Prototyping with the IVY workbench: Bridging Formal Methods and User-Centred Design*

Rafael Braga da Costa and José Creissac Campos[0000−0001−9163−580X]

University of Minho & INESC TEC, Braga, Portugal
rafael.b.costa@inesctec.pt, jose.campos@di.uminho.pt

Abstract. The IVY workbench is a model-based tool for the formal modelling and verification of interactive systems. The tool uses model checking to carry out the verification step. The goal is not to replace, but to complement more exploratory and iterative user-centred design approaches. However, the need for formal and rigorous modelling and reasoning raises challenges for the integration of both approaches. This paper presents a new plugin that aims to provide support for the integration of the formal methods based analysis supported by the tool, with user-centred design. The plugin is described, and an initial validation of its functionalities presented.

Keywords: Formal methods, user-centred design, prototyping

1 Introduction

The design of safety critical interactive systems needs to provide assurance regarding the quality and safety of the interaction. The exploratory nature of traditional User-Centred Design (UCD) approaches (cf. [2, 9]) does not necessarily guarantee a depth of analysis that provides this assurance. Formal (mathematically rigorous) modelling and verification are able to provide a thorough and repeatable analysis, but interactive systems pose particular challenges for their application (see [13, 11] for discussions about these challenges).

A number of tools has been proposed that aim to address this from different angles (see [6, 3], for reviews). Of particular interest here is the IVY workbench tool [7]. The focus of the tool's development has been to ease the application of formal modelling and verification to interactive systems design. Experience with using the tool (cf. [14, 5]) has highlighted the need to find solutions to communicate the model (for validation) and the verification results (for interpretation) to domain and human factors experts. Prototyping seems a promising approach to bridge this gap.

The contribution of this paper is thus an approach for the integration of prototyping support into the IVY workbench. The goal is to enable the creation

* *Author's version of the paper published in Human-Computer Interaction, vol. 14143 of Lecture Notes in Computer Science, pages 504-513. Springer. 2023. The final version is available at Springer via: http://dx.doi.org/10.1007/978-3-031-42283-6_27.*

of prototypes by linking early mock-ups of the user interface with their models developed in IVY. The mock-ups provide a concrete illustration of the visual appearance of the interface, while the model provides its behaviour. Together they enable the simulation of the system behaviour.

2 Background

Mock-up editors focus on the physical design of the system, allowing users and designers to identify potential problems with the interface or generating ideas for new functionalities [1]. Adobe XD and Figma provide a set of built-in components, representing different user interface (UI) controls, which designers can use to build mock-ups. More advanced features such as components with multiple states, which support a more economical modelling of how the appearance of the mock-up changes with user interactions, can also be found in some of these tools. In general, however, mock-up editors have limited support for prototyping UI behaviour, as this implies some notion of the underlying system's state. Traditionally, these tools allow designers to define navigation rules, but without any associated control logic. These rules capture the navigation from mock-up to mock-up in response to user events, such as mouse clicks, but the lack of control logic means they are static and unable to express complex behaviours.

Model-based user interface analysis tools support early detection of UI design problems [6, 3]. However, formal modelling does not integrate well with the exploratory nature of UCD approaches. Formal modelling is also outside the typical toolbox of a UCD practitioner. This creates barriers to adoption, even in situations where these tools might be valuable. The IVY Workbench supports the application of formal methods to interactive systems' design, from writing the models to interpreting the results of verification (performed using the NuSMV model checker [10]). IVY is designed for simplicity, aiming to provide modelling and analysis tools that are easily usable by non-experts and to communicate results effectively within an interdisciplinary team. Models are expressed in the MAL interactors language [8]. A detailed description of the language is outside this paper's scope. In the present context, what is relevant is that interactors have a state (a collection of typed attributes) and actions that act on that state. Attributes capture the contents of the user interface, and any relevant internal state of the device. Actions capture user-triggered events, as well as relevant internal events, which cause changes to the state.

Using the IVY tool typically involves a number of steps: model development, model validation, property development and verification, and analysis and interpretation of verification results. The validation and interpretation of results require input from domain and human factors experts. While the model captures the structure and behaviour of the user interface, a prototype representation of the interface would make it much easier to communicate the intended design and the verification results.

Tools such as Circus [12] or PVSio-web [15] place a greater emphasis on prototyping than IVY has done so far. The former on high-fidelity prototyping,

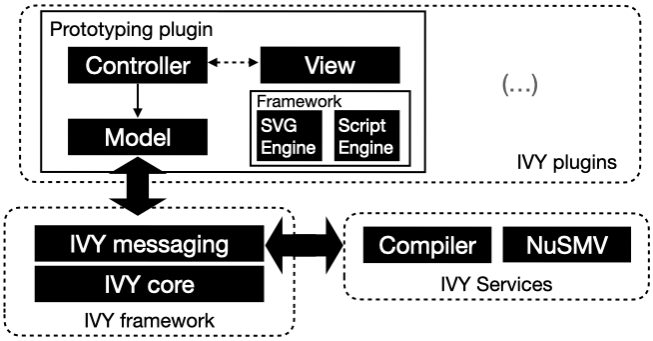

Fig. 1. Prototyping plugin's architecture

the latter focusing more on the earlier stages of design. Other authors have explored the problem of integrating formal methods and UCD in ways that are complementary to what is proposed herein. In [4] the goal is to bring informal design artefacts into a formal setting by finding ways to formally describing them, while herein the goal is to make formal artefacts accessible to designers.

3 Prototyping Plugin

To build a prototype from a MAL model, one must bring together the model, which defines the contents and behaviour of the interface, and a mock-up providing a graphical representation of that interface. Mock-ups are assumed to be vector graphics drawings in SVG [16]. This is achieved by configuring two types of bindings. Event bindings specify how the prototype responds to user interactions. They are established between user generated events in the mock-up (e.g. clicking an SVG element) and actions of the formal model. State bindings specify how the mock-up represents the state of the model. They are established between attributes in the model and the components in the mock-up. User interaction with the prototype triggers events, which cause the bound actions in the model to be *executed*. The resulting update of the model's attributes then generates changes in the prototype according to the configured state bindings.

3.1 Architecture

The prototyping plugin's architecture (see Figure 1) follows the Model-View-Controller design pattern. The *Model* holds the prototype's configuration, such as the events and states and their mapping to the MAL model. The *View* defines the UI of the plugin (using Java Swing). The *Controller* acts as a mediator between *View* and *Model*. Additionally, the *Framework* component offers utility methods. It includes two essential components: *SVGEngine* and *ScriptEngine*.

The *SVGEngine* contains methods for prototype initialization, particularly the generic SVG parser. SVG files are read using the Apache Batik library, and

the parser leverages the DOM (Document Object Model) structure thus generated to identify all SVG elements that support user interaction. These elements are assigned a UUID on their *id* property to ensure their uniqueness, allowing fast document queries. The engine has been shown to work with Inkscape, Evolus Pencil, Adobe Illustrator, and Adobe XD-generated SVG files.

The *ScriptEngine* provides support for externally loaded widgets and scripts execution support. The inclusion of support for widgets in SVG mock-ups provides additional expressive power and simplifies the creation of mock-ups. Widgets consist of a set of SVG shapes and Javascript code that specifies the widget's behaviour (e.g. which shape should be presented based on conditions). A library of such widgets is being created that, at the time of writing, includes from simple widgets such as a switch, a toggle button, a checkbox, a led light or a progress bar, to complex widgets such as a dual mode clock face (hours/alarm) or the display of the B. Braun Perfusor medical device (see Section 4).

The scripting environment is responsible for the initialisation of imported widgets. Using the Rhino Javascript engine, it extracts the widgets' properties and methods needed for the prototype's configuration. Furthermore, the environment performs the management of all widget element identifiers to ensure their uniqueness. This task prevents problems from scenarios where a user builds a prototype containing multiple instances of the same widget. The scripting environment is also responsible for binding the SVG mock-up and the MAL model. This feature is essential for combining the formal model capabilities of IVY with the mock-up. During prototype animation, the environment invokes the widgets' methods with relevant values obtained from the model. These methods use DOM queries to select SVG elements. Then, they modify the CSS properties of those elements according to the parameters received.

3.2 The Plugin's UI

The prototyping plugin's UI (see Figure 2) features the SVG renderer, the SVG sidebar and the Configuration sidebar. An animation mode is also provided. The renderer uses the Apache Batik library to render the SVG mock-up, and supports SVG elements selection by clicking. The SVG tree sidebar displays the SVG's hierarchical structure. It provides basic SVG editing functionalities, including visibility toggling, and element deletion and insertion.

The Configuration sidebar supports the definition of the binding between mock-up and model. The states tab allows users to bind SVG elements to the model's attributes. Each element has a default state representing its initial appearance. Users can modify the properties of that state, or add more states that are triggered when a specific condition is met. SVG tags, such as *g*, have a predefined set of properties that can be configured (e.g. their visibility), while externally loaded widgets allow users to configure any properties presented in their *props* object. The UI guides users in selecting valid values for the properties, listing attributes according to the type required by each property. This helps prevent binding errors. At animation time, the environment checks whether any of the defined conditions of an element matches its criteria. If this happens,

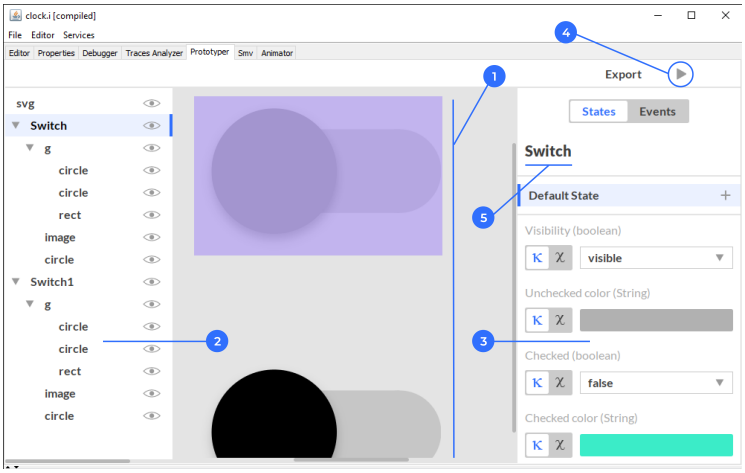

Fig. 2. User interface of the plugin – (1) SVG renderer; (2) SVG sidebar; (3) Configuration sidebar; (4) Access to animation mode; (5) Prototype configuration

then the matched state will be rendered. Otherwise, the simulation renders the default state of the element.

The events tab allows users to bind events at the SVG level (triggers) with formal model's actions. A trigger can be a user interaction or a periodic timer event. A trigger can be defined to directly execute a widget function. This offers the possibility to change the prototype's appearance without the formal model's assistance. This feature supports the creation of advanced prototypes without the need for creating complex formal models, by isolating cosmetic changes at the SVG level.

The animation window displays the prototype (in a SVG renderer), and the list of available actions (see Figure 3). The environment uses IVY's internal messaging system to communicate with the NuSMV model checker and obtain the current state of the prototype. Users can interact with the prototype through the mock-up, triggering the defined events, or with the actions in the list.

4 An example – the B. Braun Perfusor® Space

The B. Braun Perfusor® Space is a programmable infusion pump that automates the administration of drugs to patients. Although the device offers multiple configuration modes, herein we will focus on the programming of the value to be infused (VTBI). We will briefly describe the formal model, the UI mock-up, and the prototype's configuration and running.

The goal of the **formal model** is to capture the behaviour of the device. The VTBI is presented on the display (see Figure 3). Digits can be increased and decreased by pressing the up and down keys. A cursor (identified by a black square) highlights the currently selected digit. The left and right arrow keys move

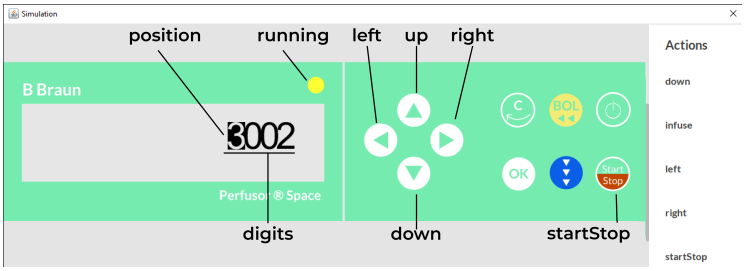

Fig. 3. Mock-up of the B. Braun Perfusor® Space UI

the cursor between digits. The start/stop button, controls drug administration. When the device is administering the drug, the running indicator lights up. In this mode, users cannot interact with the cursor, and the value represented in the display decreases over time.

The infusion pump model consists of a MAL interactor capturing the contents and behaviour of its user interface. A detailed description of the model is not feasible herein. Enough will be described to make the example clear. This amounts to explaining which attributes and actions are needed in the model. Looking at Figure 3, the following attributes are relevant: *digits* (an array holding the value of each digit in the display), *position* (an integer indicating the cursor position in the display), *running* (a boolean indicating whether the device is in infusion mode). As for actions, the model defines the following: *up*/*down* (increase/decrease the value in the selected digit), *left*/*right* (increase/decrease the *position* attribute – i.e. move the cursor), *startStop* (start/stop the infusion process), *infuse* (internal action that represents the progression of the infusion process by decreasing the volume to be infused until it reaches 0 or the device is stopped). This is expressed in MAL as (axioms ommited for brevity):

```
interactor main
  attributes
    [vis] digits: array 0..MAXDIG of int
    [vis] position: 0..MAXDIG
    [vis] running: boolean
  actions
    [vis] up down left right startStop
    infuse
```

Figure 3 presents the device's **UI mock-up**. Besides basic SVG shapes representing buttons and static information, it features two widgets: led and cursor. The led widget is used to indicate if the device is in infusion mode. It has a property (*On*) to control whether the led is light-up. The cursor widget was explicitly developed to represent the digits screen of this pump. Its main properties are *Digits* (each of the digits displayed) and *Cursor Position* (the position of the cursor in the display).

To have a **running prototype**, bindings between model and mock-up need to be defined. Configuration of the cursor widget is done by binding the attributes *digits* and *position* from the model to the properties *Digits* and *Cursor Position*, respectively. Configuration of the led widget is done by binding the *running* attribute from the model to the *On* property of this widget. The click events of the arrow buttons and the start/stop button are bound to the appropriate actions (*up*, *down*, *left*, *right*, *startStop*). Finally, the *infuse* action was bound to a timer event, which executes every one second. Once all the steps just described were performed, users were able to interact with the prototype in the simulation window. Figure 3 is in fact a screen capture of the contents of that window, annotated for readability.

5 Validation

The medical device above served as validation of the functional capabilities of the new plugin. The next phase of the project will be to validate the role of the plugin as a bridge between formal methods and human factors and domain experts. This will entail exploring its use to validate formal models by allowing domain experts to interact with the model via the prototype, thus validating the captured behaviour, as well as using the prototype to replay behaviours resulting from the verification process, to support the identification of errors.

As a first step, however, we were interested in evaluating whether the process of creating prototypes is itself accessible to non-experts. We have carried out the pilot of a user test designed to evaluate this. The test consists of setting up the prototype for the B. Braun device, given the formal model and the mock-up. The trial sessions were performed on a laptop with the IVY Workbench installed, and the required model and mock-up available. A consent form, a test script containing a brief introduction to IVY, a step-by-step guide to prototype configuration, and a final questionnaire were also made available. The step-by-step guide explained the configuration of a toggle button. The questionnaire served to collect demographic data (name, age, sex and background experience) and collect feedback on the tool. The full test procedure was the following: (1) welcome and consent form signing; (2) test script presentation and brief explanation of the tool and the activities to perform; (3) execution of the step-by-step example and clarification of any questions related to the tool; (4) execution of the actual test (recorded via screen capturing); (5) answering the questionnaire.

The pilot involved 5 participants (one male and four female) with average age 23.6 years ($\sigma = 2.1$). Participants were selected to cover a wide range of backgrounds. One participant had a software engineering background. Two participants had design backgrounds. And the last two participants did not have a background in either of the areas. Two participants indicated that they had previous experience with mock-up editors, none had experience with IVY.

All participants were able to complete the configuration of the prototype. The average time required to fulfil the task was 8 minutes 55 seconds ($\sigma = 2$m17s). Overall the pilot validated the testing protocol and a number of observations was

already possible. Users were able to quickly select the appropriated attributes of the formal model to configure the required states of the elements in the mock-up. However, they had difficulties selecting SVG elements composed from other SVG elements. Users rarely selected elements by interacting with the renderer. Instead, they prefered the SVG tree sidebar. Initially, users had some difficulties in distinguishing the differences between states and events. These were overcome as they interacted more with the tool.

As mentioned, at the end of the test a number of questions were asked about the tool. Regarding their overall impression of the system, all participants made a positive overall evaluation. As most positive aspects of the system, participants mentioned the simplicity of the UI. Three participants approved the possibility of selecting SVG elements both with the SVG tree sidebar and the renderer. One participant mentioned the attribute selection guidance in the states configuration process. As most negative aspects, participants mentioned difficulties in selecting groups of SVG elements. As most surprising aspects of the system, three participants mentioned the use of widgets in the mock-ups, two participants pointed out periodic time events, and one participants mentioned the tool's capabilities to create running prototypes with ease. As for missing functionalities, two participants pointed out drag and drop and SVG group selection in the renderer. Only two participants (those with prior experience with mock-up editors) were able to compare the functionalities of the plugin with other UI prototyping tools. Both participants appreciated the capabilities of adding behaviour to prototypes afforded by the tool.

6 Conclusion

We have presented a new plugin for the IVY workbench. The plugin aims to provide support for the integration of the formal methods based analysis, with user centred design. It proposes to achieve this through supporting the process of creating user interface prototypes by binding together the formal models used for analysis, and the mock-ups used for user centred design. These prototypes will be instrumental in, first, validating the behaviour captured by the model with domain experts and human factors experts, and, second, communicate the results of the analysis to the same stakeholders. The goal is not to replace tools like Figma, rather integrate the workflows of such tools with the formal verification workflow. In any case, in doing this we achieve prototypes that are able to exhibit a level of behaviour that is not easily achieved with mock-up editors.

Besides describing the plugin, the paper describes the initial steps taken to validate the tool. Although preliminary, the tests already emphasised the strengths and weaknesses of the new prototyping features. Users recognised the widgets' capabilities and generally found the developed UI intuitive. However, users had difficulties selecting SVG elements in the renderer, suggesting some improvements are needed to support group selection as default instead of single element selection.

Future work will include further developing the widgets library and proceeding with the evaluation of the tool. First by completing the evaluation mentioned above. Then, by exploring the role that the tool might have as a bridge between formal methods and user-centred design.

Acknowledgements This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

References

1. Beaudouin-Lafon, M., Mackay, W.: Prototyping tools and techniques. In: Jacko, J.A., Sears, A. (eds.) The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications, chap. 52, pp. 1006–1031. L. Erlbaum Associates Inc., 365 Broadway Hillsdale, NJUnited States (2002)
2. Beyer, H., Holtzblatt, K.: Contextual Design: Defining Customer-Centred Systems. Morgan Kaufmann (1998)
3. Bolton, M.L., Bass, E., Siminiceanu, R.: Using formal verification to evaluate human-automation interaction: A review. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans **43**(3), 488–503 (May 2013)
4. Bowen, J., Reeves, S.: Formal models for informal gui designs. Electronic Notes in Theoretical Computer Science **183**, 57–72 (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems (FMIS 2006)
5. Campos, J.C., Sousa, M., Alves, M.C.B., Harrison, M.D.: Formal verification of a space system's user interface with the IVY workbench. IEEE Transactions on Human-Machine Systems **46**(2), 303–316 (April 2016). https://doi.org/10.1109/THMS.2015.2421511
6. Campos, J., Fayollas, C., Harrison, M., Martinie, C., Masci, P., Palanque, P.: Supporting the analysis of safety critical user interfaces: an exploration of three formal tools. ACM Transactions on Computer-Human Interaction (2020), accepted
7. Campos, J., Harrison, M.D.: Interaction engineering using the IVY tool. In: ACM Symposium on Engineering Interactive Computing Systems (EICS 2009). pp. 35–44. ACM, New York, NY, USA (2009)
8. Campos, J.C., Harrison, M.D.: Model checking interactor specifications. Automated Software Engineering **8**(3/4), 275–310 (August 2001)
9. Carroll, J. (ed.): Scenario Based Design: Envisioning Work and Technology in System Development. Wiley (1995)
10. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification. pp. 359–364. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
11. Dix, A., Weyers, B., Bowen, J., Palanque, P.: Trends and gaps. In: Dix, A., Weyers, B., Bowen, J., Palanque, P. (eds.) The Handbook of Formal Methods in Human-Computer Interaction, chap. 3, pp. 65–88. Springer (2017)
12. Fayollas, C., Martinie, C., Palanque, P., Deleris, Y., Fabre, J.C., Navarre, D.: An approach for assessing the impact of dependability on usability: Application to interactive cockpits. Proceedings - 2014 10th European Dependable Computing Conference, EDCC 2014 pp. 198–209 (05 2014). https://doi.org/10.1109/EDCC.2014.17

13. Harrison, M.D., Campos, J.C., Loer, K.: Formal analysis of interactive systems: opportunities and weaknesses. In: Cairns, P., Cox, A. (eds.) Research Methods in Human Computer Interaction, chap. 5, pp. 88–111. Cambridge University Press (2008)
14. Harrison, M., Freitas, L., Drinnan, M., Campos, J., Masci, P., di Maria, C., Whitaker, M.: Formal techniques in the safety analysis of software components of a new dialysis machine. Science of Computer Programming **175**, 17–34 (April 2019). https://doi.org/10.1016/j.scico.2019.02.003
15. Masci, P., Oladimeji, P., Zhang, Y., Jones, P., Curzon, P., Thimbleby, H.: PVSio-web 2.0: Joining PVS to HCI. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I, pp. 470–478. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-21690-4_30
16. W3C: Scalable Vector Graphics (SVG) 2. Candidate Recommendation CR-SVG2-20181004, W3C (October 2018), https://www.w3.org/TR/2018/CR-SVG2-20181004/