Move your MDE teaching online: The MDENet Education Platform

Steffen Zschaler
szschaler@acm.org
King's College London
Department of Informatics
London, UK

Will Barnett
will.barnett@kcl.ac.uk
King's College London
Department of Informatics
London, UK

Artur Boronat
artur.boronat@leicester.ac.uk
University of Leicester
School of Computing and Mathematical Sciences
Leicester, UK

Antonio Garcia-Dominguez
a.garcia-dominguez@york.ac.uk
University of York
Department of Computer Science
York, UK

Dimitris Kolovos
dimitris.kolovos@york.ac.uk
University of York
Department of Computer Science
York, UK

Abstract

Teaching MDE is challenging, not least because the tools developed by the community can be difficult to install and configure as well as complex to master and use. To reduce the complexity for learners of MDE, enabling them to focus on the core MDE concepts, we present the MDENet Education Platform – an online, playground-based platform for learning MDE without the need for tool installation. Teachers declaratively describe learning activities, carefully controlling the complexity of the user interface learners are exposed to. We give an overview of the platform and highlight some current applications. The demonstration will show the use of the platform from the perspective of learners and teachers.

CCS Concepts

• **Social and professional topics** → **Software engineering education**; • **Software and its engineering** → *Domain specific languages*; *Software development techniques.*

Keywords

MDE, education, online, no installation, playground

ACM Reference Format:
Steffen Zschaler, Will Barnett, Artur Boronat, Antonio Garcia-Dominguez, and Dimitris Kolovos. 2024. Move your MDE teaching online: The MDENet Education Platform. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3652620.3687780

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3687780

1 Introduction

Model-Driven Engineering (MDE) [2] is a paradigm where models play a central role in the development of a software system. In terms of education, there is a consensus that MDE is a complex subject to teach [6, 15]. A particular challenge comes from the complexity and availability of suitable tools [3, 4, 6]. We focus on two challenges:

(1) *MDE tools are difficult to install and configure correctly.* Most MDE tools depend on a rich ecosystem of other tools and frameworks, all of which need to come together in the right versions and configurations for a given tool to work. **As a result, learners of MDE first have to overcome a significant hurdle in getting to a workable MDE tool installation on their computer before they can even begin to learn MDE concepts and techniques.**
(2) *MDE tools are too powerful for learners.* Even when a learner has successfully installed the MDE tools required for a particular course, they can easily become overwhelmed by the complexity of the tools themselves [22, 23]. **As a result, learners of MDE have to first learn which functionalities are relevant before they can focus on learning MDE concepts and techniques.**

These challenges create *accidental complexity* for learners of MDE. We want learners to encounter difficulties, but these should be *desirable difficulties* [1] that enhance their learning, such as guided practical engagement with the relevant concepts. Ideally, learners would be able to focus on the MDE concepts and techniques they are trying to understand, rather than first having to overcome several accidental challenges. We argue, therefore, that there is a need for MDE tools specifically for the purpose of learning MDE.

To address these challenges, we present an online playground environment for MDE learning activities—the MDENet Education Platform (EP in the rest of the paper). The web-based nature of the playground means there is no need to install anything beyond a basic web browser. The playground metaphor means that learners will only be exposed to a minimal interface focused on the files and functions required for a given learning activity. We provide a declarative language for flexibly defining learning activities. Learning

activities are packaged as GitHub repositories, enabling teacher–teacher collaboration as well as providing students with the ability to undertake the activities directly in standard IDEs if desired.

The remainder of the paper is structured as follows: We briefly recap related work in Sect. 2. Section 3 gives an overview of the EP, followed by a brief description of some recent applications in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Related work

Following a workshop at MODELS'23, an expert voice paper in SoSyM [15] recently catalogued requirements for teaching tools for modelling. Ease of installation, configuration, and use, as well as the teacher's ability to constrain what students can do are part of the requirements discussed.

No other generic playground solution for MDE exists, but there are playgrounds for specific tools. For example, the Epsilon Playground [16] enables web-based use of the various tools and languages in Epsilon [17]. It uses Functions-as-a-Service (FaaS) for its back-end functions allowing on-demand scalability and minimal running costs when the platform is not being used. Langium [25] also provides a bespoke playground service for basic language-workbench functionalities. A web-based platform for the MontiCore language workbench [19] based on JupyterLab [5] has been used for teaching the tutorials of a conference and lectures on the use and engineering of Domain Specific Languages (DSL).

In addition to the increasing number of playgrounds, there are web-based versions of IDEs such as Eclipse [9, 10] and Visual Studio Code [24]. Some code repositories use such online IDEs to provide direct access to repositories, including in educational settings. For example, GitHub Classroom offers access to Codespace IDEs (based on VSCode) for students undertaking activities provided through GitHub repositories [13].

Online MDE platforms have seen increasing interest recently—examples include AToMPM [26], Freon [27], and Gentleman [20]—though note that these tools have not been developed specifically for educational purposes. Umple [21], is an online modelling platform, focused on UML-style models and code generation from them. It is education-focused, but only provides support for a fixed set of modelling languages and tools.

3 Platform description

The EP builds on the Epsilon Playground [16] but generalises the architecture to allow the declarative description of learning activities and the flexible integration of a wide range of MDE tools. The EP also integrates with GitHub to provide a way for students to save their work and easily transition to the use of real-world MDE tools and environments. In this section, we give an overview of the key components of the EP. The EP is available on GitHub[1], including a video demonstrating an example activity. A publicly hosted version is also available at ep.mde-network.org.

The EP is a single-page web application, with most of the functionality running directly in the learner's browser. Figure 1 gives a high-level overview of the key components. A `platform server` provides the HTML and JavaScript to be executed in the learner's browser. It also runs the `Token Server` [7], which manages GitHub

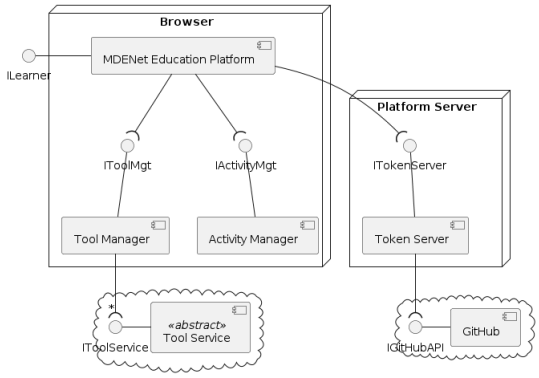

Figure 1: High-level architecture of the EP.

OAuth authentication for access to the repository underlying a learning activity. This means learners can easily save the current state of work as a commit to the underlying repository; the EP supports this directly through a "Save" button in the menu.

Three key components run in the learner's browser:

(1) The `MDENet Education Platform` is the main entry point.
(2) The `Activity Manager` is responsible for parsing and validating activity descriptions, enabling the EP to configure the appropriate user interface.
(3) The `Tool Manager` keeps track of the tool services in use by the current learning activity. These implement wrappers around MDE tools to make them accessible to the EP. They are implemented (and typically hosted) by tool providers.

Learning activities. Learning activities are stored in GitHub repositories. Two types of files have to be provided:

(1) A YAML [8] or JSON [11] file declaratively describing the configuration of the EP for the learning activity.
(2) Any other files required for the learning activity—for example, models, language grammars, meta-models, etc.

Teachers can include arbitrary files and folder structures beyond the files directly required for the learning activity. Setting up the repository to work directly with regular tools makes it possible for learners to engage through the EP as well as through a regular IDE.

To describe the set of learning activities available, a teacher uses a domain-specific language, currently encoded as a JSON schema [14] (and, thus, also accessible via YAML [8]). We provide a graphical overview of the abstract syntax of the activity-specification language in meta-model notation in Fig. 2[2].

These descriptions include:

(1) A reference to the *tools* used by the learning activity. These are referenced through their URL, where tool providers make available a hosted wrapper around their tool.
(2) A definition of the *panels* that learners should be able to interact with and the files providing the contents for these panels. Panels can be of a range of different types as provided by the different tools used in the activity. They can contain text or graphics, depending on need.

[1] https://github.com/mdenet/educationplatform-docker

[2] This is an approximation of the JSON schema adjusted for readability as a meta-model.

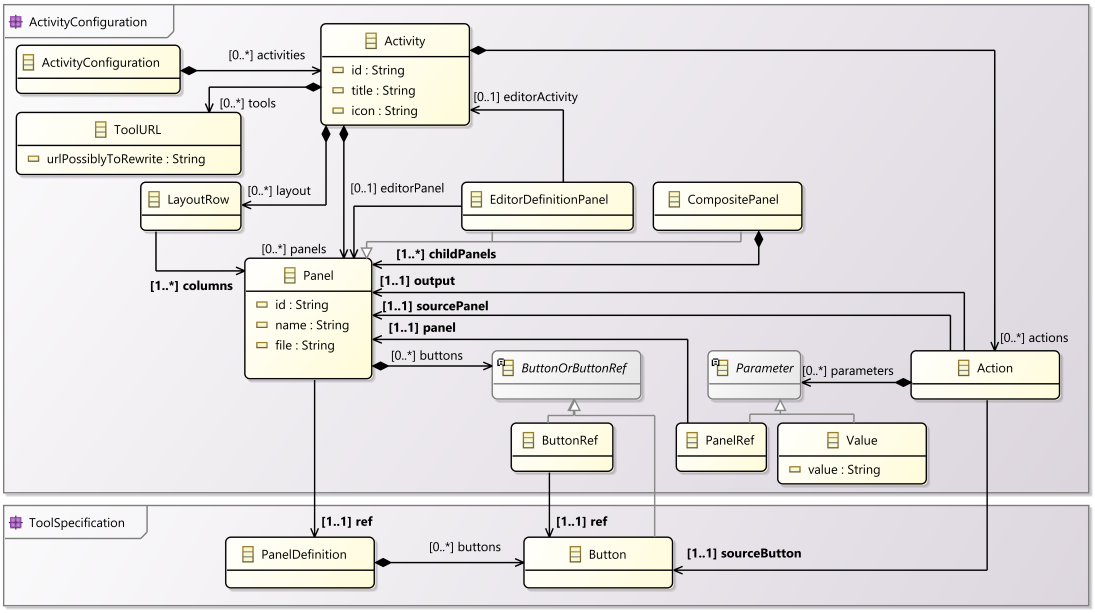

Figure 2: Meta-model of the activity-specification language

(3) A *layout* definition, identifying the panels to show on the screen and their relative locations.
(4) *Action* definitions defining what happens in response to learners clicking on buttons in the user interface.

Figure 3 shows an example of the interface that learners see when completing an activity. Here, we reuse an example from the Epsilon Playground [16] focusing on the validation language EVL [18], which we have ported to the EP.[3] In addition to a menu area (❶) on the left, there are five panels: The contents of panel ❷ are the constraints to check against the model ❸ and its metamodel ❺. Panel ❹ displays the result of evaluating the constraints for the model. The evaluation is triggered when the learner clicks the run button ▶ in Panel ❷. The console ❻ shows error messages if the EVL script cannot be parsed correctly.

Listing 1 shows the declarative definition of the learning activity in Fig. 3. After some preliminaries in Lines 2–4, the activity declares (Lines 6–7) that it will use the Epsilon tools. Lines 9–27 define the different panels that will be available to the learner, making reference to the different files that should be shown in those panels, where appropriate. Lines 29–32 describe which panels are visible and how to arrange them on the screen. Finally, Lines 34–43 define what happens when the learner clicks on the run button ▶ in Panel ❷. This implicitly makes use of a web-API function provided by the Epsilon tool service and linked to the run button by the tool specification provided by the Epsilon tool service.

Language development activities. A key MDE aspect is developing new, typically domain-specific, modelling languages. Therefore, the EP needs to be able to support learning activities on that topic. Such activities take two steps:

(1) Learners *create* a description of their modelling language (via a combination of a meta-model, a grammar, a graphical-syntax model, a generator template, validity constraints).
(2) Learners *use* the language they have defined. They can see how their language description has been translated into editors and other tools for their language.

To support such activities, the EP allows activities to be coupled dynamically within an activity-configuration file. Teachers define language-workbench activities consisting of

(1) an activity for learners to provide descriptions of their new language. A button in this activity triggers the generation of a new tool service encapsulating the learner's new language.
(2) a separate activity configured to use the tool service dynamically generated from the learner's language description.

Listing 2 shows an excerpt from an activity where learners create languages with Xtext [12]. Students write a grammar and then try out the generated editor. There are two activities:

(1) `activity-xtext` defines a panel where the learner can create the grammar. This panel uses the `xtext-grammar` panel type provided by the Xtext `tool service`, which includes a button to trigger the generation of Xtext artefacts from the grammar. In addition to the usual panel attributes, `panel-xtext` uses two attributes `editorActivity` and `editorPanel`. These refer to the second activity `activity-editor` and a panel in that activity (`panel-editor`).
(2) `activity-editor` is the activity learners use to try out their new language. Because the activity is referenced from a

[3] This primarily required creating declarative specifications of the activities and the Epsilon tools in the format required by the EP.

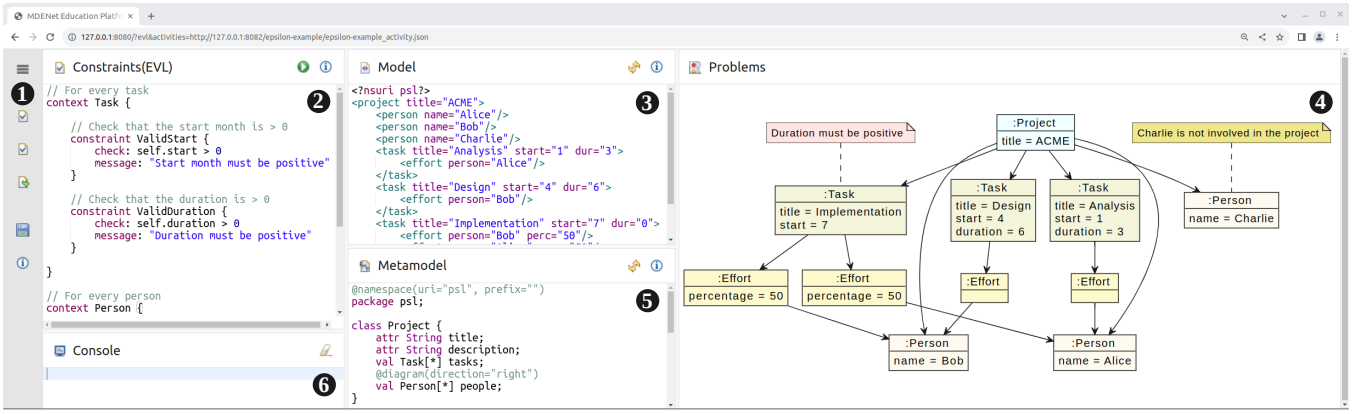

Figure 3: The Epsilon EVL example in the EP. Numbered circles indicate different parts referenced from the text.

panel in the first activity, the EP only makes `activity-editor` available through the menu if the generation action in the first activity has been used by the learner and has produced an editor. The generation action returns the URL of the newly generated tool service, which is made available to the learning activity via the `{{ID-panel-editor}}` variable, used to load the generated tool in the second activity.

MDE tools. Tool services provide the functionality that the installed tools on a developer's local machine environment would normally provide—for example, model-to-model transformation, text generation, or model validation. They make up the back-end of the EP, providing a wrapper around an existing MDE tool. A tool service comprises a (set of) tool function(s) and static resources.

The tool function provides a web-based API endpoint that conforms to the tool interface specification. The static resources a tool provider must create include: a tool configuration file, highlighting rules, and icons. Tools are provided independently of learning activities. They may be hosted on the same infrastructure as the EP, but they may also be hosted on separate infrastructure—for example, controlled by the tool provider. Teachers reference tools by their URL to use them in an activity they are creating.

4 Applications

We have successfully used the EP in several teaching contexts:

(1) *As part of a tutorial on MDE DevOps.* This live tutorial used GitHub Classroom and the EP to deliver a series of MDE activities. These used the Epsilon toolkit and GitHub Actions to demonstrate the use of MDE to drive DevOps pipelines.
(2) *As part of a university course on MDE and language engineering with Xtext.* We provided the EP as an optional alternative to using Eclipse in a 10-week university course. Activities included all stages of language development, as well as combinations of Xtext and ETL, enabling learners to build model transformations on top of their own DSML.
(3) *To develop an online playground for YAMTL.* This online playground[4] uses the EP to provide several activities to help explore the capabilities of YAMTL in a practical setting.

5 Conclusions

We have presented the MDENet Education Platform, an online platform for teaching MDE based on the playground metaphor. The platform allows teachers to declaratively describe learning activities and deliver them to learners via a GitHub repository. Learners can do the activities without the need to install any software. They are guided through the activity via a carefully restricted interface providing only the functionality needed for the current activity.

Acknowledgments

Zschaler and Barnett's work was partly funded through the UK Engineering and Physical Sciences Research Council (EPSRC) MDENet grant (EP/T030747/1). The work of Kolovos and Garcia-Dominguez was partly funded by the SCHEME InnovateUK project (#10065634).

References

[1] Elizabeth L. Bjork and Robert A. Bjork. 2011. Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. In *Psychology and the real world: Essays illustrating fundamental contributions to society*, M. A. Gernsbacher, R. W. Pew, L. M. Hough, and J. R. Pomerantz (Eds.). Worth Publishers, 56–64.
[2] Marco Brambilla, Jordi Cabot, Manuel Wimmer, and Luciano Baresi. 2017. *Model-Driven Software Engineering in Practice* (second edition ed.). Morgan & Claypool.
[3] Antonio Bucchiarone, Jordi Cabot, Richard F. Paige, and Alfonso Pierantonio. 2020. Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling* 19 (2020), 5–13. Issue 1. https://doi.org/10.1007/s10270-019-00773-6
[4] Shalini Chakraborty and Grischa Liebel. 2023. We do not understand what it says – studying student perceptions of software modelling. *Empirical Software Engineering* 28, 6 (Nov. 2023). https://doi.org/10.1007/s10664-023-10404-w
[5] Joel Chuks Charles, Nico Jansen, Judith Michael, and Bernhard Rumpe. 2022. Teaching the Use and Engineering of DSLs with JupyterLab: Experiences and Lessons Learned. In *Modellierung 2022*, Matthias Riebisch and Marina Tropmann-Frick (Eds.), Vol. P-324. Gesellschaft für Informatik e.V., 93–110. https://doi.org/10.18420/modellierung2022-014
[6] Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sebastien Mosser, Richard F. Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, Gabi Taentzer, Antonio Vallecillo, and Manuel Wimmer. 2018. How Do We Teach

[4] https://yamtl.github.io/playground/?activities=https://yamtl.github.io/playground-activities/yamtl-demo-activity.yml

```yaml
activities:
  - id: evl
    icon: evl
    title: Validate Project Plan

    tools:
    - https://ep.mde-network.org/tools/epsilon/
        tools

    panels:
    - id: panel-evl (2)
      name: Constraints(EVL)
      ref: evl
      file: psl.evl
    - id: panel-model (3)
      name: Model
      ref: flexmi
      file: psl-evl.flexmi
    - id: panel-mm (5)
      name: Metamodel
      ref: emfatic
      file: psl.emf
    - id: panel-console (6)
      name: Console
      ref: console
    - id: panel-problems (4)
      name: Problems
      ref: problem

    layout:
      area:
      - [panel-evl, panel-model, panel-problems]
      - [panel-console, panel-mm]

    actions:
    - source: panel-evl
      sourceButton: action-button
      parameters:
        emfatic: panel-mm
        flexmi: panel-model
        program: panel-evl
      output: panel-problems
```

Listing 1: Example activity definition. Numbers in circles map to the panels in Fig. 3

```yaml
    activities:
      - id: activity-xtext
        panels:
          - id: panel-xtext
            name: Grammar
            ref: xtext-grammar
            file: Turtles.xtext
            editorActivity: activity-editor
            editorPanel: panel-editor
          - ...
        ...
      - id: activity-editor
        tools: [{{ID-panel-editor}}/editor_tool.
            json, ... ]
        panels:
          - id: panel-editor
            ...
```

Listing 2: Configuring language-workbench activities

Modelling and Model-Driven Engineering? A Survey. In *Proc. 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 122–129. https://doi.org/10.1145/3270112.3270129

[7] Curity. 2024. The Token Handler Pattern for Single Page Applications. Online: https://curity.io/resources/learn/the-token-handler-pattern/, last accessed 12 June.

[8] Ingy döt Net, Tina Müller, Pantelis Antoniou, Eemeli Aro, and Thomas Smith. 2021. YAML Ain't Markup Language (YAML) revision 1.2.2. Online: https://yaml.org/spec/1.2.2/, last visited 21 May 2024. https://yaml.org/spec/1.2.2/

[9] Eclipse Foundation, Inc. 2020. Eclipse Orion. Online: https://projects.eclipse.org/projects/ecd.orion, last accessed 23 June 2024.

[10] Eclipse Foundation, Inc. 2024. Theia – Cloud and Desktop IDE Platform. Online: https://theia-ide.org/, last accessed 23 June 2024.

[11] Ecma. 2017. *ECMA-404: The JSON data interchange syntax*. Ecma International – European Association for Standardizing Information and Communication Systems, Geneva, Switzerland. https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf

[12] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: implement your language faster than the quick and dirty way. In *Companion Proc. ACM Int'l Conf. Object Oriented Programming Systems Languages and Applications (OOPSLA'10) (OOSPLA'10)*. ACM. https://doi.org/10.1145/1869542.1869625

[13] GitHub. 2024. Using GitHub Codespaces with GitHub Classroom. Online: https://docs.github.com/en/education/manage-coursework-with-github-classroom/integrate-github-classroom-with-an-ide/using-github-codespaces-with-github-classroom, last accessed 23 April 2024.

[14] JSON Community. 2020. JSON Schema Specification, version 2020-12. Online: https://json-schema.org/specification, last visited 22 May 2024.

[15] Jörg Kienzle, Steffen Zschaler, William Barnett, Timur Sağlam, Antonio Bucchiarone, Silvia Abrahão, Eugene Syriani, Dimitris Kolovos, Timothy Lethbridge, Sadaf Mustafiz, and Sofia Meacham. 2024. Requirements for Modelling Tools for Teaching. *Software and Systems Modelling* (2024). To appear..

[16] Dimitris Kolovos and Antonio Garcia-Dominguez. 2022. The Epsilon Playground. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. Association for Computing Machinery, 131–137. https://doi.org/10.1145/3550356.3556507

[17] Dimitrios Kolovos, Richard Paige, Louis Rose, and Fiona Polack. 2009. *The Epsilon Book*. Published on-line: http://www.eclipse.org/gmt/epsilon/doc/book/.

[18] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. 2009. On the Evolution of OCL for Capturing Structural Constraints in Modelling Languages. In *Rigorous Methods for Software Construction and Analysis: Essays Dedicated to Egon Börger on the Occasion of his 60th Birthday*, Jean-Raymond Abrial and Uwe Glässer (Eds.). Springer Berlin Heidelberg, 204–218. https://doi.org/10.1007/978-3-642-11447-2_13

[19] Holger Krahn, Bernhard Rumpe, and Steven Völkel. 2010. MontiCore: a Framework for Compositional Development of Domain Specific Languages. *Int'l Journal on Software Tools for Technology Transfer (STTT)* 12, 5 (Sept. 2010), 353–372.

[20] Louis-Edouard Lafontant and Eugene Syriani. 2020. Gentleman: a light-weight web-based projectional editor generator. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. Association for Computing Machinery, Article 1, 5 pages. https://doi.org/10.1145/3417990.3421998

[21] Timothy C. Lethbridge, Andrew Forward, Omar Badreddin, Dusan Brestovansky, Miguel Garzon, Hamoud Aljamaan, Sultan Eid, Ahmed Husseini Orabi, Mahmoud Husseini Orabi, Vahdat Abdelzad, Opeyemi Adesina, Aliaa Alghamdi, Abdulaziz Algablan, and Amid Zakariapour. 2021. Umple: Model-driven development for open source and education. *Science of Computer Programming* 208 (2021), 102665. https://doi.org/10.1016/j.scico.2021.102665

[22] Grischa Liebel, Omar Badreddin, and Rogardt Heldal. 2017. Model Driven Software Engineering in Education: A Multi-Case Study on Perception of Tools and UML. In *IEEE 30th Conference on Software Engineering Education and Training (CSEE&T'17)*. IEEE. https://doi.org/10.1109/cseet.2017.29

[23] Grischa Liebel, Rogardt Heldal, and Jan-Philipp Steghofer. 2016. Impact of the Use of Industrial Modelling Tools on Modelling Education. In *IEEE 29th International Conference on Software Engineering Education and Training (CSEE&T'16)*. IEEE. https://doi.org/10.1109/cseet.2016.18

[24] Microsoft. 2023. Visual Studio Code for the Web. Online: https://code.visualstudio.com/docs/editor/vscode-web, last accessed 23 June 2024.

[25] Markus Rudolph. 2023. The Langium Playground -- TypeFox Blog. Online: https://www.typefox.io/blog/langium-playground, last accessed 23 June 2024.

[26] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Huseyin Ergin. 2013. AToMPM: A web-based modeling environment. In *16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013): Companion proceedings*. 21–25.

[27] Jos Warmer and Anneke Kleppe. 2022. Freon: An Open Web Native Language Workbench. In *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering*. 30––35. https://doi.org/10.1145/3567512.3567515

A Outline of planned demonstration

The demonstration will present the EP from the perspective of a learner and a teacher.

We will begin with the learner perspective, walking through a learning activity where learners:

(1) Define a simple Xtext grammar;
(2) Generate the language infrastructure;
(3) Experiment with the resulting infrastructure; and
(4) Write a simple ETL transformation to transform models in their language into optimised models in their language.

Figures 4 and 5 show screenshots of the two stages of this part of the demonstration, respectively. Demonstration participants will be able to explore the activities themselves via a link to a GitHub Classroom repository.

In the second stage of the demonstration, we will show how the above activity would be defined by a teacher. We will do this, by walking participants through the repository that has been set up for the activity; in particular the declarative description of the learning activity. A video of a 90-minute step-by-step tutorial on how to create learning activities can be found online[5]. Rather than such a step-by-step tutorial, we will show and explain the relevant parts of the activity definition, to ensure we stay within the time available for a tool demonstration.

[5] https://www.youtube.com/watch?v=qqJI5OqJqjs

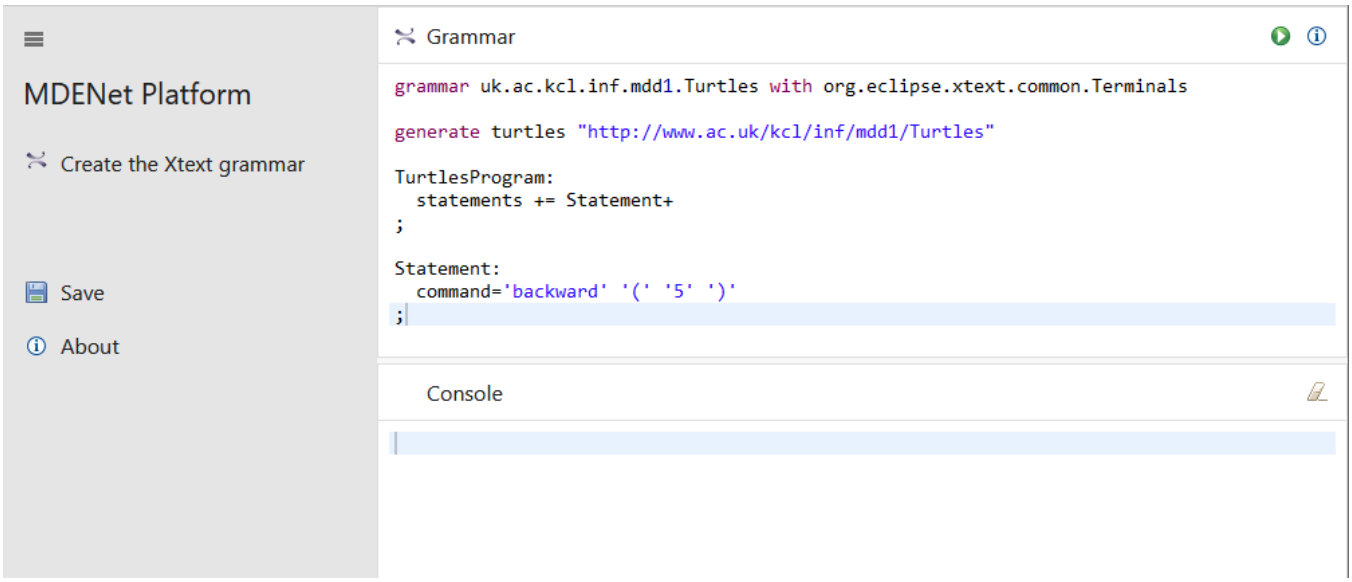

Figure 4: Xtext activity: defining the grammar

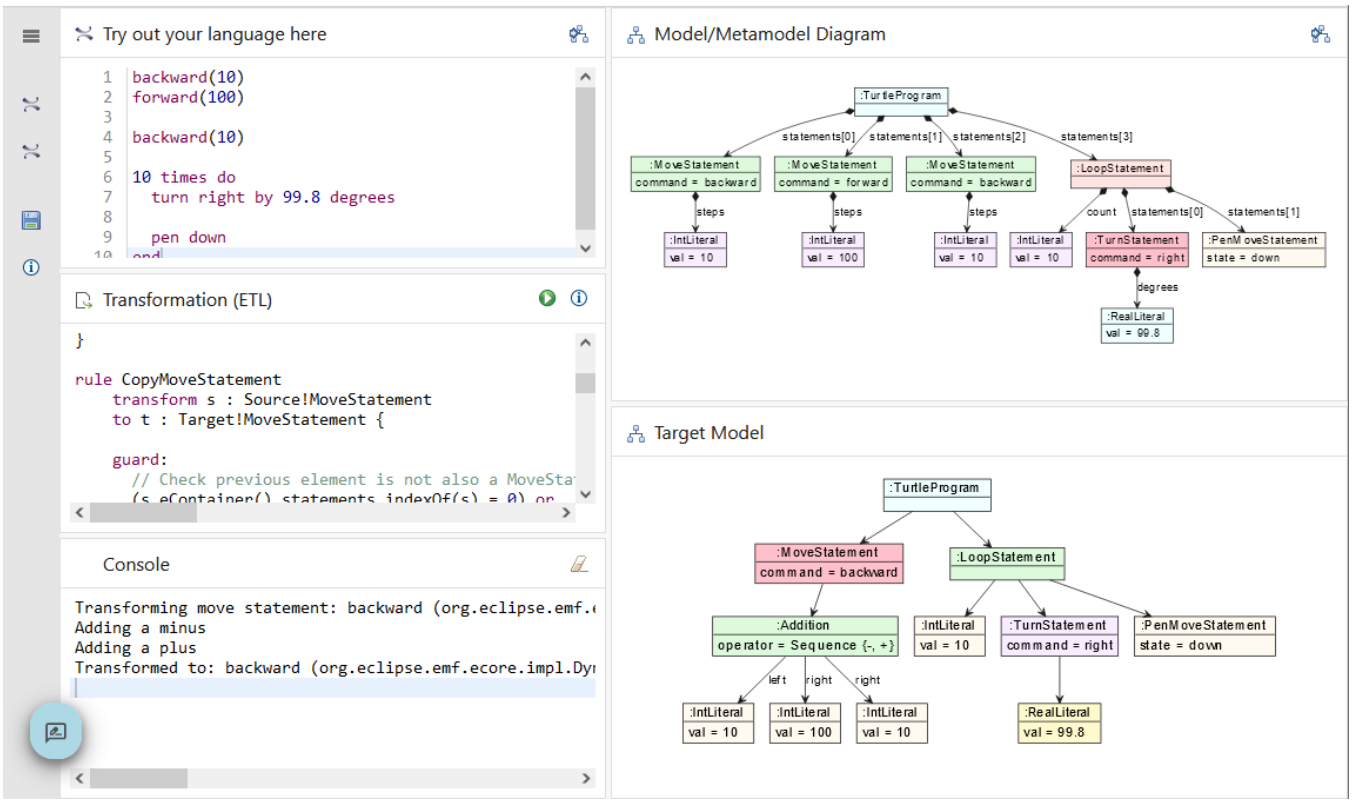

Figure 5: Xtext activity: combining a learner-defined Xtext editor with an ETL transformation