MoVE – A Standalone Modelica Vector Graphics Editor

Nicola Justus[1] Christopher Schölzel[1] Andreas Dominik[1]

[1]KITE, Technische Hochschule Mittelhessen, Giessen, Germany, {nicola.justus, christopher.schoelzel, andreas.dominik}@mni.thm.de

Abstract

Modelica models can have a graphical icon defined as a bitmap or vector graphics. Vector graphics have several benefits, the most obvious being free scaling of images from icon to poster size. With OpenModelica there already exists one open source tool that can be used for editing these vector graphics icon annotations, but it does not reach the usability comfort of professional vector graphics editing tools.

In this paper we present the Modelica Vector graphics Editor (MoVE), a standalone open source editor for Modelica's vector graphics syntax that provides many convenience features inspired by the vector graphics editor Inkscape. These features include grouping, snap to grid, move to foreground/background, rotation handles, and drawing perfect circles and squares as well as horizontal and vertical lines when holding *Shift*.

We hope that MoVE, as a part of the Modelica Tool Ensemble (MoTE), can enrich the open source ecosystem of Modelica by simplifying the creation of more elaborate vector graphics icons for Modelica models.

Keywords: JavaFX, vector graphics, open source, SVG, Inkscape, MVC, MoTE, OpenModelica

1 Introduction

Modelica is a language for modeling complex physical systems that also incorporates a graphical representation of model components into the language itself. These graphical representations come in the form of annotation statements that can either contain a link to a bitmap image or define an image using a vector graphics syntax (Modelica Assoc., 2012). Vector graphics have the advantage that they are freely scalable. This is interesting in any context where a model might not only be displayed as a small icon on a screen but also has to be presented to a larger audience on a slide or a poster.

Unfortunately, creating vector graphic icons for Modelica models is not as easy as it could be. Standard vector graphics tools such as Inkscape (Inkscape, 2016) provide a rich set of features for precise and fast interaction, such as grouping, rotation handles, sending elements to the front or back, snap to grid, or drawing straight horizontal lines and perfect circles when holding a modifier key. It would be ideal, if we could use such a tool and save the resulting image in Modelica notation. However, the Modelica annoations are not compatible with vector graphics formats such as Scalable Vector Graphics (SVG) (Dahlström et al., 2011), since there are both features in SVG that have no equivalent in the Modelica syntax and vice versa.

There are many commercial tools for Modelica but OpenModelica is the only open source choice for graphical editing of Modelica models (Fritzson et al., 2005). This graphical editor of OpenModelica is called OpenModelica Connection Editor (OMEdit) (Asghar et al., 2011). It has all features that are required to create vector graphic annotations, but does not provide the same level of user-friendliness as Inkscape or related tools. For example, non-standard rotation angles, fill patterns and line patterns can only be changed via a properties dialog that has to be opened separately for each component; the order of Elements cannot be changed (although respective entries exist in the context menu); drawing of straight horizontal lines and perfect circles is not supported; and when we began this project, OpenModelica did not even support undo-redo operations for graphical manipulations. Furthermore OMEdit generates the icon annotation as one big line of code. This is especially uncomfortable when the source code is managed through a version control system.

In this Paper we therefore present the Modelica Vector graphics Editor (MoVE), a new standalone open source Modelica vector graphic editor with a streamlined interface similar to Inkscape. In the following, we will first give a bit more detail of the context in which MoVE was created. In section two, we will then present an overview of the technologies used to create the editor followed by a discussion of the major design aspects in section three. Section three presents the major features of MoVE and section four explains current limitations leading to the conclusion in section six.

1.1 Background and Related Work

Modelica Tool Ensemble

MoVE is part of the Modelica Tool Ensemble (MoTE) (Justus, Hoppe, and Schölzel, 2017). MoTE aims to provide small user-friendly standalone appliations for editing and simulating Modelica models. With this toolset we follow the Unix philosophy to "make each program do one thing well" (McIlroy, Pinson, and Tague, 1978). Separating the different tasks needed for editing and simulating Modelica models leads to smaller applications that are easier to maintain than a full-featured development environment like OpenModelica. Additionally, users may choose to use the tools that they like and substitute the tools they do not like with other alternatives, leading to a more flexible ecosystem that can accommodate different user needs. MoVE only touches the main annotation statement of a model. To edit other parts of the model, one can, for example, use the text editor Atom (GitHub, 2016) which can provide type checking, auto-completion and error highlighting when coupled with Modelica | Editor (Mo|E), another tool in the MoTE family. Instead one could also chose to use OMEdit or the Eclipse plugin Modelica Development Tooling (MDT) (Pop et al., 2006).

Inkscape

The main source of inspiration for MoVE was the aforementioned vector graphics editor Inkscape (Inkscape, 2016). Inkscape is an open source application that can be used to create professional and complicated vector graphic images. It supports a rich set of features including alignment of elements or individual nodes, combination and cutting of multiple paths, drawing with Bezier curves, bold and italic text, importing shapes from a PDF-file, and many many more. Features that are not already included can be made available with a language-independent scripting API. These features are presented to the user mainly through toolbars and hotkeys that make the interaction fast and seamless. The native format of Inkscape is SVG (Dahlström et al., 2011), which is an XML-based format that can easily be interpreted by other tools.

MoVE does not nearly provide as many features as Inkscape, but it tries to follow the same principles for usability and precision.

2 Technologies

This chapter is a short overview over the technologies that are used for implementing MoVE. Basically MoVE is written in the programming language Scala using the graphical user interface toolkit JavaFX.

2.1 Scala

Scala is a programming language for the Java Virtual Machine (JVM). This means that it is platform-independent and that it is possible to use any Java library. Especially the library JavaFX is useful for creating a modern Graphical User Interfaces (GUIs). Scala merges object orientation with functional programming, which allows to write code faster and more flexible than in plain Java. It also brings its own set of libraries such as a parser combinator library (EPFL and Typesafe, Inc., 2016) that proved very useful for this project.

2.2 JavaFX

JavaFX is a GUI toolkit for the Java programming language. Because it is written for Java it runs on the JVM and is also usable from Scala. JavaFX is the latest toolkit for GUIs running on the JVM and incorporates many ideas of modern GUI design. JavaFX provides a special format for describing the structure of a UI. This format is called FXML and based on XML. To develop GUIs using the FXML format, Oracle provides a graphical editor for building the user interface by dragging and dropping interface components, namely the *SceneBuilder*. This makes GUI design much faster and leads to a clean separation of the layout and the actual code.

3 Design

3.1 Parser

To load existing Modelica models we have created a simple parser for Modelica source code. This parser is built using the *scala-parser-combinators* library (EPFL and Typesafe, Inc., 2016). This library allows combination of simple parsers to create more complex ones. An external parser generator is not necessary. MoVE ignores everything in the source file, except the icon annotations of all models defined in the file. This makes the parser (and MoVE) mostly independent of future language modifications, thus MoVE should work with future versions of Modelica. If the icon annotations are modified, the parser has to be modified. This should be a small effort. Additionally this assures that MoVE interacts nicely with version control systems. Since we only parse annotations, we can *guarantee* that we will not change any other part of the model.

During the parsing process, the parser generates a MoVE-specific abstract syntax tree. This tree is then transformed into shapes, that are derived from the standard JavaFX shapes. Finally this shapes are displayed in the user interface.

3.2 Model–View–Controller

JavaFX is built around the Model-View-Controller (MVC) software design pattern (Reenskaug, 1979). MVC splits the application in three parts. The first part is the model[1], which represents the business data. The model updates the view if someone changes the model. The second part is the view, which displays the data and listens on updates to the model. The third part is the controller, which connects a model with the respective view. The controller is also responsible for user interactions and transforms them into commands for the model or the view. The typical MVC workflow is depicted in Figure 1.

Because JavaFX already provides views, which contain the data representation for shapes, MoVE is designed with controllers and views. There are no explicit models. They are *hidden inside* of the JavaFX shapes.

3.3 JavaFX Elements

To display Modelica's graphical primitives (Modelica Assoc., 2012), we have created a small set of JavaFX elements. These elements are all derived from the standard JavaFX shape elements and add additional properties and behavior such as fill and stroke patterns. The JavaFX shapes

[1]Here, in section 3.2, the word "model" refers to source code of a software project that is structured with the MVC-Pattern and not to a Modelica model.

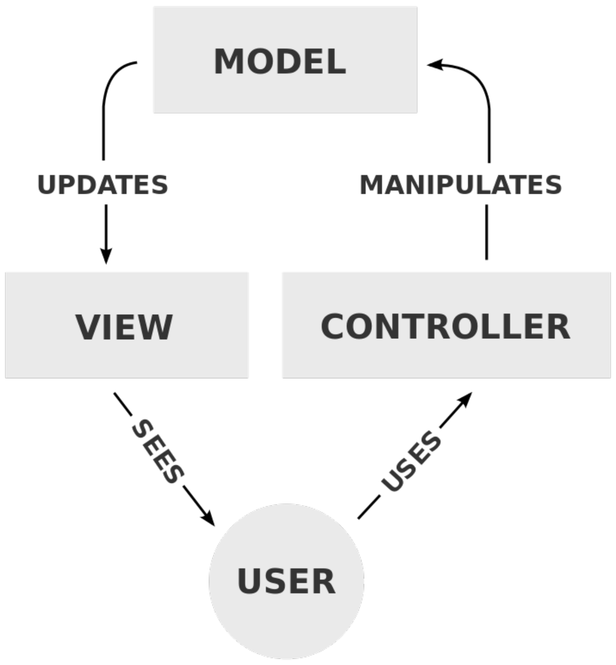
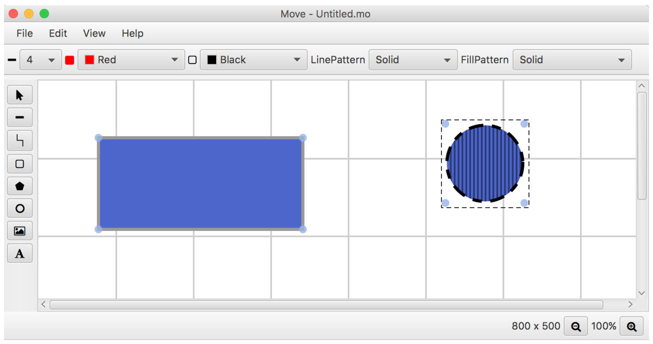


Figure 1. The Model-View-Controller (MVC) software design pattern splits an application into three parts in order to increase maintainability and extensibility (Frey, 2016).

provides the basic properties for rectangles, ellipses, lines, paths, polygons, and images.

Furthermore, for abstracting the common behavior of the shapes, they are all derived from a specific trait, which provides the common behavior. For example, all shapes that *behave like* a *rectangle* are derived from the trait *RectangleLike*. A similar trait is defined for shapes that *behave like* a *path*.

3.4 UI Overview



4 Red Black LinePattern Solid FillPattern Solid

800 x 500 100%

Figure 2. The user interface of MoVE is built with the JavaFX-framework and consists of three main toolbars: tool selection (left), tool properties (top) and zoom and size indicator (bottom).

The user interface contains 3 toolbars for interacting with MoVE (Figure 2).

The top toolbar contains controls for specifying the color of selected or newly drawn shapes. Going left to right this toolbar starts with a selector for the stroke size, followed by the color pickers for the fill color and stroke color. The color pickers are followed by a selector for the *LinePattern* and *FillPattern*. For these last two elements, all patterns defined in chapter 18.6 from (Modelica Assoc., 2012) can be selected.

The left toolbar contains the tools for selecting and moving as well as drawing the icon primitives. Going top to bottom it starts with the arrow, which is used for selecting and moving shapes. The arrow is followed by the tools for drawing lines, paths, rectangles, polygons and ellipses, and for inserting images, and text.

The bottom toolbar currently only contains two items: an indicator for the size of the draw pane and buttons for controlling the zoom. The magnifying glass with the minus zooms out and the magnifying glass with the plus zooms in.

4 Features

4.1 Code Generation

MoVE provides two code generators for the icon annotation. The first generates the annotation as one big line and writes it into the model. This is similar to OMEdit. The second code generator generates *pretty-printed* code with line breaks and indentations, which is more readable than a big line (Listing 1). This style is also better supported by version control systems as they can recognize which lines or properties have changed instead of reporting only a change of the whole annotation.

Listing 1. MoVE generates a well formatted icon annotation with line breaks and indentation.

```
model Test
  annotation(
  Icon (
    coordinateSystem(
      extent = {{0,0},{200,125}}
    ),
    graphics = {
      Rectangle(
        origin = {34,96},
        lineColor = {0,0,0},
        fillColor = {128,186,36},
        lineThickness = 4.0,
        pattern = LinePattern.Solid,
        fillPattern = FillPattern.Solid,
        extent = {{-14,8}, {14,-8}}
      ),
      Ellipse(
        origin = {75,91},
        lineColor = {0,0,0},
        fillColor = {128,186,36},
        lineThickness = 4.0,
        pattern = LinePattern.Solid,
        fillPattern = FillPattern.Solid,
        extent = {{-13,10}, {13,-10}},
        endAngle = 360
      )
    })
  );
end Test;
```

4.2 Grouping

MoVE supports grouping of multiple shapes through *Edit* → *Group* or by pressing *Ctrl+G*. Moving a shape which is part of a group moves all shapes that are part of this group (Figure 3). Ungrouping is also supported through *Edit* → *Ungroup* or by pressing *Ctrl+Shift+G*. Note that the groups are only used in MoVE and are discarded when the model is saved, because this is not supported by the icon annotation syntax of Modelica.

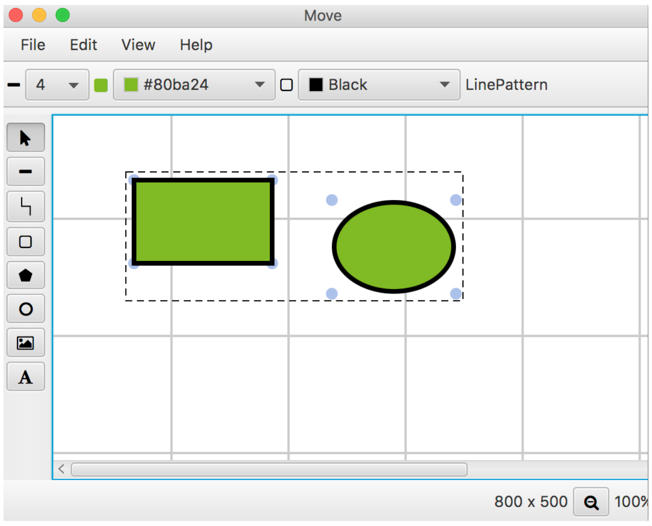


Figure 3. MoVE allows to group shapes together in the user interface, so that they can be easily moved together. These groups are lost when the annotation is saved.

4.3 Stacked Shapes

MoVE allows to move shapes into the background using *ContextMenu* → *In Background* and to move shapes into the foreground using *ContextMenu* → *In Foreground* (Figure 4). This allows easy modifying of the order of stacked shapes.

Furthermore, MoVE also handles shapes with the fill pattern *FillPattern.None* in an intuitive way. Shapes that are behind the transparent filling can still be selected. The transparent shape itself is only selected when the user clicks on the visible border.

4.4 Export as Images

MoVE enables exporting of Modelica icons either as *PNG* or as *SVG* (Figure 5). SVG is especially interesting because SVG images can be further modified in Inkscape. This is useful if the user likes to create a poster which contains a graphic from a Modelica model.

4.5 Rotation

After a double click on a shape, four red anchors appear at the corners of the shape (Figure 6). Moving the anchors rotates the shape around its center. This is more intuitive than rotating a shape by defining a specific degree value through a separate property dialog.

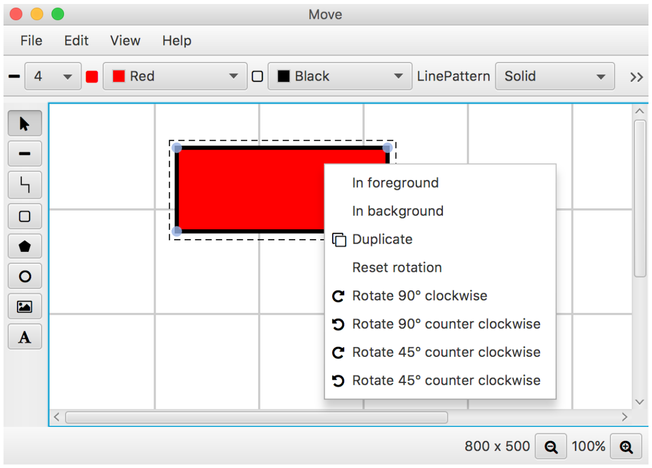


Figure 4. The context menu for a shape contains controls for rotation and stacking order.

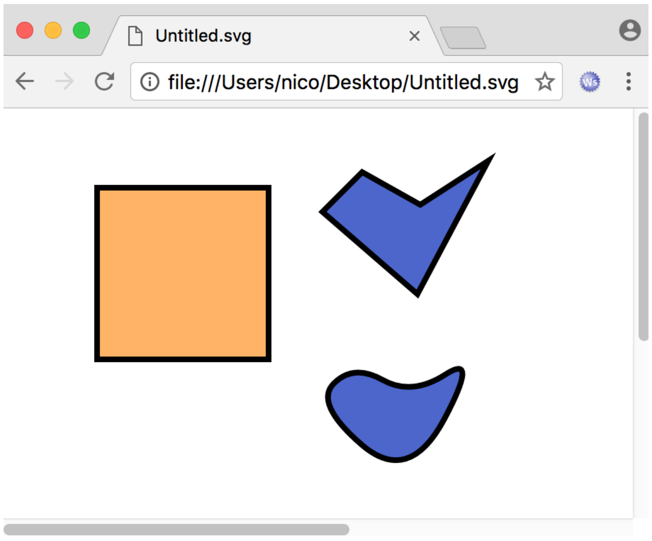


Figure 5. SVG image exported from MoVE displayed in Google Chrome.

Additionally to rotation by moving the anchors, it is possible to rotate an element using the context menu:

- *ContextMenu* → *Rotate 90° clockwise*
- *ContextMenu* → *Rotate 90° counter clockwise*
- *ContextMenu* → *Rotate 45° clockwise*
- *ContextMenu* → *Rotate 45° counter clockwise*

4.6 Snap to Grid

MoVE operates on a customizable grid. The size of the grid can be modified to fit the needs of the user. Via the menu entry *View* → *Enable snapping* or by pressing *Ctrl+A* the *snap to grid* function can be toggled. If activated, elements will snap to the precise location of the grid lines when they are moved close to such a line. This allows for a precise alignment of individual elements.

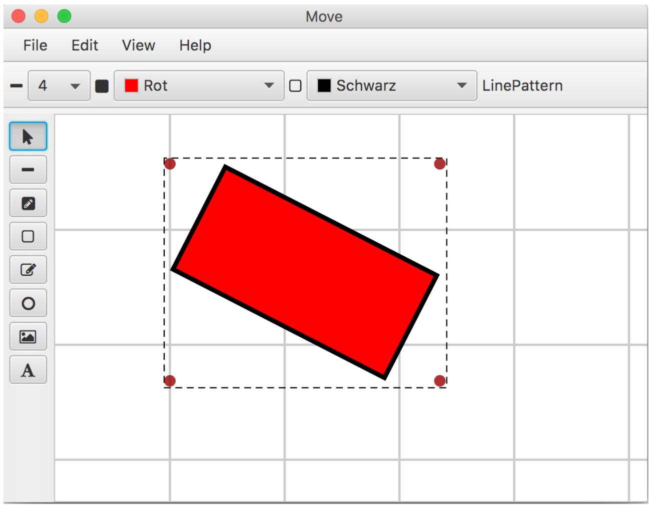


Figure 6. Arbitrary rotations can be realized in MoVE by rotation handles (red dots).

4.7 Config Files

MoVE uses simple text files as configuration files that are placed inside of `~/.move`. Application settings are placed in the file `~/.move/move.conf` and keyboard shortcuts are read from `~/.move/shortcuts.conf`. Both files can be customized with any text editor.

4.8 Additional features

When holding down *Shift* while drawing a shape, it is possible to create straight horizontal or vertical lines, perfect squares, and perfect circles (Figure 7).

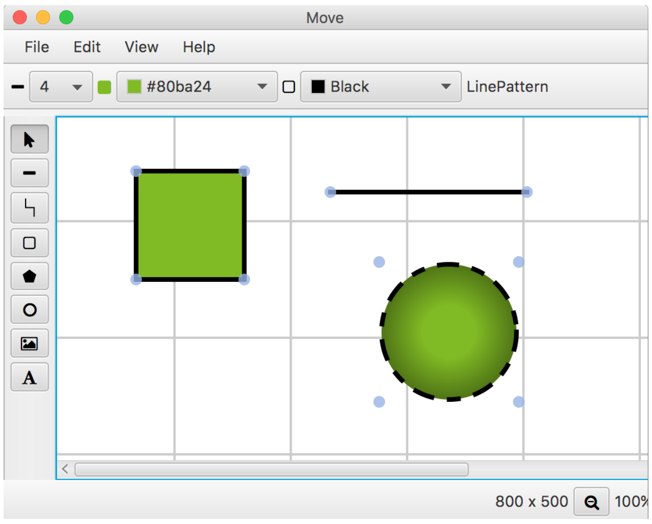


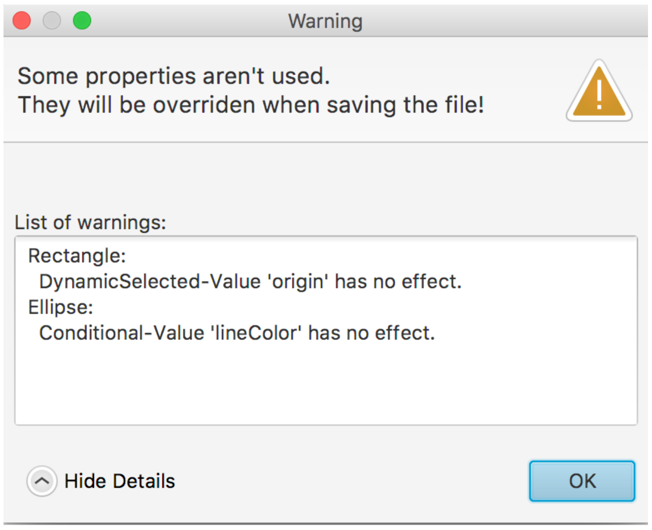
Figure 7. When holding *Shift*, MoVE will create perfect squares and circles and straight horizontal or vertical lines.

MoVE supports *undo and redo* using *Edit* → *Undo* / *Edit* → *Redo* or through the shortcuts *Ctrl+Z* and *Ctrl+Shift+Z*. It is also possible to *copy*, *paste* and *duplicate* selected shapes. Holding down *Shift* and selecting a shape selects multiple shapes.

5 Limitations

5.1 Annotations

MoVE supports every icon annotation except properties which are defined using a `if-clause` or a `DynamicSelect` statement, because the result of both statements is a dynamic value, which is only defined at runtime. These dynamic definitions do not fit into the scope of an editor for static vector graphic images. If MoVE finds properties, which are defined using this two statements, it warns the user that this properties will be overwritten by a static value after a save call (see Figure 8).


Warning

Some properties aren't used.
They will be overriden when saving the file!

List of warnings:

```
Rectangle:
  DynamicSelected-Value 'origin' has no effect.
Ellipse:
  Conditional-Value 'lineColor' has no effect.
```

Figure 8. A Warning is displayed when opening a Model whose icon annotation contains `DynamicSelect` and `if-clause` elements in MoVE.

5.2 Line Scaling

The Modelica language specification does not define the meaning of the `thickness` property of a line (Modelica Assoc., 2012). The most intuitive definition would be to assume that the thickness of a line is given in units of the coordinate system of the icon. Both Dymola and OMEdit, however, define the line thickness in terms of the coordinate system of the users screen, so that lines scale automatically when zoomed. At the moment, MoVE does not follow this behavior, because it is unintuitive and cannot be reproduced when the image is exported to SVG or Portable Network Graphics (PNG).

5.3 Placing Connectors

MoVE currently does not support placing connectors in the icon, because this would require parsing and altering connector definitions in the model. Loading and saving models with MoVE does not affect existing connector placements. MoVE is only a graphical editor for the main annotation statement of Modelica models and leaves the rest of the code untouched. Connector placement would add another layer of complexity to the tool that goes beyond its intended scope.

We are currently working on another tool in the MoTE family named Modelica Diagram Editor (MoDE), which will be used for the graphical composition of Modelica models and could also be used handle the connector placement (Hoppe, 2016).

5.4 Inherited Annotations

Modelica allows inheritance of icon annotations. The inherited annotations are currently not displayed in MoVE. This feature was postponed to future versions, because it would require parsing of several files and inspection of inheritance hierarchies.

6 Conclusions

In this paper we presented a new graphical editor for Modelica icon annotations. In contrast to other open source alternatives, the user interface of MoVE is specifically designed to make editing and creating vector graphic icons for Modelica models as easy and fast as creating a vector graphic image with tools such as Inkscape. MoVE builds on the modern platform-independent framework JavaFX. It has many convenience features such as grouping, snap to grid, move to foreground/background, rotation handles, and drawing perfect circles and squares as well as horizontal and vertical lines when holding *Shift*. It is also designed to work well with version control systems so that changes to individual elements can be captured. Except for dynamic elements, it supports every part of the icon definition in the Modelica language specification.

There are many possibilities for future improvement which can be drawn from the feature set of Inkscape such as component and node alignment or the combination and cutting of multiple paths. Ideally, these features could be brought to MoVE by a (partial) import of SVG graphics. This would allow to create icons in Inkscape and convert them into Modelica code so that they are used directly in Modelica models. For this, one would need to define a subset of SVG that is translatable to Modelica and somehow restrict the user in Inkscape to only use this subset. Futhermore, if MoVE should be able to place and display connectors of the model, the parser needs to be extended and additional parts of the model have to be altered.

We hope that this tool can enrich the open source ecosystem of Modelica and will enable more elaborate vector graphic icons for Modelica libraries. MoVE is part of a larger ensemble of tools called MoTE, which also features an integration of Modelica compiler features into a structured text editor.

The projects are open source and hosted on GitHub:
`https://github.com/thm-mote/`

References

Asghar, Syed Adeel et al. (2011). “An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation”. In: *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, pp. 739–747.

Dahlström, Erik et al. (2011). *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation REC-SVG11-20110816. W3C. URL: `https://www.w3.org/TR/SVG/`.

EPFL and Typesafe, Inc. (2016). *scala-parser-combinators*. GitHub Repository. URL: `https://github.com/scala/scala-parser-combinators` (visited on 12/09/2016).

Frey, Regis (2016). *The model, view, and controller (MVC) pattern relative to the user*. URL: `https://en.wikipedia.org/wiki/File:MVC-Process.svg` (visited on 12/07/2016).

Fritzson, Peter et al. (2005). “The OpenModelica Modeling, Simulation, and Development Environment”. In: *Proceedings of the 46th Scandinavian Conference on Simulation and Modeling (SIMS)*. Trondheim, Norway.

GitHub (2016). *Atom*. URL: `https://atom.io` (visited on 11/01/2016).

Hoppe, Marcel (2016). *Modelica Diagram Editor*. URL: `https://github.com/THM-MoTE/MoDE` (visited on 12/20/2016).

Inkscape (2016). *Inkscape — Draw Freely*. URL: `https://inkscape.org` (visited on 12/09/2016).

Justus, Nicola, Marcel Hoppe, and Christopher Schölzel (2017). *Modelica Tool Ensemble (MoTE)*. URL: `https://github.com/thm-mote` (visited on 03/28/2017).

McIlroy, M. D., E. N. Pinson, and B. A. Tague (1978). “Unix Time-Sharing System: Foreword”. In: *The Bell System Technical Journal* 57.6, pp. 1899–1904.

Modelica Association (2012). *Modelica - A Unified Object-Oriented Language for Systems Modeling*. Language Specification. Version 3.3.

Pop, Adrian Dan Iosif et al. (2006). “OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging”. In: *Proceedings of the 5th International Modelica Conference*. Vienna, Austria, pp. 459–465.

Reenskaug, Trygve (1979). *Thing-Model-View-Editor — An Example from a planningsystem*. technical note. Xerox PARC.