugin as the definition of both. When a graphical element is selected, the possible domain elements which are eligible for mapping are suggested and selectable. Domain classes can be mapped to shape classes. Their properties can be mapped to the defined dynamic labels. Domain references can be mapped to connectors. Classes containing instances of other classes can be mapped to compartment shapes. If one of the contained classes is mapped to a subcompartment of its parent, a property of it has to be selected to be displayed in the list view in the subcompartment. References can be mapped to subcompartments, too. Domain classes and relationships can be selected for appearing in the tool palette [Par20].

2.21 MetaBup

MetaBup presents an example-based approach for generating graphical modeling tools and proposes a novel technique for this generation based on Sirius. Instead of building meta models first and then describing its concrete syntax, MetaBup aids domain specialists in building examples with graphical general-purpose modeling tools like Microsoft Powerpoint, Dia, or yED. Sketches drawn with one of those applications need to have a legend for the used symbols. They

are parsed into an intermediate textual representation. A central assumption is, that every representation of a domain class has variable slots and references to other domain objects. Heuristics then try to derive the metamodel from these fragments, e.g. by assigning cardinalities based on the maximal amount of observed connections, by analyzing the spatial relationships between objects, where overlapping or containment implies composition in the metamodel. Alignments or adjacency to a specific side of a node are extracted, too, as well as styles of edges (color, line width, source, and target decorations). When containment, adjacency, or overlapping is detected between two domain elements, a reference between the two corresponding classes is added to the intermediate representation. Several iterations and refinements. The graphical representation is inferred from the intermediate representation by an automatic mapping to a metamodel representing graphical modeler parts. This instance is then transformed via a model to model transformation to a Sirius viewpoint specification file for the resulting editor, which then generates an Eclipse plugin [LFGGL16].

2.22 Related Tools and Approaches

There are several other related tools or approaches, which shall be named in this work to give an extended and complete overview:

- Clooca [HHFN13] describes an approach of generating a web application for modeling instances of metamodels specified in Ecore. Unfortunately, no comprehensive documentation or other work can be found for this approach. The source code[11] can't be built as well.
- [AGK09] presents a deep modeling approach (Melanee) with an integrated graphical representation, but does not go into detail about how exactly the representations are specified or customizable.
- [TB18] provides a way of visualizing models described in Xtext, but without editing functionality.
- [CM03] describes a grammatical approach for diagram parsing and error correction according to graph grammars. This approach could potentially be extended for the purpose of generating modeling editors.

3 Results and trends

For a more accessible analysis, the landscape of analyzed tools is visualized (Figure 1 in the Appendix) and sorted by the year of the first release or description according to the cited sources. Their execution domain as well as tools influencing other tools are displayed, too. Influence is defined as either a tool being the base for a reimplementation or being used by another tool to generate the graphical editor or parts of it.

Two main findings can be derived from the analysis and this visualization. The first one concerns the execution environment of graphical modeler generators. Pounamu was the last tool that was released as a standalone application.

[11] `https://github.com/clooca/clooca`, 29.11.2020

Since then, all tools are either plugins to IDEs or web applications. Especially noticeable is the Eclipse ecosystem with a majority of tools being implemented as plugins to this IDE. Since 2014, the majority of tools is executed in web environments. All of these web applications rely on backend components and employ a client-server structure. Another finding is that there are several tools (Spray, Eugenia, Cinco, Sirius), which build abstraction layers on top of other tools. A thesis that can be stated is, that there's a trend for trading off complexity for simplicity. GMF serves as a base library for multiple other tools. As already stated, the metamodels of GMF are rather complex and the re-generation of code can be error-prone. Abstraction layers build a way of using the provided functionality more easily but entail potentially having to refrain from the whole expressiveness of the methodological base.

4 Conclusion

This work posed two research questions. The first, asking about the landscape of approaches and tools for (semi-)automatically generating graphical modeling editors was answered in Section 2. 21 approaches and tools were presented, laying focus on product generation, specification of graphical representations, and the mapping from metamodels or other formalisms to graphically modeled elements. Based on this description, the second research question, which asked about trends or developments in this universe, could be answered. Considering the execution domain, there's a clear trend towards deploying graphical editors in plugin structures, mainly in the Eclipse ecosystem as well as deploying them as web applications. Furthermore, several approaches build abstraction layers on top of others to ease the usage, partly with the caveat of reducing the expressiveness of the abstracted approaches or toolings.

References

AGK09. C. Atkinson, M. Gutheil, and B. Kennel. A Flexible Infrastructure for Multilevel Language Engineering. *IEEE Transactions on Software Engineering*, 35(6):742–755, 2009.

BB20. Mélanie Bats and Stéphane Bégaudeau. Sirius Web: 100% open source cloud modeling platform | EclipseCon 2020, October 2020. [Online; accessed 13. Dec. 2020, `https://www.eclipsecon.org/sites/default/files/slides/EclipseCon2020-SiriusWeb.pdf`].

BGK+. Christian Brand, Matthias Gorning, Tim Kaiser, Jürgen Pasch, and Michael Wenz. Graphiti - Development of High-Quality Graphical Model Editors. [Online; accessed 15. Dec. 2020, `https://www.eclipse.org/graphiti/documentation/files/EclipseMagazineGraphiti.pdf`].

CJD17. Enrique Chavarriaga, Francisco Jurado, and Fernando Díez. An approach to build XML-based domain specific languages solutions for client-side web applications. *Computer Languages, Systems and Structures*, 49:133–151, 2017.

CM03. Sitt Sen Chok and Kim Marriott. Automatic Generation of Intelligent Diagram Editors. *ACM Trans. Comput.-Hum. Interact.*, 10(3):244–276, September 2003.

Dav03. James Davis. GME: The Generic Modeling Environment. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, page 82–83, New York, NY, USA, 2003. Association for Computing Machinery.

Ecl19a. Eclipse Foundation. Graphical Modeling Framework/Tutorial/Part 1 - Eclipsepedia, Nov 2019. [Online; accessed 7. Dec. 2020, `https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1`].

Ecl19b. Eclipse Foundation. Graphical Modeling Framework/Tutorial/Part 2 - Eclipsepedia, Nov 2019. [Online; accessed 7. Dec. 2020, `https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_2`].

EEHT05. Karsten Ehrig, Claudia Ermel, Stefan Hänsgen, and Gabriele Taentzer. Generation of Visual Editors as Eclipse Plug-Ins. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, page 134–143, New York, NY, USA, 2005. Association for Computing Machinery.

Fou19a. Eclipse Foundation. GEMS Metamodeling Tutorial - Eclipsepedia, Nov 2019. [Online; accessed 11. Dec. 2020, `https://wiki.eclipse.org/GEMS_Metamodeling_Tutorial`].

Fou19b. Eclipse Foundation. GEMS Stylesheet Tutorial - Eclipsepedia, Nov 2019. [Online; accessed 11. Dec. 2020, `https://wiki.eclipse.org/GEMS_Stylesheet_Tutorial`].

Fou19c. Eclipse Foundation. Sirius/Tutorials/StarterTutorial - Eclipsepedia, Nov 2019. [Online; accessed 12. Dec. 2021, `https://wiki.eclipse.org/Sirius/Tutorials/StarterTutorial`].

Fou20. Eclipse Foundation. Eclipse Documentation - Current Release, December 2020. [Online; accessed 6. Dec. 2021, `https://help.eclipse.org/2020-12/index.jsp?nav=%2F51`].

GB16. Markus Gerhart and Marko Boger. Concepts for the model-driven generation of graphical editors in Eclipse by using the Graphiti framework. *International Journal of Computer Techniques*, 3(4):11 – 20, July 2016.

GHHL08. John Grundy, John Hosking, Jun Huh, and Karen Li. Marama: An eclipse meta-toolset for generating multi-view environments. In *30th International Conference on Software Engineering (ICSE 2008)*, pages 819–822, 01 2008.

Gro09. Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley, Upper Saddle River, N.J. ; Boston, Mass., USA, 2009.

HHFN13. Shuhei Hiya, K. Hisazumi, A. Fukuda, and Tsuneo Nakanishi. clooca : Web based tool for Domain Specific Modeling. In *Demos/Posters/StudentResearch@MoDELS*, 2013.

ISI21. ISIS/Vanderbilt University. WebGME Documentation - Release 1.0.0, January 2021. [Online; accessed 7 Feb. 2020, `https://webgme.readthedocs.io/_/downloads/en/latest/pdf/`].

Jet20a. JetBrains. MPS Help System - Diagramming Editor, December 2020. [Online; accessed 2. Dec. 2021, `https://www.jetbrains.com/help/mps/diagramming-editor.html`].

Jet20b. JetBrains. MPS Help System - Structure, December 2020. [Online; accessed 2. Dec. 2021, https://www.jetbrains.com/help/mps/structure.html].

KGDRP17. Dimitrios S. Kolovos, Antonio García-Domínguez, Louis M. Rose, and Richard F. Paige. Eugenia: towards disciplined and automated development of GMF-based graphical model editors. *Software and Systems Modeling*, 16(1):229–255, February 2017. The final publication is available at Springer via http://dx.doi.org/10.1007/s10270-015-0455-3.

KLR96. Steven Kelly, Kalle Lyytinen, and Matti Rossi. MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In Panos Constantopoulos, John Mylopoulos, and Yannis Vassiliou, editors, *CAiSE*, volume 1080 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 1996.

LF19. Philip Langer and Martin Fleck. Diagrams in web and space with Eclipse GLSP(Graphical Language Server Platform), October 2019. [Online; accessed 5. Dec. 2020, https://www.eclipsecon.org/sites/default/files/slides/Diagrams%20in%20web%20and%20space%20with%20GLSP%20%28Shared%29.pdf].

LFGGL16. Jesús López Fernández, Antonio Garmendia, Esther Guerra, and Juan Lara. Example-Based Generation of Graphical Modelling Environments. In *European Conference on Modelling Foundations and Applications*, volume 9764, pages 101–117, July 2016.

LMB+01. Akos Ledeczi, M Maroti, A Bakay, Gabor Karsai, J Garrett, C Thomason, G Nordstrom, J Sprinkle, and Péter Völgyesi. The Generic Modeling Environment. *Workshop on Intelligent Signal Processing, Budapest, Hungary*, January 2001.

LMVL01. Á. Lédeczi, M. Maróti, P. Völgyesi, and A. Ledeczi. The Generic Modeling Environment - Technical Report. 2001. https://www.cse.msu.edu/~chengb/CSE891/Techniques/GME/GMEReport.pdf.

LV02. Juan de Lara and Hans Vangheluwe. AToM3: A Tool for Multi-formalism and Meta-modelling. In Ralf-Detlef Kutsche and Herbert Weber, editors, *Fundamental Approaches to Software Engineering*, pages 174–188, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

Lyb19. Michael Lybecait. *Meta-Model Based Generation of Domain-Specific Modeling Tools*. PhD thesis, TU Dortmund, 2019.

MBL07. Zoltán Molnár, Daniel Balasubramanian, and Akos Ledeczi. An Introduction to the Generic Modeling Environment. 01 2007.

Met16. MetaCase. MetaEdit+ Workbench User's Guide, May 2016. [Online; accessed 2. Dec. 2020, https://www.metacase.com/support/45/manuals/mwb/Mw.html].

MKK+14. M. Maróti, Tamas Kecskes, R. Kereskényi, Brian Broll, Péter Völgyesi, L. Jurácz, T. Levendoszky, and Akos Ledeczi. Next generation (Meta)modeling: Web- and cloud-based collaborative tool infrastructure. *CEUR Workshop Proceedings*, 1237:41–60, 01 2014.

NLKS18. Stefan Naujokat, Michael Lybecait, Dawid Kopetzki, and Bernhard Steffen. CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools. *International Journal on Software Tools for Technology Transfer (STTT)*, 20(3):327–354, June 2018.

Par20. Joshua Partlow. Modeling SDK for Visual Studio - Domain-Specific Languages, Dec 2020. [Online; accessed 12. Dec. 2020, https:

//docs.microsoft.com/en-us/visualstudio/modeling/modeling-sdk-for-visual-studio-domain-specific-languages?view=vs-2019].

Poh05. Risto Pohjonen. Metamodeling Made Easy – MetaEdit+ (Tool Demonstration). In Robert Glück and Michael Lowry, editors, *Generative Programming and Component Engineering*, pages 442–446, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

RDC+20. Fatima Rani, Pablo Diez, Enrique Chavarriaga, Esther Guerra, and Juan de Lara. Automated Migration of EuGENia Graphical Editors to the Web. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery.

TB18. Marcel Toussaint and Thomas Baar. Enriching Textual Xtext-DSLs with a Graphical GEF-Based Editor. In Alexander K. Petrenko and Andrei Voronkov, editors, *Perspectives of System Informatics*, pages 394–401, Cham, 2018. Springer International Publishing.

VMP14. V. Viyović, M. Maksimović, and B. Perisić. Sirius: A rapid development of DSM graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, pages 233–238, 2014.

Wen20. Michael Wenz. Graphiti - Overview | The Eclipse Foundation, December 2020. [Online; accessed 11. Dec. 2020, https://www.eclipse.org/graphiti/documentation/overview.php].

WSM06. J. White, D. Schmidt, and Seán Mulligan. The Generic Eclipse Modeling System. 2006.

WSNW07. Jules White, Douglas C. Schmidt, Andrey Nechypurenko, and Egon Wuchner. Introduction to the Generic Eclipse Modeling System. *Eclipse Magazine*, 6:11–19, January 2007.

ZGH04. Nianping Zhu, John Grundy, and John Hosking. Pounamu: A Meta-Yool for Multi-View Visual Language Environment Construction. pages 254 – 256, 10 2004.

ZNS19. Philip Zweihoff, Stefan Naujokat, and Bernhard Steffen. Pyro: Generating Domain-Specific Collaborative Online Modeling Environments. In Reiner Hähnle and Wil van der Aalst, editors, *Fundamental Approaches to Software Engineering*, pages 101–115, Cham, 2019. Springer International Publishing.

Zwe15. Philip Zweihoff. Cinco Products for the web. Master's thesis, TU Dortmund, 2015.

Appendix - Visualization of the Timeline and Inter-Dependencies Between Analyzed Approaches

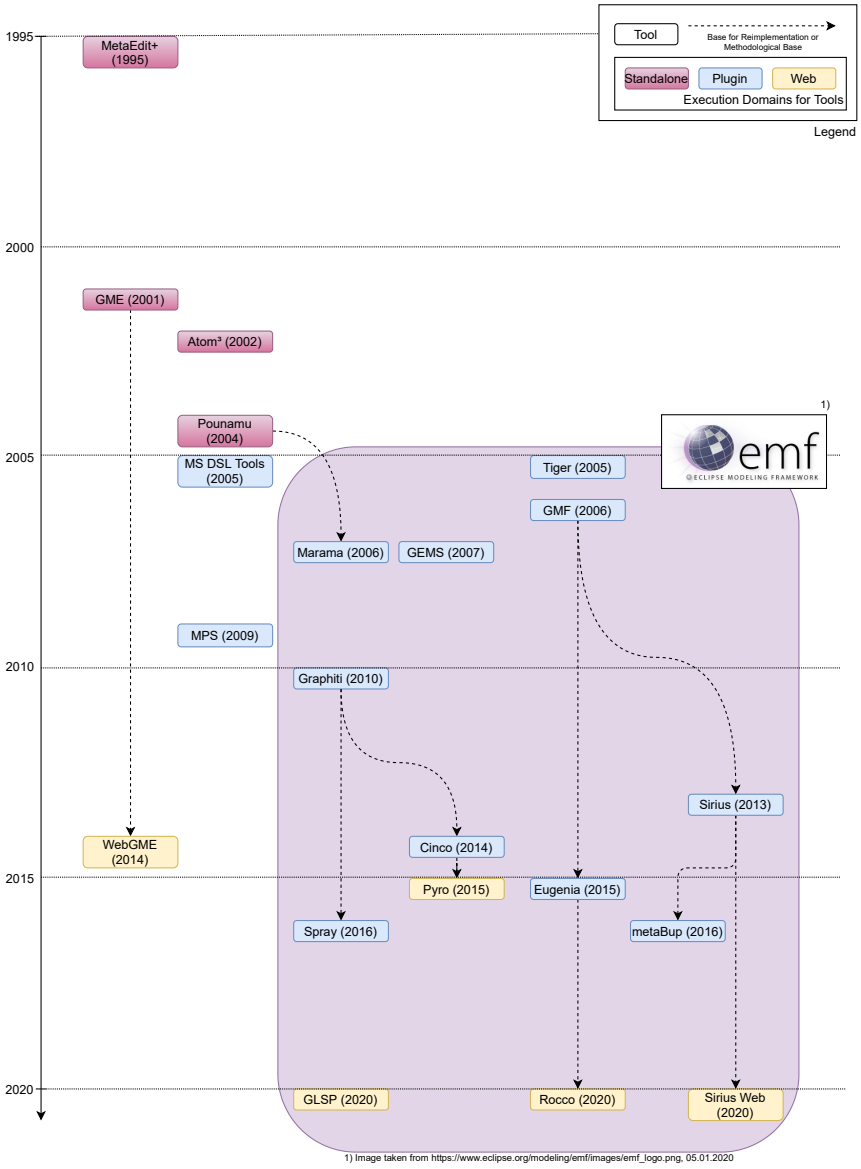

Fig. 1. Overview of the landscape of generators for graphical modelers