BMotionWeb: A Tool for Rapid Creation of Formal Prototypes

Lukas Ladenberger(✉) and Michael Leuschel

Institut für Informatik, Universität Düsseldorf, Düsseldorf, Germany
{ladenberger,leuschel}@cs.uni-dusseldorf.de

Abstract. The application of formal methods to the development of reliable interactive systems usually involves a multidisciplinary team with different roles and expertises (e.g. formal engineers, user interface designers and domain experts). While formal engineers provide the necessary expertise in formal methods, other roles may not be well versed in formal methods, such as user interface engineers or domain experts; consequently barriers may arise while working in a multidisciplinary team. For instance, communication problems and challenges in the rigorous use of formal method tools. Tools like BMotion Studio may reduce these barriers by creating visualizations of formal specifications, however, lacks features needed for the analysis of interactive systems. In this paper, we present a novel graphical environment that continues the ideas of BMotion Studio called *BMotionWeb* to provide support for the rapid creation of *formal prototypes*. A formal prototype links a mockup of a graphical user interface or device to an animated formal specification with the aim of providing lightweight formal validation of interactive systems. In order to demonstrate the application of BMotionWeb, we provide two case studies: a formal prototype of a simple phonebook software and a cruise control device.

Keywords: Formal methods · Animation · Visualization · Rapid prototyping · Validation · Mockup · Interactive user interface

1 Introduction

Formal methods are often applied in the field of safety-critical systems. They allow the specification and analysis of systems based on mathematical techniques with the main goal to ensure *reliability* and *robustness* of the system. The application of formal methods for the development of safety-critical systems usually involves a multidisciplinary team with different roles and expertise (e.g. formal engineers, domain experts and end users). Nowadays, however, safety-critical systems, such as, medical devices, airplane cockpits, or railway- and nuclear plant control systems, typically include interactive user interfaces (UI). Thus, the development of safety-critical systems also requires to properly account for user's cognition and to ensure the *usability* of the system. This task is typically performed by UI engineers. However, UI engineers, domain experts and end

R. De Nicola and E. Kühn (Eds.): SEFM 2016, LNCS 9763, pp. 403–417, 2016.
DOI: 10.1007/978-3-319-41591-8_27

users are rarely trained in formal methods. Consequently barriers could arise while working in a multidisciplinary team which can compromise the success of the project. For example, it can be challenging to find a common language for discussing potential system and design issues. The use of a formal specification as a basis for discussion requires significant knowledge about the mathematical notation of the respective formal method which non-formal method experts typically not have. Moreover, formal method tools may become inaccessible to non-formal method experts. On the other hand, formal engineers typically have no experience in common UI engineering techniques. As a consequence, there is a great demand for tools that can significantly reduce these barriers applying formal methods for developing interactive systems.

One tool that faces these barriers is BMotion Studio [9], a graphical environment for creating visualizations of formal specifications. While BMotion Studio provides a very convenient and fast approach to create simple visualizations of formal specifications, it makes it difficult to use and apply it when validating *interactive* systems. A reason for this is the limited reuse of existing components (e.g. interactive graphical elements and advanced UI techniques) and the lack of validation features needed for the analysis of UIs, such as logging of user interactions and deployment of visualizations.

In this paper we present a novel graphical environment called *BMotionWeb* that builds on the ideas of BMotion Studio to provide support for the lightweight validation of interactive systems by combining techniques known from the formal- and UI-engineering discipline: *animation* [8] and *mockup*. Animation allows the user to inspect the behavior of a formal specification by "executing it". Mockup is a common technique in the field of UI design to describe a model of a device or software UI. Combining an animation tool and a mockup, the formal specification becomes a "tool" in a real sense: it serves as a *formal prototype* that binds the intended functionality of the system to an interactive UI or device. BMotionWeb contributes new features for the rapid creation and lightweight validation of formal prototypes: (1) a visual editor that allows UI engineers to create mockups of a system UI or device; (2) the necessary technique to link a mockup with an animated formal specification; (3) and UI validation features such as logging of user interactions, visualizing the behavior of UI elements and deployment of formal prototypes. Throughout the paper, we demonstrate the application of BMotionWeb based on two case studies: a formal prototype of a simple phonebook software and a cruise control system.

The paper is organised as follows: Sect. 2 describes the architecture of BMotionWeb. In Sect. 3 we describe our approach for creating formal prototypes using BMotionWeb based on two case-studies. In Sect. 4 we demonstrate the application of a formal prototype for the lightweight validation of interactive systems. Finally, we compare our work with related work in Sect. 5 and conclude in Sect. 6. For more information and resources we refer the reader to our project website: http://stups.hhu.de/ProB/FormalPrototyping. The website contains the case-studies (specifications and formal prototypes) and interactive online-versions of the formal prototypes.

2 BMotionWeb

BMotionWeb builds on the ideas of BMotion Studio [9] to provide support for the rapid creation of formal prototypes. BMotion Studio is a graphical environment for creating domain specific visualizations of formal specifications. It is based on Eclipse and GEF (Graphical Editing Framework) [19] and comes as a separate plug-in for Rodin [2] with support for the Event-B specification language [1]. BMotionWeb is a complete rewrite based on web technologies. Figure 1 gives an overview of the architecture of BMotionWeb. The overall architecture is subdivided into a client front-end and a server back-end, where WebSockets [6] are used to realise the communication between client and server. The client and server can be run in a standalone application based on electron[1], a framework for building cross-platform desktop applications using JavaScript or as separate processes (either on the same machine or on different machines).

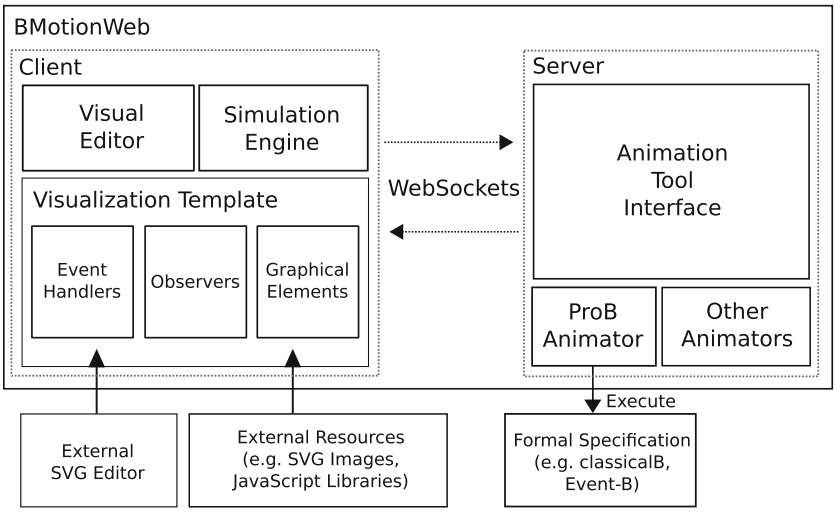


Fig. 1. Architecture overview of BMotionWeb for ProB

In the following subsections we describe the components of the client and server in more detail.

2.1 Server Back-End

The server is entirely written in Java and provides an *animation tool interface* capable of integrating external animation tools with BMotionWeb. Currently, BMotionWeb integrates the ProB animator [11] that supports among others classical B [20] and Event-B [1]. The aim of the animation tool interface is to provide

[1] http://electron.atom.io/.

communication between the client front-end and the respective animation tool via WebSockets. For instance, to obtain information about the animated formal specification (e.g. values of state variables or results of evaluating formulas) and to trigger transitions (e.g. executing operations in classical B or events in Event-B).

2.2 Client Front-End

The client front-end consists of a *visual editor* and a *simulation engine* and is entirely written in JavaScript. The aim of the visual editor is to facilitate the creation and editing of *visualization templates*, whereas the simulation engine is responsible for executing visualization templates.

Visualization Template. At the heart of a formal prototype one finds a *visualization template*. A visualization template uses web-technologies to describe the mockup and its bindings to the animated formal specification of a formal prototype. In particular, it is composed of *graphical elements*, *observers* and *event handlers*:

- A *graphical element* is based on SVG (Scalable Vector Graphics) [22] and HTML [23], two markup languages which provide widgets like shapes, images, labels, tables and lists.
- An *observer* observes a specific part of the animated formal specification during the simulation. BMotionWeb provides various observers. For instance, a *formula observer* that binds a formula (e.g. an expression or a variable) to a graphical element and allows the tool to compute a visualization for any given state of the animated formal specification by changing the properties of the graphical element (e.g. the colour or position) according to the evaluation of the formula in the respective state.
- Finally, an *event handler* wires an interactive action to a graphical element. As an example, BMotionWeb provides an *execute event handler* that binds a classical B operation or an Event-B event to a graphical element and executes the operation or event respectively when the user clicks on the graphical element.

BMotionWeb also provides a JavaScript API for scripting observers and event handlers. Indeed, the use of web-technologies and especially the possibility to reuse existing resources like SVG images and external JavaScript libraries enables users to create formal prototypes for a wider range of systems.

Visual Editor. Figure 2 shows a snapshot of the visual editor. The editor consists of a palette for creating graphical elements, like shapes, labels, images and input fields and a view for managing the properties, observers and event handlers of graphical elements. Graphical elements can be added to a canvas which provides features known from modern graphical editors like drag and drop, undo/redo, copy/paste and zooming.

Simulation Engine. The simulation engine allows users to interact with the formal prototype and to explore its behavior. For this purpose, it renders a visualization template and manages the communication between the mockup and the animated formal specification. In particular, it sends requests from an observer (e.g. evaluating a formula) to the animation tool via the animation tool interface on the server side and forwards the returned results of the animation tool back to the observer. In addition, it triggers state changes in the animated formal specification based on user actions like clicking on an graphical element that wires an execute event handler.

Apart from the formal prototype view, several additional views for analysing a formal prototype have been made available from the ProB animator. For instance, a view that shows the values of variables and constants for the current and previous state of the animated formal specification and a view that lists all enabled and disabled transitions based on the current state of the animated formal specification.

3 Creating Formal Prototypes

In the following two subsections we give more details about the features of BMotionWeb based on two case studies: a classical B specification of a phonebook software (Sect. 3.1) and an Event-B specification of a cruise control device (Sect. 3.2). Both case studies are supported by code examples in which observers and event handlers are described using the BMotionWeb JavaScript API.

3.1 Formal Prototype of a Phonebook Software

In this section we demonstrate our approach based on a classical B specification of a phonebook software. The phonebook allows users to manage persons and telephone numbers and provides the following functionalities: adding and deleting persons with an associated number, searching for numbers and activating or deactivating persons. Moreover, the user can lock the phonebook which results in deactivating the phonebook, i.e. the user can not add new entries. The aim of this case-study is to exemplify the creation of software UI mockups, as well as to demonstrate how the connection between a software UI mockup and an animated formal specification can be established with BMotionWeb.

Mockup Software User Interfaces. Figure 2 shows a snapshot of the visual editor of BMotionWeb while creating the mockup of the input form of the phonebook software. As can be seen in Fig. 2, the mockup of the input form is composed of different graphical elements like input fields, buttons, a checkbox, shapes and labels. The UI engineer can change the properties of the selected graphical element by means of the properties view located on the right side of the editor. For instance, in Fig. 2, the phonenumber input field is selected. Thus, related properties like a property for defining the placeholder or the ID of the input field are made available to the UI engineer. Further, the input field element

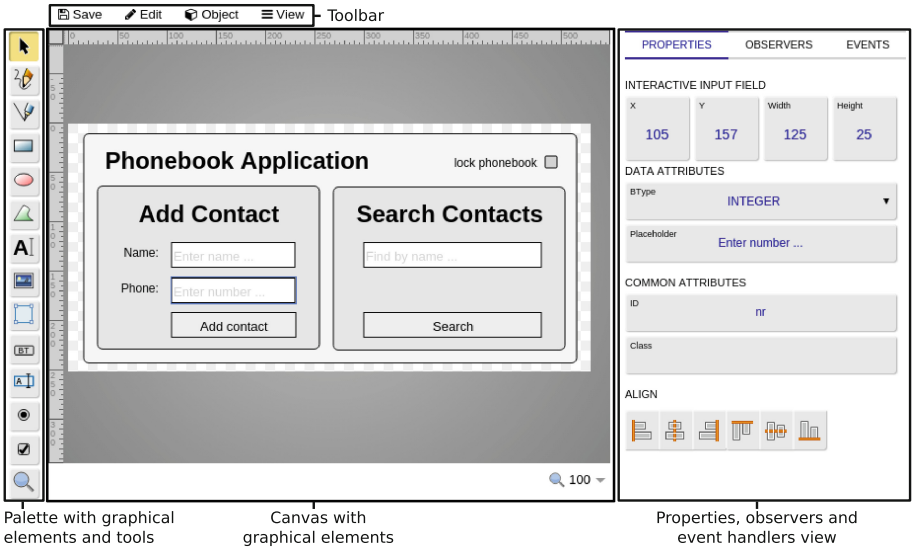


Fig. 2. UI mockup of the phonebook software in visual editor of BMotionWeb

provides a property that defines a classical B set like *INTEGER* or *NAT* or a custom set which comes from the animated formal specification. Defining a set causes a validation of the entered input, i.e. the input field checks whether the entered input is an element of the set or not.

```
1 bms.executeEvent({
2  selector:"#btAdd",
3  name:"add",
4  predicate: function(ui) {
5   var name = ui.find("#name");
6   var nr = ui.find("#nr");
7   return"name="+ name.val() +
8          "& nr="+ nr.val();
9 }});
```

Listing 1. Execute event handler for "Add contact" button (JavaScript)

```
1 add(name, nr) =
2  PRE
3   name ∈ STRING ∧
4   name ∉ dom(db) ∧
5   nr ∈ NATURAL ∧
6   lock = FALSE
7  THEN
8   db := db ∪ {name ↦ nr}
9 END;
```

Listing 2. Phonebook *add* Operation (classical B)

The values of the interactive graphical elements (e.g. the entered input of an input field or the value of a checked checkbox) can be used for defining event handlers. For instance, the "Add contact" button shown in Fig. 3 is wired to the event handler defined in Listing 1. Lines 1 and 2 state that we register a new *execute event handler* on the graphical element that matches the selector

"#btAdd" (The prefix "#" is used to match a graphical element by its ID.[2]), i.e. the graphical element that represents the "Add contact" button. Line 3 states that the event handler should execute the *add* operation of the classical B specification of the phonebook software (see Listing 2). In lines 4 to 9 we define a JavaScript function that returns a predicate determining the parameters of the *add* operation. The returned predicate (lines 7 to 8) is composed of the values of the name and number input fields (line 5 and 6). Figure 3 shows a snapshot of the formal prototype of the phonebook software where the user hovers the "Add contact" button. We have defined the event handler for the "Search" button in a similar fashion.

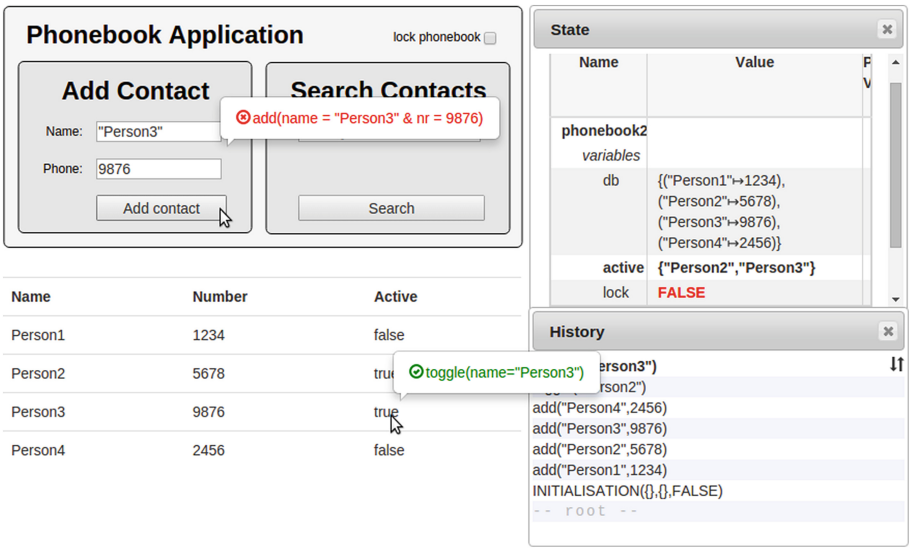


Fig. 3. Formal prototype of phonebook software

Listing 3 shows the *formula observer* for observing the *lock* variable of the phonebook specification (see Listing 4). Line 1 and 2 state that we register a new formula observer on the name and number input fields (#name and #nr) and on the "Add contact" button (#btAdd). Line 3 states that the observer should observe the variable *lock* during the simulation. In lines 4 to 6 we define a trigger function that is called whenever a state change has occurred. The reference to the matched graphical element (e) and the state values of the observed formulas (v) are passed as arguments to the trigger function. In line 6 we define the trigger action: the observer sets the *disabled* property of the graphical elements to the value of the lock variable (v[0]). Since we have set the *translate* property of the

[2] BMotionWeb makes use of the jQuery selector syntax to link event handlers and observers to graphical elements. For more information about the jQuery selector API we refer the reader to http://api.jquery.com/category/selectors/.

formula observer to *true* (see line 4) the value is automatically translated into a JavaScript object.

```
1 bms.observe("formula", {
2  selector:"#name,#nr,#btAdd",
3  formulas: ["lock"],
4  translate: true,
5  trigger: function(e, v) {
6   e.prop("disabled", v[0]);
7  }
8 });
```

Listing 3. Formula observer for lock variable (JavaScript)

```
1 VARIABLES db, active, lock
2 INVARIANT
3     lock : BOOL &
4     db : STRING +-> NATURAL &
5     active : POW(STRING)
6 INITIALISATION
7     db := {} || active := {} ||
8     lock := FALSE
```

Listing 4. Variables of phonebook specification (classical B)

Mockup Dynamic Data-Structures. In formal specification languages like classical B, the software is often modeled with data-structures like sets and relations. These data-structures typically contain a *dynamic* number of elements. For instance, the database of the phonebook is modeled as a relation between persons and numbers, where the size of the database increases or decreases, whenever the user adds or deletes a phonebook entry. In this section, we demonstrate the use of an external JavaScript library to connect HTML elements like tables and lists with dynamic data-structures like sets or relations.

In order to establish a connection between HTML elements and an animated formal specification we make use of the JavaScript MVC (Model View Controller) framework AngularJS [7].[3] AngularJS provides *controllers* and *directives*. A controller defines the data and behavior of HTML templates, whereas a directive can attach a specified behavior to an existing HTML element. As an example, consider the controller *pCtrl* in Listing 5. In lines 4 to 10 we register a new *formula observer* which observers the two state variables *db* and *active* (see Listing 4). The values of the variables are assigned to the *scope* of the controller and updated whenever a state change occurred in the animated formal specification (line 8 and 9). Moreover, in lines 12 to 14 we assign a helper function *isActive* to the scope that takes a person as its parameter and returns true whenever the person is in the *active* set. Otherwise it returns false.

A scope can be made available to an HTML template using the *ngController* directive as demonstrated in line 1 in Listing 6. Once the scope has been attached to the template, the values of the two variables *db* and *active* can be used within the template. For instance, in line 7 we assign a *ngRepeat* directive which creates a table row (lines 7 to 15) once per element from the *db* relation. Each row gets

[3] We choose AngularJS because BMotionWeb is also based on AngularJS, however, we could also use other JavaScript MVC libraries as well.

its own scope, where the current element (*el*) of the *db* relation is set to the row's scope. Thus, we can access the name ({{*el[0]*}}), the number ({{*el[1]*}}) and the status of each element ({{*isActive(el[0]*}})) and show them in the respective columns of the row (lines 8 to 14). In addition, we assign to each status column an *executeEvent* directive which creates a new execute event handler with *toggle* as the operation's name and *name="{{el[0]}}"* as the operation's predicate. The lower left side of Fig. 3 demonstrates the HTML table during the simulation of the phonebook formal prototype.

```
1  angular.module('phone', [])
2  .controller('pCtrl', function() {
3
4   bms.observe("formula", {
5    formulas: ["db","active],
6    translate: true,
7    trigger: function(v) {
8     $scope.db = v[0];
9     $scope.act = v[1];
10  }});
11
12  $scope.isActive = function(n) {
13   return $scope.act.indexOf(n) > -1;
14  }
15
16 });
```

Listing 5. Controller for database view (JavaScript)

```
1  <table ng-controller="pCtrl">
2   <tr>
3    <th>Name</th>
4    <th>Number</th>
5    <th>Active</th>
6   </tr>
7   <tr ng-repeat="el in db">
8    <td>{{el[0]}}</td>
9    <td>{{el[1]}}</td>
10   <td execute-event
11    name="toggle"
12    predicate='name="{{el[0]}}"'>
13    {{isActive(el[0])}}
14   </td>
15  </tr>
16 </table>
```

Listing 6. Template for database view (HTML)

3.2 Formal Prototype of a Cruise Control Device

A common way to develop mockups is to create graphical sketches using the classical paper-and-pencil approach. In this section we demonstrate the application of BMotionWeb for reusing such graphical sketches for the creation of formal prototypes. For this purpose, we use an Event-B specification of a cruise control system (CCS) and a graphical sketch of a car cockpit including an exemplar of a device (see Fig. 4) as a case-study. A CCS is an automotive system implemented in software which automatically controls the speed of a car. The CCS device provides buttons to switch the CCS system on/off, to set the current speed of the car as the target speed of the CCS system and to increase and decrease the target speed. In addition, the speedometer provides information about the state of the CCS system and about the target speed in dependence on the current car speed.

Using the visual editor of BMotionWeb, UI engineers can select a picture (e.g. a graphical sketch or a photograph) of a device or a UI as a starting point

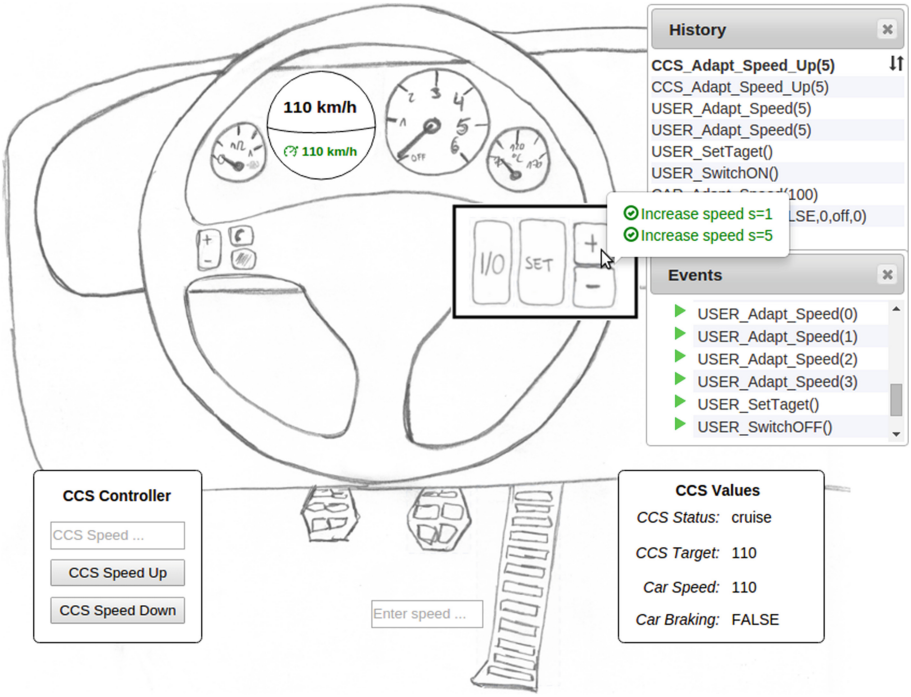


Fig. 4. Formal prototype of cruise control device

for creating a formal prototype. Once a picture is selected it can be extended with additional graphical elements. For this purpose, BMotionWeb contributes an *interactive area* graphical element which can be placed over the picture. An interactive area allows UI engineers to bind an execute event handler or a value of a variable to a specific area of a picture. As an example, Fig. 4 shows a snapshot of the formal prototype of the CCS device based on a graphical sketch, where the user hovers the "+" button on the graphical sketch. An interactive area overlays the button and binds an execute event handler (see Listing 7) that wires the event *USER_Adapt_Speed* (see Listing 8), one with with the predicate *s=1* and one with the predicate *s=2*. In addition, the speedometer of the graphical sketch binds the two variables *car_speed* (the current speed of the car) and *ccs_target* (the target speed of the CCS system).

4 Validating Formal Prototypes

The use of a formal prototype for validation can take place at different stages of the development process. On the one hand, a formal prototype can be created of existing specifications (e.g. when the system is already implemented) as demonstrated in Sect. 3. On the other hand, a formal prototype can be maintained and used for validation at earlier stages of the development process, e.g. as a

```
bms.executeEvent({
 selector:"#btSpeedUp",
 events: [{ name: "USER_Adapt_Speed",
            predicate: "s=1" },
          { name: "USER_Adapt_Speed",
            predicate: "s=5" }],
 label: function(evt) {
  return "Increase speed" +
           evt.predicate";
}});
```

Listing 7. Execute event handler "increase target speed" (JavaScript)

```
event USER_Adapt_Speed
 any
  s
 where
  @g1 ccs_status = cruise
  @g2 s ∈ ℤ
  @g3 ccs_target + s ≥ 0
 then
  @a1 ccs_target := ccs_target+s
end
```

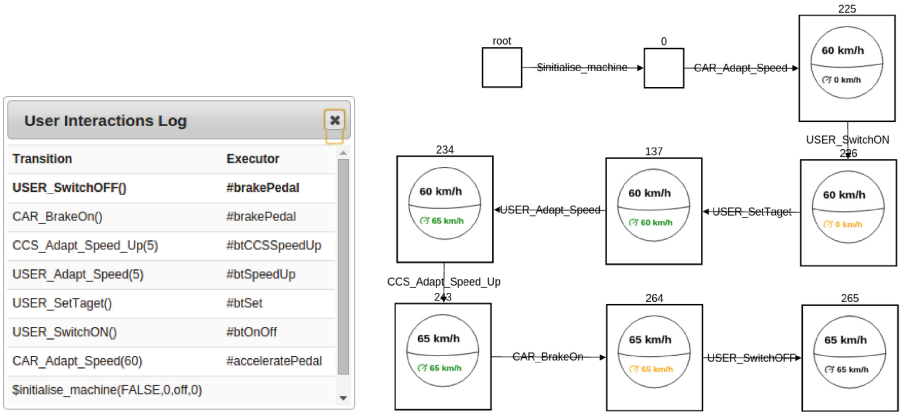
Listing 8. CCS "increase target speed" event (Event-B)

by-product of the developed formal specification. In the following we describe different application examples of a formal prototype to support the validation process of interactive systems.

Formal Prototype as a Base for Communication. A common understanding of the system in a multidisciplinary team is crucial for the success of the project. Indeed, it is important for the formal engineer to get feedback from the UI engineer for further development of the formal specification. On the other hand, the UI engineer needs to check whether his expectations are met in the formal specification. However, rarely all persons involved in a project are versed in formal methods. The application of a formal prototype can overcome this challenge: it allows UI engineers to validate the behavior of the system and the system's UI or device by interacting with a realistic prototype rather than by examining a substantial amount of mathematical formulas. Moreover, a formal prototype can be used to demonstrate features of the system's UI or device and to discuss validation results (e.g. system and design issues).

Deployment of Formal Prototypes. Running the client and server components of BMotionWeb as web-server processes allows the deployment of formal prototypes online. This can be in particularly useful for accessing a formal prototype from other devices, such as tablets and mobile phones and for sharing a formal prototype with other stakeholders (e.g. during an online project meeting). For example, a UI engineer could demonstrate a specific scenario of the system's UI or device by interacting with the formal prototype. All updates made on the formal prototype are automatically reflected to other stakeholders that have also opened the same formal prototype.

Logging and Visualizing User Interactions. BMotionWeb contributes new features for logging and visualizing user interactions. Figure 5 demonstrates the *user interactions log* view that lists the so far executed transitions of a simulation, where each transition shows the id of the graphical element that triggered the



(a) User Interactions Log (b) Transition Diagram

Fig. 5. User interactions log and transition diagram of a CCS device scenario

transition. The user can toggle between the states of a simulation by clicking on an entry of the list.

Based on the user interactions log, a *transition diagram* visualizing the behavior of graphical elements for a specific scenario can be generated. In order to generate a transition diagram, we apply the following approach: (1) the user selects the graphical elements for the transition diagram from the formal prototype; (2) we determine the observers of the selected graphical elements and derive their formulas (which can be simple variables or expressions) that are required to draw the state of the selected graphical elements; (3) for each state of the user interactions log we compute the representation of the selected graphical elements according to the value of the formulas in the respective state. As an example, Fig. 5(b) illustrates the behavior of the graphical element that represents the speedometer of the CCS device. Each rectangle represents a state, whereas a directed edge between two rectangles represents a transition labelled with the associated transition name.

Other Validation Features. BMotionWeb integrates other features that may support the validation of formal prototypes. For instance, a *projection diagram* can be generated for individual graphical elements following the approach presented in [10]. The objective of a projection diagram is to support human analysis of the system by highlighting relevant aspects of it, while hiding information that is not relevant from the diagram based on the state space of the animated formal specification.

Several additional views for analysing a formal prototype are made available from the ProB animator, e.g. a model-checking view with the goal to automatically check properties of the system like deadlock freeness and invariant preservation.

5 Related Work

Brama [21], AnimB [13] and the ProB based tool presented in [3] allow developers creating UIs of Event-B specifications using graphical elements provided by Flash[4] (e.g. labels and images, as well as interactive graphical elements like input fields and radio buttons). In Brama and AnimB the mapping between an animated formal specification and a UI is realised with the built-in programming language *ActionScript*, whereas the ProB based tool requires the developer to write Java as the gluing code. Although the use of Flash seems to be a promising tool for creating rich interactive prototypes, it involves some disadvantages for the developer: Since Flash is a self-contained tool the developer requires skills for using it. Furthermore, the developer requires additional programming skills for writing the gluing code that maps a state to the UI (e.g. ActionScript for Brama and AnimB and Java for [3]).

Other tools which provide comparable concepts to that of BMotionWeb are: JeB [24], an animator that provides an HTML5 canvas allowing the creation of UIs for Event-B specifications and WebASM [25], a web-based tool that brings the CoreASM [5] animator into the web and allows developing UIs for ASM [4] specifications. Similar to the previously mentioned Flash based tools, JeB and WebASM requires programmings skills to map a state to the UI. Moreover, the Flash- and web-based tools lack features for the validation of UIs (e.g. logging of user interactions).

PVSio-web [17] is a web environment including the animation engine PVSio [14] and a visual editor for creating interactive prototypes for the PVS formalism [18]. The visual editor of PVSio-web allows users to choose an image that represents the layout of the UI and to place interactive areas over it (e.g. areas to execute events and to display variable values). In contrast to PVSio-web, BMotionWeb allows users to compose a UI prototype of different graphical elements (e.g. images, labels and shapes). Moreover, in BMotionWeb variable values can also be mapped to the different attributes of the graphical elements using the observer concept of BMotionWeb.

The authors in [16] present three alternatives to extend the animation capabilities of VDMPad [15]. Especially the "Lively Walk-Through" approach can be compared to our work. It combines VDM animation with a UI to provide lightweight validation features for VDM specifications. For this, the approach introduces its own language called "LiveTalk" to wire interactive actions (e.g. executing an VDM operation) to UI widgets. In contrast to the "Lively Walk-Through" approach, in BMotionWeb at best no additional languages are required to create interactive actions (e.g. executing classicalB operations or Event-B events). Another difference between BMotionWeb and "Lively Walk-Through" is that the latter lacks of linking state variables to the UI.

6 Conclusion

In this paper we have presented BMotionWeb, a novel graphical environment for the *rapid* creation of *formal prototypes*. A formal prototype combines a mockup

[4] http://www.adobe.com/devnet/flash.html.

of a UI or device with the intended functionality of an animated formal specification. Thus, we eliminate (at least to a large degree) the need to implement and maintain the functional part of a prototype, e.g. using additional programming languages. BMotionWeb provides a visual editor that facilitates the creation of formal prototypes and different features for the lightweight validation of interactive systems, such as logging of user interactions, visualizing the behavior of graphical elements and deployment of formal prototypes. Since a formal prototype is based on web-technologies, external web-resources like third party JavaScript libraries and SVG images can be used to create formal prototypes for a wider range of interactive systems. We have demonstrated the application of BMotionWeb based on two case studies: a formal prototype of a classical B specification of a simple phonebook software and an Event-B specification of a cruise controller. The case studies (specifications and formal prototypes) and interactive online-versions of the formal prototypes have been made available at our project website.[5]

BMotionWeb can be used at any stage of the development process to support the validation of interactive systems. In summary, we believe that BMotionWeb can be useful for the following purposes: (i) to get a common understanding of the system between formal method and non-formal method experts; (ii) to demonstrate features of the system's UI or device; (iii) to discuss validation results (e.g. system and design issues).

Future Work. In future, we plan to apply BMotionWeb to create formal prototypes of other case studies, especially case studies coming from industrial projects. In this context, our aim is to integrate other animation tools with BMotionWeb to address a wider range of interactive systems. First experiments towards supporting the CoreASM animator [5] have already been made. We also plan to develop more features for the lightweight validation of interactive systems, such as A/B testing to compare two variants of a system UI or device. Finally, we plan to consider other techniques and tools to support the validation process, like the ProB constraint solver [12], e.g. to intelligently disable/enable graphical elements.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. Softw. Tools Technol. Transfer **12**(6), 447–466 (2010)
3. Bendisposto, J., Leuschel, M.: A generic flash-based animation engine for ProB. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 266–269. Springer, Heidelberg (2006)
4. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-level System Design and Analysis. Springer Science & Business Media, New York (2012)

[5] http://www.stups.hhu.de/ProB/FormalPrototyping.

5. Farahbod, R., Gervasi, V., Glässer, U.: CoreASM: an extensible ASM execution engine. Fundamenta Informaticae **77**(1–2), 71–103 (2007)
6. Fette, I., Melnikov, A.: The websocket protocol (2011)
7. Green, B., Seshadri, S.: AngularJS. O'Reilly Media Inc., California (2013)
8. Hazel, D., Strooper, P., Traynor, O.: Requirements engineering and verification using specification animation. In: Automated Software Engineering, p. 302 (1998)
9. 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)
10. Ladenberger, L., Leuschel, M.: Mastering the visualization of larger state spaces with projection diagrams. In: Butler, M., Conchon, S., Zaïdi, F. (eds.) Formal Methods and Software Engineering. LNCS, vol. 9407, pp. 153–169. Springer, Switzerland (2015)
11. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. Softw. Tools Technol. Transfer (STTT) **10**(2), 185–203 (2008)
12. Leuschel, M., Bendisposto, J., Dobrikov, I., Krings, S., Plagge, D.: From animation to data validation: the ProB constraint solver 10 years on. In: Boulanger, J.-L. (ed.) Formal Methods Applied to Complex Systems: Implementation of the B Method, Chap. 14, pp. 427–446. Wiley ISTE, Hoboken (2014)
13. Mtayer, C.: AnimB Homepage. http://www.animb.org/. Accessed 12 Jan 2015
14. Munoz, C.A.: Pvsio reference manual. National Institute of Aerospace (NIA), Formal Methods Group, 100 (2005)
15. Oda, T., Araki, K.: Overview of VDMPad: an interactive tool for formal specification with vdm. In: Proceedings of International Conference on Advanced Software Engineering and Information Systems (ICASEIS) (2013)
16. Oda, T., Yamamoto, Y., Nakakoji, K., Araki, K., Larsen, P.G.: VDM animation for a wider range of stakeholders. Grace Technical reports, p. 18 (2015)
17. Oladimeji, P., Masci, P., Curzon, P., Thimbleby, H.: PVSio-web: a tool for rapid prototyping device user interfaces in PVS. In: FMIS2013 (2013)
18. Owre, S., Rushby, J.M., Shankar, N.: PVS: a prototype verification system. In: Kapur, D. (ed.) Automated Deduction–CADE-11. LNCS (LNAI), vol. 607, pp. 748–752. Springer, Heidelberg (1992)
19. Rubel, D., Wren, J., Clayberg, E.: The Eclipse Graphical Editing Framework (GEF). Addison-Wesley Professional, Boston (2011)
20. Schneider, S.: The B-Method: An Introduction. Palgrave Oxford, Oxford (2001)
21. 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)
22. W3C SVG Working Group. Scalable Vector Graphics (SVG) 1.1 (2nd edn.), August 2011. http://www.w3.org/TR/SVG11/
23. W3C SVG Working Group. HTML5, A vocabulary and associated APIs for HTML and XHTML, October 2014. http://www.w3.org/TR/html5/
24. Yang, F.: A Simulation Framework for the Validation of Event-B Specifications. Ph.D. thesis, Université de Lorraine (2013)
25. Zenzaro, S., Gervasi, V., Soldani, J.: WebASM: an abstract state machine execution environment for the web. In: Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. LNCS, vol. 8477, pp. 216–221. Springer, Heidelberg (2014)