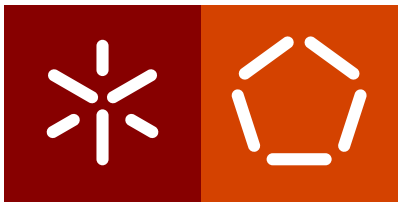
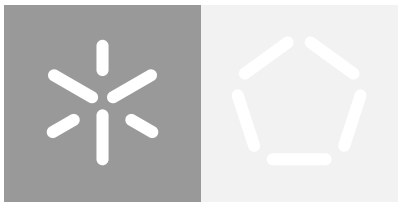


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rafael Braga Gomes da Costa

Animating user interface prototypes with formal models

November 2020



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rafael Braga Gomes da Costa

Animating user interface prototypes with formal models

Master dissertation
Integrated Master in Informatics Engineering

Dissertation supervised by
José Creissac Campos
Rui Couto

November 2020

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to my supervisor, Professor José Creissac Campos, for his guidance, availability, commitment, and continuous assistance. Without his exceptional suggestions, this work would not have been possible.

I would also link to thank my co-supervisor, Professor Rui Couto, for his help and valuable ideas that vastly contributed to the final result.

I am incredibly grateful to my family for their unconditional support, affection and for pushing me in the right direction throughout my life.

I would like to express my profound gratitude to my girlfriend, Mariana Farias, for always giving me the inspiration and motivation that I need.

My special thanks to Diogo Silva, for providing me relevant feedback that helped me to achieve better results.

I am very grateful to my friends for their support and companionship throughout my life and for giving me countless memorable moments that allowed me to stay focused.

Last but not least, I wish to extend my special thanks to all participants that completed a survey to help me gather valuable data for this dissertation.

ABSTRACT

The *User Interface (UI)* provides the first impression of an interactive system and should, thus, be intuitive, in order to guide users effectively and efficiently in performing their tasks. User interface prototyping is a common activity in UI development, as it supports early exploration of the UI design by potential users.

UI quality plays a crucial role in safety-critical contexts, where design errors can potentially lead to catastrophic events. Model-based analysis approaches aim to detect usability and performance issues early in the design process by leveraging formal analysis. They complement prototyping, which supports user involvement, but not an exhaustive analysis of the designs.

The IVY Workbench emerges as a model-based analysis tool intended for non-expert usage. The tool was originally focused on supporting modelling and verification, but more recently an effort began to combine the formal model capabilities with UI mock-ups, to produce more interactive prototypes than traditional mock-up editors support.

This work addresses the enhancement of the prototyping features of the IVY Workbench. The improvements of such features include the creation of a dynamic widget library that can vastly improve the quality of prototypes. Such a library, however, should be compatible with several mock-up editors to attract a broader design community.

The results of this work include an analysis of alternative prototyping tools, identifying potential features that can enhance the IVY Workbench, the creation of a dynamic widget library that is compatible with several mock-up editors, and several improvements to IVY's prototyping plugin, including the addition of code exporting functionalities. Usability tests were conducted to validate the new features of the tool, with positive results. Two mobile applications were also created, allowing users to test prototypes in their mobile devices.

Keywords: User interface, prototype, user-centred design, widget.

RESUMO

A UI proporciona o primeiro contacto entre um utilizador e um sistema interativo. Assim, a UI deverá ser capaz de guiar o utilizador na execução das suas tarefas, de um modo eficiente e eficaz. A prototipagem de interfaces é uma atividade comum no processo de desenvolvimento de UIs, já que permite a exploração antecipada do *design* de uma UI com potenciais utilizadores.

A UI tem um papel bastante relevante no contexto de sistemas críticos, onde falhas no *design* podem gerar eventos catastróficos. As metodologias de análise baseadas em modelos procuram detetar potenciais falhas de usabilidade e desempenho, em fases iniciais do processo de desenvolvimento, através de análise formal. Estas metodologias complementam o processo de prototipagem, que suporta o envolvimento dos utilizadores mas não oferece uma análise exaustiva do *design*.

A IVY Workbench surge como uma ferramenta de análise baseada em modelos que visa suportar utilizadores sem grandes conhecimentos de análise formal. Embora originalmente focada na modelação e verificação, surgiu recentemente um esforço para combinar as capacidades da análise formal com *mock-ups* da UI. O objetivo é produzir protótipos com maior nível de interação do que os produzidos pelos tradicionais editores de *mock-ups*.

O presente trabalho apresenta melhorias das capacidades de prototipagem da ferramenta IVY Workbench. Estas melhorias incluem a criação de uma biblioteca de *widgets* dinâmicos, que aperfeiçoa a qualidade dos protótipos desta ferramenta. Esta biblioteca deverá ser compatível com múltiplos editores de *mock-ups*, de modo a atrair uma vasta comunidade de *designers*.

Os resultados deste trabalho incluem uma análise de alternativas de ferramentas de prototipagem, onde são identificadas funcionalidades que podem aprimorar a ferramenta IVY Workbench; a criação de uma biblioteca de *widgets* dinâmicos, compatível com inúmeros editores de *mock-ups*; assim como várias melhorias efetuadas no *plugin* de prototipagem desta ferramenta, incluindo a adição de funcionalidades de exportação de código fonte. Foram realizados testes de usabilidade para validar as novas funcionalidades da ferramenta com utilizadores, onde foram obtidos resultados positivos. Finalmente, foram criadas duas aplicações móveis que permitem que os utilizadores testem os protótipos nos seus dispositivos móveis.

Palavras-Chave: User interface, protótipo, user-centred design, widget.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

ACRONYMS

A

API Application Programming Interface.

C

CSS Cascading Style Sheets.

D

DOM Document Object Model.

G

GUI Graphical User Interface.

H

HCI Human-computer Interaction.

HTML Hyper Text Markup Language.

I

ICO Interactive Cooperative Objects.

J

JPEG Joint Photographic Experts Group.

M

MAL Modal Action Logic.

MVC Model-View-Controller.

P

PNG Portable Network Graphics.

PVS Prototype Verification System.

S

SVG Scalable Vector Graphics.

U

UCD User-centred Design.

UI User Interface.

UUID Universally Unique Identifier.

UX User Experience.

W

W3C World Wide Web Consortium.

X

XML Extensible Markup Language.

1

INTRODUCTION

It is of crucial importance for user interfaces to be well designed and intuitive, thereby leading to correct system usability and promoting user acceptance and productivity.

Building prototypes is a typical techniques used to help improve user interface quality. In many cases these prototypes consist of mock-ups of the envisaged design. When considering safety-critical systems, however, more detailed prototypes, capturing also the behaviour of the systems, are needed. Models of user interface behaviour can be useful in this regard.

This work addresses the prototyping of user interfaces. The current chapter aims to introduce the domain of user interface prototyping and the motivation of model-based approaches for critical systems design, with a particular focus on the IVY Workbench tool.

1.1 CONTEXTUALIZATION

The UI plays a crucial role in the field of interactive systems because it is the component of the system that users can hear, see, touch, understand or interact with. The UI should be easy to use and intuitive, promote efficient system use, to increase productivity and satisfaction. By contrast, a poorly designed UI leads to exasperation, frustrations and may even lead users to abandon the system permanently Galitz (2007). Besides, in the field of critical systems, where human lives are at risk, a bad UI design could lead to potentially catastrophic events. Consequently, it is indispensable to design the system by taking into account the requirements of the users.

In this context, prototyping emerges as an essential step of the design process. It provides a quick and inexpensive first impression of the system before its release, allowing users and designers to identify usability issues early. Therefore, prototypes provide a method for designers to evolve the system into a final result that meets the users' demands.

Prototypes can take many forms, from rough sketches to complex animations built with software tools. Some prototyping tools focus on the visual representation, producing mock-ups with several levels of detail that can closely resemble the final look of the system, the so-called mock-up editors.

However, in safety-critical fields, even if the prototype is detailed enough to capture all relevant features of the system, it may not present the required level of analysis thoroughness. Model-based approaches surpass these issues by involving a formal analysis of the system behaviour alongside with its design. For this reason, tools of this approach tend to be often complex and designed for experts.

The IVY Workbench (Campos and Harrison, 2008; Couto and Campos, 2019) is a model-based tool currently under development. It provides a set of plugins with multiple features intended for a multidisciplinary team. One of its main goals is to provide a simple environment for non-experts.

1.2 Motivation

Conceiving well designed and intuitive systems can be an expensive and time-consuming task, requiring early planning, testing, and evaluation with the users. In safety-critical contexts, this process grows in complexity, demanding model-based tools for proper prototyping of the system behaviour.

However, the model-based solutions found so far require extensive knowledge of the tool (cf. Campos et al., 2020). Furthermore, they require designers to understand the internal model of the system behaviour when building prototypes for the system UI. Thus, the current solutions for this approach target domain experts and programmers.

This work proposes a more powerful model-based approach with the IVY Workbench tool. A combination of the prototyping features of the mock-up editors and the formal analysis of this tool can potentially produce highly-detailed prototypes with large sets of user interaction. Moreover, the usage of these features expands the target fields of this tool, enabling prototype creation for non-expert users.

1.3 Aim of the work

The main goal of this work is to enhance the prototyping features of the IVY Workbench with the definition of a widget framework capable of introducing dynamic behaviour to the prototypes' interface and validate it with the creation of a collection of programmable and easy to use widgets. We envisage that these widgets will be embedded into *SVG (Scalable Vector Graphics)* files, which are often used in mock-up editors. This approach will allow designers to build mock-ups with dynamic functionalities in their preferred tools and later import these mock-ups into IVY Workbench.

This work also aims to bring together prototyping techniques and model-based tools, to achieve a user-friendly environment for user interface prototyping in the IVY Workbench. The goal is to explore relevant features of mock-up editors and provide an abstraction of the

user interface models, in order to support synergistic but independent development of both layers, thus promoting task division within a multidisciplinary team. Furthermore, cognitive and usability tests of this tool should be carried out with non-expert users to validate these subjects.

The current work also addresses tests for compatibility of the tool's prototypes with the major mobile platforms, such as Android and iOS. The purpose is to promote prototype usability, allowing users to test and validate prototypes in their own mobile devices.

To summarize, the goals of this work are the following:

- Design a widget framework capable of introducing dynamic behaviour to prototypes.
- Validate this framework by creating a collection of programmable widgets.
- Explore relevant features of prototyping tools that can improve the IVY Workbench prototyping functionality.
- Provide support for importing prototypes from multiple prototyping tools.
- Automatize the prototyping features of the IVY Workbench.
- Perform usability tests with non-expert users.
- Evaluate cross-platform compatibility of the prototypes by performing tests on mobile devices.

1.4 Document structure

The remaining of the dissertation includes the following chapters:

- Chapter 2 reviews previous studies regarding the user-centred design methodology, prototyping techniques and prototyping tools. This chapter ends with a comparison of those tools according to their prototyping features.
- Chapter 3 explores features to include in the IVY Workbench prototyping plugin, describing their requirements and the technologies needed.
- Chapter 4 presents the contribution made to the prototyping features of the IVY Workbench.
- Chapter 5 provides an application example of the new prototyping features. This chapter also details the results obtained from the tests with non-expert users.
- Chapter 6 presents the conclusions regarding the additional prototyping features and the studies conducted in this project, as well as prospects for future work.

2

STATE OF THE ART

Use errors derived from user interface design flaws are a significant concern in safety-critical contexts. Prototypes provide a means to improve the design process by helping users and designers in the early identification of usability issues (Beaudouin-Lafon and Mackay, 2002). In critical systems design, however, prototyping is not enough to provide the required level of assurance. Even if the prototype is detailed enough to capture all relevant aspects of the system, it may not present the required level of analysis thoroughness. A correct application of human factors intrinsic to UCD and the usage of prototyping tools supporting this methodology could conceivably augment the safety level of these systems (Cacciabue, 2004).

This chapter describes the UCD methodology and its relevance for critical systems design, as well as the chief prototyping techniques. Ultimately, it also reports examples of two types of tools that support prototyping: model-based tools and mock-up editors.

2.1 USER-CENTRED DESIGN

This section describes a design process focused around users' needs called user-centred design, and provides a brief explanation about alternatives centred around technologies, the so-called technology-centred design.

Traditionally, systems have been designed and developed from a technology-centred view (Endsley and Debra, 2011). As technology grows, more and more components are added, like displays and sensors. This style of approach, centred around the technologies, assigns a great responsibility to the users since they are left to keep up with the exponential growth in complexity of the system. Because humans can only process a certain amount of information at once, this becomes a pitfall, especially in the design of critical systems, such as those found in the medical and avionics domains. A good design should focus more on users intentions and their working conditions rather than technologies, since the opposite tends to induce usability errors.

UCD (ISO, 1999) emerges as an alternative to the complexity and error induced by the technology-centred design methodologies. This approach's prime focus is to make systems

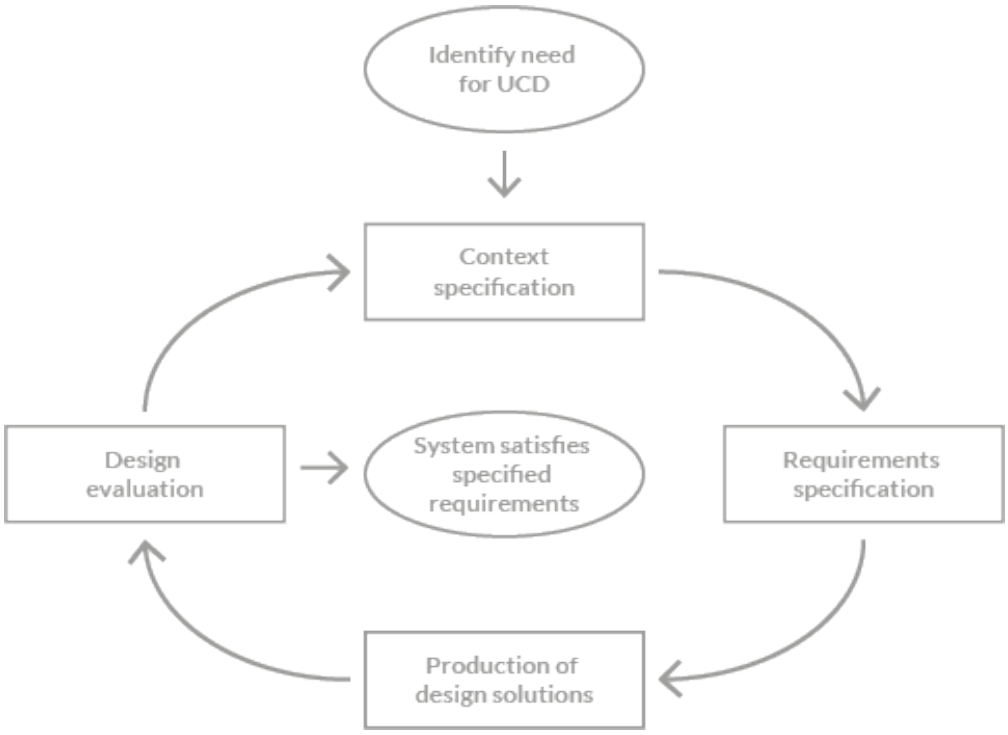


Figure 2.1: UCD process and its activities represented as rectangles (image adapted from ISO (1999)).

usable by incorporating human factors, ergonomics knowledge and balancing the allocation of functions and responsibilities between users and technology. One of the main principles of this methodology is the active involvement of users that provide a valuable source of knowledge about their demands, while also increasing their interaction with developers. Furthermore, UCD is an iterative process that minimizes the risk of poorly designed systems by testing preliminary design solutions against real scenarios and user requirements. Lastly, another fundamental principle of the UCD is the involvement of a multidisciplinary team that should be sufficiently diverse to make relevant design decisions. The correct application of UCD and its principles assists developers to produce systems that support users according to their goals, tasks and needs while motivating them to learn. The benefits of this methodology can include improved user satisfaction, enhanced quality of work, increased productivity, reductions of learning curves and training costs.

UCD process should start at the most beginning stage of the project and repeated iteratively until the system satisfies all the requirements, as depicted in Figure 2.1. This process should have the following four inherent design activities:

- The **context specification** relates to the identification of the characteristics and roles of the intended users. Furthermore, it describes the tasks the users are to perform and the environment in which they are to work with the system.

- The **requirements specification** is a description of user and organizational requirements. In this activity takes place the division of the system tasks between humans and technology.
- The **production of design solutions** activity relates to the collection and management of both multidisciplinary design proposals and user feedback. Moreover, it iteratively evolves the design solutions in response to the collected feedback until the design goals are met.
- The **design evaluation** is the activity that tests and validates the design solutions of the system to ensure that they satisfy the requirements of the users, the tasks and the environment.

To summarize, UCD is an iterative process that focuses on users tasks and characteristics. It receives input from a multidisciplinary team to explore diverse design ideas from various fields, such as ergonomics and cognitive science. Lastly, it collects user feedback from interaction with generated design solutions. Prototypes play a crucial role in the latter subject, by providing a concrete and inexpensive representation of the system, and are described with detail in the next section.

2.2 PROTOTYPING

A prototype is a concrete representation of part or all of a system, used to envision and reflect on the final product (Beaudouin-Lafon and Mackay, 2002). A prototype can be defined differently according to the application area. In architecture, for instance, a plausible prototype is a scaled-down representation of the building. By contrast, prototypes for interactive systems, although with limited information, should present a full-scaled interface.

Prototypes support creativity and innovation (Viswanathan and Linsey, 2009), helping designers to generate ideas, specify design problems, and gather information about users and their habits. They promote communication and interaction between designers, managers, developers and customers. Furthermore, they allow early evaluation by collecting user feedback throughout the design process with usability tests. Therefore, they are useful to capture user requirements (Deininger et al., 2017), anticipate possible improvements, and identify and mitigate errors (Devadiga, 2017), both early and late in the design process.

Prototyping plays a crucial role in the design process, especially in the research field of *Human-computer Interaction (HCI)*, being used in the early stages to explore several design options and direct the development further through iterations (Elverum and Welo, 2014). It is also one of the most significant activities of UCD by allowing users to evaluate the product throughout the design process (Rocha Silva et al., 2015).

This section begins with the introduction of four dimensions for analysing prototypes. Then, it describes three prototyping approaches: *rapid*, *iterative* and *evolutionary*.

2.2.1 *Prototyping dimensions*

According to Beaudouin-Lafon and Mackay (2002), prototypes and its techniques have four dimensions:

- *Representation* describes their form.
- *Precision* measures their level of detail when compared to the final product.
- *Interactivity* describes the user interaction with the prototype.
- *Evolution* describes their expected life cycle.

Representation

Prototypes take different forms, from rough sketches to complex computer animations, to serve many purposes. Each form is useful for designers and users in several ways. This section distinguishes two types of prototype representations: **offline** and **online** (Beaudouin-Lafon and Mackay, 2002).

Offline prototypes do not require a computer and usually include paper sketches (Figure 2.2) and videos. They provide a quick and inexpensive first impression of the system that allows designers to explore several options without becoming overly attached to the first solution. For this reason, these prototypes are mainly used in the early stages of design and are typically thrown away once they fulfil their purpose. Another chief feature of **offline prototypes** is that developing them does not require any particular set of skills, promoting multidisciplinary involvement in the design process (Bähr, 2013).

Online prototypes, also referred to as software prototypes, run on a computer. They typically include computer animations, interactive videos, and programs created with scripting languages. Therefore, **online prototypes** usually resemble the future product (Johansson and Arvola, 2007). However, this representation of prototypes is typically more expensive than the **offline** type and may require skilled and domain-specific professionals to implement their features. Additionally, they narrow the number of design proposals, as opposed to **offline** ones. For these reasons, **online prototypes** are more practical in the latter stages of design.

Offline prototypes provide a rapid iteration cycle, quick and low detailed solution. Whereas **online prototypes** focus more on the implementation, requiring high costs, both in skill and time. One must consider the purpose of the prototype at each stage of the design process and choose the best-suited representation.

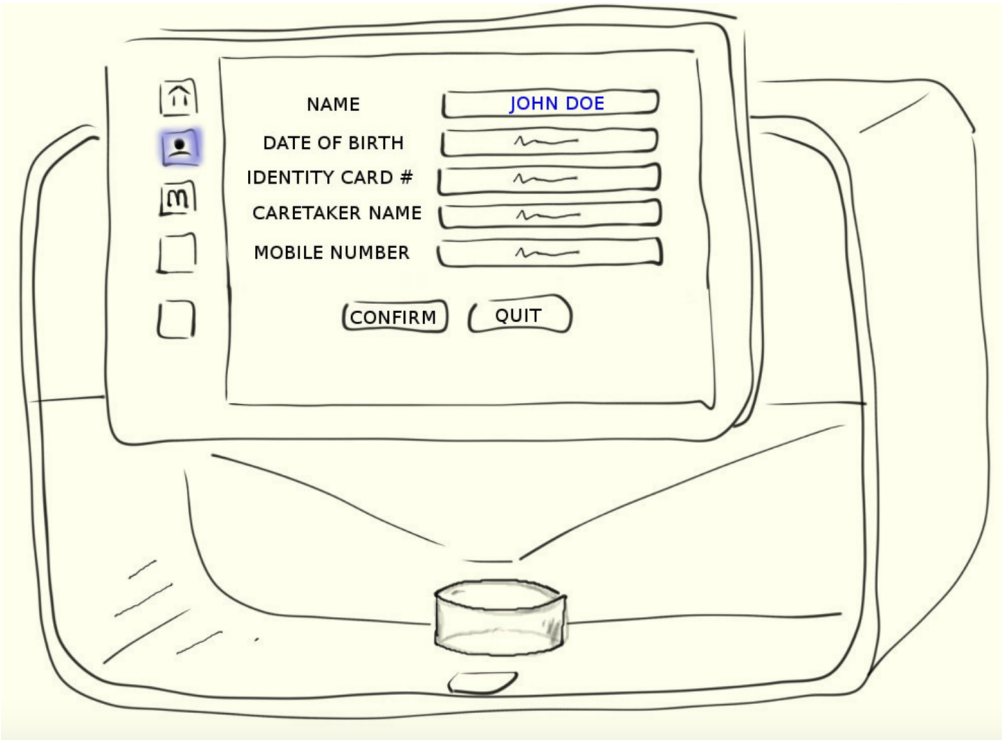


Figure 2.2: Paper sketch mock-up representing a sign up page.

Precision

Prototyping is an iterative process that entails the design and redesign of a system into a functional product that can be evaluated by the target users (Suleri et al., 2019). Hereupon, the term *precision* regards to the level of detail or **fidelity** of a prototype. Imprecise prototypes, herein called **low-fidelity prototypes**, offer a version of a product open to further discussion of design ideas. **Medium-fidelity prototypes** provide a more detailed look with limited functionality (Engelberg and Seffah, 2002). **High-fidelity prototypes** provide a very close look of the final product.

Low-fidelity prototypes are useful to identify design issues at very early stages of the design process (Carter and Hundhausen, 2010). They provide a quick and inexpensive way to explore an extensive set of new design ideas, as they are usually implemented with paper sketches or screen mock-ups. They also stimulate communication between designers and users and avoid costly redesigns.

Medium-fidelity prototypes focus on the interactive aspects, namely the navigation between components, functionality and layout. They also provide a more detailed look than **lower-fidelity** versions. However, both the look and feel aspects are limited when compared to the final product. This type of prototypes is also less time consuming and have much lower costs as compared with **higher-fidelity** versions.

Lastly, **High-fidelity prototypes** offer a way for users to test a close version of the system UI, both in terms of functionality and representation. Therefore, they are more expensive than the **low-fidelity** ones, both in cost and time. Ergo, they are usually targeted to the latter stages of the design process. **High-fidelity prototypes** are very useful for the identification of design problems where **low** and **medium-fidelity prototypes** may not suffice. Paper sketches or screen mock-ups could not convey the full sense of the system behaviour and do not provide an adequate environment of user testing and execution (Virzi et al., 1996). Furthermore, in safety critical domains, for instance, the requirements and precision of prototypes are usually high since there is the risk of fatal accidents (Rottermanner et al., 2018).

The level of fidelity of prototypes usually increases as the design process iterates, and more details are set. Lower fidelity levels provide an inexpensive way for designers to expand the universe of design solutions rapidly, whereas higher levels refine the solutions into the final product. Figure 2.3 illustrates these concepts by providing a comparison between low and high-fidelity prototypes of a mobile application page. Notice that the **low-fidelity** version is represented by a wireframe, emphasizing the main components of the UI but not their look and feel. By contrast, the **high-fidelity** variant represented by a mock-up presents a more detailed version of the UI, including icons and the style of its components. Section 2.3.1 discusses and provides some examples of wireframing and mock-up tools, the so-called mock-up editors.

Interactivity

Effective interaction design is crucial in HCI systems but challenging to implement since it implies the definition of how a system should be used. Although many systems provide attractive visuals, they fail on user interaction (Beaudouin-Lafon and Mackay, 2002). Usually, higher levels of interactivity bring user's satisfaction, effectiveness and efficiency towards the designed system (Teo et al., 2003). Hereupon, designers must hold a deep understanding of users and their working practises.

Prototypes have three different classes, according to their interactivity level: **fixed**, **fixed-path**, and **open** (Beaudouin-Lafon and Mackay, 2002). **Fixed prototypes** do not support user interaction and are merely a preview of the system. A set of video clips or precomputed animations are examples of fixed prototypes. **Fixed-path prototypes** provide little user interaction, usually triggered by a specific action. A typical example of this implementation consists of a set of clickable mock-ups, mainly used to simulate the UI flow and navigation between components of the designed application (Figure 2.4). Finally, **open prototypes** provide more support for user interaction by allowing some level of control flow, working and behaving like the target system, although with limitations, such as poor error-handling, limited user input and performance.

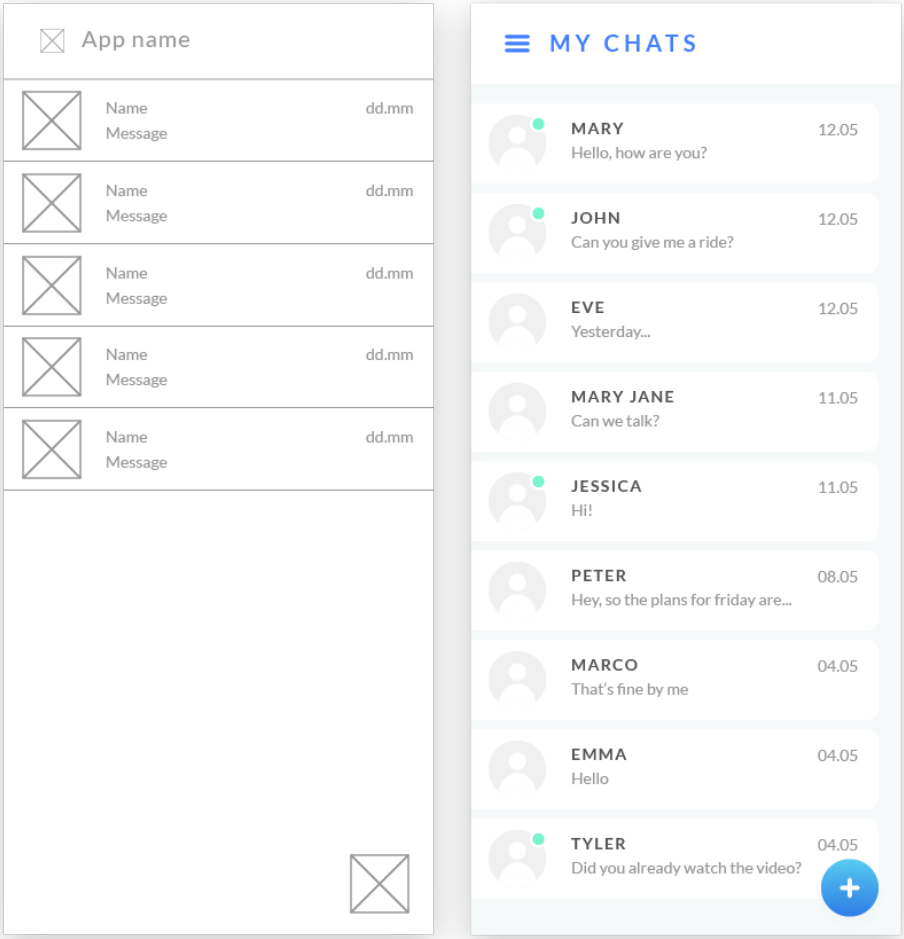


Figure 2.3: Comparison between **low** and **high-fidelity prototypes**.

These three classes of prototypes address different levels of interactivity. **Fixed prototypes** are a simple illustration of the system, whereas **fixed-path prototypes** offer the experience of what the interaction might look like, in certain situations. Finally, **open prototypes** are useful to determine how users will interact with the system.

Evolution

Evolution defines the lifespan of a prototype. According to their lifespan, prototypes can be **rapid**, **iterative**, or **evolutionary** (Beaudouin-Lafon and Mackay, 2002).

The primary role of **rapid prototyping** is to create prototypes quickly and to shorten the design evaluation cycle, encouraging the design team to explore an extensive set of new ideas before reaching a point in the design process where changes are costly (Liou, 2007). Although

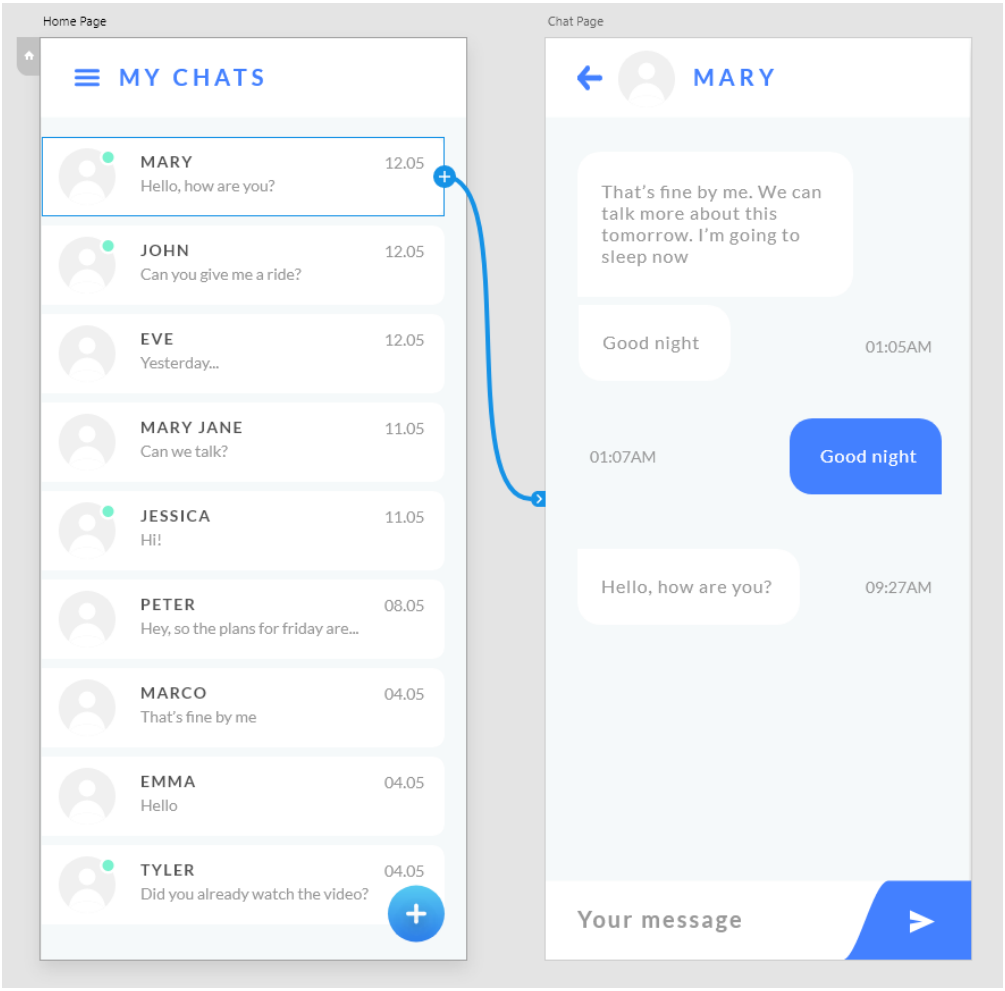


Figure 2.4: Example of a prototype consisting of two clickable mock-ups. When a user clicks on a list item of the home page, he or she will be redirected to the chat page (prototype made with Adobe XD).

rapid prototypes promote early experimental usage and evaluation, they are usually discarded as the design process iterates, and solutions that better meet the users' requirements are found. For this reason, they are often called **throwaway prototypes** (Nguyen-Duc et al., 2017). **Rapid prototyping** techniques include both offline, like paper prototypes, and online approaches, such as wireframes and interactive mock-ups. Paper prototypes are a valuable offline technique in the early stages of the design process (Soute et al., 2017), being a low-cost and fast way for designers to communicate, explore and evaluate interface designs (Bailey et al., 2008). Wireframes are representations of the skeletal structure of the UI that provide an initial perception of the hierarchy and relationship between its components (Llema and Vilela-Malabanan, 2019). Lastly, mock-ups can be used at different levels of detail to provide a representation of the system, allowing designers to concentrate on the physical design of

the UI, like layout positioning (Beaudouin-Lafon and Mackay, 2002). Mock-ups also enable designers to distinguish and detail specific areas of concern while leaving others open to further discussion (Camburn et al., 2017).

Iterative prototyping is a process that evolves with the design process with increasing levels of fidelity (Wood and Romero, 2010), commencing with exploratory and simple prototypes and iterating towards complex models of the system (Gervet et al., 1999). As the process iterates, each successive prototype uncovers new design opportunities or refines some ideas made with previous ones, oscillating between creation and feedback with experimentation (Dow et al., 2009). **Iterative prototyping** techniques include software tools and environments that require a higher level of expertise when compared to the **rapid prototyping** approaches described earlier (Beaudouin-Lafon and Mackay, 2002). Model-based tools are one of those techniques that aim to facilitate the creation of UI's and to reduce costs of interactive systems development (Machado et al., 2017). Section 2.3.2 provides some tool examples and a more detailed explanation of the model-based methodology.

Evolutionary prototypes are a particular case of iterative prototypes only applicable to software products, with the purpose to evolve into a part or all of the system (Beaudouin-Lafon and Mackay, 2002). Due to their nature, they are very challenging to implement when compared with the approaches described earlier, requiring careful planning about their underlying software architecture (Hertel and Dittmar, 2017). One chief advantage of this approach is that it allows users to test the product even in the early stages of the design process (Guida et al., 1999). However, this process tends to introduce over commitment due to the amount of time invested in one solution, encouraging designers to refine and work towards a particular solution instead of exploring more alternatives. Evolutionary prototyping techniques imply the use of architectural solutions such as *Model-View-Controller (MVC)* and other design patterns.

Each one of the described prototyping approaches has its own set of unique features that make them appropriate to different contexts. Table 2.1 summarizes the main principles of the mentioned approaches by clarifying their advantages and disadvantages.

2.3 PROTOTYPING TOOLS

Prototypes serve different goals and thus take many forms at each stage of the design process. Therefore, one must consider the purpose of the prototype and the best-suited prototyping tool for the current design question (Beaudouin-Lafon and Mackay, 2002).

This section details and provides examples of two distinctive types of tools that support prototyping: mock-up editors and model-based tools. The goal is to evaluate potential prototyping features that could substantially improve the tool related to this project, the IVY Workbench. The combination of features from these two types of prototyping promotes

Table 2.1: Advantages and disadvantages of rapid, iterative and evolutionary prototypes.

Approach	Advantages	Disadvantages
Rapid	Low time consumption and inexpensive; support for exploring new ideas; low evaluation cycle.	Discarded as the design process iterates.
Iterative	Evolves with the design process; explore and refine ideas.	Higher costs and evaluation cycles when compared to rapid approaches.
Evolutionary	Early testing of the product; becomes part or all of the product.	Very high costs and evaluation cycles; challenging to implement; discourages exploration of new solutions.

multidisciplinary team working, which is one of the standards of the UCD methodology (ISO, 1999). Each prototyping tool detailed in this section is categorized according to the dimensions introduced in Section 2.2. This categorization will exclude the representation dimension since all tools produce online prototypes..

2.3.1 *Mock-up editors*

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 (Beaudouin-Lafon and Mackay, 2002). There are several tools of this type that support prototyping currently available in the market. However, since the current project is related to interactive systems, it is more relevant to consider tools that can produce interactive prototypes rather than fixed mock-ups that do not support user interaction. Pencil Project[1], Adobe XD[2], InVision[3] and Figma[4] are some examples of mock-up editors that provide support for this feature and are widely used by the design community.

Pencil Project played a crucial role in the latest version of the IVY Workbench since it was the selected tool for creating prototypes to be imported into IVY. For this reason, it is included in the analysis. The other tree tools are not free or open-source, unlike Pencil. Nevertheless, Adobe XD offers a free plan for prototype creation, even if with limited features. Missing features include cloud storage, access to fonts and unlimited prototype sharing between teams, which have little relevance to this project. Thus, the prototyping tools considered in this comparative analysis are Pencil Project and Adobe XD.

1 http://pencil.evolus.vn/, accessed 06-July-2020
2 https://www.adobe.com/products/xd/details.html, accessed 06-July-2020
3 https://www.invisionapp.com/, accessed 06-July-2020
4 https://www.figma.com/, accessed 06-July-2020

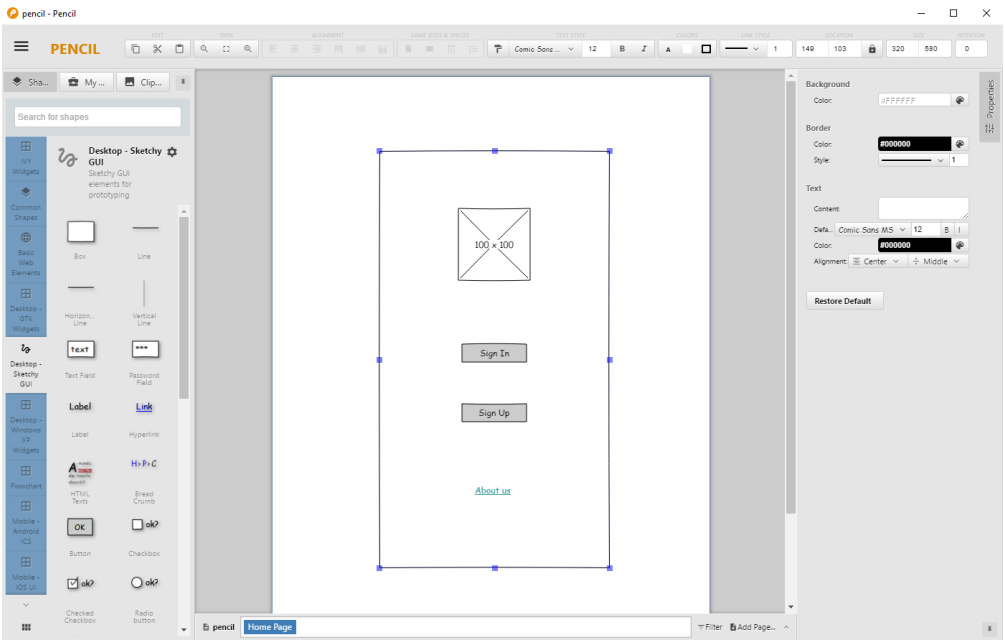


Figure 2.5: Prototyping environment of the Pencil Project. The current prototype represents a home page built with the collection of widgets available on the left panel.

Pencil Project

Pencil is a free and open-source *Graphical User Interface (GUI)* prototyping tool designed for simplicity that provides an extensive set of built-in widgets, like basic drawing shapes, flowchart elements and UI controls. Many of these follow the style of the leading desktop and mobile platforms, allowing designers to develop mock-ups more familiar to users. Furthermore, experienced users can easily create custom widgets and shared them with the Pencil community. Designers can use these widgets and shapes to build wireframes and mock-ups with ease (Figure 2.5).

Additionally, this tool provides support for system navigation by allowing designers to split wireframes or mock-ups into a set of pages. Elements of these pages can then receive click events that trigger the navigation to other pages, thus enabling the simulation of the UI flow. Essentially, this feature allows designers to define static routes that encompass the navigation of a system without any logic associated to them. However, this tool offers no built-in support for running the produced simulations with these features. Instead, designers need to export all pages to a single web page that allows users to interact with the prototype in a browser. Given these features, the prototypes produced with this tool are fixed-path in terms of interactivity.

Designers can quickly create prototypes only with the composition of the widgets provided by the tool. This characteristic makes Pencil Project an adequate choice for rapid and

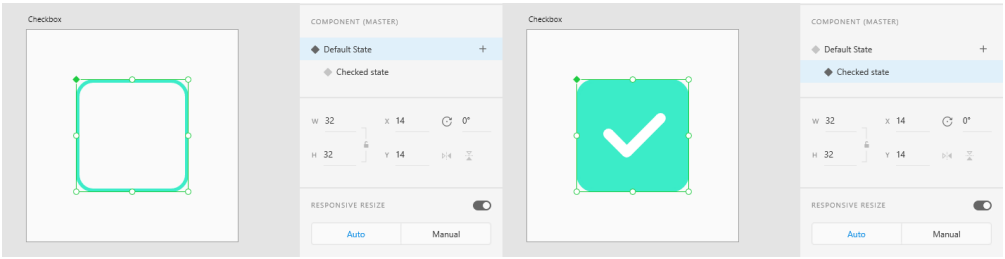


Figure 2.6: Definition of different states for a checkbox component with Adobe XD.

inexpensive prototype building. That said, the widget collection itself can limit designers to achieve highly-detailed mock-ups since the widgets are not fully customizable. Designers should then focus on presenting to users a preview of the system's layout and interaction, rather than its final look. Thus, the prototypes built with this tool fall in the medium-fidelity category. As the design process iterates and the necessity of adding more detail arises, these prototypes will probably be discarded since they cannot evolve into high-fidelity versions. Therefore, Pencil Project is a rapid prototyping tool in terms of prototype evolution.

Adobe XD

Adobe XD (Schwarz, 2017), or Adobe Experience Design, is a tool for highly-detailed mock-up building. It exploits much of the features of other Adobe tools, like Illustrator[5] and Photoshop[6], that allow designers to create mock-ups for the final stages of the design process. This tool offers three main features: design, prototyping, and sharing.

The design feature offers a diverse set of highly customizable elements built to assemble a mock-up of a UI. Additionally, designers can import several widget libraries made by the Adobe XD community to speed up the process of prototype building. Several mock-ups can be created with these features, allowing designers to present each page of the expected UI in an organized fashion. Moreover, it is possible to create reusable components of drawings, allowing several instances to inherit all their base properties. These instances can then add their unique attributes without affecting their base component. Constraints can also be attached to the elements of a page, making them adjust accordingly as the screen resizes, which promotes responsiveness. Another notable design feature is that drawings can have multiple states. A state might represent a hover or a selected status that is triggered by a specific user action, such as a click event, keyboard press or even a voice command. This feature helps designers to define several states of a specific component without having to redraw the entire mock-up multiple times. Figure 2.6 depicts one example of this feature.

5 https://www.adobe.com/products/illustrator.html, accessed 14-January-2020
6 https://www.adobe.com/products/photoshop.html, accessed 14-January-2020

The prototyping feature enables the creation of prototypes that support user interaction by allowing pages to be connected, in order to simulate the flow of the UI. Each interaction reacts to the same triggers defined with the design component of this tool. Although this navigation has no logic associated with it, designers can create several animations between the pages, thence achieving prototypes that closely resemble the final version of the UI. Furthermore, this tool offers built-in support for running the defined prototype simulations. The tool also allows users to export prototypes to *Hyper Text Markup Language (HTML)*. With this feature, the tool transforms the internal SVG structure into HTML and *Cascading Style Sheets (CSS)* code. However, this code is not optimized since the generated CSS most often relies on absolute position properties. Besides, the exported results do not offer any internal logic. Lastly, the Adobe XD mobile app available in iOS and Android allows users to test the created prototypes in their mobile devices.

The sharing feature encourages teamwork by allowing designers to share a link of the produced prototype and all its evolution history. Other team members can then proactively add reviews and contribute new ideas to improve the product.

Adobe XD offers support for building high-fidelity prototypes. However, it also allows designers to create low-fidelity versions for the early stages of the design process. The level of fidelity of these versions can increase as the process iterates, making Adobe XD a tool suitable for iterative prototyping approaches. Although this tool offers code exporting functionalities to HTML and CSS, it is not an evolutionary prototyping tool since its prototypes do not provide any logic. Even so, this tool can produce fixed-path prototypes that support several user event triggers, navigation animations and multiple screen dimensions support.

2.3.2 *Model-based tools*

Model-based analysis tools enable user-centred design methods that integrate formal verification technologies, facilitating early detection of underlying user interface problems (Szekely et al., 1996). This section includes three model-based tools: CIRCUS, PVSio-Web, and IVY Workbench. These tools were chosen because each tool covers distinctive UI issues, supports UCD methodology and has prototyping, verification and validation capabilities (Campos et al., 2020).

CIRCUS

CIRCUS (Fayollas et al., 2014), which stands for Computer-aided-design of Interactive, Resilient, Critical and Usable Systems is a development environment that embeds both system and task modelling functionalities. This environment helps to achieve the design and development of interactive critical systems, and it is best suited for software engineers, system designers and human factors specialists. CIRCUS features range from the formal

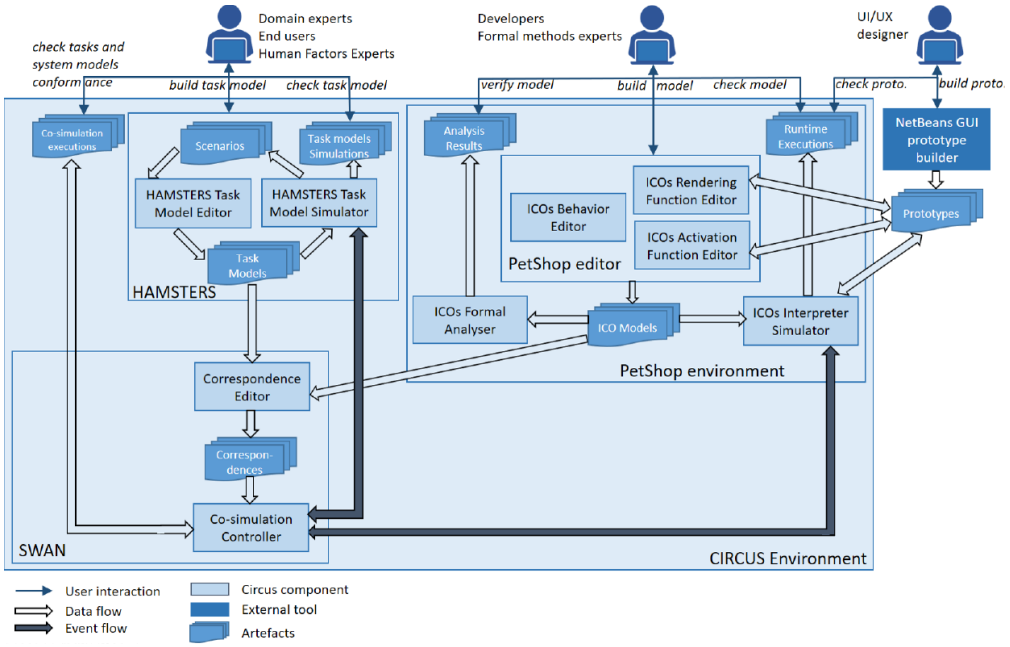


Figure 2.7: Architecture of the CIRCUS environment (source: Campos et al. (2020)).

verification of the system's behaviour to the assessment of compatibility between users' tasks and that behaviour. The environment, as illustrated in Figure 2.7, integrates three tools:

- **HAMSTERS** (Fayollas et al., 2014) allows the editing and simulation of task models, ensuring consistency, coherence, and conformity amidst users' tasks and the sequence of actions necessary to operate interactive systems. It offers a graphical notation to structure users' goals and sub-goals into hierarchical task models, while also denoting temporal operators that express qualitative temporal relationships between tasks.

- **PetShop** (Palanque et al., 2009) is a tool for system modelling and prototyping, where developers specify the behaviour and the appearance of interactive systems, using the *Interactive Cooperative Objects (ICO)* notation (Navarre et al., 2001). ICO uses object-oriented concepts to define the structural or static features of the system. In this tool, prototypes are an assembled set of objects featuring four components: behaviour, presentation, and two functions (activation and rendering) that make the bridge between the cooperative object and the presentation component. Finally, the presentation component is built using Java widget libraries or by invoking Java code dedicated to it.

- **SWAN** (Campos et al., 2020) is a tool for the co-execution of PetShop and HAMSTERS models, allowing developers to establish correspondences between tasks and system behaviours.

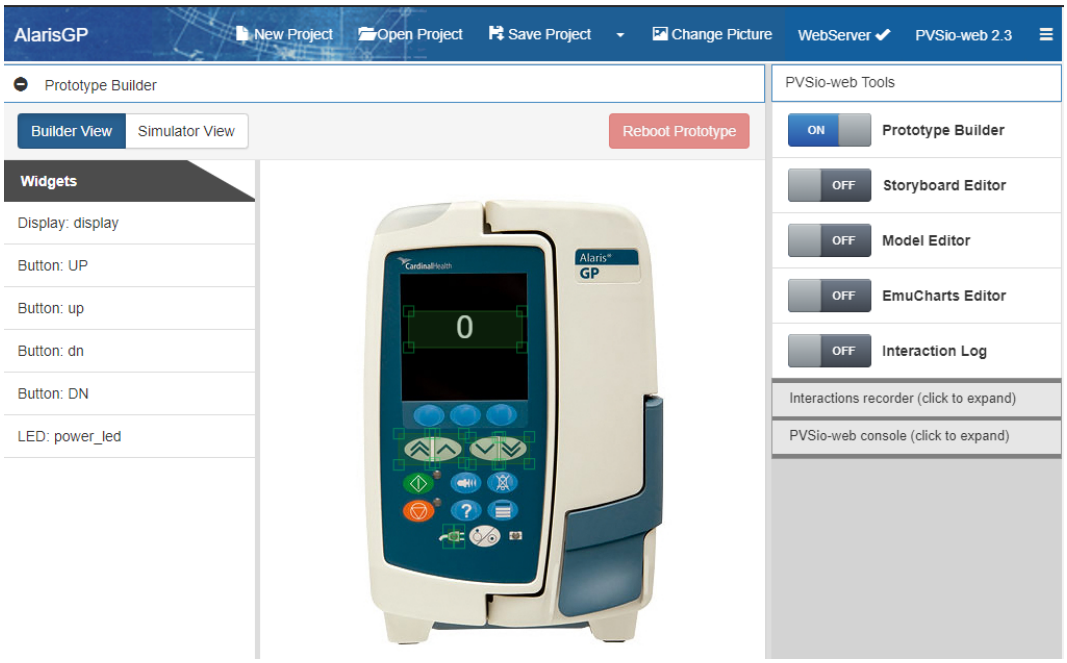


Figure 2.8: Prototyping environment of the PVSio-Web. Green areas over the prototype image represent the widgets depicted on the left panel. The right panel lists all the available environments of this tool.

The CIRCUS environment produces prototypes designed to evolve into a part or all of the system. These prototypes fit in the evolutionary category by following the ICO methodology that is adaptable to the Model-View-Controller design pattern (Navarre et al., 2000). Finally, this tool uses interface builders which makes it possible to design both high-fidelity and open prototypes.

PVSio-Web

PVSio-Web[7] provides a formal method based, graphical front-end for prototyping and modelling interactive systems (Masci et al., 2015). Users can rapidly generate realistic prototypes by uploading an image representing the UI of the system into this tool. Afterwards, they can define programmable areas over the image and link them to the formal model that specifies the human-system interaction logic (Mauro et al., 2016).

This tool combines multiple development environments aimed for multidisciplinary teams of UI engineers, domain experts and software analysts. These environments allow users with different backgrounds and expertise levels to cooperate with the same underlying formal models. There are seven fundamental environments incorporated into this tool:

7 http://www.pvsioweb.org/, accessed 14-January-2020

- The **Prototype builder** provides a graphical environment with functions for defining the visual aspect of the prototype, as well as for creating programmable overlay areas that enable user interaction with the prototype. These overlay areas correspond to two types of widgets: input and output widgets. Input widgets, react to user actions and translate them into formal model expressions. By contrast, output widgets illustrate how to render the internal prototype behaviour (Figure 2.8).

- The **Simulator** executes the developed prototypes and logs user interactions with them. This environment translates user actions over the input widgets into model expressions, evaluates these expressions and renders all the returned results into output widgets on the Web browser.

- The **Storyboard editor** facilitates preliminary mock-up prototype development. It allows developers to load images representing different screens, and to define input widgets on these images. Transitions between screens can then be made by linking these widgets with user actions.

- The **EmuCharts editor** is a visual editor and code generator for creating executable formal models, using a graphical notation based on Statecharts (Harel, 1987). With this module, developers can define states, transitions, variables representing relevant properties of the system, and generate executable models from the visual diagram.

- The **Model editor** is a component for editing formal models, providing syntax highlighting, auto-completion, search, and compile functionalities. It also incorporates a file browser that allows developers to select, rename, delete, and create files and directories.

- The **Property providing assistant** includes the PVSio environment and the *Prototype Verification System (PVS)* (Owre et al., 1992) theorem prover. PVSio is used during simulations to evaluate PVS expressions generated by the **Simulator** environment. Whereas PVS offers a formal analysis of use-related safety properties of the prototype.

- The **Co-Simulator engine** creates a communication infrastructure that enables the exchange of simulation events and data between PVS models and other models developed with different simulation frameworks, like Simulink[8] (Masci et al., 2014).

PVSio-Web produces **high-fidelity prototypes** since it allows developers to import images from any source that represent the visual appearance of UI. In terms of interactivity, the prototypes created with this toolkit provide large sets of user interaction coupled with internal logic associated with the formal model, therefore falling in the **open prototype**

8 https://www.mathworks.com/products/simulink.html, accessed 14-January-2020

category. Finally, this toolkit designs **iterative prototypes** intending to evolve as the design process iterates without becoming part or all of the system.

IVY Workbench

The IVY Workbench (Campos et al., 2016) supports the development of models of interactive systems, the formulation of required properties of the behaviour of these systems, and their verification through the NuSMV model checker (Cimatti et al., 2002). The tool generates counter-examples acting as scenarios for analysis when this verification fails.

Models follow the *Modal Action Logic (MAL)* interactor language, which describes how available actions change the state of the system (Ryan et al., 1991; Couto and Campos, 2019). The MAL model describes the structure of the system with a set of *attributes*, possible *actions* and *rules* expressing its behaviour (*axioms*).

IVY is designed for simplicity, aiming to provide representation and analysis tools easily usable by interface developers and to communicate results effectively within an inter-disciplinary team of software engineers and formal method experts. Lastly, it adopts a plugin-based architecture to establish a flexible development environment (Couto and Campos, 2019). In its current version (2), the tool includes five interoperable plugins:

- The **MAL editor** supports the usual editing facilities like syntax highlighting, code completion, undo/redo, and cut and paste. The editor also presents the model as a tree view in a side panel, that enables easy navigation of the model's structure.
- The **Properties editor** supports the formulation of properties of the model, assisting with pattern and template selection. This editor is useful for the verification of assumptions about the expected system's behaviour (Campos and Harrison, 2008). The verification step produces counter-examples when any property fails, that can be later analysed.
- The **Traces Analyser** offers a visual representation of the produced counter-examples by the verification step when a property fails. It makes it possible to analyse the sequence of actions that prove the falseness of the failed property, and therefore explore alternative paths.
- The **Animator plugin** allows users to choose the sequence of executed actions, starting from the initial state of the model. It implements two representations: tabular and state-based. The tabular one uses columns to represent states and lines for actions and attributes. The state-based one represents each interactor as a lifeline. Each lifeline is the sequence of states of the execution trace for that interactor.
- The **Prototyper plugin** focuses on the design of the prototype's UI (Araújo et al., 2019; Araújo, 2019). It operates in two different modes: edit and animate modes.

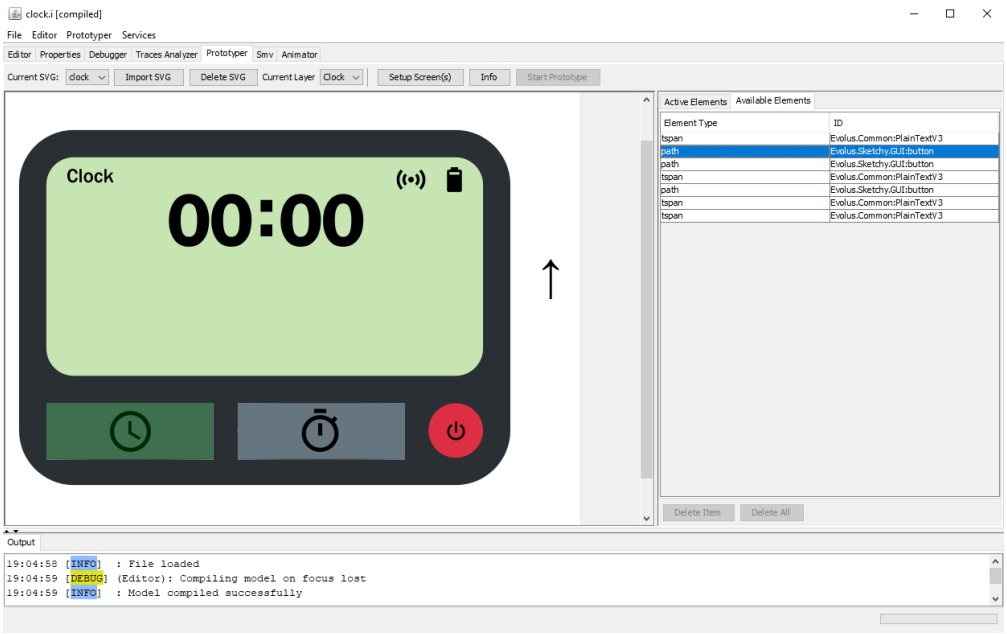


Figure 2.9: Representation of a device with clock and chronometer modes, built with the prototyping plugin of the IVY Workbench. The right panel presents the interactable elements available for assigning actions. Any interaction with this panel reflects in the prototype (the dark green highlight over the clock icon).

The edit mode allows designers to import the visual representation of the system, as one or multiple SVG files representing several screens, and link this representation to the behaviour defined with the MAL editor. Then, the plugin identifies all the interactable elements of the SVG file, allowing a map between these elements and the model's actions and attributes. Actions associated with elements allow further user interaction with the prototype, supporting mouse clicks and keyboard press events. Attributes of the model can either be presented with textual elements or toggle the several prototype's screens according to their values. The animate mode simulates the user interaction with the prototype by following all the configurations made in the edit mode. In the running animation, users can either interact with the prototype itself or select between the available actions of the model presented in a side panel (Figure 2.9).

The IVY Workbench allows developers to define the behaviour and visual aspect of the prototype's UI and therefore build **open prototypes**. The prototypes are **iterative** since both the model and the UI components used by this tool can evolve as the design process iterates but not become part of the real system. Finally, the latest version of this tool is dependent on the Pencil Project exported SVG files to identify interactable elements and only offers partial support to other sources. Thus, like Pencil, the prototypes built with this tool have **medium-fidelity** precision.

Table 2.2: Categorization of the studied prototyping tools according to precision, interactivity and evolution.

Tool	Precision	Interactivity	Evolution
Pencil Project	Medium-fidelity	Fixed-path	Rapid
Adobe XD	High-fidelity	Fixed-path	Iterative
CIRCUS	High-fidelity	Open	Evolutionary
PVSio-Web	High-fidelity	Open	Iterative
IVY Workbench	Medium-fidelity	Open	Iterative

2.4 SUMMARY

UCD places the user at the centre of the design process, from the initial analysis of user requirements to testing and evaluation. This approach improves user acceptance and satisfaction by making systems easier to understand and use, being therefore crucial for safety-critical domains.

Prototyping is an iterative process, essential for interactive systems design. Prototypes support UCD by allowing users and designers to experience earlier versions of the real system, identify potential usability and functional issues and to improve the design accordingly. They may take several forms, provide highly-detailed representations of the system, and have different life cycles according to their goals.

This chapter described two types of tools that support prototyping: mock-up editors and model-based tools. The first type of tools focuses on the visual representation of the system. The second one excels for critical systems design by prototyping the UI and its core behaviour. The studied tools serve different purposes in the design process, being therefore complementary rather than competitive. Each one of the mentioned tools was categorized in Sections 2.3.1 and 2.3.2 according to the prototyping dimensions introduced in Section 2.2. Table 2.2 summarizes this categorization.

Pencil Project is a tool best-suited for earlier stages of the design process, producing medium-fidelity prototypes inexpensively and with ease, capable of representing the overall look of the system's UI. Whereas, Adobe XD allows designers to build high-fidelity prototypes with large sets of user interaction. CIRCUS defines both the UI and its behaviour and aims for the production of evolutionary prototypes meant to become part of the real system. Therefore, each iteration of the design process with this tool is expensive both in skill and time. PVSio-Web offers a flexible prototyping environment, useful for a multidisciplinary team. IVY Workbench, like CIRCUS and PVSio-Web, creates prototypes by defining the system's representation with its core usability features. Moreover, it offers a more automated analysis through model checking, requiring lesser levels of formal methods expertise (Araújo et al., 2019; Campos et al., 2020) when compared to the other studied model-based tools.

Prototyping support in the IVY Workbench has some downfalls like Pencil Project dependency, requiring exported SVG files from this tool to further identify user interactable elements. Consequently, the prototypes achieved with IVY have medium-fidelity precision. A combined approach between the formal methodologies inherent to this tool and the prototyping features related to mock-up editors that are familiar to non-experts could potentially promote its multidisciplinarity. Lastly, the current version of IVY has some limited user interaction with the produced prototypes when compared to CIRCUS and PVSio-Web. One plausible method for solving this limitation is to develop a library of dynamic widgets.

The next chapter describes the features of the studied tools that may be attractive for the enhancement of the prototyping capabilities of IVY and introduces the requirements and technologies of the dynamic widget library.

3

REQUIREMENTS OF INTERFACE PROTOTYPING

This chapter describes the potential features of the prototyping tools detailed in Chapter 2 that can considerably improve the prototyping capabilities of the IVY Workbench. Furthermore, the chapter describes the architecture of IVY's prototyping plugin and identifies its limitations. From the features of the other tools and the identified issues, a list of requirements is derived, in conjunction with the technologies needed to fulfil them.

3.1 PROTOTYPING TOOLS REVISITED

Besides the IVY Workbench, Chapter 2 detailed other tools that support prototyping: CIRCUS and PVSio-Web as model-based tools; Pencil Project and Adobe XD as mock-up editors. Each one of these tools has its specific prototyping features and covers different concerns intrinsic to the design process. This section revisits these tools to identify potential features to improve the prototyping capabilities of IVY.

3.1.1 *Pencil Project*

Pencil Project is easy to use and rapidly produces medium-fidelity prototypes. One of its relevant features is offering support for the development of widget libraries. Designers can then import these libraries into the tool and add widgets to the prototype. Hereupon, a library of dynamic widgets made for IVY should support being imported into Pencil, allowing designers to produce prototypes with dynamic behaviour quickly and later import them into IVY.

3.1.2 *Adobe XD*

Adobe XD supports the reuse of components, allowing designers to define multiple states for a specific element of the prototype, something which is a relevant feature to include in the plugin, as it would enhance the functionality and usability of the prototypes. Another

striking feature of this tool is its capability to run prototypes in Android and iOS platforms, through a companion mobile application. This feature allows users to test and validate dynamic prototypes on their mobile devices. Lastly, the transformation of SVG layouts into file formats such as HTML is a prominent feature to add to the IVY Workbench. However, the development of such features is out of the scope of this project and left for future work.

3.1.3 *PVSio-Web*

PVSio-Web supports dynamic behaviour in the prototypes by providing widgets with scripting functionalities. However, this tool uses static raster images as the source format of its prototypes' mock-ups. This hinders a tight integration with mock-ups developed by designers. Since raster images do not provide any internal structure, users must define the widgets that specify the behaviour component of a prototype in PVSio-Web explicitly. On the contrary, IVY takes more advantage of the output of the mock-up editors by working with SVG files that allow IVY users to import mock-ups. This feature correlates with the main goal of the new prototyping approach of the IVY tool: the creation of a framework of dynamic widgets. Another relevant feature of this tool is how it assists users with mapping the model to the prototype. It automatically assigns widgets' events to actions in the model sharing the same name. Once again, this is only useful for domain experts, aware of the system's model. The IVY Workbench could assume an improvement of this feature by assisting users without any knowledge of the model with auto-complete functionalities or listings of the available actions accompanied by their documentation.

3.1.4 *CIRCUS*

Lastly, CIRCUS aims to produce prototypes designed to evolve into the real system, consequently requiring skilled programmers and domain experts. Evolutionary features, such as code generation are of great interest in the improvement of IVY capabilities.

3.1.5 *IVY's Prototyping Plugin*

The current version of the prototyping plugin of the IVY Workbench successfully manages the mapping of the actions and attributes of the formal model with the visual representation of the UI. The overall architecture of the plugin, depicted in Figure 3.1, is divided in two packages: *Gui* and *Backend*.

The *Gui* package contains classes that define the UI of the plugin. The *Backend* package contains the classes that hold the internal infrastructure of the plugin. The plugin uses the *Parser* class to process SVG documents and extract the elements that can receive configu-

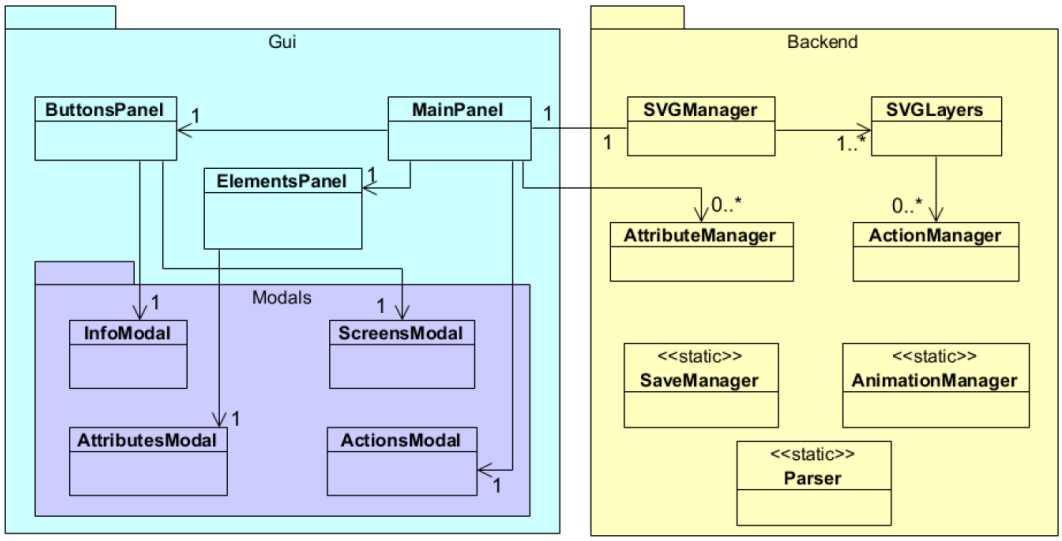


Figure 3.1: Architecture of the current version of IVY's prototyping plugin (Araújo, 2019).

rations. These elements are identified by SVG metadata originated by the Pencil Project tool. Currently, the parsing algorithm only extracts elements marked as labels or buttons. Furthermore, the *Parser* verifies the existence of SVG layers in the imported document. This verification is also performed by analysing Pencil metadata in SVG group elements. The imported SVG document is then divided into the identified layers that are stored in the *SVGLayers* class. Lastly, other relevant classes of the architecture are the *AttributeManager* and the *ActionManager* classes. These classes hold the mapping between the extracted elements and the attributes and actions of the formal model, respectively. This mapping only includes the association between attributes and labels, and the association between actions and buttons.

The plugin allows the importing of mock-ups in the SVG format, aiming to offer designers the possibility to work in their preferred mock-up editor tools. However, the current version is dependent on SVG files originated from Pencil Project. This dependency is an issue for three reasons. First, it limits designers to one tool. Second, it causes IVY prototypes to share the same definitions of the medium-fidelity prototypes of the Pencil Project, which prevents the achievement of high-fidelity versions. Lastly, as the Pencil Project evolves and future updates are released, so can its SVG file format change. This can lead to the entire rewrite of the parsing algorithm, as it forces the contributors of the IVY plugin to always keep up with the updates of the mock-ups tool.

Besides the mentioned dependency, there are opportunities for further enhancements of the plugin's prototyping environment, in particular since it does not support dynamic widgets. Currently, if a user wants to define multiple states of a specific element of a page, he or she must replicate the design of that page for each of those states. Each replicated page is then associated with a layer by the described parser. To achieve the desired results,

users need to perform configurations in each generated layer. Using the plugin's multi-page support, users can then toggle between the several layers by assigning values of the model to them. This feature, however, is an inefficient and tedious solution, especially when prototyping large and complex interfaces that contain several states. In such a case scenario, the prototyping plugin will require the configuration of multiple layers to achieve the desired results. The new version advocates for a new approach where it is possible to introduce dynamic behaviour to the elements of the prototype, preventing page replication and improving the quality of the produced prototypes.

Lastly, the mapping of the attributes and actions of the formal model is also minimal, since it only allows the association of actions with buttons and attributes with labels. However, there could be cases where it is profitable to assign both actions and attributes of the formal model to any SVG element. Moreover, this approach does not offer a clear distinction between these two types of associations.

To summarise, the issues of the current version of the prototyping plugin are the following:

- Dependency of an external tool.
- Does not escalate well as the complexity of prototypes increases.
- Limited set of SVG elements that can receive configurations.
- Poor mapping between elements and the actions and attributes of the formal model.

3.2 REQUIREMENTS

As mentioned before, the current version of the IVY Workbench has some downfalls like the dependency on Pencil Project. Furthermore, there are some features of alternative prototyping tools capable of enhancing the prototyping abilities of IVY. The following list of requirements derives from these subjects:

1. **The prototyping plugin must allow designers to import SVG files from multiple sources.** SVG files can have distinct internal structures according to the mock-up editors that generated them. The current version of IVY can only extract and work with specific elements present in Pencil-generated SVG files. This requirement aims to solve this dependency. The goal is to produce a parsing algorithm able to extract all the elements of a SVG file generically. This improvement will allow designers to work in their favourite tools.
2. **The plugin must be capable of producing high-fidelity prototypes.** This feature is an addition to the first requirement. The prototypes achieved with Pencil Project have a medium-fidelity precision since they present the chief aspects of their UI layout and

navigation but not the final look and feel. Therefore, the parsing algorithm must be able to extract elements from high-fidelity sources as well.

3. **The plugin must support a framework of dynamic widgets.** This framework should be capable of considerably enhancing the usability of the produced prototypes. The widgets should react to every single change made in the values of attributes and actions of the model, modifying the visual appearance of prototypes accordingly. Moreover, this framework should assist in multi-page prototyping designs. The creation of dynamic widgets supporting several states can prevent page replication, providing, therefore, a more intuitive design experience. Finally, these widgets should considerably improve the *User Experience (UX)* by introducing large sets of user interactions that closely resembles the real system.

4. **The widgets should be as generic as possible and not only applicable to a specific model.** The goal is to develop widgets able to introduce dynamic behaviour to prototypes independently of their formal models. The more generic the widgets are, the less the probability of building new widgets for each created formal model. However, there could be some situations where more advanced and model-specific widgets are required to achieve the desired results. For example, widgets such as checkboxes or toggle buttons can be used for developing prototypes of a diverse set of mobile applications, whereas medical devices such as heart rate monitors require more specific widgets to define their functionalities.

5. **The plugin must be intuitive and made for non-expert users.** The plugin must provide the means for users to easily combine the visual appearance of the prototype with the formal analysis of the model. This process can assist users with hints about the correct attributes to be sent to the widgets. Furthermore, all the functionalities of the widgets should provide understandable documentation to users and therefore guide them to the correct building of prototypes. Lastly, usability tests of the plugin should be conducted with non-expert users to evaluate their acceptance and validate the new prototyping features.

6. **The prototypes should be compatible with multiple platforms.** The plugin should be able to export prototypes in a format compatible with web and mobile platforms to promote usability.

3.3 TECHNOLOGIES

The IVY Workbench offers support for UI prototyping by allowing designers to import files in the SVG format, access their elements and attach attributes and actions to them. This

Figure 3.2: Shape associated with the SVG document of Listing 3.1.

section describes the base technologies surrounding the prototyping plugin as developed by Araújo (2019) (see also Araújo et al., 2019) that are still relevant to the new planned features. It also details the new technologies required to fulfil the list of requirements enumerated in Section 3.2.

3.3.1 *SVG*

SVG[1] is a free, open and standardized file format for vector graphics, developed and maintained by the *World Wide Web Consortium (W3C)*, which is the foremost international standards organization for the web. It is based on *XML (Extensible Markup Language)* and explicitly designed to work with other web standards such as the DOM[2]. This file format provides three types of graphic objects: vector graphic shapes (such as paths consisting of straight lines and curves), images and text. These graphical objects offer a diverse set of operations such as grouping, styling, transforming and composition.

SVG usage has many benefits over raster image formats like *PNG (Portable Network Graphics)* or *JPEG (Joint Photographic Experts Group)*. The chief advantage is that SVG images do not suffer a quality loss when scaled or zoomed. These files are also human-readable, easily created, compressed, scripted, and edited with a text editor. Finally, they can store a broad set of settings by embedding editor metadata[3].

It is possible to introduce dynamic and interactive behaviours to SVG drawings. Animations can be defined and triggered by embedding a set of components into the SVG content. Another option is the use of a supplemental scripting language to manipulate the SVG DOM, which provides complete access to all elements, attributes and properties.

1 https://www.w3.org/TR/SVG/, accessed 14-January-2020
2 https://developer.mozilla.org/en-US/docs/Web/SVG, accessed 14-January-2020
3 https://inkscape.org/pt/develop/about-svg/, accessed 14-January-2020

A basic example of a SVG shape is illustrated in Figure 3.2, while the code that generated it is represented in Listing 3.1. In this example, a *rect* and a *circle* are defined inside a *defs* tag. This tag does not render any particular shape. Instead, it stores SVG tags as constants. Later, several instances of those shapes are created with *use* tags which allow SVG elements duplication and the deep cloning of all their attributes. It is, however, important to note, that new attributes can be added to *use* tags, such as the transform attribute, as depicted in the bottom half of the example. Other relevant aspects of the illustrated example are the *g* elements and namespace attributes. *g* elements allow SVG object grouping that can be later referenced by *use* elements. Moreover, children objects inside *g* elements inherit any attributes or transformations applied to them. Namespaces (in the example the *xmlns* namespace is declared) allow additional metadata such as attributes or element tags to be inserted into the SVG document. This SVG property is handy for the development of the dynamic widget framework by providing support for inserting additional widget information inside any SVG document.

Listing 3.1: Example of a SVG document.

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org
    /1999/xlink" viewBox="0 0 125.06 96.12">
  <defs>
    <rect id="round-rect" x="40.06" y="11" width="69" height="14" rx="7"
       ry="7" transform="translate(-2.12 19.91) rotate(-15)" fill="#7397
       f1"/>
    <circle id="ball" cx="118.06" cy="6.5" r="7" fill="#7397f1"/>
  </defs>
  <g id="group" data-name="Layer 1">
    <circle cx="51.61" cy="49.28" r="45" fill="#57c7d5"/>
    <text transform="translate(22.75 54.77)" style="font-size:21px;fill:#
       fff;font-family:Consolas-Bold, Consolas;font-weight:700">&amp;lt;svg&amp;
       gt;</text>
    <use xlink:href="#round-rect"/>
    <use xlink:href="#ball"/>
    <use xlink:href="#round-rect" transform="translate(-40 65)"/>
    <use xlink:href="#ball" transform="translate(-107 60)"/>
    <use xlink:href="#ball" transform="translate(-90 23)"/>
    <use xlink:href="#ball" transform="translate(-24 33)"/>
  </g>
</svg>
```

SVG is the current file format used in the IVY Workbench prototyping plugin. Several design platforms use this file format which is an advantage since one of the goals of IVY is to work with a vast diversity of mock-up editors and consequently adapt to a broader design community. Designers can then import files of this format into IVY for adding the dynamic behaviour to mock-ups, leading to more interactive and realistic prototypes Furthermore, its ability to embed scripts is a fitting feature for the development of the framework of

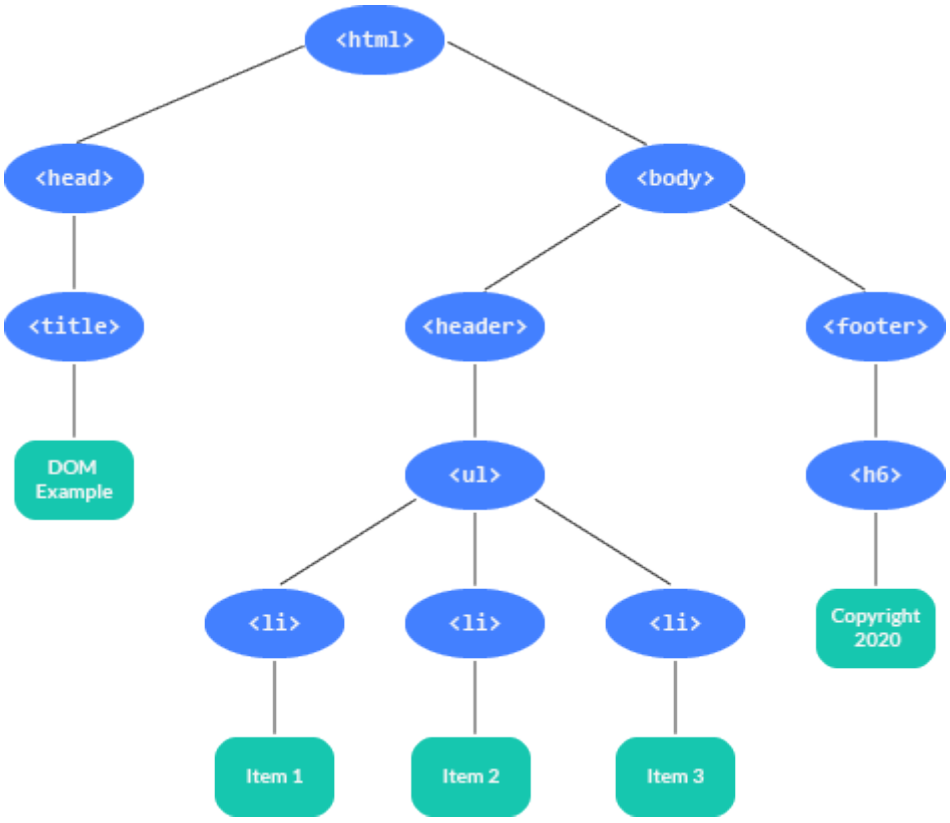


Figure 3.3: DOM tree associated with the HTML document of Listing 3.2.

dynamic widgets. Lastly, the SVG is a file format that is compatible with multiple platforms, such as web browsers and mobile devices, which makes it suitable for the fulfilment of the cross-compatibility requirement.

3.3.2 *DOM*

The DOM (Jakobson, 2014) is a language-independent *Application Programming Interface (API)* intended for use with *HTML (Hyper Text Markup Language)* or XML. The DOM API represents a document in the indicated formats as a logical tree, allowing application programs to change its structure, style and content (by changing, deleting, or adding elements) or even add event handlers to its elements that react to user input events such as mouse clicks. Each DOM tree has a root node that subsequently has one or more leaf nodes, with zero or more intermediates, that can be easily accessed by attributes such as the *id* or tag names. This feature provides an effective way to traverse all the elements contained in a document for further manipulation of their properties. Figure 3.3 illustrates an example of the generated DOM tree of an HTML document (depicted in Listing 3.2).

Listing 3.2: Example of a HTML document.

```
<html>
  <head>
    <title>DOM example</title>
  </head>
  <body>
    <header>
      <ul>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
      </ul>
    </header>
    <footer>
      <h6>Copyright 2020</h6>
    </footer>
  </body>
</html>
```

The IVY Workbench uses this technology for retrieving elements of the SVG document of a prototype. These elements can then be linked to attributes of the formal model and receive user input events. It also allows for the development of traditional mock-up editors features such as the addition and removal of SVG elements. Other features, such as the changing of the visibility of an element, can be implemented as well. Lastly, it enables the manipulation of the prototype's appearance during its simulation. Therefore, the DOM technology emerges as one of the core components of the prototyping features of the IVY Workbench.

3.3.3 *Apache Batik*

Apache Batik[4] is a Java-based toolkit for SVG manipulation that is the core of the Prototyper plugin for offering support for SVG importing and rendering. It also provides a Java implementation of the SVG DOM with its typical features. Studies conducted by Araújo (2019) (see also Araújo et al., 2019) concluded that this toolkit was the best-suited choice for the prototyping features of the IVY Workbench, leading to its inclusion in the creation of the Prototyper plugin. This reason serves, therefore, as a motivation for its continued use.

4 https://xmlgraphics.apache.org/batik/, accessed 19-January-2020

3.3.4 *Rhino*

Rhino[5] is an open-source implementation of JavaScript written entirely in Java, usually embedded into Java applications to provide scripting to end-users. Furthermore, it allows the implementation of Java interfaces, as well as the extension of Java classes, with JavaScript objects. This library plays a crucial role in the development of the framework of dynamic widgets by providing support for scripting evaluation, execution of JavaScript functions with a list of parameters, and features for passing Java objects into the scripting environment. However, Rhino is an implementation of the JavaScript core language, designed to be used in server-side or desktop applications, hence lacking the built-in support for document manipulation, since the DOM is a browser technology. Still, Rhino offers methods for converting Java objects into JavaScript objects. Therefore, the scripting environment provided by Rhino can receive the DOM implementation generated by the Apache Batik to access and modify its structure.

There are other currently available scripting libraries, such as Nashorn[6]. Nonetheless, the Apache Batik library requires the use of Rhino at runtime when rendering SVG documents that contain scripts. This, combined with the fact that Rhino provides the required features for the framework development, serve as a motivation for its usage.

3.4 SUMMARY

The current version of the IVY Workbench has some prototyping downfalls, for instance, the dependency on Pencil Project and limited prototyping behaviour since it does not support widgets. This chapter evaluated potential features of other prototyping alternatives that could considerably enhance the IVY tool. PVSio-Web includes dynamic widgets and user assistance when creating prototypes. The Pencil Project provides the means for creating widget libraries that allow non-expert users to build prototypes. Adobe XD supports the definition of multiple states of the prototype, as well as the simulation of prototypes on mobile devices. Lastly, CIRCUS offers support for evolutionary prototypes. These features rise as striking complements to include in the IVY Workbench.

The current chapter also enumerated a list of requirements of the new features and described the technologies required to fulfil them. The SVG file format offers the opportunity for designers to import into IVY prototypes originated by their preferred tools. However, mock-up editors can produce SVG files with distinct internal structures. The addition of dynamic widgets to mock-ups aggravates this subject even further because mock-up editors can modify the internal structure of SVG files and consequently remove relevant information

5 https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino, accessed 14-January-2020
6 https://www.oracle.com/technical-resources/articles/java/jf14-nashorn.html, accessed 14-July-2020

about its widgets. Thus, the new IVY version should present a solution that makes it compatible with a wide variety of mock-up editors. Apache Batik is used in this process, being capable of generating a Java version of the DOM originated from the SVG structure. This library is also responsible for reading SVG files for their subsequent rendering in the prototyping plugin. Dynamic widgets can be added to prototypes by including scripts containing intended functionalities in SVG files. Lastly, the Rhino library is the component that completes the framework since it can evaluate the scripts received in the prototype and execute their functions. By converting the Java DOM into a JavaScript object, it grants the IVY widgets the capabilities to offer scripting functions that can modify the SVG content of the prototype during its simulation.

The following chapter focuses on the development of the new features that will fulfil all the proposed requirements, detailing the process of generic SVG parsing, the creation of the dynamic widget framework and all the new implemented functionalities.

4

THE PROTOTYPING PLUGIN

The new version of the IVY Workbench has improvements in its prototyping plugin to enhance the fidelity and user interaction of its prototypes. This chapter describes the development process of the new features, which fulfil the requirements put forward in Chapter 3 and solve the issues of the previous prototyping plugin, described in Section 3.1.5. The chapter starts by describing the developed approach to prototypes' building, as well as the workflow and the new internal architecture of the plugin. Next, it details two fundamental components of the plugin: the generic parser and the dynamic widget framework. Lastly, the chapter ends with a description of the added features.

4.1 APPROACH

The prototypes of IVY Workbench have two major components: the formal model of the interactive system and the UI mock-up. To build these prototypes, one must bring together these two components by configuring two kinds of mappings: events and states. Figure 4.1 presents a schematic version of this process.

Events specify how the prototype responds to user interactions and internally execute actions of the formal model. By contrast, states relate to the attributes of the formal model and specify how the prototype reacts to changes in this model. Each element of the mock-up has a set of properties that compose its state. This set varies according to the element type. A state property defines a SVG parameter or a widget parameter value that can either be controlled by a constant value or a model's attribute. Sections 4.5.2 and 4.6.3 describe these configurations in more detail.

During the prototype simulation, the user interaction with the prototype triggers the defined events. Each event executes its mapped action on the model leading to the update of the model's attributes. Consequently, the configured states react to these changes in the model and update their properties accordingly. These properties are then passed as parameters to methods that update the prototype appearance to achieve the desired results.

This new approach offers a clear distinction between the configurations of the actions and attributes of the model when compared to the old version described in Section 3.1.5.

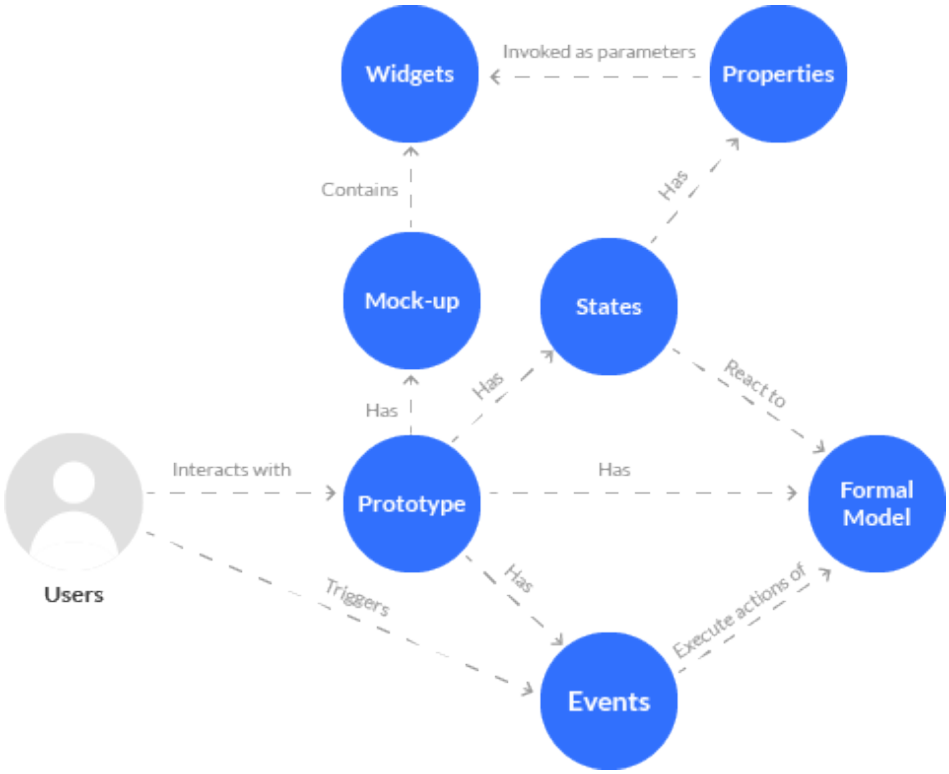


Figure 4.1: Schematic representation of the primary components of IVY Workbench prototypes.

Moreover, it provides a more robust set of configurations, since each element of a mock-up can be associated with actions and attributes of the formal model, by configuring their states and events. Lastly, the developed approach takes the prototyping capabilities of the plugin one step further by adding the possibility to configure SVG properties.

4.2 WORKFLOW

The developed prototyping plugin complements the IVY Workbench with prototypes' animation capabilities. To be appropriately used, users should load a formal model of the interactive system and the UI mock-up into the tool.

When a user imports the mock-up of the UI, the plugin uses its generic parser to process it and the built-in scripting environment to initialize its dynamic widgets. After this initialization, the plugin renders the prototype. Users can then configure the prototype by defining its events and states with the mapping between the mock-up and the actions and attributes of the model. Lastly, users can start the prototype simulation that receives all the configurations made. Users can repeat the tasks of the mentioned workflow process if the

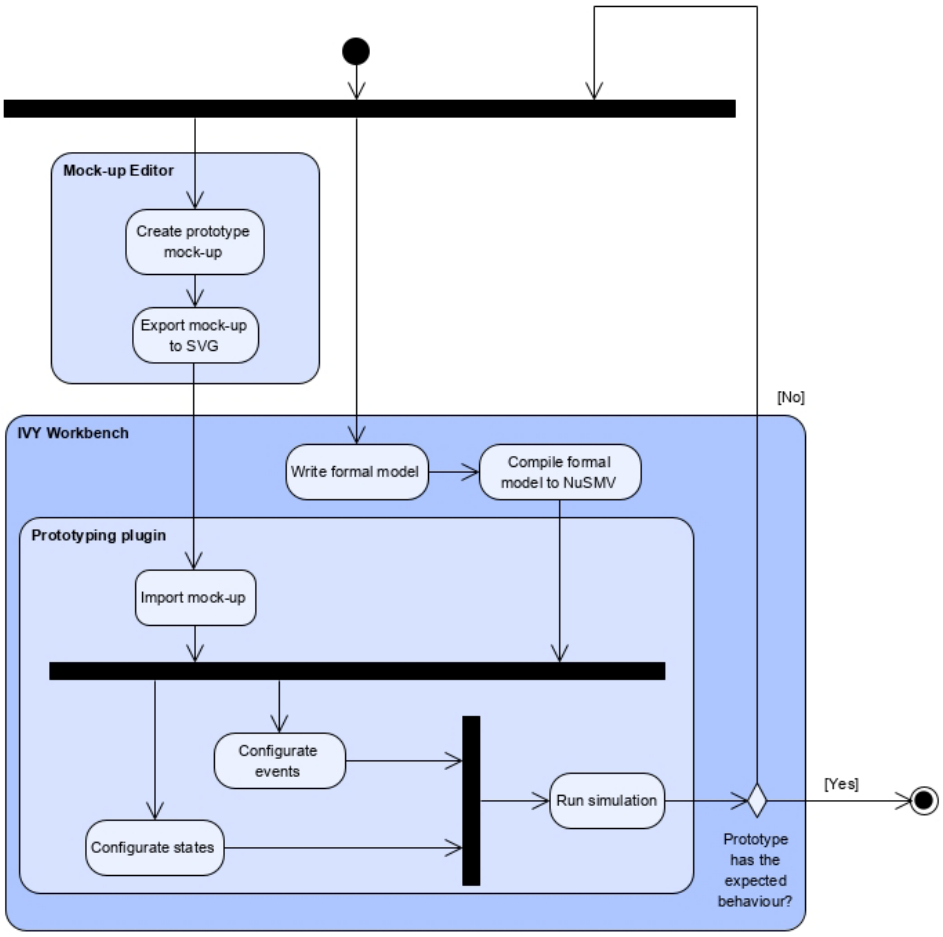


Figure 4.2: Activity diagram representing the workflow of the prototyping plugin.

prototype simulation does not achieve the desired results. Figure 4.2 presents an activity diagram that illustrates the tasks and activities of the described workflow.

4.3 Architecture of the plugin

The new architecture of the prototyping plugin follows an Oriented Object approach like the other components of the IVY Workbench. The architecture also performs a separation of concerns between the UI classes and business classes by following the MVC pattern. Figure 4.3 illustrates a class diagram of the prototyping plugin architecture. Due to the complexity of the architecture, the class diagram only presents the chief classes of the plugin and omits their attributes and methods. The architecture has three main packages: the *Model*, *View* and *Controller* packages.

The *Model* package contains classes that hold the prototype configurations made by the user, such as the events and states and their mapping to the formal model. States have

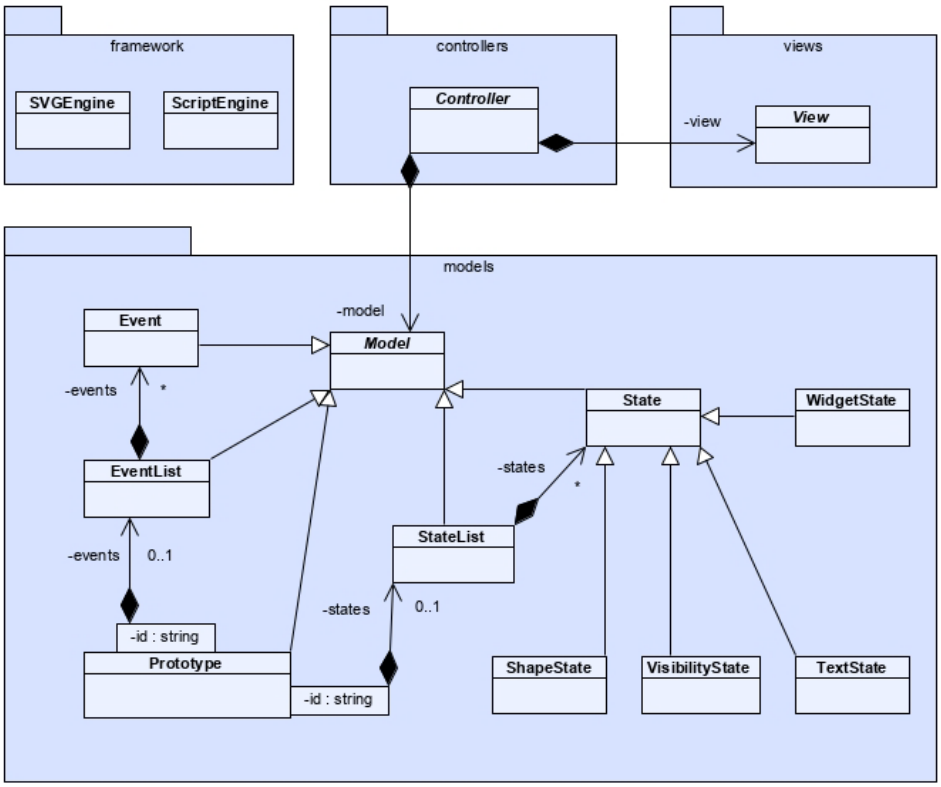


Figure 4.3: Simplified class diagram of the prototyping plugin architecture.

different properties according to their SVG element. For this reason, distinct states were developed with an inheritance pattern. The View package contains classes that specify the UI of the plugin. In this case, these classes hold Swing components that compose the appearance of the new features described in Section 4.6. Both the *View* and *Model* classes are entirely independent of each other. *Controller* classes, contrarily, act as mediators between *Views* and *Models*. These classes add listeners to views to receive user interactions, such as click events. Moreover, the *Controllers'* classes observe changes in the *Models* and update the UI, respectively. Since the plugin architecture is quite complex, *Controllers* can use sub-controllers to separate these tasks into smaller ones.

Besides the *Model*, *View* and *Controller* packages, there is also the *Framework* package. This package mostly offers static methods for supporting the dynamic widget framework. Its two essential components reside in the *SVGEngine* and *ScriptEngine* classes. The *SVGEngine* contains methods for prototype initialization, such as the generic SVG parser detailed in Section 4.4, as well as methods for SVG manipulation. Lastly, the *ScriptEngine* class provides

support for widget initialization and an API for scripting execution. The next sections describe these two essential components of the new version with more detail.

4.4 Generic SVG parser

The current version of the IVY Workbench plugin is dependent on the SVG file format exported by Pencil Project. The developed SVG parser inspects the imported files for elements that contain specific attributes characterizing widgets of that tool, such as buttons and labels. Only those elements can later receive formal model configurations for the prototype simulation. This approach has several downfalls, covered in Chapter 3.

The new version of the IVY Workbench follows a distinct approach by leveraging the DOM structure generated by Apache Batik to identify all the elements that support user interaction. Instead of extracting specific elements, the new parser considers all SVG elements of the imported document as valid components for the prototype. There are, however, some exceptions of SVG elements that do not offer user interaction, such as *script*, *filter*, *defs*, and *title* elements. The new parser, developed herein, does not extract these elements and all their child elements to prevent further configurations on them. This process removes some misconceptions that could be introduced to non-expert users lacking the knowledge about SVG elements that do not provide user interaction. On the contrary, all elements supporting user interaction are assigned a *Universally Unique Identifier (UUID)* on their *id* property to ensure their uniqueness. This method allows fast document queries by *ids* of SVG elements.

One particular exception of the parsing process is the handling of *use* elements. There are no guarantees of style modifications of the instances created with these elements when applying CSS properties to them, unless those are explicitly requested using CSS inheritance. Therefore, any style modification of a *use* element without this request during the prototype simulation will not be applied and rendered, preventing the desired results. The parser takes the following steps to solve the unique case introduced by *use* elements:

- Extract the original element with the reference identifier in the *use* element.
- Perform a deep copy of the extracted element.
- Add all properties assigned to the *use* element to that element. This step ensures that the new element receives attributes such as the visibility and position, supported by *use* elements.
- Replace the *use* element with the new element in the SVG document.

Lastly, when the parser detects a *script* element, it passes the control to the Script Engine for the proper dynamic widget initialization.

To validate the parser abilities of the new IVY version, several identical SVG shapes from distinct sources were imported into the workbench. The process included Pencil Project, Adobe Illustrator, Adobe XD, and Inkscape. The study consisted of creating a mock-up with a set of SVG basic shapes, such as rectangles and circles. Both Adobe tools and Inkscape produced similar results: a SVG document containing *rect* and *circle* tag elements. Therefore, the developed parser successfully extracted all the desired elements from these tools' documents. Pencil Project, however, produced a much different document. This tool declared all SVG shapes in a *defs* element. Then it created *use* elements to render those shapes. Also, it added several custom SVG elements to the document. The new parser successfully discarded these tags since it only processes SVG elements that can support user interactions. Furthermore, the described process to handle *use* tags successfully replaces all these elements with the original shapes included in the *defs* tags. This study provided positive insights about the generic parser correctness and proper extraction of SVG elements that support user interaction.

The new SVG parser successfully solves the dependency on the Pencil Project. Furthermore, it extracts a more diverse set of SVG elements, as opposed to the old parser that only extracted labels and buttons. Lastly, the new parser does not divide the SVG document into layers, since the developed framework of dynamic widgets provides a more effective method for building prototypes in the IVY Workbench. This framework is described with detail in the next section.

4.5 DYNAMIC WIDGET FRAMEWORK

The developed dynamic widget framework has two major components: the library of dynamic widgets and the scripting environment of the IVY Workbench. This framework aims to improve the quality of prototypes and removes the necessity of configuring multiple layers (described in Section 3.1.5) to achieve the desired results. Since each widget has its own set of dynamic properties, designers no longer need to replicate parts of the UI mock-up and to add configurations to these parts.

This section commences with an overview of the required structure of the widgets that compose the developed library. Then, it details the functionalities of the scripting environment responsible for processing the dynamic widgets in prototypes. The section ends with a description of the studies made to check the compatibility of the developed library with mock-up editors.

Listing 4.1: Structure of the switch widget.

```
<svg ...>
  <g xmlns:ivy="http://ivy.di.uminho.pt/ivy" id="ivy-widget:Switch"
     ivy:id="ivy-widget:Switch" ivy:widget="Switch">
    <script type="application/javascript"> //<![CDATA[
      var Switch = (function() {
        <!-- Private auxiliary function -->
        function setColor(color) { ... }

        <!-- Checks or unckecks the switch -->
        function setChecked(value) { ... }

        return {
          setChecked: setChecked,
          props: {
            setChecked: {
              params: [{
                name: "Checked",
                type: "boolean",
                desc: "<html>Checks or unchecks the switch.<br>Checks the
                    switch if the received value<br>is true and unchecks
                   it otherwise.</html>",
                default: false
        }]}}};
      })();
    //]]> </script>
    <!-- SVG elements of the widget -->
  </g>
</svg>
```

4.5.1 *Dynamic widget structure*

The dynamic widget framework requires an extensive library of widgets. The more diverse and complete the library, the better the capabilities of the framework to achieve high-fidelity prototypes for multiple contexts. Each widget of this library, however, must follow specific requirements to be correctly interpreted by the framework. This section describes these requirements by detailing the structure of a switch widget developed for the framework.

The dynamic widgets are built by embedding Javascript code into SVG files to specify their behaviour. Each widget is defined by a *g* element containing a *script* element and one or more SVG elements representing its appearance. Listing 4.1 presents the code for a switch widget that follows this structure, while Figure 4.4 illustrates the appearance of the widget regarding its checked state variations. The *script* functions and many SVG elements composing this widget were omitted in this example for the sake of simplicity. The full code is provided in Appendix A.1.

Figure 4.4: Representation of the different possibilities of the switch widget checked state.

Widgets are defined in SVG documents, inside *g* elements using the IVY namespace (assigned with *xmlns:ivy="http://ivy.di.uminho.pt/ivy"*). The *ivy:widget* attribute provides information about the root element of the widget and its name to be presented to users. While the *ivy:id* attribute assists in situations where the mock-up editors modify the *ids* of the constructed SVG document. These aspects are discussed further in the sections below.

The widgets follow the Javascript Revealing Module pattern (Osmani, 2012). This pattern allows the definition of modules that can encapsulate variables and methods. The module also provides a set of public pointers to its methods and variables in a *return* section. Only these methods and variables can be accessed outside of the module. In the example of the switch widget, only the method *setChecked* and the object *props* are accessible outside of its module. By contrast, the method *setColor* remains a private function that is only recognized inside of the *Switch* module. This pattern has great relevance, as more and more widgets are included in a particular prototype. Conflicts between widget properties' and methods' names might happen, for example, when a widget is used multiple times. This can compromise the results of a prototype simulation. The Revealing Module pattern solves this issue by granting the widget encapsulation in its model. Furthermore, it only allows the scripting environment of IVY to access the revealed public API of the widget.

Besides the list of public methods, the widgets should also expose their method parameters. The *props* object is a collection containing information about the parameters of their public methods. These parameters are exposed as widgets properties by the plugin. In the example provided in Listing 4.1, the parameter of method *setChecked* is associated with a property of name *Checked*. It expects a value of type *boolean* and has its default value set to *false*. All this information guides users through the process of prototype configuration. The *type* value prevents users from passing invalid formal model parameters to the methods of the widgets. Furthermore, users might not need to specify all parameters of the widgets' methods. The *default* value assists these situations, providing the default value to be passed into methods that were not configured. Lastly, the prototyping plugin uses the *desc* value to provide tool tips to users on a widget's property.

4.5.2 *IVY scripting environment*

The scripting environment of the IVY Workbench holds the responsibility for the proper initialization of the widgets of the user imported mock-up. Using Rhino, the environment can extract the API and the properties of the widgets, necessary for the configuration of the prototype simulation. Furthermore, the environment checks if the received widgets follow the Revealing Module pattern and the structured described in Section 4.5.1, discarding any widget that fails these constraints.

There are some scenarios where the Revealing Module pattern may not suffice. For instance, a user might be attempting to execute a prototype containing the same widget multiple times. In this case, there will be two or more modules with the same name leading to override issues. Another issue that arises is the duplication of the *ids* of the widgets' elements that lead to simulation errors. These errors occur because the widget's scripts use DOM queries by *id* to grab specific SVG elements of their structure to modify their properties. Consider the scenario where a user loads into the mock-up two switch widgets having the structure depicted in Listing 4.1. This process results in two modules with the same name (Switch module) and two *ids* with the same value ("ivy-widget:switch"). To fix these issues, the environment performs the management of the widgets' module names to ensure their uniqueness. It follows an approach typically used by filename systems by appending a sequential number to a repeated module name. These unique names can then be prefixed to each *id* of the widget's elements. The repeated names and *ids* are also replaced in the widget's scripts.

The scripting environment is also responsible for the execution of the methods of the widgets during simulations. It supports the execution of these methods with a list of parameters. These parameters are the same ones exposed with the Revealing Module pattern in the *props* object. The prototyping plugin presents these parameters as properties that assemble the state of the widget. Users can control these properties by assigning constant values or attributes of the formal model to them. This feature is essential for combining the inherent formal model capabilities of IVY with the visual representation prototypes. During the simulation, the environment executes the methods of the widgets with the configured parameters. As mentioned, these methods use DOM queries to select SVG elements. Then, they modify the CSS properties of those elements according to the values of the received parameters. Lastly, the Rhino library used to execute the Javascript code of the widgets does not have built-in DOM support. This limitation means that any DOM query performed by the widgets will fail at runtime. However, this library does offer support for converting a Java object into a Javascript object. The developed environment uses this feature to convert the DOM structure generated by Apache Batik into a Javascript object. This process ensures the expected behaviour of the widgets.

4.5.3 *Compatibility with mock-up editors*

Listing 4.2: Structure of a user defined widget library for Pencil.

```
<Shapes xmlns="http://www.evolus.vn/Namespace/Pencil"
        xmlns:p="http://www.evolus.vn/Namespace/Pencil"
        xmlns:svg="http://www.w3.org/2000/svg"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        id="id of the library"
        displayName="library name"
        description="description of the widget"
        author="author name"
        url="url of the library/">
  <Shape id="widget id" displayName="widget name" icon="widget icon">
    <p:Content xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://
       www.w3.org/1999/xlink">
    <!-- widget structure goes here -->
    </p:Content>
  </Shape>
  <!-- more widgets can be added here -->
</Shapes>
```

One of the goals of the IVY Workbench is to offer a way for designers to reuse previous work done building mock-ups. One of the motivations for the generic SVG parser stemmed from this. Through the creation of such a feature, designers can user their favourite mock-up editors to create prototypes. By providing an online version of the dynamic widget library, designers can also import IVY widgets with these editors. However, it is essential to test if the mock-up editors do not alter the SVG structure or Javascript code added to mock-ups, which would mean they might modify the widgets. If the structure of a given widget is compromised, the scripting environment will not be able to initialize it, leading to incomplete prototyping simulations. Therefore, a study was conducted where a switch widget (presented in Listing 4.1) was imported into a mock-up. The study covered the following mock-up editors: Inkscape 0.92.4, Pencil Project 3.1.0, Adobe Illustrator CC 22.0.1 and Adobe XD 28.9.12.2.

Inkscape does not modify any internal structure of the widgets, keeping both IVY namespaces, *id* attributes and *script* elements intact. Therefore, this tool (or any other tools sharing the same characteristics) does not require any addition of dedicated features to the developed library.

Pencil Project is the only one of the studied tools that does not provide a built-in mechanism for users to import SVG documents. Instead, users need to create a new project and create a prototype with the widgets of this tool. However, users can define a custom library of widgets and import it into this tool. This library must be defined in a file called

Definition.xml and must follow the structured detailed in Listing 4.2. When a user imports a library into Pencil Project, the widgets in the library will become available for creating prototypes. Pencil Project modifies the *ids* of the widgets but keeps custom SVG namespace attributes intact. Hence, the scripting environment of IVY can recover the original *ids* with the assistance of *ivy:id* attributes. Then, the environment adds these recovered *ids* to the elements of the widget.

Adobe Illustrator and Adobe XD share similar SVG manipulation mechanisms. Both remove *script* nodes and custom namespaces of produced SVG documents. Nevertheless, both these tools keep the *ids* intact. Due to this feature, it is possible to rebuild the desired widget with only the information provided by the elements' *ids*. The scripting environment checks for dynamic widgets on the imported prototype by analysing the *id* and *ivy:id* properties of its SVG elements. If any of these properties start with the prefix *ivy-widget:*, the environment extracts the widget name and loads its Javascript code from the widget library. The properties *id* and *ivy:id* started with *ivy:* mark child elements of the widget that need to receive the process mentioned in Section 4.5.2. After the described process, the content of the widget becomes similar to one presented in Listing 4.1. Listing 4.3 presents the generic widget structure that solves restrictions posed by the studied mock-up editors.

Listing 4.3: Widget structure compatible with the studied mock-up editors.

```
<g xmlns:ivy="http://ivy.di.uminho.pt/ivy" id="ivy-widget:Widget Name"
   ivy:id="ivy-widget:Widget Name" ivy:widget="Widget Name">
    <elementName id="ivy:elementId" ivy:id="ivy:elementId"/>

    <!-- Remaining SVG elements of the widget -->
</g>
```

The described process solves the issues presented by the studied mock-up editors. The information contained in the property *id* makes the developed widgets compatible with editors that remove custom SVG namespaces. By contrast, the *ivy:id* property offers compatibility with editors that modify the *ids* of SVG elements.

4.6 Added features

Sections 4.4 and 4.5 detailed internal additions to the plugin that augment the prototyping capabilities of the IVY Workbench. The current section describes the newly introduced user features built upon these internal additions. Figure 4.5 illustrates the overall appearance of the prototyping plugin, as well as its added features. These include: the SVG renderer, the SVG sidebar, the States and Events sidebar, the Prototype simulation window and the dynamic widget collection.

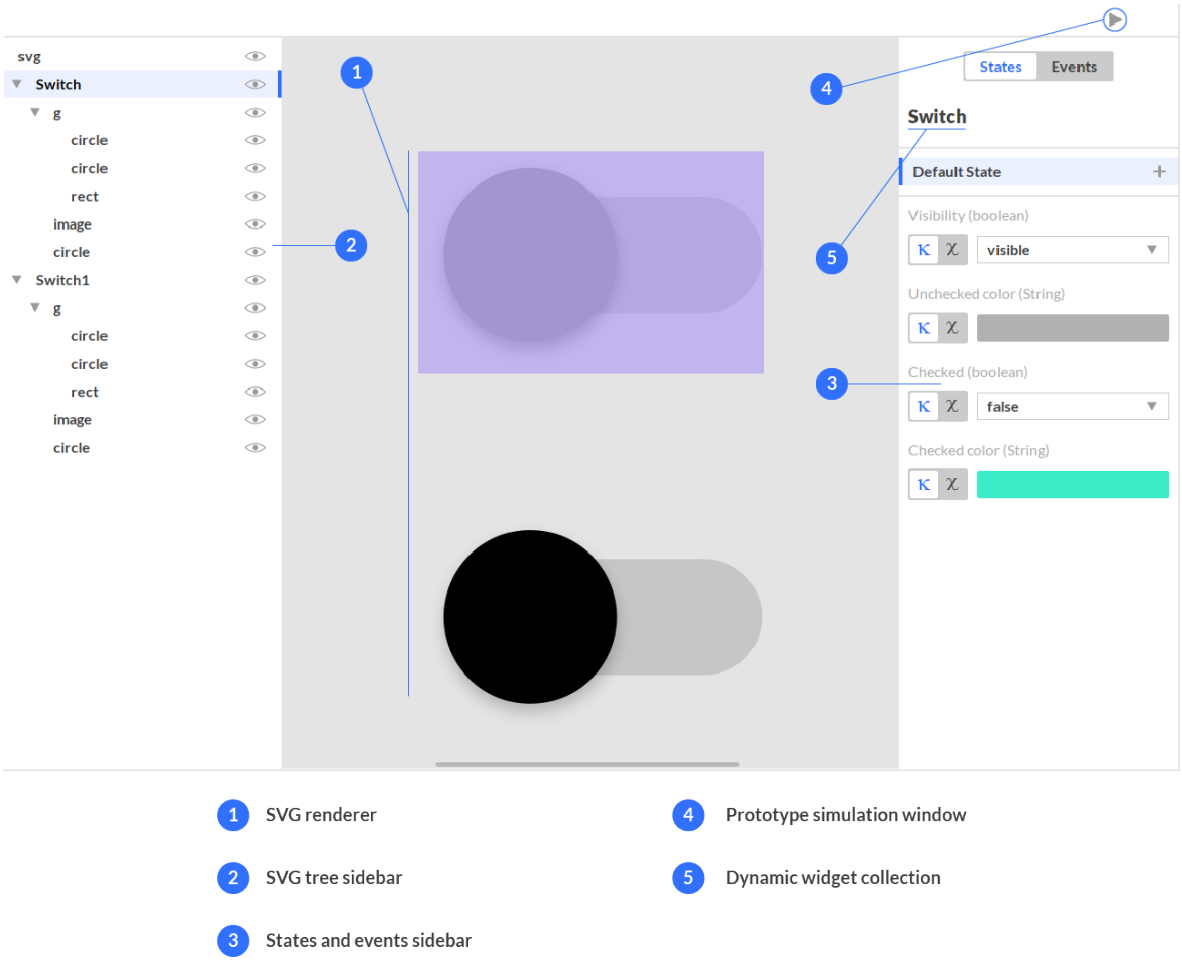


Figure 4.5: Prototyping plugin interface.

4.6.1 *SVG renderer*

This component uses the Apache Batik library to render the SVG document of a prototype. Users can select elements of the prototype by clicking on the rendered document. This interaction creates a purple overlay on the selected element, as depicted in Figure 4.5. The renderer also highlights the selected element on the SVG tree sidebar and updates the states and events sidebar accordingly.

4.6.2 *SVG tree sidebar*

The SVG tree sidebar displays the hierarchic structure of the prototype's SVG document with the exclusion of SVG elements that do not support user interaction. Furthermore, it exhibits the root elements of dynamic widgets by their widget names. Like the SVG renderer,

any user selection on this sidebar is also reflected in other components of the plugin. In this case, the selection reflects in the SVG renderer and the states and events sidebar.

This component also provides basic SVG editing functionalities. These can be activated when a user right-clicks on an element of the sidebar, which prompts a popup menu. The provided functionalities are the following:

- **Visibility toggling** – Each element on the sidebar is represented by its tag name (or widget name) and an eye icon that illustrates its current visibility status. If the eye icon is not visible, then the element is not visible on the renderer as well. Users can toggle an element's visibility with the popup menu, which grants the ability to select elements placed behind the interacted element. This functionality is most useful when configuring a prototype that represents several pages of an application, for instance. The pages can be structured in the SVG document as a group of layers. In this case, this feature allows users to manipulate the visibility of each layer and configure each page separately.
- **SVG insertion** – This feature allows users to insert SVG content into the rendered prototype. It offers two options of insertion: before or after the clicked element's position in the document.
- **SVG deletion** – This functionality allows users to delete unnecessary SVG elements from the prototype.

4.6.3 *States and events sidebar*

The states and events sidebar allows users to configure the behaviour of the prototype. As the name suggests, this sidebar holds two types of configurations: prototype's states and events.

The states configuration allows users to define one or more states for a specific SVG element of the prototype. Each element has a default state representing its initial appearance. Users can modify all the properties that are intrinsic to that state or add more states that trigger when a specific condition is met. A conditional state is triggered when a model's attribute matches a criterion based on its value. As illustrated in Figure 4.6, the conditional "State 1" will be triggered when the attribute *on1* has its value set to *true*. At simulation time, the environment checks if any of the conditional states of an element matches its criteria. If this happens, then the matched state will be rendered. Otherwise, the simulation renders the default state of the element.

States also have specific properties according to the type of the configured element. *g* elements allow users to define their visibility; shape elements, such as circles and rectangles, have an additional fill property; text elements have the same properties of shape elements

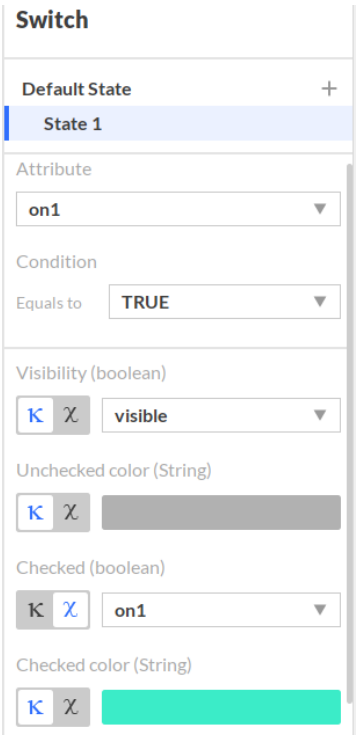


Figure 4.6: Conditional state configuration of a switch widget.

with the addition of a text property that can be used to display the values of an attribute, and widget elements allow users to configure their properties. Figure 4.6 shows an example of the configuration of the states of a switch widget (detailed in Section 4.5.1). The widget has four state properties to control: visibility, checked, checked colour and unchecked colour. The plugin guides users to select valid values for each one of these properties. The checked and unchecked colours only allow colour values that can be set with a colour picker that prompts when a user clicks on the colour button. The visible and checked properties require boolean attributes of the formal model. These can be selected by choosing one of the available options listed in a combo box. The plugin only lists attributes having the same type as the one required by each property to prevent usability errors. Lastly, when a users moves the mouse over a property name label, the plugin shows a tool tip containing a brief explanation about that property. All this information is defined inside the widget structure, as mentioned in Section 4.5.1.

The events configuration allows users to specify multiple events for the elements of the prototype. An event has a trigger, associates with an action of the formal model and executes a function of the prototype's widgets. A trigger can be defined by user interactions (the new version supports click, keyboard press and hover events), or by periodic timer events. Users can specify the duration in seconds of those timers. A practical example that illustrates the benefits of this feature is a chronometer that updates its display every second. The function

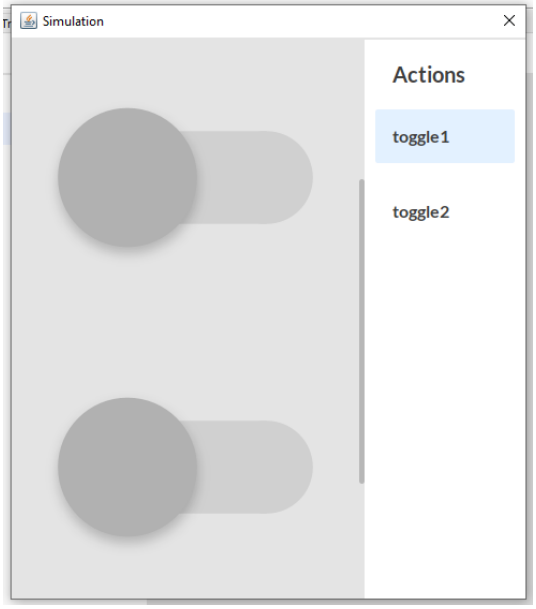


Figure 4.7: Simulation window of the prototyping plugin.

component offers the possibility to alter the appearance of the prototype without the assist of the formal model. This feature supports the creation of advanced prototypes without the need for creating complex formal models.

4.6.4 *Prototype simulation window*

When a user clicks on the simulation button presented in Figure 4.5, the plugin commences initializing the simulation environment. The environment receives the SVG document, widgets, events, states and the formal model of the prototype. When the initialization finishes, the environment launches the simulation window, as depicted in Figure 4.7. This window has an SVG renderer that displays the prototype and a right sidebar that contains the list of available actions of the formal model. Users can either interact with the prototype itself, triggering the defined events, or with any of the actions in the sidebar. Lastly, the simulation environment uses IVY's internal messaging system to communicate with the NuSMV model checker and receive the updates of the prototype's formal model.

4.6.5 *Dynamic widget collection*

The developed collection of dynamic widgets contains widgets of two distinct types. One type consists of what are herein called *external widgets*. Their structure is described in Section

4.5.1 and they can provide complex behaviour to the prototypes created with the plugin. The created widgets of this type are the following:

- **Switch** – A representation of a toggle button inspired by the Android Material Design guidelines. It offers mechanisms to control its checked, visibility and colour properties.
- **Checkbox** – Provides the same properties as the switch widget but offers a distinct UI.
- **ClockChrono** – Offers two distinctive modes: clock mode and chronometer mode. In clock mode, it provides a display that presents its hours and minutes properties. In chronometer mode, this widget adds the value of its seconds' property to its display.
- **Led** – A circle shape that updates its *On* property with a boolean variable. It also allows users to modify their *On*/*Off* colours.
- **ProgressBar** – Represents a horizontal bar that allows users to update its progress with a value between 0 and 100.
- **Cursor** – This widget was developed to simulate the behaviour of the B. Braun Perfusor medical device, although it can also simulate other prototypes due to its generic implementation. This widget represents a numeric display with a fixed number of digits and a cursor that marks the currently selected digit. Users can use this widget to move the cursor position to the left or the right. Chapter 5 provides a more detailed description of this widget and an example of its configuration.

The second type of widgets consists of SVG elements. These are called *internal widgets* herein, because it is the prototyping plugin that defines their behaviour. Each type of SVG elements has its own set of properties to control. Currently, the prototyping plugin distinguishes three distinct groups of this type of widgets:

- **Visibility widgets** – Represent SVG elements such as *image* and *g* tags. These widgets allow users to control its visibility property with a boolean variable.
- **Shape widgets** – These widgets inherit the visibility property from the visibility widgets and also allow users to control their fill property. They represent SVG shapes such as *rects*, *circles* and *paths*.
- **Text widgets** – Represent SVG *text* elements and can display any variable value as its *textContent* property. They also inherit all the properties of shape and visibility widgets.

The two types of dynamic widgets provide a very flexible method for users to define the behaviour of a prototype. For complex models, users can create prototypes with

external widgets that offer a more concrete solution for a specific problem. By contrast, the internal widgets present a minimalist solution for users to add behaviour to prototypes or to complement the functionalities of the external widgets.

4.7 Evolutionary prototyping

The introduced features described earlier in this chapter enhanced the iterative prototyping capabilities of the IVY Workbench. Besides these features, some efforts were made to introduce evolutionary prototyping functionalities, such as support for code export. One of the primary motivations for this work was to present to users the opportunity to evaluate their prototypes in multiple platforms.

The support developed for evolutionary prototypes allows the export to HTML code. This file format is optimal for cross-platform evaluation because it allows users to test prototypes in the browser and mobile devices. Furthermore, it vastly reduces the export process by allowing the embedding of the UI mock-up as a SVG element. Lastly, it supports Javascript code which offers a similar DOM API to the one used with Apache Batik and grants the possibility to add behaviour to the prototype.

This section describes the prototype export process to HTML. It starts by describing the export of the formal model, states and events to Javascript code. This code, in combination with the SVG structure of the UI mock-up are embedded in HTML code for achieving evolutionary prototypes. Ultimately, this section ends by detailing the developed Android and iOS mobile applications solely made for users to evaluate the exported prototypes on their mobile devices.

4.7.1 *Formal model export*

The developed evolutionary prototypes contain the formal model as Javascript code. The formal model of the IVY Workbench has four primary components: defines, attributes, actions and axioms. The approach created to export these components into Javascript code views defines as constants, attributes as code variables, actions as methods, and axioms as a set of code instructions of a method.

The model's axioms express how the actions affect the values of the attributes. The approach considers four distinct forms for axioms, depicted in Listing 4.4, whereas Listing 4.5 presents the conversion into Javascript code of these forms.

The first axiom is the only one that is not associated with an action, and therefore not associated with any method in the resulting code. In the formal model terminology, this axiom defines the initial state of the model. The purpose of this axiom is to provide an initial value to every attribute of the model. The second axiom defines a condition that must be met

Listing 4.4: Different forms of axioms.

```
[] instruction1 & instruction2 & ... & instructionN
per(actionName) -> condition1 & condition2 & ... & conditionN
[actionName] instruction1 & instruction2 & ... & instructionN
condition1 & condition2 & ... & conditionN -> [actionName] instruction1 &
    instruction2 & ... & instructionN
```

for the action to executes. In Javascript, this process can be done by adding the negation of the condition to the corresponding method. If the condition fails, then the method is exited by a return statement. This portion of code must be placed at the beginning of the method. The third axiom represents a set of instructions that always execute when a specific action is called. Lastly, the fourth axiom is a set of instructions that only executes if a condition is met. In Javascript, this process is done with an if statement.

The axioms could also contain a function called *keep*, that receives attributes as parameters. Its purpose is to record attributes that are not modified in a specific axiom and to make the formal model deterministic. The algorithm discards this function. Moreover, the axioms mark the current attribute value by adding the character "'" to their names. As an example, consider the integer attribute *inc*. For an axiom to increment the value of this attribute by one, the instruction would be the following: $inc' = inc + 1$. When the algorithm detects this kind of instructions, it discards the "'" character.

To better understand the whole model export process, consider the formal model presented in Listing 4.6.

The export algorithm commences by extracting the defines of the formal model, if it has any, and converts these into Javascript constant values. Next, it processes the attributes by transforming these into Javascript variables. Then, it identifies the actions and stores their names as dictionary keys. The values associated with these keys will later receive the instructions resulting from the process of extracting the model's axioms. This dictionary structure is essential because one formal model could have multiple axioms associated with one action. Afterwards, the algorithm initializes the variables. This step is done by analyzing the initialization axiom. In Listing 4.6, this axiom is depicted in line number 7. Lastly, the algorithm processes every set of instructions of the remaining axioms and adds the resulting code to the dictionary. In the end, it traverses this dictionary and adds the full code of each method to the code containing the variables and their initialization. For the model in Listing 4.6, the developed algorithm produces the Javascript code presented in Listing 4.7.

The formal models of the IVY Workbench also support attributes represented by fixed-size arrays. Suppose the algorithm detects one of such attributes. In that case, it adds a Javascript array to the produced code and initializes this array with the size equals to the number of elements declared in the formal model. The arrays of the formal model also support negative

Listing 4.5: Generic conversion of axioms into Javascript code.

```
// First Axiom
instruction1;
instruction2;
...
instructionN;

function actionName() {
    // Second Axiom
    if (!(condition1 && condition2 && ... && conditionN)) {
        return;
    }

    // Third Axiom
    instruction1;
    instruction2;
    ...
    instructionN;

    //Fourth Axiom
    if (condition1 && condition2 && ... && conditionN) {
        instruction1;
        instruction2;
        ...
        instructionN;
    }
}
```

Listing 4.6: Formal model of a switch.

```
1 interactor main
2  attributes
3   [vis] checked: boolean
4  actions
5   [vis] toggle
6  axioms
7   [] checked=false
8   [toggle] checked'=!checked
```

Listing 4.7: Javascript code generated from the switch model.

```
var checked;
checked=false;

function toggle() {
    checked=!checked;
}
```

indexes which are not directly supported in Javascript. At the moment, the developed version of the algorithm only supports positive index arrays. Therefore, the support for negative indexes is left for future work.

4.7.2 *States and events export*

Besides the formal model, it is essential to export the configurations made for assigning the states and events to the prototype. These two types of configurations are transformed into Javascript code and added to the code generated by the formal model export process.

The generated code from the states is included in a method called *update*. This method is invoked when an event is triggered and executes the properties of all configured states. In code, these properties are widgets' methods calls or SVG modifications with the DOM API.

The event configurations are transformed into document event listeners. As mentioned, this type of configurations has an action of the formal model. Therefore, these listeners execute their action by invoking its generated method from the formal model export process. Lastly, the listeners execute the *update* method to update the prototype accordingly.

The DOM API used in the IVY Workbench has some differences from the browser DOM API. One example is the modification of the text property of an SVG element. In the browser, this modification is done by directly accessing the *textContent* property of the *Element* object, whereas the API provided by the Apache Batik library requires invoking the *setTextContent* method. Consequently, this kind of operations are incompatible and will result in an error during the execution of the exported prototypes. To address this, the export process adds a small compatibility script that adds extra functionalities to Javascript objects, such as the *Element* object. This process can only be achieved because the Javascript language supports the addition of object properties in runtime. For the example mentioned, the code presented in Listing 4.8 grants the correct compatibility between the Apache Batik DOM and the browser DOM.

Listing 4.8: Example of the compatibility between the Apache Batik library and the browser DOM.

```
Element.prototype.setTextContent = function(text) {
    this.textContent = text;
};
```

The exported prototype is a HTML document that contains the SVG structure of the UI mock-up and the combination of the compatibility methods, formal model, states and events inside a single script. The exported prototypes follow a separation of concerns intrinsic of the MVC pattern. The *View* is the SVG structure and the *Model* is the generated code from the formal model. Both these components are completely independent of each other. The

Controller is the code generated from the states and events configurations. This component receives user input and updates the *Model*. Lastly, it reflects these changes in the *View* by invoking the generated *update* method.

4.7.3 *Android and iOS mobile applications*

Besides browser compatibility, the produced work includes Android (Figure 4.8a) and iOS (Figure 4.8b) mobile applications that offer support for users to run the exported prototypes in their mobile devices. The Android application was developed in Kotlin and the iOS application in Swift. Both these applications use the *WebView* component to render the prototype exported in the HTML file format. The only requirement was to enable the Javascript in this component to ensure the correctness of the prototype execution.

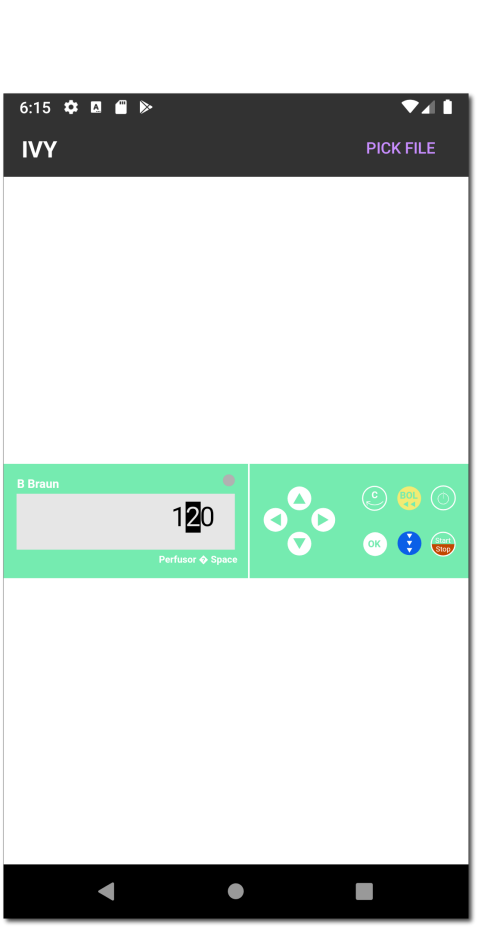
As depicted in Figure 4.8, the developed applications provide a way for users to pick an HTML file from their mobile's files. A more optimal solution would have been to build an infrastructure that supports the creation of user accounts and shared links. In the IVY Workbench users could save their projects in their account and create a shared link of these projects. Later, they could log into their account in the IVY mobile applications and load any of their projects. Alternatively, they could share the links of their projects with other users that do not have an IVY account. This solution is out of the scope of this project and is left for future work.

4.8 SUMMARY

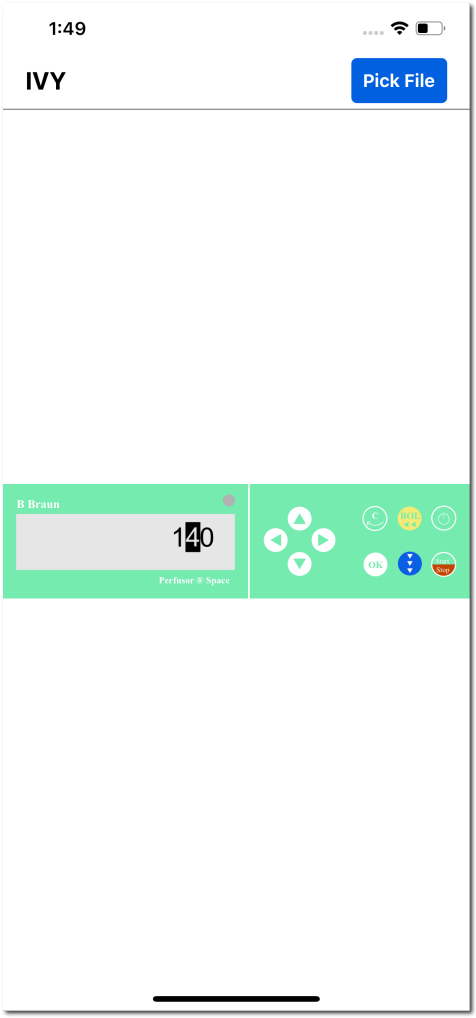
The new IVY version includes updates in its Prototyper plugin that improve the quality of the produced prototypes. The generic SVG parser allows users to use their favourite mock-up editors to build the mock-up of the prototype. It also augments the fidelity of the tool's prototypes since it does no longer requires Pencil Project's medium-fidelity prototypes. Moreover, the new parser allows users to configure each element of the prototype (excluding non-interactable elements such as filters and scripts), as opposed to the limited set of elements of the old version.

The scripting environment of the IVY Workbench is an essential component of the developed widget framework, providing methods for initializing widgets of a prototype and executing scripting functions during simulations. The developed framework vastly improves the capabilities of the tool by removing the necessity of configuring several layers to achieve the desired results.

The new features add basic mock-up editors' functionalities such as manipulation of SVG attributes, insertion and elimination of elements. Furthermore, they introduce dynamic behaviour to prototypes by leveraging the functionalities of the widget framework. This



(a) Android mobile application.



(b) iOS mobile application.

Figure 4.8: Developed mobile applications.

behaviour stems from the events and states configurations that listen to user interactions and react to changes in the formal model, respectively. The events and states augment the configuration capabilities of the plugin's older version and provide a more clear separation of the formal model's attributes and actions. Lastly, the introduced evolutionary features provided a way for users to evaluate prototypes in the browser and their mobile devices. For this last topic, the HTML proved to be a useful file format that readily allowed the embedding of the UI mock-up and logic behaviour expressed with Javascript code.

The next chapter provides a more accurate view of the capabilities of the new features by detailing some practical prototyping examples.

5

PLUGIN VALIDATION AND EVALUATION

The present chapter explores the features added to the prototyping plugin, which have been described in Chapter 4. It does this by describing the prototype of a medical device, the B. Braun Perfusor ® Space, which covers most of the mentioned features, including two external widgets: led and cursor. Lastly, this chapter covers the details of usability tests with non-expert users.

The goal of presenting the prototype is to illustrate the new features introduced to the prototyping plugin of IVY Workbench. The goal of the usability tests was to validate the new features of the tool.

5.1 B. BRAUN PERFUSOR ® SPACE

The B. Braun ® Perfusor Space[1] is a syringe infusion pump that allows the configuration of the amount of a specific drug to be administrated over time. This medical device offers multiple configuration modes. However, the prototype of the device focuses only on the configuration of the quantity of drug to be administered. Moreover, the prototype includes an option that simulates the drug administration at one drug unit per second. For the sake of simplicity, the prototype does not cover other configurations offered by this device, such as drug selection, administration rate and patient weight specifications.

Figure 5.1 depicts the mock-up that was developed to represent the user interface of the infusion pump. The goal is for the prototype to capture the following behaviour of the target system:

- When a user performs a click in the left or right arrow buttons, then the cursor moves left or right respectively. The cursor position is identified by a black background in the digits screen. The cursor also changes the digit foreground to white.
- When a user performs a click in the up or down arrow buttons, then the digit value of the current cursor position increases or decreases respectively.

1 https://www.bbraun.com/en/products/b/perfusor-space.html, accessed 26-September-2020

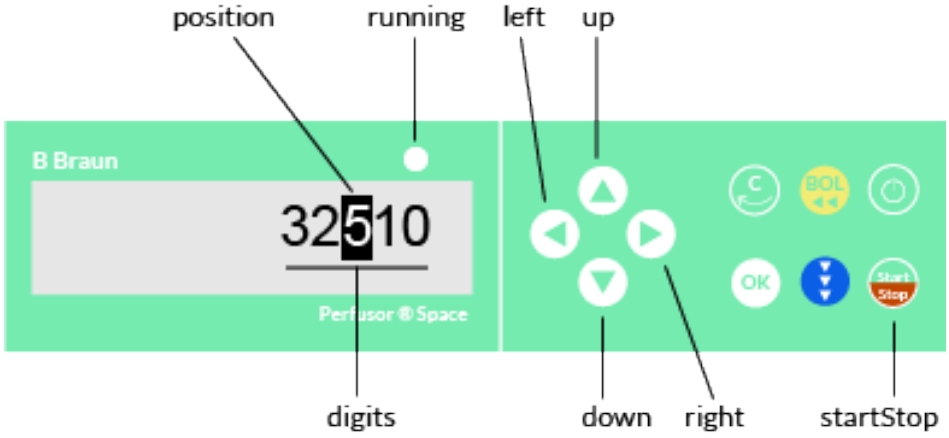


Figure 5.1: Infusion pump mock-up with the respective association between its elements and the formal model's attributes and actions. The formal model also contains the *infuse* action that is triggered every one second.

- When a user performs a click in the start/stop button, then the prototype commences/stops the drug administration. In this mode, users cannot interact with the cursor, and the value represented in the digits screen decreases one unit every second. The circle above the screen has a white background colour when the administration is running and a grey background colour otherwise.

This prototype requires a formal model that specifies its behaviour in order to animate the mock-up. The following sections describe these components and the configurations required to develop the prototype in the prototyping plugin.

5.1.1 *Formal Model*

The formal model on which the prototype is based (see Appendix A.2) has three main components: the attributes that describe the system's structure, the possible actions performed in the system, and the axioms that represent rules for expressing system's behaviour. However, for users of the prototyping plugin, the relevant components are the attributes and actions of the model (the mapping between them and the mock-up is presented in Figure 5.1). The attributes are the following:

- ***digits*** – This attribute holds the value of each digit of the prototype. An array of five integer values represents it.
- ***position*** – As the name suggests, it represents the position of the cursor in the digits screen.

- ***running*** – This attribute has a boolean type and indicates if the device is in infusion mode.

Lastly, the actions of the developed model are the following:

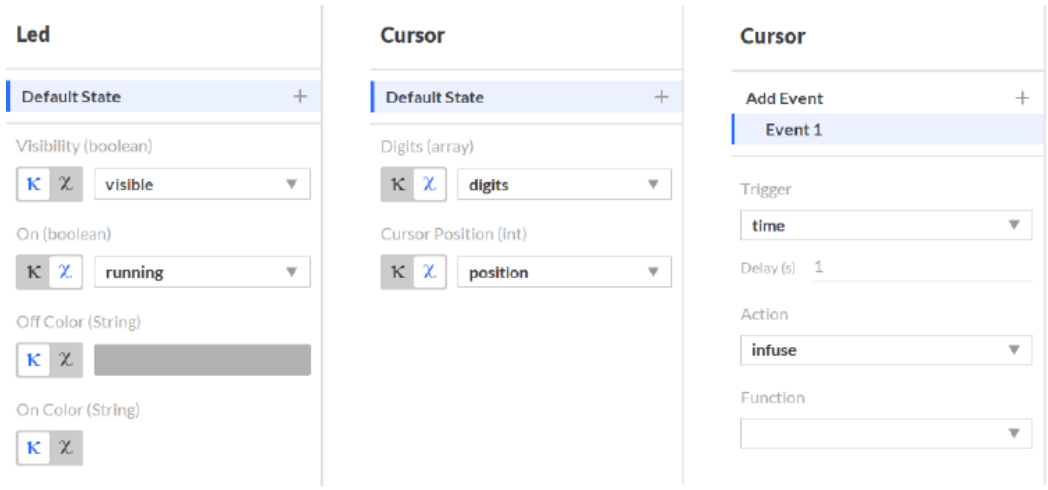
- ***up/down*** – Increases/decreases the value in *digits* array at the index equal to the current value of the *position* attribute.
- ***left/right*** – Increases/decreases the value of the *position* attribute.
- ***startStop*** – Initiates or stops the infusion process. This action toggles the *run* attribute's value.
- ***infuse*** – This action only runs if the *running* attribute has its value equal to *true*. It decreases the value of the whole *digits* array. It stops executing, by changing the value of *running* to *false*, when all elements of the *digits* array are equal to 0.

5.1.2 *UI Mock-up*

The UI mock-up, presented in Figure 5.1 contains several SVG elements that are relevant to the development of the prototype.

Two of these elements are the external widgets led and cursor. The led widget provides a way to indicate if the device is in infusion mode with its *On* property. This property expects a boolean and therefore is controlled by the *running* attribute of the formal model. The cursor widget is responsible for the animation of the digits screen. Its main properties are *Digits* and *Cursor Position*. The first property expects an array of integers and is controlled by the *digits* attribute of the formal model. When this widget receives the integer array, it creates as many SVG text elements as the size of the received array. The result is the display of the reversed received array. For instance, if the array received has the values [1, 0, 4, 5, 6], then the result in the screen will be 65401. The second property expects an integer value and is assigned to the *position* attribute of the formal model. This property applies a SVG filter to the text element in the current position. The result is a black rectangle, as shown in Figure 5.1 in the digit "5".

The remaining SVG elements of the mock-up are treated as internal widgets by IVY's prototyping plugin. The prototype of this device only requires the configuration of the elements that assemble the arrow buttons and the start/stop button. It is essential to mention, however, that these buttons have multiple elements. For instance, the arrow buttons have a group (*g* tag) containing a white circle and an arrow icon. Only these group elements need to received configurations to achieve the desired results.



(a) State configurations of the led widget.

(b) State configurations of the cursor widget.

(c) Configuration of the periodic event.

Figure 5.2: Configurations of the external widgets led and cursor.

5.1.3 *Prototype Configurations*

The last two sections described the two major components that assemble the prototype of the mentioned medical device. This section details the steps required to link these components to achieve the desired results. After the files containing the formal model and the UI mock-up are imported into the IVY Workbench, users need to define the states and events in the prototyping plugin by following these steps:

1. **Configure the state of the cursor widget**. For this step, users need to assign the attributes *digits* and *position* to the properties *Digits* and *Cursor Position*, as depicted in Figure 5.2b.

2. **Configure the state of the led widget**. The *On* property of this widget should be assigned to the *running* attribute of the model. Moreover, the default *On* colour of this widget is green, which could not be perceptible since the device shares the same colour. To prevent such issues, users should assign a white colour value to this property. Figure 5.2a illustrates these configurations.

3. **Add event handling to the buttons of the mock-up**. Users should add event handlers to the click events of the arrow buttons and the start/stop button. In the prototyping plugin, the default trigger for an event is a click. Therefore, users only need to select the appropriate action (*up*, *down*, *left*, *right*, *startStop*) of the model in the action combo box of the events sidebar.

4. **Configure the periodic event for the infusion mode**. This event can be assigned to the cursor widget. Users need to change the event trigger to time. This selection will display a new input value that controls the interval in seconds that triggers this event, as depicted in Figure 5.2c. The default value is one second which is the one required for the prototype. Lastly, users need to select the action *infuse*.

When all the steps just described are performed, users can interact with the prototype by clicking in the *play* button. This action launches the simulation window, which should present the prototype of the medical device with the expected behaviour.

5.2 USABILITY TESTS

The prototype described in the previous section served as the validation of the overall capabilities of the new prototyping features of the IVY Workbench. However, it was considered relevant to carry out a usability test to validate the new features with non-expert users. The goals of such usability tests were the following: verify if any user was capable of eagerly producing a prototype, and derive a list of future improvements for the workbench. The usability tests were carried out according to the script provided in Appendix B.2. A consent form (provided in Appendix B.1) was also collected.

The experiment script started with a brief explanation of the context of the tool and its features. This document also included a step-by-step guide on how to create a simple prototype: a toggle button. The test itself consisted on the creation of the prototype described in Section 5.1. Lastly, the test ended with a brief questionnaire to collect demographic data (name, age, sex and whether the user had background experience with mock-up editors) and to collect information on the usability of the tool.

The consent form informed users about the data collected in the usability test. This document ensures that users agree to share the required data by the experiment script and the recording of the session.

This section describes the whole procedure of the usability tests. This description includes the conditions of the tests and every step performed with the users. Next, the section provides information on the participants and ends with a discussion about the collected data and results.

5.2.1 *Procedure*

The usability tests were performed on a laptop that already contained the environment setup with the IVY Workbench. Both the step-by-step guide and the actual test required the importing of two files: the formal model and the UI mock-up. Both these files were previously created since this step is related to experienced users. These files were saved

in the same directory and were imported before the first contact with the users. This step ensured that the IVY Workbench saved the files' directory as the default working directory. Therefore, in the usability tests, users were appropriately directed to the files directory. This method avoided unnecessary work in the process of locating the required files.

The full procedure of the usability tests, as described in the script, included the following steps:

1. **Beginning of the process.** The process started with a brief explanation of the tool and the activities to perform in the tests. Afterwards, users were notified about the required data and the recording of the session. If they agreed with the requirements, they were asked to sign the consent form. All the participants involved in the performed tests agreed to share the required data.

2. **Experiment script reading.** Users were asked to read the experiment script document. In this phase, they were encouraged to ask questions about the document and the goals of the tests.

3. **Step-by-step example.** As stated above, the script included a step-by-step guide on how to create the prototype of a toggle button. The formal model of this prototype had one boolean attribute and one action that toggled the value of that attribute. The UI mock-up contained the switch widget described in Section 4.5. The prototype required the assignment of the *Checked* property of the widget's state with the attribute of the model. The prototype also required a click event configuration on the widget that executes the action of the formal model. The goal of this step-by-step guide was to familiarize the users with the tool before the actual test. In this phase of the usability tests, users were encouraged to ask questions about the tool's features. Users were also accompanied by the script document in the process of developing the example prototype. During this process, it was also emphasized the relevant/irrelevant tool's features for the example and the official test.

4. **Actual test.** The test itself consisted on the creation of the prototype described in Section 5.1. This test had a mandatory and a complementary part. The first part consisted on the prototyping of the amount of drug to be administered, whereas the complementary part included the prototyping of the infusion process. This division was perform to evaluate if the users were able to complete at least the mandatory part. In this test, users were also accompanied by the script document since this document contains the required steps for building the prototype. Before the test commenced, screen recording with the Windows Xbox Game Bar application was initiated. Then, users were asked to build the prototype that fulfils the requirements of the mandatory part. Afterwards, users began the complementary part. Lastly, when they finished the prototype, they were asked to stop the screen recording.

Table 5.1: Participants involved in the usability tests.

Participant	Age	Sex	Previous experience	Part I	Part II	Time (minutes)
1	25	F	YES	✓	✓	10:08
2	25	F	NO	✓	✓	12:19
3	24	M	YES	✓	✓	06:38
4	24	F	NO	✓	✓	07:40
5	20	F	NO	✓	✓	07:52

5. **Filling of the questionnaire.** The last phase of the usability tests included the filling of the questionnaire presented at the end of the script's document. The questionnaire contained seven questions about the usability with the tool and is presented in Appendix B.2. Question 6 is related to users with background experience in mock-up editors. Therefore, only these users were asked to answer this question.

5.2.2 *Participants*

The usability tests were conducted in the region of Braga, Portugal and involved 5 participants. The participants included one male and four females. Their average age was 23.5 years. Furthermore, two participants had background experience with mock-up editors. Table 5.1 depicts this information and includes the information about the conclusion of parts I and II of the test and the time required by each participant. As shown on the table, all participants were able to complete the two parts. Lastly, the average time required to fulfil the mentioned parts was 8 minutes and 51 seconds.

5.2.3 *Collected data and results*

By analysing Table 5.1 it can be concluded that all participants were able to successfully develop a prototype that fulfils the requirements of the two parts of the test. Therefore, the results of the usability tests suggest that the tool provides the means for non-expert users to build prototypes of interactive systems by bringing together mock-ups and formal models.

It was expected that all users with previous experience with mock-up editors completed the prototype in the shortest time, since these users would generally have more knowledge about the SVG structure. Although one of this type of users proved this hypothesis, another user of this type complete the prototype in the second-longest time. This event suggests that a more in-depth knowledge about SVG structure might not be so relevant in the process of developing prototypes with this tool.

The following observations were listed from the screen recordings of the official tests with the participants:

- **Observation 1.** The users quickly selected the appropriated attributes of the formal model to configure the required states of the elements of the mock-up.
- **Observation 2.** The users had difficulties in the selection of elements that were a composition of other SVG elements. In the prototype of the medical device, these difficulties were raised in the selection of the arrow buttons.
- **Observation 3.** In the process of developing the prototype, users rarely selected elements of the mock-up by interacting with the renderer. Instead, they usually selected the elements with the SVG tree sidebar.
- **Observation 4.** The users had some difficulties in distinguishing the differences between states and events. These difficulties were overcome as they interacted more with the tool.

As mentioned, the questionnaire included seven questions about the usability with the tool. The following list presents these questions and the answers provided by the participants:

1. **What was your overall impression of this system?** All participants made a positive overall evaluation of the system, recognizing its practical functionalities and its simplicity.
2. **What aspects of this system did you like the most?** Most participants liked the colour scheme and the simplicity of the UI. Three participants approved the possibility of selecting SVG elements with the SVG tree sidebar and the renderer. One participant mentioned the attribute selection guidance in the states configuration process.
3. **What aspects of this system did you like the least?** In general, the participants found difficulties in selecting groups of SVG elements. Two participants suggested that only one file should be imported instead of separate files containing the formal model and the UI mock-up.
4. **Were any features that you were surprised to see?** Three participants were surprised with the functionalities of the external widgets. Moreover, two participants pointed out the periodic time events. Lastly, one of the participants acknowledged the tool's capabilities in creating prototypes with ease.
5. **What features did you expected to find but are missing in this system?** Two participants did not found any feature that was missing in the tool. One of the participants chose not to answer this question. Lastly, two participants pointed out drag and drop functionalities and the missing of a SVG group selection in the renderer.

6. **Compare the capabilities of adding behaviour to prototypes of this tool with other tools that you have used.** This question was only answered by the two participants with previous experience with mock-up editors. Both participants acknowledged the capabilities of behaviour addition of the tool. One of the participants compared Pencil Project (described in Section 2.3.1) with this tool and claimed that the IVY Workbench allows the creation of prototypes with more complex behaviour.

7. **Would you recommend this system to your colleagues?** All participants said they would recommend this tool to their colleagues.

The results of the usability tests proved satisfactory because each participant was able to fulfil not only the mandatory part of the official test but also the complementary part where more advanced features of the tool were covered. This fact seems to indicate that the IVY Workbench is a tool that has the potential to support multidisciplinary teams, where even non-expert users can develop prototypes for interactive systems.

The tests also emphasized the strengths and weaknesses of the new prototyping features. Users recognized the capabilities of the external widgets and generally found the developed UI intuitive. However, users had difficulties in selecting SVG elements in the renderer. This suggests some improvements in the renderer are needed, to support group selection as default instead of single element selection. This process includes the propagation of click events one level higher of the SVG hierarchy. Lastly, another improvement is the addition of project files to the tool. With this feature, users will be able to create projects that include the two main components of a prototype: formal model and UI mock-up, hence reducing the importing phase work.

5.3 SUMMARY

This chapter covered the development of a relatively complex prototype of the medical device B. Braun Perfusor ® Space. This example prototype served to explore the new prototyping features of the tool. The prototype also included a practical application of two external widgets: led and cursor. These widgets proved useful to achieve the desired simulation results that will not be possible only with a formal model and a mock-up. Although the prototype had some level of complexity, it shared the same workflow as simple prototypes: loading of the required files and configuration of the states and events.

The present chapter also covered the details of the tests conducted with users. These tests provide indication that the IVY Workbench is a tool that makes it possible for non-expert users to develop prototypes of interactive systems. In particular, the results indicate that the tool has the potential to support multidisciplinary teams, which is one of the chief aspects of the UCD methodology described in Section 2.1. These tests also pointed out a few

improvements that should be added to the tool, namely SVG group selection in the renderer and support for project files.

6

CONCLUSION

Prototyping is an iterative process, essential for interactive systems design that provides an inexpensive earlier evaluation of the system, and supports the concepts of the UCD methodology. In the context of safety-critical interactive computing systems development, it is useful to combine it with the development of formal models that allow a more exhaustive analysis of system behaviour.

This work addressed the synergies between two types of approaches: mock-up editors and model-based tools. The first approach focuses on the visual representation of the system, while the second one combines the appearance of the system with formal analysis. However, the solutions found so far for the model-based approach are complex and not suited for non-experts. The current work evaluated these issues on the IVY Workbench tool and identified potential features to improve it. These features were collected from the studied mock-up editors (Pencil Project and Adobe XD) and model-based tools (PVSio-Web and CIRCUS). The contributed features of this project were added to the prototyping plugin of the IVY Workbench.

6.1 RESULTS

The IVY Workbench prototyping plugin uses the SVG file format to work with a broader design community than tools that resort to coding or raster images, since the majority of mock-up editors commonly uses this format. However, distinct mock-up editors could provide different types of SVG structures. Along these lines, this project included the development of an engine that was able to process SVG files from different sources. To validate the effectiveness of this engine, several identical SVG shapes from distinct sources were imported into the workbench.

The SVG file format also granted the possibility of extending the prototyping capabilities with the introduction of dynamic widgets, since this format allows the embedding of Javascript code. These widgets provide the means for adding dynamic behaviour to SVG mock-ups. One of the challenges of this project consisted in the creation of a framework

of dynamic widgets. This framework included the definition of a widget API and the development of a scripting environment that was able to execute these widgets during prototype simulations. Tests were also performed to validate the compatibility of these widgets with several mock-up editors. These tests suggested the widget recovery at runtime from mock-ups since some mock-up editors modify both the SVG structure and the Javascript code.

Furthermore, the developed widgets have two types: external and internal. The external type includes the widgets created for a library of dynamic widgets that contain Javascript code and aim for complex behaviour or model-specific functionalities. The internal widgets are created by the prototyping plugin. These widgets are any SVG shape, such as *rect*, *text* or *g* elements and allow the manipulation of several SVG properties. The main goal of this type of widgets is to allow users to configure each component of the UI mock-up. Moreover, these widgets are also independent of the formal model and reduce the complexity in the process of creating prototypes.

The present work presented an approach for developing prototypes with IVY and combine the features of formal models with UI mock-ups. The approach consists on the configuration of states and events. States are related to the formal model attributes and define the way prototypes react to changes in the values of these attributes. Events are associated with formal model actions and user interactions, such as clicks or user-defined timers. This approach proved resourceful to develop prototypes for interactive systems. One example was provided in Chapter 5 where the prototyping of the B. Braun Perfusor ® Space device was detailed.

Besides the features for improving the prototyping capabilities of the IVY Workbench environment, code generation functionalities were was also added to the tool. This process included the translation of the formal model and the configurations of states and events into Javascript code. Then, the tool exports its prototypes by including, in a HTML file, the SVG mock-up and the Javascript code. This file format proved useful since it is cross-compatible with multiple platforms. Consequently, Android and iOS mobile applications were developed that use the *WebView* component to render the exported prototypes.

Moreover, usability tests with non-expert users were conducted, to validate the new prototyping features of IVY. The results were satisfactory since all users were able to fulfil all the requirements of the tests, proving good indications of IVY's support of multidisciplinary teams, which is one of the chief principles of UCD methodology. The tests also helped identify a number of improvements that should be added to the tool in future work.

Before the additions made by this project, the IVY Workbench was able to produce medium-fidelity prototypes since it could only process SVG files from Pencil Project. Its prototypes were iterative since they offered the possibility to evolve, as the design process iterates, but do not become part of the system. The overall prototyping features of the

Table 6.1: Updated comparison table against the new features of IVY Workbench.

Tool	Precision	Interactivity	Evolution
Pencil Project	Medium-fidelity	Fixed-path	Rapid
Adobe XD	High-fidelity	Fixed-path	Iterative
CIRCUS	High-fidelity	Open	Evolutionary
PVSio-Web	High-fidelity	Open	Iterative
IVY Workbench	High-fidelity	Open	Evolutionary

previous version of IVY Workbench were compared with other tools and the results are presented in Table 2.2 (see Section 2.4).

With the new prototyping features, IVY Workbench can produce high-fidelity prototypes. This improvement stemmed from the development of the engine that can process SVG structures from distinct sources. Additionally, with the code export functionalities to HTML and Javascript, the IVY Workbench achieved evolutionary prototypes intended to become part of a system. Table 6.1 presents the updated version of the comparison table from Section 2.4.

Lastly, the new prototyping plugin developed in this project solved the issues of the last version, described in Section 3.1.5. The contributed solutions were the following:

- A SVG parser was created that can extract any SVG element from distinct sources, which removed the dependency of an external tool. This new parser also allows the configuration of any SVG element as opposed to the old solution that only allowed Pencil Project's labels and buttons.
- The addition of the dynamic widget framework reduced the complexity of the prototypes' building process, where it was required to replicate parts of the mock-up and configure each one of these parts to achieve the desired results.
- A more flexible prototypes' configuration approach was developed, which enables users to assign both actions and attributes of the formal model to elements of the UI mock-up. This approach also enriched on the prototype configuration capabilities of the old version by adding conditional states, event triggers, dynamic widget functions, and allowing the control of SVG properties.

6.2 Future Work

As previously mentioned in this document, future work should include improvements to fix the difficulties felt by the users involved in the usability tests. These improvements include the addition of project files to the tool. These should contain both the formal model and the

UI mock-up. Moreover, the developed SVG renderer should support SVG group selection and drag and drop functionalities.

More tests should also be conducted that cover more mock-up editors to evaluate their compatibility with the developed SVG engine and the external widgets. The environment could also extend the set of SVG properties supported by the internal widgets. Furthermore, the extension of the external widget library could vastly improve the creation of prototypes of diverse fields.

Another striking improvement concerning the widgets is the creation of a platform that offers the full documentation of the widget library. This platform could also allow expert users to upload new dynamic widgets. Such a platform has the potential of creating a community of advanced IVY users that can continuously promote the library of dynamic widgets. The more diverse and extended this library becomes, the more complete and diverse the tool's prototypes will be.

Improvements can also be made in the process of exporting prototypes. Currently, the prototypes are exported to HTML files that embed Javascript code and the UI mock-up as SVG. A more interesting solution is to transform the SVG layout into HTML and CSS. Extending this process to other file formats would also be a remarkable improvement.

Lastly, the workbench should also support the creation of user accounts and shared links. This feature could improve the development of prototypes in large teams and could also improve the UX of the developed mobile applications.

BIBLIOGRAPHY

J. M. Araújo, R. Couto, and J. C. Campos. A generator of user interface prototypes for the ivy workbench. In *2019 International Conference on Graphics and Interaction (ICGI)*, pages 32–39, Nov 2019. doi: 10.1109/ICGI47575.2019.8955088.

João Araújo. Gerador de protótipos de interfaces gráficas. Master's thesis, Universidade do Minho, 2019.

Benjamin Bähr. Rapid creation of sketch-based native android prototypes with "blended prototyping". In *Mobile HCI 2013—Workshop on Prototyping to Support the Interaction Designing in Mobile Application Development*, 2013.

Brian Bailey, Jacob Biehl, Damon Cook, and Heather Metcalf. Adapting paper prototyping for designing user interfaces for multiple display environments. *Personal and Ubiquitous Computing*, 12:269–277, 03 2008. doi: 10.1007/s00779-007-0147-2.

Michel Beaudouin-Lafon and Wendy Mackay. Prototyping tools and techniques. In Julie A. Jacko and Andrew Sears, editors, *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, chapter 52, pages 1006–1031. L. Erlbaum Associates Inc., 365 Broadway Hillsdale, NJUnited States, 2002.

Pietro C Cacciabue. *Guide to applying human factors methods: Human error and accident management in safety-critical systems*. Springer Science & Business Media, 2004.

Bradley Camburn, Vimal Viswanathan, Julie Linsey, David Anderson, Daniel Jensen, Richard Crawford, Kevin Otto, and Kristin Wood. Design prototyping methods: state of the art in strategies, techniques, and guidelines. *Design Science*, 3:e13, 2017. doi: 10.1017/dsj.2017.10.

J. C. Campos, M. Sousa, M. C. B. Alves, and M. D. Harrison. 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. ISSN 2168-2305. doi: 10.1109/THMS.2015.2421511.

J. Creissac Campos and M. D. Harrison. Systematic analysis of control panel interfaces using formal tools. In T. C. Nicholas Graham and Philippe Palanque, editors, *Interactive Systems. Design, Specification, and Verification*, pages 72–85, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-70569-7.

J.C. Campos, C. Fayollas, M.D. Harrison, C. Martinie, P. Masci, and P. Palanque. Supporting the analysis of safety critical user interfaces: an exploration of three formal tools. *ACM Transactions on Computer-Human Interaction*, 2020. accepted.

A. S. Carter and C. D. Hundhausen. How is user interface prototyping really done in practice? a survey of user interface designers. In *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 207–211, Sep. 2010. doi: 10.1109/VLHCC.2010.36.

Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification*, pages 359–364, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45657-5.

Rui Couto and José Creissac Campos. Ivy 2: A model-based analysis tool. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367455. doi: 10.1145/3319499.3328228. URL https://doi.org/10.1145/3319499.3328228.

Michael Deininger, Shanna R. Daly, Kathleen H. Sienko, and Jennifer C. Lee. Novice designers' use of prototypes in engineering design. *Design Studies*, 51:25 – 65, 2017. ISSN 0142-694X. doi: https://doi.org/10.1016/j.destud.2017.04.002. URL http://www.sciencedirect.com/science/article/pii/S0142694X17300273.

N. M. Devadiga. Tailoring architecture centric design method with rapid prototyping. In *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*, pages 924–930, 2017.

Steven P. Dow, Kate Heddleston, and Scott R. Klemmer. The efficacy of prototyping under time constraints. In *Proceedings of the Seventh ACM Conference on Creativity and Cognition*, C&C '09, page 165–174, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605588650. doi: 10.1145/1640233.1640260. URL https://doi.org/10.1145/1640233.1640260.

Christer Elverum and Torgeir Welo. The role of early prototypes in concept development: Insights from the automotive industry. *Procedia CIRP*, 21:491–496, 12 2014. doi: 10.1016/j.procir.2014.03.127.

Mica R. Endsley and Jones G. Debra. *Designing for Situation Awareness: An Approach to User-Centered Design, Second Edition*. CRC Press, Inc., USA, 2nd edition, 2011. ISBN 1420063553.

Daniel Engelberg and Ahmed Seffah. A framework for rapid mid-fidelity prototyping of web sites. In Judy Hammond, Tom Gross, and Janet Wesson, editors, *Usability: Gaining a Competitive Edge*, pages 203–215. Springer US, Boston, MA, 2002. ISBN 978-0-387-35610-5. doi: 10.1007/978-0-387-35610-5_14. URL https://doi.org/10.1007/978-0-387-35610-5_14.

Camille Fayollas, Célia Martinie, Philippe Palanque, Yannick Deleris, J.-C Fabre, and David Navarre. An approach for assessing the impact of dependability on usability: Application to interactive cockpits. *Proceedings - 2014 10th European Dependable Computing Conference, EDCC 2014*, pages 198–209, 05 2014. doi: 10.1109/EDCC.2014.17.

Wilbert O Galitz. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons, 2007.

Carmen Gervet, Yves CASEAU, and Denis Montaut. On Refining Ill-Defined Constraint Problems: A Case Study in Iterative Prototyping. In *PACLP Practical Applications of Constraint Logic Programming*, London, United Kingdom, 1999. URL https://hal.umontpellier.fr/hal-01742389.

Giovanni Guida, Gianfranco Lamperti, and Marina Zanella. *Software Prototyping in Data and Knowledge Engineering*. Kluwer Academic Publishers, USA, 1999. ISBN 0792360168.

David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231 – 274, 1987. ISSN 0167-6423. doi: https://doi.org/10.1016/0167-6423(87)90035-9. URL http://www.sciencedirect.com/science/article/pii/0167642387900359.

Henrik Hertel and Anke Dittmar. Design support for integrated evolutionary and exploratory prototyping. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '17, page 105–110, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350839. doi: 10.1145/3102113.3102145. URL https://doi.org/10.1145/3102113.3102145.

ISO. Human-centred design processes for interactive systems. Standard, International Organization for Standardization, Geneva, CH, June 1999.

Gabriel Jakobson. Collaborative web browsing system having document object model element interaction detection. https://patents.google.com/patent/US8769017B2/en, July 1 2014. US Patent 8769017B2.

Maria Johansson and Mattias Arvola. A case study of how user interface sketches, scenarios and computer prototypes structure stakeholder meetings. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but Not as We Know It - Volume*

1, BCS-HCI '07, page 177–184, Swindon, GBR, 2007. BCS Learning & Development Ltd. ISBN 9781902505947.

Frank W. Liou. *Rapid Prototyping and Engineering Applications (Dekker Mechanical Engineering)*. CRC Press, Inc., USA, 2007. ISBN 0849334098.

Charisa F. Llema and Cenie M. Vilela-Malabanan. Design and development of mlerws: A user-centered mobile application for english reading and writing skills. *Procedia Computer Science*, 161:1002 – 1010, 2019. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2019.11.210. URL http://www.sciencedirect.com/science/article/pii/S1877050919319210. The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.

Marina Machado, Rui Couto, and José Creissac Campos. Modus: model-based user interfaces prototyping. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2017, Lisbon, Portugal, June 26-29, 2017*, page 111–116, 2017. doi: 10.1145/3102113.3102146. URL http://doi.acm.org/10.1145/3102113.3102146. n/a .

Paolo Masci, Yi Zhang, Paul Jones, Patrick Oladimeji, Enrico D'Urso, Cinzia Bernardeschi, Paul Curzon, and Harold Thimbleby. Combining pvsio with stateflow. In Julia M. Badger and Kristin Yvonne Rozier, editors, *NASA Formal Methods: 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 – May 1, 2014. Proceedings*, volume 8430, pages 209–214. Springer International Publishing, Cham, 2014. ISBN 978-3-319-06200-6. doi: 10.1007/978-3-319-06200-6_16. URL http://dx.doi.org/10.1007/978-3-319-06200-6_16.

Paolo Masci, Patrick Oladimeji, Yi Zhang, Paul Jones, Paul Curzon, and Harold Thimbleby. PVSio-web 2.0: Joining PVS to HCI. In Daniel Kroening and Corina S. Păsăreanu, editors, *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pages 470–478. Springer International Publishing, 2015. ISBN 978-3-319-21690-4. doi: 10.1007/978-3-319-21690-4_30. URL http://dx.doi.org/10.1007/978-3-319-21690-4_30.

Gioacchino Mauro, Harold Thimbleby, Andrea Domenici, and Cinzia Bernardeschi. Extending a user interface prototyping tool with automatic MISRA C code generation. In *Proceedings of the Third Workshop on Formal Integrated Development Environment, F-IDE@FM 2016, Limassol, Cyprus, November 8, 2016*, pages 53–66, 2016. doi: 10.4204/EPTCS.240.4. URL https://doi.org/10.4204/EPTCS.240.4.

David Navarre, Philippe Palanque, Rémi Bastide, and Ousmane Sy. Structuring interactive systems specifications for executability and prototypability. In *Proceedings of the 7th*

International Conference on Design, Specification, and Verification of Interactive Systems, DSV-IS'00, page 97–119, Berlin, Heidelberg, 2000. Springer-Verlag. ISBN 3540416633.

David Navarre, Philippe Palanque, Rémi Bastide, and Ousmane Sy. A model-based tool for interactive prototyping of highly interactive applications. In *Proceedings of the 12th International Workshop on Rapid System Prototyping*, RSP '01, page 136, USA, 2001. IEEE Computer Society.

Anh Nguyen-Duc, Xiaofeng Wang, and Pekka Abrahamsson. What influences the speed of prototyping? an empirical investigation of twenty software startups. In Hubert Baumeister, Horst Lichter, and Matthias Riebisch, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 20–36, Cham, 2017. Springer International Publishing. ISBN 978-3-319-57633-6.

Addy Osmani. *Learning JavaScript Design Patterns: A JavaScript and jQuery Developer's Guide*. " O'Reilly Media, Inc.", 2012.

S. Owre, J. M. Rushby, and N. Shankar. Pvs: A prototype verification system. In Deepak Kapur, editor, *Automated Deduction—CADE-11*, pages 748–752, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-47252-0.

Philippe Palanque, Jean-François Ladry, David Navarre, and Eric Barboni. High-fidelity prototyping of interactive systems can be formal too. In *Human-Computer Interaction. New Trends*, volume 5610, pages 667–676, 07 2009. doi: 10.1007/978-3-642-02574-7_75.

Thiago Rocha Silva, Jean-Luc Hak, and Marco Antonio Winckler. A Review of Milestones in the History of GUI Prototyping Tools. In *15th IFIP TC.13 International Conference on Human-Computer Interaction (INTERACT 2015)*, pages pp. 1–12, Bamberg, Germany, September 2015. URL https://hal.archives-ouvertes.fr/hal-01343040.

Gernot Rottermanner, Markus Wagner, Martin Kalteis, Michael Iber, Peter Judmaier, Wolfgang Aigner, Volker Settgast, and Eva Eggeling. Low-fidelity prototyping for the air traffic control domain. In Raimund Dachselt and Gerhard Weber, editors, *Mensch und Computer 2018 - Workshopband*, Bonn, 2018. Gesellschaft für Informatik e.V. doi: 10.18420/muc2018-ws12-0401.

Mark Ryan, José Fiadeiro, and Tom Maibaum. Sharing actions and attributes in modal action logic. In Takayasu Ito and Albert R. Meyer, editors, *Theoretical Aspects of Computer Software*, pages 569–593, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. ISBN 978-3-540-47617-7.

Daniel Schwarz. *Jump Start Adobe XD*. Sitepoint, 1 edition, 2017. ISBN 0995382611.

Iris Soute, Tudor Vacaretu, Jan De Wit, and Panos Markopoulos. Design and evaluation of rapido, a platform for rapid prototyping of interactive outdoor games. *ACM Trans. Comput.-Hum. Interact.*, 24(4), August 2017. ISSN 1073-0516. doi: 10.1145/3105704. URL https://doi.org/10.1145/3105704.

Sarah Suleri, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke. Eve: A sketch-based software prototyping workbench. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI EA '19, page 1–6, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359719. doi: 10.1145/3290607.3312994. URL https://doi.org/10.1145/3290607.3312994.

P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools: the mastermind approach. In Leonard J. Bass and Claus Unger, editors, *Engineering for Human-Computer Interaction: Proceedings of the IFIP TC2/WG2.7 working conference on engineering for human-computer interaction, Yellowstone Park, USA, August 1995*, pages 120–150. Springer US, Boston, MA, 1996. ISBN 978-0-387-34907-7. doi: 10.1007/978-0-387-34907-7_8. URL https://doi.org/10.1007/978-0-387-34907-7_8.

Hock-Hai Teo, Lih-Bin Oh, Chunhui Liu, and Kwok-Kee Wei. An empirical study of the effects of interactivity on web user attitude. *International Journal of Human-Computer Studies*, 58(3):281 – 305, 2003. ISSN 1071-5819. doi: https://doi.org/10.1016/S1071-5819(03)00008-9. URL http://www.sciencedirect.com/science/article/pii/S1071581903000089.

Robert A. Virzi, Jeffrey L. Sokolov, and Demetrios Karis. Usability problem identification using both low- and high-fidelity prototypes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '96, page 236–243, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917774. doi: 10.1145/238386.238516. URL https://doi.org/10.1145/238386.238516.

V. K. Viswanathan and J. S. Linsey. Enhancing student innovation: Physical models in the idea generation process. In *2009 39th IEEE Frontiers in Education Conference*, pages 1–6, 2009.

W3C. Scalable vector graphics (svg) 2. https://www.w3.org/TR/SVG/, 2018. [Online; accessed 14-January-2020].

Simon Wood and Pablo Romero. User-centred design for a mobile learning application. In *Proceedings of the 3rd Mexican Workshop on Human Computer Interaction*, MexIHC '10, page 77–84, San Luis Potosí, S.L.P, MEX, 2010. Universidad Politécnica de San Luis Potosí.

A

LISTINGS

A.1 SWITCH WIDGET

Listing A.1: Full code of the switch widget.

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org
   /1999/xlink" viewBox="0 0 60 100">
  <g  xmlns:ivy="http://ivy.di.uminho.pt/ivy" id="ivy-widget:Switch"
     ivy:id="ivy-widget:Switch" ivy:widget="Switch">
    <script type="application/javascript">
    //<![CDATA[
          var Switch = (function() {
            var isChecked = false;
            var visibility = "";
            var uncheckedColor = "#b1b1b1";
            var checkedColor = "#3becc8";

            function setUncheckedColor(color) {
              uncheckedColor = color;

              if (!isChecked) {
                setColor(color);
              }
            }

            function setCheckedColor(color) {
              checkedColor = color;

              if (isChecked) {
                setColor(color);
              }
            }
```

```
function setColor(color) {
  var thumb = document.getElementById("ivy:switch-thumb");
  var trackElements = document.getElementById("
      ivy:switch-track");
  var circles = trackElements.getElementsByTagName("circle");
  var rect = trackElements.getElementsByTagName("rect").item
      (0);

  thumb.setAttribute("style", "fill: " + color);
  circles.item(0).setAttribute("style", "fill: " + color);
  circles.item(1).setAttribute("style", "fill: " + color);
  rect.setAttribute("style", "fill: " + color);
}

function setChecked(checked) {
  var thumb = document.getElementById("ivy:switch-thumb");
  var shadow = document.getElementById("ivy:switch-shadow");
  var leftCircle = document.getElementById("
      ivy:switch-left-circle");
  var rightCircle = document.getElementById("
      ivy:switch-right-circle");
  var shadowR = parseInt(shadow.getAttribute("width")) / 2 -
      parseInt(thumb.getAttribute("r"));

  isChecked = checked;

  if (isChecked) {
    thumb.setAttribute("cx", rightCircle.getAttribute("cx"));
    setColor(checkedColor);
  } else {
    thumb.setAttribute("cx", leftCircle.getAttribute("cx"));
    setColor(uncheckedColor);
  }

  shadow.setAttribute("x", parseInt(thumb.getAttribute("cx"))
      - parseInt(thumb.getAttribute("r")) - shadowR);
}

function toggle() {
  isChecked = !isChecked;
  setChecked(isChecked);
}
```

```
            function setVisible(visible) {
              visibility = visible ? "" : "hidden";

              if (visibility == "") {
                document.getElementById("ivy-widget:Switch").
                    removeAttribute("visibility");
              } else {
                document.getElementById("ivy-widget:Switch").setAttribute
                    ("visibility", visibility);
              }
            }

            return {
              toggle: toggle,
              setChecked: setChecked,
              setVisible: setVisible,
              setUncheckedColor: setUncheckedColor,
              setCheckedColor: setCheckedColor,
              props: {
                setChecked: {
                  params: [
                    {
                      name: "Checked",
                      type: "boolean",
                      desc: "<html>Checks or unchecks the switch.<br>
                          Checks the switch if the received value<br>is
                          true and unchecks it otherwise.</html>",
                      default: false
                    }
                  ]
                },
                setVisible: {
                  params: [
                    {
                      name: "Visibility",
                      type: "visibility",
                      desc: "Hides or display the switch.",
                      default: "visible"
                    }
                  ]
                },
                setUncheckedColor: {
                  params: [
```

```
                    {
                      name: "Unchecked color",
                      type: "color",
                      desc: "<html>Changes the color of the switch
                         when<br>it is in a unchecked state.</html>",
                      default: "#b1b1b1"
                    }
                  ]
                },
                setCheckedColor: {
                  params: [
                    {
                      name: "Checked color",
                      type: "color",
                      desc: "<html>Changes the color of the switch
                         when<br>it is in a checked state.</html>",
                      default: "#3becc8"
                    }
                  ]
                }
              }
            };
          })();
        //]]>
    </script>
    <g ivy:id="ivy:switch-track" id="ivy:switch-track" opacity="0.4"
       fill="#b1b1b1">
      <circle cx="20" cy="30" r="8" id="ivy:switch-left-circle" ivy:id="
         ivy:switch-left-circle"/>
      <circle cx="44" cy="30" r="8" id="ivy:switch-right-circle" ivy:id="
         ivy:switch-right-circle"/>
      <rect x="20" y="22" width="24" height="16"/>
    </g>

    <!-- Base 64 data omitted for the sake of simplicity -->
    <image id="ivy:switch-shadow" ivy:id="ivy:switch-shadow" x="5" y="16"
        width="30" height="30" opacity="0.2" xlink:href="data:image/
       png;base64"/>
    <circle cx="20" cy="30" r="12" ivy:id="ivy:switch-thumb" id="
       ivy:switch-thumb" fill="#b1b1b1"/>
  </g>
</svg>
```

A.2 B. Braun Perfusor ® Space Formal Model

Listing A.2: Formal model of the medical device B. Braun Perfusor Space.

```
defines
  MAXDIG = 4
  MAXINT = 9
types
  int = 0..MAXINT
  digit = 0..MAXDIG

interactor main
  attributes
    [vis] digits: array 0..MAXDIG of int
    [vis] position: 0..4
    [vis] running: boolean
  actions
    [vis] up
    [vis] down
    [vis] left
    [vis] right
    [vis] startStop
    [vis] infuse
  axioms
    [] digits[0]=0 & digits[1]=0 & digits[2]=0 & digits[3]=0 & digits
       [4]=0 & position = 0 & running = false

    per(up) -> digits[position] < MAXINT & running = false
    position = 0 -> [up] digits[0]' = digits[0] + 1 & keep(position,
       digits[1], digits[2], digits[3], digits[4], running)
    position = 1 -> [up] digits[1]' = digits[1] + 1 & keep(position,
       digits[0], digits[2], digits[3], digits[4], running)
    position = 2 -> [up] digits[2]' = digits[2] + 1 & keep(position,
       digits[0], digits[1], digits[3], digits[4], running)
    position = 3 -> [up] digits[3]' = digits[3] + 1 & keep(position,
       digits[0], digits[1], digits[2], digits[4], running)
    position = 4 -> [up] digits[4]' = digits[4] + 1 & keep(position,
       digits[0], digits[1], digits[2], digits[3], running)

    per(down) -> digits[position] > 0 & running = false
    position = 0 -> [down] digits[0]' = digits[0] - 1 & keep(position,
       digits[1], digits[2], digits[3], digits[4], running)
```

```
position = 1 -> [down] digits[1]' = digits[1] - 1 & keep(position,
    digits[0], digits[2], digits[3], digits[4], running)
position = 2 -> [down] digits[2]' = digits[2] - 1 & keep(position,
    digits[0], digits[1], digits[3], digits[4], running)
position = 3 -> [down] digits[3]' = digits[3] - 1 & keep(position,
    digits[0], digits[1], digits[2], digits[4], running)
position = 4 -> [down] digits[4]' = digits[4] - 1 & keep(position,
    digits[0], digits[1], digits[2], digits[3], running)

per(right) -> position > 0 & running = false
[right] position' = position - 1 & keep(digits, running)

per(left) -> position < MAXDIG & running = false
[left] position' = position + 1 & keep(digits, running)

[startStop] running'=!running & keep(digits, position)

per(infuse) -> running = true
digits[0] = 0 & digits[1] = 0 & digits[2] = 0 & digits[3] = 0 &
    digits[4] = 0 -> [infuse] running' = false & keep(digits, position
    )
digits[0] > 0 -> [infuse] digits[0]' = digits[0] - 1 & keep(running,
    position, digits[1], digits[2], digits[3], digits[4])
digits[0] = 0 & digits[1] > 0 -> [infuse] digits[0]'=9 & digits[1]'=
    digits[1] - 1 & keep(running, position, digits[2], digits[3],
    digits[4])
digits[0] = 0 & digits[1] = 0 & digits[2] > 0 -> [infuse] digits
    [0]'=9 & digits[1]'=9 & digits[2]'=digits[2] - 1 & keep(running,
    position, digits[3], digits[4])
digits[0] = 0 & digits[1] = 0 & digits[2] = 0 & digits[3] > 0 -> [
    infuse] digits[0]'=9 & digits[1]'=9 & digits[2]'=9 & digits[3]'=
    digits[3] - 1 & keep(running, position, digits[4])
digits[0] = 0 & digits[1] = 0 & digits[2] = 0 & digits[3] = 0 &
    digits[4] > 0 -> [infuse] digits[0]'=9 & digits[1]'=9 & digits
    [2]'=9 & digits[3]'=9 & digits[4]'=digits[4] - 1 & keep(running,
    position)
```

B

USABILITY TESTS DOCUMENTS

B.1 CONSENT FORM

Formulário de Consentimento

Obrigado por participar na nossa investigação.
Nós vamos gravar esta sessão para permitir que colaboradores do HASLab/INESC TEC a visualizem e beneficiem dos seus comentários.
Por favor leia o texto abaixo e assine onde indicado.

Compreendo que esta sessão de testes de usabilidade vai ser gravada.
Eu dou permissão aos colaboradores do HASLab/INESC TEC para utilizarem esta gravação unicamente para utilização interna, com o propósito de melhorar os projetos em teste.

Assinatura: _____

B.2 EXPERIMENT SCRIPT

IVY Workbench - Guião Experimental

"Antes de começarmos, gostaríamos de agradecer a sua participação neste estudo. Esta sessão vai durar cerca de 10 minutos. Durante esse tempo, vai interagir com uma plataforma que permite criar protótipos de sistemas interativos – IVY Workbench".

1 Introdução

A ferramenta IVY Workbench é uma plataforma que permite criar protótipos de sistemas interativos. Os protótipos criados a partir desta ferramenta são divididos em duas componentes: um modelo formal que expressa o comportamento do sistema e uma *mock-up* que ilustra a interface com o utilizador desse mesmo sistema. Pretende-se, com recurso a este guião experimental, explorar as capacidades de prototipagem desta ferramenta através do desenvolvimento de um protótipo do dispositivo médico B. Braun Perfusor ® Space[1]. Para isso, este guião apresenta uma breve descrição das funcionalidades da ferramenta necessárias para a construção do protótipo desejado, assim como um exemplo passo a passo de um protótipo de um *switch*. No final deste guião apresenta-se um breve questionário de usabilidade da ferramenta.

2 Descrição das funcionalidades

A ferramenta IVY Workbench (ilustrada na Figura 1) possui diferentes ambientes de execução organizados por separadores. Dentro destes, dá-se ênfase aos ambientes "Editor" e "Prototyper". O primeiro ambiente diz respeito à construção de modelos formais que expressam o comportamento de um sistema. Neste guião, este ambiente apenas será utilizado para importar o modelo formal, já que este ambiente destina-se a utilizadores mais experientes desta ferramenta. Do modelo formal em concreto, apenas é importante reter a informação que este contém ações e atributos. As ações modificam o valor dos atributos e os atributos correspondem a variáveis que expressam o comportamento do protótipo.

O segundo ambiente é o ponto fulcral deste guião e será utilizado para adicionar o comportamento definido num modelo formal a uma *mock-up* para sua posterior animação. As Figuras 1 e 2, ilustram os componentes mais relevantes

[1] `https://www.bbraun.pt/pt/products/b/perfusor-space.html`

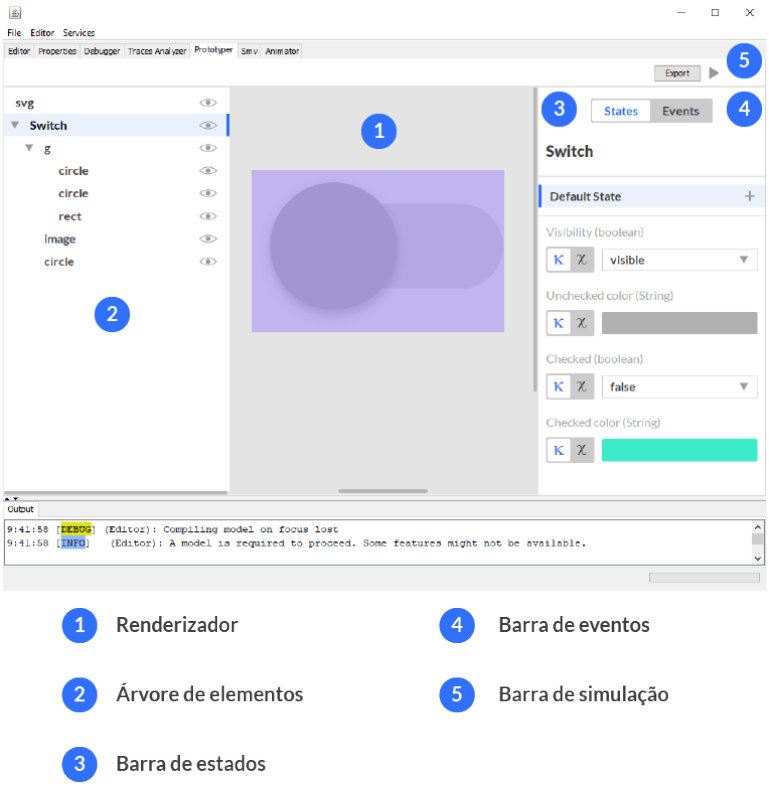


Figura 1: Ilustração da ferramenta IVY Workbench.

deste ambiente de execução. As funcionalidades destes componentes relevantes para este guião são as seguintes:

- **Renderizador** – Este componente é responsável por apresentar a *mock-up* da interface do sistema a prototipar. Sempre que é efetuado um clique num elemento da *mock-up*, este torna-se automaticamente selecionado através de um fundo semi opaco roxo.
- **Árvore de elementos** – Apresenta a hierarquia interna de todos os elementos que constituem a *mock-up* importada. Este componente é uma alternativa ao renderizador para selecionar elementos. No entanto, este permite selecionar grupos de elementos e *widgets*. Os *widgets* são elementos pertencentes à plataforma IVY Workbench que podem ser utilizados para construir *mock-ups*. Como regra geral, estes são apresentados na árvore com um nome mais sugestivo que os outros elementos e iniciado por uma letra maiúscula. O principal objetivo destes elementos é o de

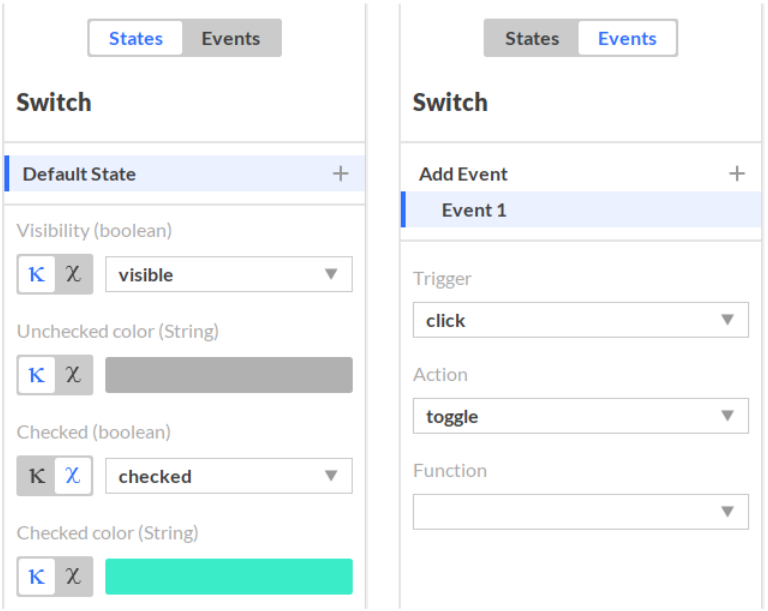


Figura 2: Barras de estados e eventos.

apresentar um conjunto de configurações mais específicas para o protótipo a desenvolver. Na Figura 1, pode-se verificar que a árvore de elementos possui um destes widgets chamado "Switch". A árvore de elementos possui outras funcionalidades como a inserção de sub *mock-ups*, mas que não serão relevantes para este guião.

- **Barra de eventos** – Os eventos estão correlacionados com as interações do utilizador com o protótipo, tais como cliques ou movimento do rato, e sobre que ações do modelo formal executar sempre que ocorre uma dessas interações. Na versão atual desta ferramenta pode-se configurar como interações ("Trigger") o clique ou movimento do rato, assim como o pressionar do tecla ou um temporizador que é acionado num determinado período em segundos. Além disso, a *combo box* "Action" permite selecionar que ação do modelo executar. Finalmente a *combo box* "Function" diz respeito a executar determinadas componentes dos *widgets* das *mock-ups*, mas não é relevante para os objetivos deste guião. A barra de eventos permite também adicionar um conjunto ilimitado de eventos a um dado elemento através do ícone "+". No entanto, esta funcionalidade serve para resolver protótipos mais complexos pelo que não será necessária para este guião.

- **Barra de estados** – Os estados dizem respeito ao modo de como o protótipo é modificado sempre que um determinado evento é despoletado. A barra de estados lista o conjunto de possíveis propriedades que podem ser configuradas para um elemento em específico. Cada propriedade pode ser controlada por um valor constante (identificado pelo ícone "K") ou por um atributo do modelo formal (identificado pelo ícone "x"). Uma propriedade atribuída por um valor constante (por exemplo o valor de uma cor) mantém-se constante durante toda a simulação do protótipo. Já uma propriedade controlada por um atributo reflete o valor desse atributo durante toda a simulação. Como exemplo, considere-se um atributo que controla os minutos de um relógio. Se esse atributo for atribuído a um elemento de texto da *mock-up*, então durante a simulação o texto é atualizado sempre que o valor desse atributo é modificado. Tal como os eventos, também é possível adicionar mais sub estados através do ícone "+". Esta funcionalidade também se destina a protótipos mais avançados pelo que não será relevante para este guião.

- **Barra de simulação** – Neste guião esta barra apenas será utilizada para simular as configurações efetuadas para a construção do protótipo. Ao efetuar-se um clique no ícone *play*, a ferramenta deverá apresentar uma janela com o protótipo desenvolvido.

3 Switch

Para se conhecer melhor as funcionalidades desta ferramenta, segue um exemplo de uma criação de um protótipo relativamente simples detalhada com todos os passos necessários. O protótipo em questão corresponde a um *switch*, ou *toggle button* que deverá ser modelado de forma a apresentar as variações da Figura 3.

O modelo formal deste protótipo é relativamente simples, possuindo apenas a ação *toggle* e o atributo *checked*. Este atributo possui apenas dois valores possíveis que indicam se o *switch* está ativo ou inativo. Relativamente à ação, esta alterna o valor do atributo mencionado. Posto isto, a sequência de passos necessários para a construção deste protótipo é a seguinte:

1. **Importação dos ficheiros necessários** – Comece por importar o ficheiro "switch.i" no separador "Editor". De seguida importe o ficheiro "switch.svg" que contém a *mock-up* no separador "Prototyper".

2. **Configuração dos estados** – Selecione o *widget Switch* na árvore de elementos. Na barra de estados pode verificar algumas propriedades deste elemento que podem ser controladas. Dentro destas propriedades pode-se verificar a propriedade *checked*. Selecione o ícone "x" de modo a que este possa ser controlado por um atributo. Dentro da caixa de atributos disponíveis deverá aparecer como única opção o atributo *checked*. Selecione esse atributo.

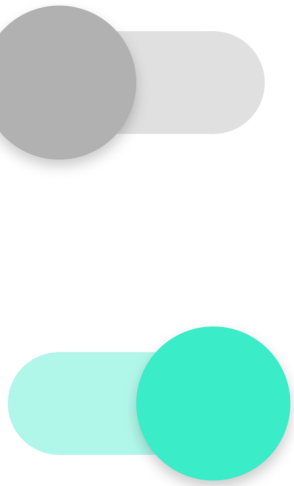


Figura 3: Resultados esperados do protótipo *switch*.

3. **Configuração dos eventos** – Este protótipo apenas necessita de um evento que pode estar associado a um clique e que irá alternar o *switch* entre o estado ativo e inativo. Tendo o elemento *Switch* selecionado na árvore de elementos selecione o separador "Events". Por defeito estará selecionada a opção clique como a ação do utilizador que despoleta este evento. Na caixa relativa à ação ("Action") selecione a única ação disponível do modelo.

4. **Simulação do protótipo** – Após os passos efetuados acima, clique no ícone *play* na barra acima da *mock-up*. Deverá ser apresentada uma janela semelhante à Figura 4. Esta janela apresenta o protótipo desenvolvido e a lista de ações possíveis deste protótipo. O protótipo deverá apresentar o resultado desejado sempre que se clica diretamente na *mock-up* ou na ação *toggle*.

Como passos opcionais deste protótipo, feche a janela de simulação e modifique as cores do *Switch* quando este está ativo (*checked*) e inativo (*unchecked*).

4 B. Braun Perfusor ® Space

Este dispositivo médico funciona como uma bomba de infusão para seringas e permite criar configurações acerca da quantidade de um fármaco a ser aplicado num determinado período de tempo. Este dispositivo possui bastantes modelos de configuração. No entanto, o objetivo deste guião é criar um protótipo que

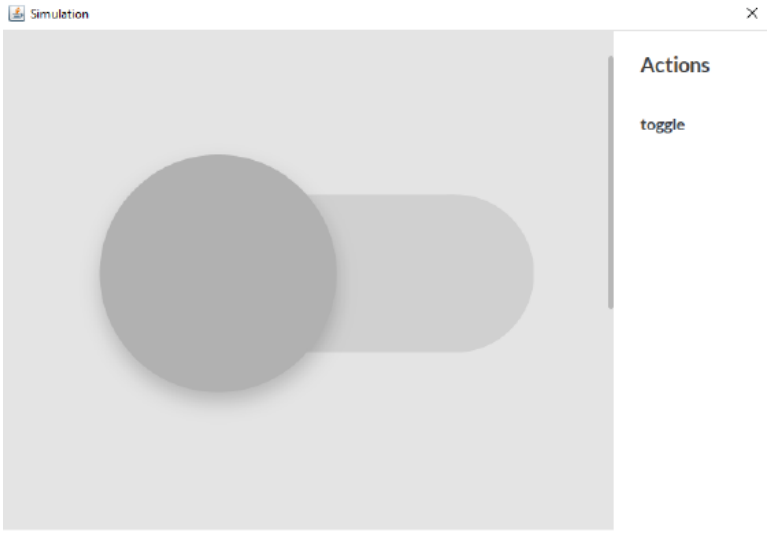


Figura 4: Janela de simulação do protótipo *switch*.

simule a quantidade de um fármaco a aplicar. O resultado deverá ser semelhante ao ilustrado na Figura 5. Nesta figura, denote-se o ecrã com 5 dígitos com um cursor (identificado pelo fundo preto e dígito a branco) e os botões "cima", "baixo", "esquerda" e "direita". O protótipo deverá apresentar os seguintes aspetos:

- Quando é efetuado um clique no botão "cima", deverá ser aumentado o valor do dígito na posição atual do cursor.
- Quando é efetuado um clique no botão "baixo", deverá diminuir o valor do dígito na posição atual do cursor.
- O cursor deverá mover-se para a esquerda ou direita, sempre que se efetuar um clique nos botões "esquerda" ou "direita".

O modelo formal possui as ações *up*, *down*, *left* e *right*. Além disso, contém o atributo *digits* que diz respeito aos valores dos dígitos do ecrã do dispositivo, e o atributo *position* que diz respeito à posição do cursor. A *mock-up* contém o *widget Cursor* que possui propriedades que auxiliam a construção do protótipo em questão. Posto isto, efetue os seguintes passos para desenvolver o protótipo desejado:

1. Importe o ficheiro "bBraunPerfusor.i" no separador "Editor" e o ficheiro "bBraunPerfusor.svg" no separador "Prototyper".

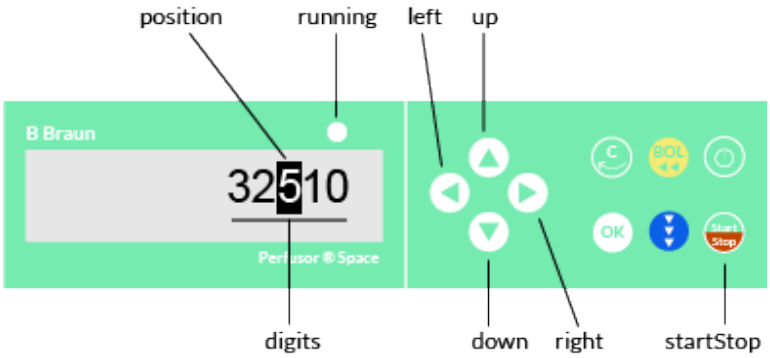


Figura 5: *Mock-up* da bomba de infusão e respetiva associação entre os seus elementos e as ações e atributos do modelo formal.

2. Configure o estado do *widget Cursor*. Deverá configurar as suas duas propriedades (*digits* e *position*) com os atributos adequados.

3. Configure os eventos dos botões mencionados com recurso às ações *up*, *down*, *left* e *right*. Estes eventos deverão ser executados sempre que se efetua um clique. NOTA: os botões são constituídos por mais que um elemento. Pode, por isso, ser proveitoso selecionar o conjunto dos seus elementos (definidos por um elemento g na árvore de elementos).

4. Simule o protótipo e verifique se este apresenta o comportamento esperado.

Considere agora as ações *startStop* e *infuse*. A primeira ação indica o início ou o fim do processo de infusão. Já a ação *infuse* simula a aplicação do fármaco. Neste caso, esta última ação corresponderá a uma diminuição do valor apresentado no ecrã do dispositivo. Considere ainda o atributo *running* que indica se o processo de infusão está em execução. Pretende-se, como objetivo complementar ao proposto acima, acrescentar o seguinte comportamento ao protótipo:

- Quando se clicar no botão "start/stop" do dispositivo, este deve iniciar/terminar a infusão.

- Quando a infusão está em execução o *led* do dispositivo (representado por um círculo acima do ecrã) deverá estar ligado. Por conseguinte, este *led* deverá estar desligado quando a infusão é terminada.

- Quando a infusão está em execução, então o valor apresentado no ecrã deverá diminuir a cada 1 segundo.

Face ao comportamento complementar anunciado execute os seguintes passos:

1. Configure o evento do botão "start/stop" para executar ou parar a simulação. Deverá utilizar a ação *starStop* do modelo.
2. Configure o estado *widget Led* de modo a que este apresente o comportamento esperado. Para isso configure a sua propriedade *On* de modo a esta ser controlada pelo atributo *running*. NOTA: poderá ser necessário mudar o valor da cor ativa (*On Color*) deste elemento de modo a que este seja perceptível na simulação.
3. Configure a execução da infusão através de um evento despoletado por um *timer* de 1 segundo, na *combo box* relativa ao *Trigger*. Pode adicionar este evento ao *widget Cursor*. Este evento deverá executar a ação *infuse*.
4. Simule o protótipo e verifique se este apresenta o comportamento esperado.

5 Questionário de usabilidade

Participante: _____

Data de nascimento: __/__/_____

Sexo: M__ F__

Possui alguma experiência de utilização de editores de *mock-ups*? Sim__ Não__

Data: _____

1. Qual foi a sua impressão em geral do sistema?

2. Quais foram os aspetos do sistema que mais gostou?

3. Quais foram os aspetos do sistema que menos gostou?

4. O sistema continha algumas funcionalidades que o/a deixou surpreendido/a?

5. Existem algumas funcionalidades que esperava encontrar nesta ferramenta, mas que lhe estejam a faltar?

6. Compare a capacidade de adição de comportamento a protótipos desta ferramenta com as de outras ferramentas que tenha utilizado.

7. Recomendaria o uso deste sistema?