VisB: A Lightweight Tool to Visualize Formal Models with SVG Graphics

Michelle Werth and Michael Leuschel(✉)

Institut für Informatik, Universität Düsseldorf, Universitätsstr. 1,
40225 Düsseldorf, Germany
{michelle.werth,michael.leuschel}@hhu.de

Abstract. Visualization is important to present formal models to domain experts and to spot issues which are hard to formalise or have not been formalised yet. VisB is a visualization plugin for the ProB animator and model checker. VisB enables the user to create simple visualizations for formal models. An important design criterion was to re-use scalable vector graphics (SVG) generated by off-the-shelf graphic editors using a lightweight and easy-to-use annotation mechanism. The visualizations can be used to formal models in B, Event-B, Z, TLA+ and Alloy.

1 Introduction and Background

The animator and model checker ProB [3] supports both classical B and Event-B, as well as several other formalisms (Z, Alloy and TLA+) which are translated to B. Animation allows the user to experiment with a model, inspecting states, and interactively choose events or operations to execute. Animation is very useful to validate functional behaviour of a model, but also to uncover unexpected behaviour related to issues or requirements the modeller has not yet thought about. Here *graphical visualization* of the current state of a formal model is often essential so that a human can more quickly validate the behaviour or spot unexpected behaviour. To cite Bryan Cantrill:[1] *"The visual cortex is unparalleled at detecting patterns."* and *"The value of visualization is not merely providing answers but especially provoking new questions."*

There are several visualization tools for formal models such as PVSio-Web [7] for PVS, various co-simulation tools for VDM such as [6], and JEB [8], AnimB[2] or Brama [5] for Event-B. There have been several visualization based on ProB in the past, such as the animation functions of [4], BMotionStudio [2] or BMotionWeb [1]. The animation function feature is based on declaring a set of images and writing a B expression which generates a matrix of image numbers. It is still available in current versions of ProB, but it is hard to generate larger, visually appealing visualizations. BMotionStudio still exists within Rodin for Event-B, but is not available for other formalisms and it can be cumbersome to generate

[1] https://www.slideshare.net/bcantrill/visualizing-systems-with-statemaps.
[2] Available at http://wiki.event-b.org/index.php/AnimB.

© Springer Nature Switzerland AG 2020
A. Raschke et al. (Eds.): ABZ 2020, LNCS 12071, pp. 260–265, 2020.
https://doi.org/10.1007/978-3-030-48077-6_21

complex visualizations using its editor. BMotionWeb is based on web technologies, and allows to generate very refined visualizations. However, its learning curve is quite steep, and due to its heavy use of web technology and associated frameworks can no longer be maintained by the ProB team. This situation was the starting point for the development of the present VisB technology: it should be both easy to use and maintain, it should not be bound to an editor but allow a user to generate the images using off-the-shelf applications or even to re-use existing images.

2 VisB Principles and Architecture

The core idea of VisB is to use SVG files as the basis of the visualization. An SVG file is shown in Listing 1.1. Such files can be produced by most off-the-shelf editors and their textual XML representation can be programmatically generated.

```
1 <svg height="200" width="200">
2   <circle id="button" cx="100" cy="100" r="80"
3     stroke="black" stroke-width="3" fill="green" />
4 </svg>
```

Listing 1.1. Small SVG file (`button.svg`)

Moreover, SVG files can contain object identifiers (such as `button` for the circle in Listing 1.1) and it is possible (e.g. using jQuery and JavaScript) to load an SVG file and programmatically find objects from an identifier and set attributes of the found objects, and immediately display the changes. This is the basis of VisB, whose core is written in Java, JavaFX and JavaScript, and whose architecture is shown in Fig. 1.

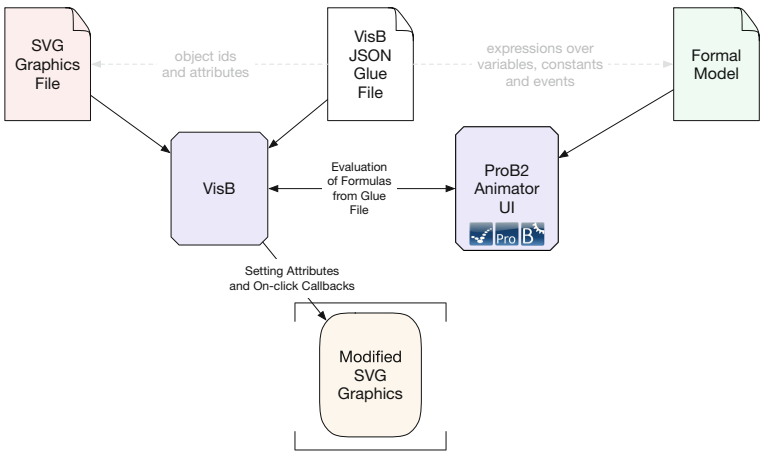

Fig. 1. VisB architecture

This architecture makes VisB easy to maintain because it allows the ProB team to mostly use Java and JavaFX in development, while cutting down the interactions with web languages (such as JavaScript) to a bare minimum.

The link to the formal model is provided by a lightweight glue file (see Listing 1.2), that provides two lists. VisB-items consist of SVG object identifiers, attributes, and expressions that provide the value the attribute should take depending on the state of the formal model. VisB-events link formal model events (aka operations or actions, depending on the formalism) to object identifiers. These events are executed when the object is clicked by the user.

The motivation was to keep the foundation of VisB simple, and not to require the user to learn any new programming language (e.g., JavaScript, Flash, ...). The user just has to know relevant expressions or variables from the formal model and corresponding object identifiers in the SVG graphics file. Moreover, VisB works for all of ProB's supported state-based formalisms (B, Event-B, Z, TLA+, Alloy) in an identical fashion.

```
{
  "svg":"button.svg",
  "items":[
    {
      "id":"button",
      "attr":"fill",
      "value":"IF button=TRUE THEN \"green\" ELSE \"red\" END"
    }
  ],
  "events":[
    {
      "id":"button",
      "event":"press_button",
       "predicates":[
         "status=TRUE"
       ]
    }
  ]
}
```

Listing 1.2. Minimal example for VisB file

3 VisB Examples

One of the simpler examples of a VisB file is shown in Listing 1.2. The corresponding machine contains a bool variable and an operation called press_button that changes the status of this variable. We use the fact that ProB allows IF-THEN-ELSE and LET for expressions (to simplify the syntax of the VisB file). In Listing 1.2 the fill attribute of the SVG object with the identifier "button" is changed to green whenever the button variable "button" in the corresponding machine is set to true. This is realized with the IF-THEN-ELSE expression in the value attribute. For the visualization this means that the circle's color is changed from red to green, when the operation press_button is executed. Thanks to the VisB-events, the user can also execute press_button directly by clicking on the SVG object with the identifier "button".

Fig. 2. Example of VisB visualization of lift model

The first visualisation created with VisB can be seen in Fig. 2. In the formal model, the state of a lift is represented by three variables: the current floor, an integer value between the ground floor and the top floor (`topf`), the current direction of the lift and a boolean variable indicating whether the door is open or not. In addition, the lift controller maintains the status of calling buttons inside the lift and on each floor. To cater for different number of floors, represented by the constant `topf`, we have made use of the SVG "visibility" attribute to hide unused floors (see right of Fig. 2). Note that each floor is represented by five graphical objects. To avoid having to hide each object of a given floor individually, we have grouped the objects for each floor together. VisB can then be used to hide or show all objects of a floor in one go, as shown in Listing 1.3.

```
... {
      "id":"gFloor_2",
      "attr":"visibility",
      "value":"IF topf>=2 THEN \"visible\" ELSE \"hidden\" END"
    }, ...
```

Listing 1.3. VisB item with grouping of SVG elements

```
... {
      "id":"lift",
      "attr":"y",
      "value":"IF cur_floor=2 THEN \"3.207\" ELSIF cur_floor=1 THEN \"76.
          974\" ELSIF cur_floor=0 THEN \"150.474\" ELSE \"224.574\" END"
    },
    {
      "id":"door_right",
      "attr":"y",
      "value":"IF cur_floor=2 THEN \"3.207\" ELSIF cur_floor=1 THEN \"76.
          974\" ELSIF cur_floor=0 THEN \"150.474\" ELSE \"224.574\" END"
    },
    {
      "id":"door_left",
      "attr":"y",
      "value":"IF cur_floor=2 THEN \"3.207\" ELSIF cur_floor=1 THEN \"76.
          974\" ELSIF cur_floor=0 THEN \"150.474\" ELSE \"224.574\" END"
    }, ...
```

Listing 1.4. Change "y" Attribute of the lift

Unfortunately, not all attributes can be changed for groups of SVG objects in this way. For example, the x and y coordinates cannot be changed for groups. Hence, to achieve the vertical movement of the lift cabin, we need three VisB items, each changing the attribute y to the same value (see Listing 1.4).

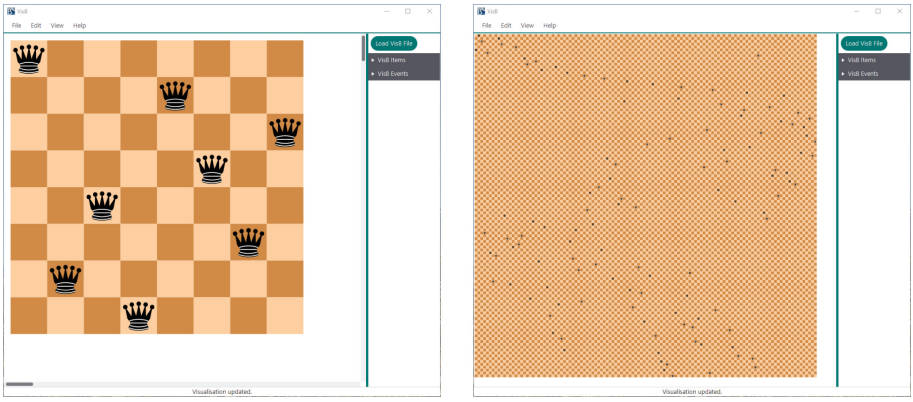

Fig. 3. Example of VisB visualization of N-queens problem

A solution to this drawback is to use embedded SVGs (i.e., nested SVG graphics embedded in the master SVG file) where it is possible to change the coordinates of those embedded SVGs. We have used this for the VisB visualization of the n-queens problem, partially shown in Listing 1.5, where the VisB items needed for the visualization of one given queen is shown. (Note, that the value of the second VisB item is not complete.) Additionally, each chess tile has a VisB event which triggers a B event to place a queen on that tile.

```
... {
      "id": "svgQueen1",
      "attr": "visibility",
      "value" : "IF 1:dom(queens) THEN \"visible\" ELSE \"hidden\" END"
    },
    {
      "id": "svgQueen1",
      "attr": "y",
      "value" :"IF  1|->2:queens THEN \"45\" ELSIF 1|->3:queens THEN \"90
          \" ELSIF 1|->4:queens THEN \"135\" [...] ELSIF 1|->20:queens
          THEN \"855\" ELSE \"0\" END"
    },
    {
      "id": "svgQueen1",
      "attr": "fill",
      "value" : "IF is_attacked(1) & 1:dom(queens) THEN \"red\" ELSE \"
          black\" END"
    }, ...
```

Listing 1.5. Example of VisB items for one queen in n-queens problem

For the n-queens problem, we programatically created the VisB file for the chess field and queens, which enabled us to visualize bigger chess fields (120×120), as you can see on the right in Fig. 3.

A more complex example can be found in our ABZ 2020 case study article in the present proceedings, where SVGs were received from coordinators of the case study and used to visualize various classical B and Event-B models.

In conclusion, thus far we seem to have met our goals of developing lightweight, easy-to-use and easy-to-maintain visualization technology, which nonetheless is flexible enough for creating simple academic visualizations up to complex, full-fledged industrial applications. VisB is available for download at:

https://www3.hhu.de/stups/prob/index.php/VisB

References

1. Ladenberger, L.: Rapid creation of interactive formal prototypes for validating safety-critical systems. Ph.D. thesis (2016)
2. Ladenberger, L., Bendisposto, J., Leuschel, M.: Visualising event-B models with B-motion studio. In: Alpuente, M., Cook, B., Joubert, C. (eds.) FMICS 2009. LNCS, vol. 5825, pp. 202–204. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04570-7_17
3. Leuschel, M., Butler, M.J.: ProB: an automated analysis toolset for the B method. STTT **10**(2), 185–203 (2008). https://doi.org/10.1007/s10009-007-0063-9
4. Leuschel, M., Samia, M., Bendisposto, J., Luo, L.: Easy graphical animation and formula viewing for teaching B. In: The B Method: From Research to Teaching, pp. 17–32 (2008)
5. Servat, T.: BRAMA: a new graphic animation tool for B models. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 274–276. Springer, Heidelberg (2006). https://doi.org/10.1007/11955757_28
6. Thule, C., Lausdahl, K., Gomes, C., Meisl, G., Larsen, P.G.: Maestro: the INTO-CPS co-simulation framework. Simul. Model. Pract. Theory **92**, 45–61 (2019)
7. Watson, N., Reeves, S., Masci, P.: Integrating user design and formal models within PVSio-web. In: Masci, P., Monahan, R., Prevosto, V. (eds.) Proceedings Workshop Formal Integrated Development Environment. EPTCS, vol. 284, pp. 95–104 (2018)
8. Yang, F., Jacquot, J., Souquières, J.: JeB: safe simulation of event-B models in JavaScript. In: Muenchaisri, P., Rothermel, G. (eds.) Proceedings APSEC 2013, pp. 571–576. IEEE Computer Society (2013)