Scientific Work: Creating a web-based model transformation UI (with GLSP)

Florian Weidner
Philipps-University Marburg, Germany
Department of Mathematics and Computer Science, Software engineering group
May 06, 2025

I. Motivation and introduction

In software engineering, often Model-Driven Engineering (MDE) is used to increase development productivity and quality. Concepts are modeled closer to the domain, so that they describe important aspects of a solution with human-friendly abstractions. The models can also be used to generate application fragments, that can be directly used as source code. In the process of MDE, many activities need to transform source models into different target models, while following a set of transformation rules. This model transformation process is based on algebraic graph transformations. A metamodel is used to model the structure and rules of the concept. The resulting transformation language can provide automatic model creation, development and maintenance activities. [9] One framework to use MDE is Eclipse Modeling Framework (EMF) by the Eclipse Foundation. It provides a basis for application development, using modeling and code generation facilities. Much frameworks build upon EMF, providing various MDE tools like code generators, graphical diagramming, model transformation or model validation. [10] One model transformation framework is Henshin. [12] It tries to provides model transformation capabilites with a high level of usability. [11] For metamodels it uses EMF Ecore files and for instance models EMF XMI files. The framework enables transformations on XMI instance files with a defined transformation language. It provides a graphical and textual syntax to create these transformation rules. [12] Henshin can be used as a eclipse plugin. Eclipse makes it easy to access, but especially for new users, the heavy editor makes the use of Henshin unintuitive.

Therefore the goal exists to create a graphical option to use the Henshin model transformations without the overhead of the heavy eclipse editor. A web-based graphical editor would make the use of Henshin even more accessible and intuitive.

Graphical Language Server Platform (GLSP) is a open-source framework by the Eclipse Foundation to develop custom diagram editors for distributed web-applications. [8] It can be used in Eclipse Desktop IDE, Eclipse Theia, Visual Studio Code and embedded in any website. With these fuctionalities, GLSP fits to create an accessible, intuitive application to create and apply Henshin model transformations.

II. Background

In this section, the theoretical background of the project and used technologies are described.

A. Eclipse Foundation

The Eclipse Foundation is a not-for-profit, member-supported corporation that provides an environment for individuals and organizations for collaborative and innovative software development. [6] The Eclipse Foundation grew out of the publication of the Eclipse Integrated Development Environment (IDE) code from IBM in 2001. The Eclipse Foundation itself was founded in 2004. The new organization was founded to continue the development of Eclipse IDE as an open source platform. Over time many different projects were initiated by the organization in the Eclipse environment, all running under the Eclipse Public License. [7, 6] In the recent years, the key initatives of the Eclipse Foundation are contributing to european digital souvereignty, enhancing security measures, innovating Software-Defined Vehicle (SDV), organizing community events and improving their most popular projects. Popular projects are for example the Jakarta EE, a ecosystem for cloud native applications with java, Eclipse Temurin, providing open source Java Development Kits and the Eclipse IDE. [2] In total, the Eclipse Foundation hosts more than 400 open source projects, supported 14 european research projects in 2024 and has 117 organisations participating in commits [2]

The scope of this work remains within the Eclipse Foundation ecosystem. All frameworks used are projects from the Eclipse Foundation. The used frameworks are described in the sections II-B, II-C and II-D.

The Eclipse IDE is not the main project but it is still an important part of the Eclipse infrastructure. It is divided into four main components: Equinox, the Platform, the Java Development Tools (JDT) and the Plug-in Development Environment (PDE). Together they provide everything to develop and extend Eclipse-based tools. Equinox and the Platform are the core of the Eclipse IDE. With expanding the core with the JDT or other plugins, the IDE can be used to develop different programming languages, like Java, C/C++ or PHP. [10] Eclipse provides different packages to download, depending on the use case. One package is the Eclipse Modeling Tools package by the Eclipse Modeling Project. It provides tools and runtimes to build model-based applications. It can be used to graphically design domain models and test those models by creating and editing dynamic instances. Also Java code can be generated from the models to get a scaffold, that can be used to create application on top. [4] The base of the Eclipse Modeling Tool is EMF (section II-B). Other modeling tools and projects, that are built on top of the EMF core functionality, provide capabilites for model transformation, database integration, or graphical editor generation. [10]

B. Eclipse Modeling Framework (EMF)

> „Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for building tools and other applications based on a structured data model.“ [5]

Eclipse Modeling Framework (EMF) is the core part of the Eclipse Modeling Project and unifies the representation of models in UML, XML and Java. You can define your model in one of these formats and use EMF to generate the other formats.

EMF consists of three components. The EMF core part, provides Ecore meta models, a runtime support for the models, and a basic API for manipulating EMF objects generically. Ecore meta models are used to describe the structure of a model. [3] They can be serialized in XML Metadata Interchange (XMI) 2.0, as Ecore XMI and have the file extension *.ecore*. There are several Ecore classes to represent a model, here are the most important ones:

- **EClass**: A class in the model that is identified by a name, containing attributes and references to other classes. It can also refer to a number of other classes as its supertypes to support inheritance.[10]
- **EAttribute**: An attribute of a class, that are identified by a name and have a type.[10]
- **EDataType**: A simple data type like *EString*, *EBoolean* or *EJavaClass*[10]
- **EReference**: A reference to another class, containing a link to an instance of that class.[10]

Together Steinberg et al. called these classes the Ecore kernel. In figure 1 you can see the kernel classes and their relations. these classes are enough to define simple models. **EAttribute** and **EReference** have a lot of similarities. They both define the state of an instance of an **EClass** and have a name and a type. For that Ecore provides a common interface for both, called **EStructuralFeature**. Ecore can also model behavioral features of classes as **EOperation** using **EParameter**. Related classes are grouped into packages called **EPackage**. It is represented by the root element when the model is serialized. [10]

The second component of EMF is EMF.Edit. It provides generic reusable classes to build viewers and editors for EMF models. With these classes EMF meta models can be displayed in JFace viewers, that are part of the Eclipse UI. [3] The Eclipse IDE can display a Ecore model in a tree viewer. Eclipse acceses the data over the *ITreeContentProvider* interface to navigate the content and the*ILabelProvider* interface to provide the label text and icons for the displayed objects. The properties of objects are displayed in a Property

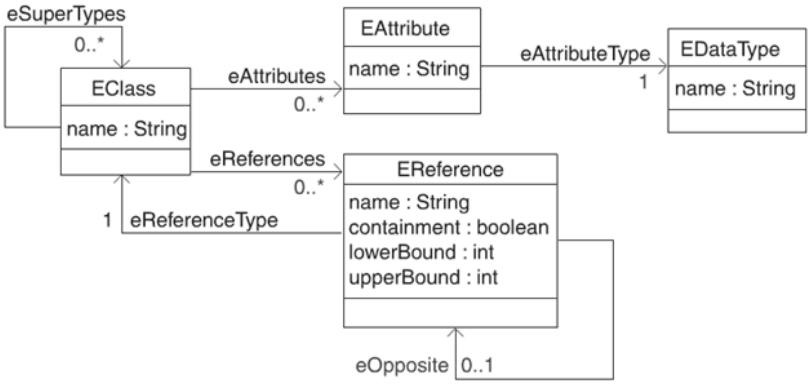

Figure 1. The Ecore kernel. Image obtained from [10]

Sheet over the *IPropertySourceProvider*, where the user can edit the model. EMF.Edit also provides undo and redo operations when creating or editing a instance model. For that it uses a command framework with commands like a *AddCommand*, *SetCommand* or *CopyCommand*. [10]

The third component is EMF.Codegen. It can generate Java code for a complete editor for EMF instance models of a Ecore meta model. It provides different generation options. So unlike EMF.Edit, that just provides generic classes for Ecore models, EMF.Codegen directly generates complete editors with a UI. [3] The generation can be done over a wizard in the Eclipse IDE or by using the command line interface. [10] The generation can be sepertated in to three levels. The first level is to generate Java interfaces and implementations for the Ecore model classes and a factory and package implementation class. The second level generates specific *ItemProviders* to edit instance models based on the meta model.The classes are structured like the EMF.Edit component for the Ecore models. The third level generates a structured editor with UI that works like the Ecore editor in the Eclipse IDE and can be a starting point for customization. [3] There are many frameworks that build on top of EMF using these generation capabilites to create further modeling functionality. One of these frameworks is Henshin, that is described in section II-C.

C. Henshin

One part of the Eclipse Modeling Project for model transformations is Henshin. It can be used as a plugin in the Eclipse IDE or as an SDK. It provides a graphical and textual syntax to define model transformation rules and apply them on EMF XMI instance models. It can be used for endogenous transformations, where EMF model instances are directly transformed, and exogenous transformations, where new instances are generated from given instances using a trace model. It also brings efficient in-place execution of transformations using a interpreter with debugging support and a performance profiler. Henshin also provides confilct and dependency analysis, and state space analysis for verification. [12]

Henshin builds on top of EMF. It uses an Ecore meta model to define the structure of the transformation rules, resulting in a serialized XMI file with the file extension *.hensin*, that can therefore be edited in the Eclipse tree editor. [12] The meta model of the transformation rules uses another Ecore meta model that models the model structure of the domain, to type the nodes, egdes and attributes of the rules. [1] In figure 2 you can see the Henshin transformation rule meta model. A rule consits of a Right-Hand Side (RHS) Graph, a Left-Hand Side (LHS) Graph and attribute conditions. Additionally mappings between the LHS

and RHS Graph are defined between nodes. The mapping of the edges is done implicitly by the mapping of the source and target nodes. [1] Henshin uses Units to control the order of rule applications. With Units, control structures can be defined. Also parameters can be passed from the previous executed rule to the next one to have a controlled object flow. Henshins tansformation language is based on algebraic graph transformations, complying with the syntactical and semantic structure of rules and transformation units. This ensures a language usable for formal verification or validation. [1]

In the Eclipse IDE, rules can also be edited in a graphical editor. The rules are displayed as a single graph, calculated from the LHS and RHS Graphs. The nodes and edges are annotated with *<<preserve>>*, *<<create>>*, *<<delete>>*, *<<forbid>>* or *<<require>>* to indicate what happens to the nodes and edges when applying the rule. These annotations can be directly edited in the graphical editor and the the LHS and RHS Graphs are then adapted to the change. Also multible Negative Application Conditions (NACs), Positive Application Conditions (PACs) and Paramters can be specified directly. [12] When a set of transformation rules are specified, they can be applied to an EMF XMI instance model, by using a wizard in the Eclipse IDE. There the source model, the rule and its parameters can be selected. The result of the transformation can be seen in a new XMI instance file. [12] Next to the graphical editor, Henshin also provides a textual syntax to define transformation rules and units. In a *.henshin_rule* file with the keyword **rule** a name and paramters, a new rule can be described. If you want to define a node, you can use the keyword **node** with a its action keyword like **create** or **preserve** to specify the action of the node in the transformation.

The Henshin SDK consists of multible packages oriented to the package structure of EMF. Next to a model, edit and editor package, it provides an interpreter package, that contains a default engine to execute model transformations.

D. Graphical Language Server Platform (GLSP)

GLSP is a framework that provides components for the development of GUIs for web-based diagram editors. [8]

III. Related Work/Software

Similar already existing tools for model checking: Henshin, Groovy...

A. Implementation

IV. Conclusion

References

[1] Thorsten Arendt et al. “Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations”. In: *Model Driven Engineering Languages and Systems*. Ed. by Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 121–135. ISBN: 978-3-642-16145-2.

[2] Eclipse Foundation. *2024 Annual Community Report*. en. https://www.eclipse.org/org/foundation/reports/annual_report.php. Accessed: 2025-6-2. 2024.

[3] Eclipse Foundation. *Eclipse Modeling Framework*. Accessed: 2025-06-02. 2025. URL: https://eclipse.dev/emf/.

[4] Eclipse Foundation. *Eclipse Modeling Project*. Accessed: 2025-06-02. 2025. URL: https://eclipse.dev/modeling/.

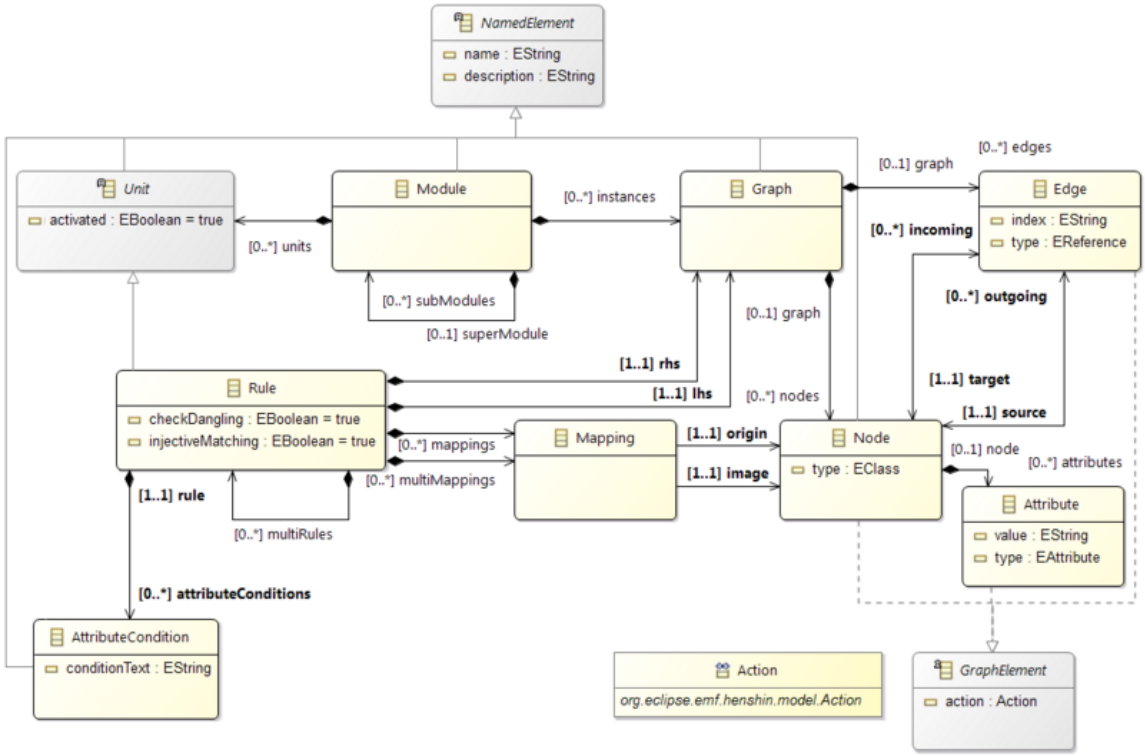

Figure 2. Henshin transformation rule meta model. Image obtained from [12]

[5] Eclipse Foundation. *Eclipse Modeling Framework*. https://github.com/eclipse-emf/org.eclipse.emf. 2025.

[6] Michael Jastram. *Die Eclipse Foundation wird 20 Jahre alt: Die interessantesten SE-Projekte*. de. https://www.se-trends.de/eclipse-foundation-wird-20-jahre-alt/. Accessed: 2025-6-2. Dec. 2024.

[7] Alexandra Kleijn. *Eclipse und die Eclipse Foundation*. Accessed: 2025-6-2. 2006. URL: https://www.heise.de/-221997.

[8] Philip Langer and Tobias Ortmayr. *Eclipse GLSP*. https://github.com/eclipse-glsp/glsp. 2025.

[9] S. Sendall and W. Kozaczynski. "Model transformation: the heart and soul of model-driven software development". In: *IEEE Software* 20.5 (2003), pp. 42–45. DOI: 10.1109/MS.2003.1231150.

[10] David Steinberg et al. *EMF: Eclipse Modeling Framework 2.0*. 2nd. Addison-Wesley Professional, 2009. ISBN: 0321331885.

[11] Daniel Strüber et al. "Henshin: A Usability-Focused Framework for EMF Model Transformation Development". In: *Graph Transformation*. Ed. by Juan de Lara and Detlef Plump. Cham: Springer International Publishing, 2017, pp. 196–208. ISBN: 978-3-319-61470-0.

[12] Daniel Strueber. *Henshin*. https://github.com/eclipse-henshin/henshin/wiki. 2025.

V. Acronyms

GLSP Graphical Language Server Platform
EMF Eclipse Modeling Framework
MDE Model-Driven Engineering
UI User Interface
GUI Graphical User Interface (UI)
IDE Integrated Development Environment
SDV Software-Defined Vehicle
JDT Java Development Tools
PDE Plug-in Development Environment
SDK Software Development Kit
API Application Programming Interface
UML Unified Modeling Language
XMI XML Metadata Interchange
XML Extensible Markup Language
LHS Left-Hand Side
RHS Right-Hand Side
NAC Negative Application Condition
PAC Positive Application Condition