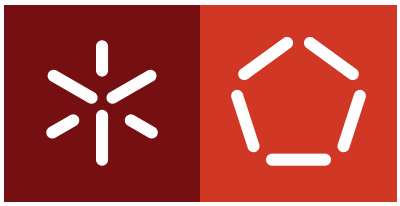


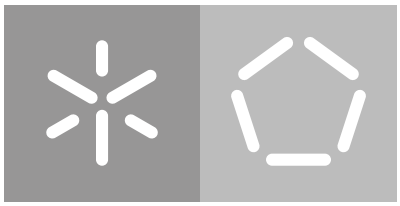
Universidade do Minho
Escola de Engenharia
Departamento de Informática

João Miguel Matela Aidos Manso de Araújo

Gerador de protótipos de interfaces gráficas para o IVY Workbench

Dissertação

January 2020



Universidade do Minho
Escola de Engenharia
Departamento de Informática

João Miguel Matela Aidos Manso de Araújo

Gerador de protótipos de interfaces gráficas para o IVY Workbench

Dissertação

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
José Francisco Creissac Freitas Campos
Rui Miguel Silva Couto

January 2020

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

GERADOR DE PROTÓTIPOS DE INTERFACES GRÁFICAS

RESUMO

A interface de um programa é um elemento importante na experiência que o utilizador tem com o software, pois constitui o principal método de interação com a lógica do programa. A existência de métodos fiáveis de verificação de sistemas de software permite o a conceção destes de acordo com a especificação e, em casos mais críticos, evitar erros com consequências graves. Estes métodos rigorosos, no entanto, contrastam com a prática mais comum no desenho de interfaces. Um dos métodos mais utilizados para o desenho e avaliação de interfaces é a prototipagem. Os protótipos permitem transmitir aspetos do design da interface e até avaliar a sua usabilidade, mas não oferecem as garantias sobre o seu funcionamento que os métodos de verificação oferecem.

O IVY Workbench é uma ferramenta que suporta a modelação do comportamento de sistemas interativos e a verificação formal dos mesmos. A ferramenta contém um conjunto de *plugins* que suportam o processo de modelação e análise, incluindo um editor de modelos, um verificador de propriedades e um animador. Este último permite visualizar e interagir com os modelos, mas não suporta associá-los a *mockups* representativos das interfaces.

A interação com os modelos facilita a sua validação por parte de quem os está a desenvolver. Não facilita, no entanto, a comunicação com os potenciais *clientes* do sistema modelado, para quem um protótipo será um meio mais eficaz de comunicação.

Neste documento propõe-se uma solução para o problema acima, assente no desenvolvimento de um novo *plugin* capaz de suportar a construção e animação de protótipos de sistemas interativos modelados no IVY. É descrito todo o processo de desenvolvimento, desde o levantamento de requisitos, até exemplos de aplicação que permitem demonstrar as novas funcionalidades existentes.

PALAVRAS-CHAVE

Interfaces com o Utilizador, Prototipagem, Modelação e análise formais

A GENERATOR OF USER INTERFACE PROTOTYPES

ABSTRACT

A program's interface is the most important element in the user's experience with the software, because it is the primary method through which the user interacts with the program's logic. The availability of reliable software verification methods allows its conception according to specification and, in critical cases, to avoid errors with grave consequences. These rigorous methods, however, are in contrast with the traditional approaches to user interface design. One of the most used methods for designing and evaluating interfaces is prototyping. Prototypes allow communication of design and usability aspects of a software system, but don't offer any guarantees about their behaviour.

IVY Workbench is a tool that supports modeling interactive system's behavior and their respective formal verification. The tool contains a set of plugins that enable the modeling and analysis process, including a model editor, a trace analyzer and an animator. This last feature allows visualization and interaction with models but doesn't support linking them with mockups that represent them.

Model interaction makes it easier for developers to validate its behavior. However, it doesn't facilitate the communication of the system's behavior to potential clients, to which a prototype would be the most efficient communication path.

In this document a solution to the above problem is presented, based on the development of a new plugin that will support the construction and animation of prototypes of interactive systems modeled in IVY. The document describes the undertaken development process, from requirements elicitation to practical examples that help demonstrate the new available features.

KEYWORDS

User Interfaces, Prototyping, Formal analysis and modeling

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGENS

LISTA DE ACRÓNIMOS

API Application Programming Interface. 1, 20

CTL Computational Tree Logic. 1, 6

DI Departamento de Informática. 1, 5

GUI Graphical User Interface. 1, 22, 36

HTML HyperText Markup Language. 1, 20, 22, 25, 26

JPEG Joint Photographic Experts Group. 1, 22, 25, 26
JSON JavaScript Object Notation. 1, 20, 25, 26, 40

KLM Keystroke-Level Modelling. 1, 11

MAL Model Action Language. x, 1, 2, 5, 6, 8, 47
MVC Model-View-Controller. 1, 32, 33

ODT Open Document Format for Office Applications. 1, 20, 25

PDF Portable Document Format. 1, 20, 25
PNG Portable Network Graphics. 1, 20, 22, 25
PVS Prototype Verification System. 1, 12

SMIL Synchronized Multimedia Integration Language. 1, 28
SVG Scalable Vector Graphics. vii, 1, 20, 25–34, 36, 38–41, 43, 46, 53, 55–57

UM Universidade do Minho. 1, 5

W3C World Wide Web Consortium. 1, 26, 55

XML Extensible Markup Language. 1, 25–27, 30, 55

1

INTRODUÇÃO

A interface de um programa é um elemento importante na experiência que o utilizador tem com o software, pois constitui o principal método de interação com a lógica do programa. A existência de métodos fiáveis que nos forneçam formas de analisar de modo rigoroso o comportamento da interface torna-se indispensável quando é necessário fornecer garantias sobre a qualidade de um sistema interativo (Campos, 2014). Em certos sistemas, como equipamento médico, a aplicação destes métodos de modelação e verificação formal permite detectar potenciais problemas de interação e, assim, evitar erros com consequências graves. Os modelos formais, no entanto, não permitem comunicar de forma simples o design da interface. Métodos como a prototipagem auxiliam a definir o âmbito e as limitações de sistemas nas suas diversas fases de desenvolvimento.

1.1 PROTOTIPAGEM

O principal método disponível para auxiliar no desenho e avaliação de uma interface é o protótipo. Define-se um protótipo como uma representação concreta de uma parte ou a totalidade de um sistema interativo (Maler et al., 2004). O protótipo pode ser elaborado em diferentes fases do desenvolvimento de um sistema. Dependendo da fase em que for criado, pode representar um esboço com o objetivo de definir as linhas gerais do sistema, ou uma simulação detalhada para obter dados de usabilidade sobre determinadas interações. Regra geral, protótipos concebidos em fases iniciais de um projeto são de complexidade reduzida, enquanto que protótipos concebidos já durante o desenvolvimento replicam uma parte ou a totalidadade do sistema, podendo até ser integrados no sistema final.

É possível distinguir duas abordagens para a representação de protótipos:

- Baixa Fidelidade - habitualmente desenhados durante as fases iniciais do desenvolvimento, têm o objetivo de definir de forma básica o aspeto da interface do sistema e a forma da a utilizar. Estes protótipos devem ser iterados regularmente até se atingir um resultado satisfatório. Devido à sua natureza temporária, normalmente são concebidos em papel, notas, *whiteboards* ou *mockups* (isto é, esboços). Este tipo de protótipos são facilmente acessíveis a todos os envolvidos no processo de conceção.

- Alta Fidelidade - descrevem o sistema em maior detalhe que os protótipo de baixa fidelidade. Recorrem a ferramentas informáticas para melhor descrever o protótipo. Normalmente, utiliza-se esta abordagem numa fase menos preliminar do desenvolvimento de um sistema. No entanto, protótipos de alta fidelidade podem igualmente ser utilizados em fases iniciais para facilitar a comunicação de ideias. Devido ao maior grau de complexidade e ao uso de ferramentas para a descrição mais detalhada do protótipo, este não é tão acessível a pessoas fora do domínio.

Existem ferramentas de criação de *mockups* com o objetivo de criar representações visuais de sistemas. Alguns exemplos incluem o Balsamiq Mockups[1] e o Pencil Project[2]. Estas ferramentas focam-se principalmente na comunicação do aspeto do sistema e não no detalhe do comportamento ou interações que este pode suportar. De um modo geral, geram protótipos de baixa fidelidade.

Os protótipos permitem transmitir o design da interface e até avaliar a sua usabilidade. No entanto, não oferecem garantias sobre o seu funcionamento. Na conceção de sistemas críticos a análise do comportamento é uma ferramenta essencial para assegurar a inexistência de falhas. Nestes ambientes, anomalias no comportamento podem resultar em erros com consequências graves. Nestes casos, recorre-se à utilização de modelos mais formais.

1.2 IVY WORKBENCH

O IVY Workbench é uma ferramenta que adota técnicas formais para a modelação da lógica de um sistema (Campos and Harrison, 2009). O objetivo principal é a verificação do comportamento do sistema fases iniciais da conceção. Para facilitar a escrita, o IVY Workbench será abreviado para IVY durante o decorrer do documento.

O IVY adota como linguagem para descrever os sistemas, a MAL (Campos and Harrison, 2001). Esta linguagem permite descrever uma interface em termos de atributos e ações. Os atributos modelam o conteúdo da interface, as acções as interações que os utilizadores podem realizar na interface e eventos internos do sistema interativo. O comportamento do sistema é modelado indicando o efeito que as ações têm nos atributos do modelo, na prática, definindo uma máquina de estados. A análise do comportamento manifestado pelo modelo do sistema auxilia desenvolvedores e *designers* a detetar falhas mais rapidamente e a iterar sobre *designs* existentes.

Para comunicar o resultado da análise, o IVY utiliza representações gráficas do comportamento do modelo. É também possível construir essas representações interativamente, explorando uma animação do modelo. Na Figura 1 apresenta-se a representação tabular da animação de um modelo. Esta representação constitui uma das funcionalides que o IVY

1 https://balsamiq.com/wireframes/desktop/, visitado a 5 de julho de 2019
2 https://pencil.evolus.vn/, visitado a 5 de julho de 2019

*	1	2	3	4
memory	closed	closed	closed	opened
gstate	closed	opening	opened	closing
timeout	5	5	4	5
action	nil	remotebtn	opendoorsn	remotebtn

Figura 1: Representação tabular da execução de um modelo descrito no IVY

dispõe para análise de modelos. Observe-se que a tabela contém informações sobre o valor que os atributos tomam em cada um dos estados de execução e as ações executadas. Os dados apresentados constituem informação valiosa para indivíduos versados na análise de modelos formais, mas são de pouca utilidade para outras pessoas envolvidas na conceção do sistema. Para estes últimos, um protótipo seria uma forma melhor de apresentar o comportamento da interface.

Neste documento propõe-se uma solução ao problema de comunicação mencionado, acrescentando a capacidade de construção de protótipos ao IVY Workbench.

1.3 OBJETIVOS

Propõe-se o desenvolvimento de uma nova funcionalidade que acrescentará capacidades de construção de protótipos ao IVY. A criação dos protótipos deverá passar pela importação de *mockups* do sistema descrito, gerados em ferramentas de desenho de interfaces. Esses *mockups* serão depois mapeados ao modelo especificado no IVY. Com o mapeamento realizado, poder-se-á iniciar a animação do mesmo, onde a execução do modelo será visualizada pelos *mockups* importados. Na sua essência, pode-se descrever esta nova funcionalidade como uma visualização gráfica da funcionalidade de execução do modelo descrita na secção anterior.

Neste documento serão expostos todos os detalhes sobre a conceção e implementação desta nova funcionalidade no IVY.

1.4 ESTRUTURA

O documento encontra-se estruturado em seis capítulos. No capítulo 2 será descrito o IVY Workbench, o seu modo de funcionamento e as funcionalidades em oferta. Esta análise será acompanhada por um exemplo da linguagem de especificação dos modelos, para compreender as capacidades de representação do comportamento dos mesmos. Ainda no mesmo capítulo será realizada uma análise a ferramentas de análise de sistemas semelhantes ao IVY, com um foco no método utilizado para a representação visual dos sistemas. O objetivo passa por retirar ideias e conceitos sobre como esta representação será implementada na

nova funcionalidade. Após esta análise preliminar e uma compreensão do modo de funcionamento do IVY e das suas capacidades, serão elaborados um conjunto de requisitos funcionais para a nova funcionalidade. Estes requisitos irão direcionar o desenvolvimento da nova funcionalidade para melhor cumprir os objetivos impostos e mitigar o problema da comunicação dos modelos descritos no IVY. No capítulo 3 será especificado o *mockup* que contem a representação gráfica do modelo, como este deverá ser estruturado, o(s) programa(s) que o utilizador deverá utilizar para a sua criação e as bibliotecas que suportarão a manipulação do mesmo. No capítulo 4 procede-se à descrição da nova funcionalidade, da sua arquitetura e o seu modo de funcionamento. No capítulo 5 serão expostos um conjunto de exemplos para complementar a análise da funcionalidade implementada e demonstrar as capacidades da mesma. Por último, na conclusão será realizado um resumo de todo o conteúdo exposto no documento, uma análise dos resultados obtidos e discussão de trabalho futuro.

2

REQUISITOS PARA UM PLUGIN DE PROTOTIPAGEM

Nesta secção analisam-se três ferramentas relevantes para os objectivos da dissertação. Será analisada a ferramenta sobre a qual o desenvolvimento da nova funcionalidade irá incidir, o IVY Workbench. Pretende-se detalhar as funcionalidades que já existem, e as lacunas que este projeto se propõe a resolver.

Serão também analisadas o CogTool e o PVSio-Web, duas ferramentas de análise de interfaces com objetivos semelhantes ao IVY, mas com abordagens diferentes tanto quanto ao tipo de análise suportado, como quanto ao modo de representação da lógica da interface. Esta análise será efetuada no contexto de recolha de requisitos.

Finalmente, com os conceitos retirados da análise efectuada, serão identificados os requisitos para um plugin que permita gerar protótipos a partir dos modelos do IVY.

2.1 IVY WORKBENCH

Nesta secção será detalhado o IVY, a especificação dos modelos em MAL e as funcionalidades disponíveis para a animação de modelos, mais concretamente o *plugin* AniMAL.

2.1.1 *Especificação*

O IVY Workbench é uma ferramenta dedicada à modelação e análise de comportamento de interfaces (Campos and Harrison, 2009). Foi desenvolvida no DI da UM recorrendo ao *model checker* NuSMV como motor de verificação. A ferramenta está programada em Java. O objetivo da ferramenta passa pela análise do comportamento de um sistema a partir de um modelo deste. Para este propósito, recorre a MAL para descrever os modelos (Ryan et al., 1991). A linguagem MAL permite descrever:

- Ações - definem as interações possíveis do sistema;
- Atributos - definem as variáveis do sistema;
- Axiomas - definem o comportamento do sistema com base nas ações e atributos definidos.

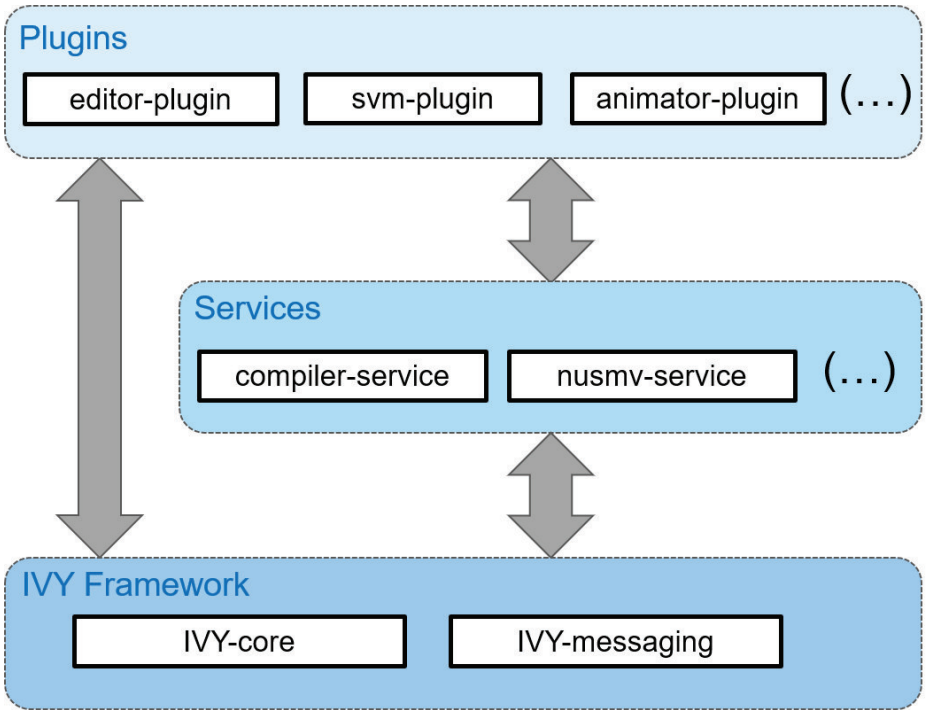


Figura 2: Arquitetura do IVY Workbench (Couto and Campos, 2019)

Os modelos podem ser validados face a propriedades da interface expressas em CTL (Clarke et al., 1983). O IVY utiliza um compilador que analisa os modelos especificados em MAL e os compila para a linguagem de entrada do *model checker* NuSMV (Cimatti et al., 2002), para que seja realizado o passo de verificação formal das propriedades definidas em CTL. Após a verificação, o IVY permite analisar os resultados da mesma (em caso de falha, na forma de comportamentos da interface que não obedecem à propriedade verificada), assim como modificar essas mesmas propriedades e acrescentar novas para investigar cenários adicionais de utilização.

Para auxiliar na verificação dos modelos existem o visualizador de traços (Trace Analyzer) e o animador de modelos (AniMAL). O Trace Analyzer permite a análise dos resultados da verificação das propriedades. Já o AniMAL permite interagir com o modelo e criar uma sequência de ações, partindo do estado inicial do mesmo. Esta funcionalidade é útil para melhor visualizar que ações são possíveis em que estados, prevenindo assim possíveis falhas no modelo.

A Figura 2 apresenta a atual arquitetura do IVY. Nesta, é possível observar a estrutura modular da ferramenta. Ao nível mais baixo, o núcleo (IVY Framework) fornece as funcionalidades essenciais, como a infraestrutura da interface com o utilizador e o sistema de comunicação entre *plugins* e serviços. No nível intermédio existem os serviços que fornecem funcionalidades essenciais à própria aplicação. Estes são usados pelos *plugins* e não são expostos ao utilizador. Por último, na camada superior existem os *plugins* que fornecem as

ferramentas com que o utilizador irá interagir, como o editor de modelos, ou visualizador de propriedades. A comunicação entre os *plugins* e os serviços pode ser efetuada de duas formas (Couto and Campos, 2019):

- Sistema de mensagens, onde se implementa o padrão de publicador-subscritor. Os *plugins* podem criar canais, subscrever a canais existentes, publicar mensagens e receber notificações dos canais subscritos.
- Sistema de memória partilhada, onde um *plugin* pode colocar informação na forma de um par chave-valor. Esta informação pode ser acedida por qualquer outro *plugin*.

A nova funcionalidade será desenvolvida como um novo *plugin* do IVY.

2.1.2 *Exemplo*

Com a intenção de demonstrar as funcionalidades que o IVY dispõe, desenvolveu-se o exemplo de um portão de garagem, descrito na Listagem 2.1.

O portão é controlado por um comando e dois sensores e pode encontrar-se num dos seguintes estados:

- Aberto
- Fechado
- A abrir
- A fechar

O comando é utilizado para abrir e fechar o portão e cada um dos sensores deteta se o portão está aberto ou fechado. Caso o portão esteja a efetuar uma ação, como a abrir ou fechar, também é possível pressionar o comando para reverter a ação.

A definição do modelo consiste em quatro partes: os tipos, os atributos, as ações e os axiomas. Analisando a listagem 2.1, o tipo encontra-se definido na segunda linha. Este é depois associado a atributos do modelo, e indica os valores que estes podem tomar. No modelo do portão aqui representado, na linha 6 verifica-se que o atributo `situacao` pode tomar valores do tipo `Movimento`.

Depois pode-se definir as ações que se podem realizar: a ação relacionada com o pressionar do comando, o `Ac` na linha 8, e as ações relacionadas com os sensores, `Ia` na linha 9 e `If` na linha 10. De notar a notação `[vis]` presente em algumas das definições para demarcar ações efetuadas pelos utilizadores e atributos presentes na interface.

```
1  types
2   Movimento = {a_abrir, a_fechar, aberto, fechado}
3
4  interactor main
5   attributes
6    [vis] situacao: Movimento
7   actions
8    [vis] Ac
9    Ia
10   If
11  axioms
12   [] situacao = fechado
13   situacao = fechado -> [Ac] situacao' = a_abrir
14   situacao = a_abrir -> [Ac] (situacao' = a_fechar)
15   situacao = a_fechar -> [Ac] (situacao' = a_abrir)
16   situacao = a_fechar -> [If] (situacao' = fechado)
17   situacao = a_abrir -> [Ia] (situacao' = aberto)
18   per(Ia) -> situacao = a_abrir
19   per(If) -> situacao = a_fechar
20   situacao = aberto -> [Ac] (situacao' = a_fechar)
```

Listagem 2.1: Modelo de um portão de garagem em MAL

Por último, da linha 12 até ao final encontram-se os axiomas, que definem o comportamento do sistema. A fim de melhor perceber a estrutura dos axiomas, apresenta-se um exemplo retirado da linha 15 da listagem 2.1

A declaração à esquerda da implicação (->) indica uma guarda. Se esta for verdadeira, a ação presente à direita da implicação poderá ser executada. A ação encontra-se rodeada por parêntesis retos. A expressão a seguir à ação, rodeada por parêntesis, indica qual será o estado após a execução da ação. A pelica é utilizada para referir o valor do atributo após a execução da ação. Traduzindo a expressão para linguagem corrente, se o portão estiver a fechar, ao pressionar o comando, o portão irá reverter a ação que estava a ocorrer e passará para o estado `a_abrir`.

Observa-se também uma notação adicional na Listagem 2.1. Nas linhas 18 e 19 é utilizado o operador ‘‘`per`‘‘. Este operador é utilizado na definição de pré-condições para a execução de ações. Por exemplo, na linha 18, a ação `Ia` (correspondente ao despoletar do sensor do portão aberto) só pode ser executada se o portão estiver no estado `a_abrir`.

Após a definição do modelo, é possível compilá-lo para a linguagem de entrada do *model checker* NuSMV, como especificado no início da secção. De seguida, é possível efetuar a verificação de propriedade e a análise dos resultados, recorrendo ou ao *Trace Analyzer* ou ao AniMAL. Na secção seguinte será detalhado o AniMAL, devido aos objetivos entre este

e o *plugin* planeado: ambos pretendem facilitar a análise do comportamento do modelo através da sua demonstração no momento de execução.

2.1.3 *AniMAL*

O AniMAL constitui um dos vários *plugins* do IVY que acrescentam funcionalidades de análise e verificação aos modelos introduzidos. O nome AniMAL deriva da conjugação dos termos Animador e MAL, e indiciam o propósito que este *plugin* serve. Este utiliza um formato tabular para demonstrar, passo a passo, o valor que os atributos tomam na execução do modelo. O utilizador define quais as ações a executar que irão fazer progredir o estado de execução.

Na Figura 3 encontra-se um exemplo de uma sequência de execução possível do modelo exposto na Listagem 2.1. A sequência pode ser visualizada na tabela apresentada no lado direito da interface. A sequência de execução pode ser visualizada na tabela apresentada no lado direito da interface. Nesta tabela, cada coluna representa um estado na execução do modelo, enquanto nas linhas são representados os valores que os atributos tomam em cada estado. À esquerda encontram-se os controlos. Na secção entitulada "ACTIONS" selecionam-se as ações que o utilizador quiser executar. Ligeiramente à direita, na secção "STATE INFO", apresentam-se as alterações que irão ocorrer caso a ação selecionada ocorra. No caso da Figura apresentada, a ação `Ac` encontra-se selecionada e o "STATE INFO" demonstra o valor que `estadoActual` irá tomar caso a seleção se execute. É ainda possível ver na figura uma secção "CONSTRAINTS", que permite controlar quais os estados considerados na execução/simulação do modelo, mas que não é relevante para a presente discussão.

As representações gráficas utilizadas no AniMAL focam-se na representação dos modelos, das suas ações e atributos. Embora sejam úteis para análise, não suportam a comunicação do design da interface a não-peritos. Tal seria melhor conseguido com uma representação focada no design da interface (ou seja, com protótipos da interface). Pretende-se, agora, uma abordagem mais flexível e de mais fácil integração com o processo de conceção e desenvolvimento de sistemas interativos. Esta necessidade constitui o catalisador que impulsiona a criação de um novo *plugin*.

Para auxiliar na fase inicial de conceção do *plugin*, nas secções seguintes serão analisadas ferramentas de modelação de sistemas com abordagens distintas para a representação dos mesmos. O estudo destas ferramentas irá auxiliar na recolha de requisitos.

2.2 COGTOOL

O CogTool (John, 2012) é uma ferramenta de modelação e análise de interfaces focada na análise da eficiência da interação. Para tal, incluí uma arquitectura cognitiva que pro-

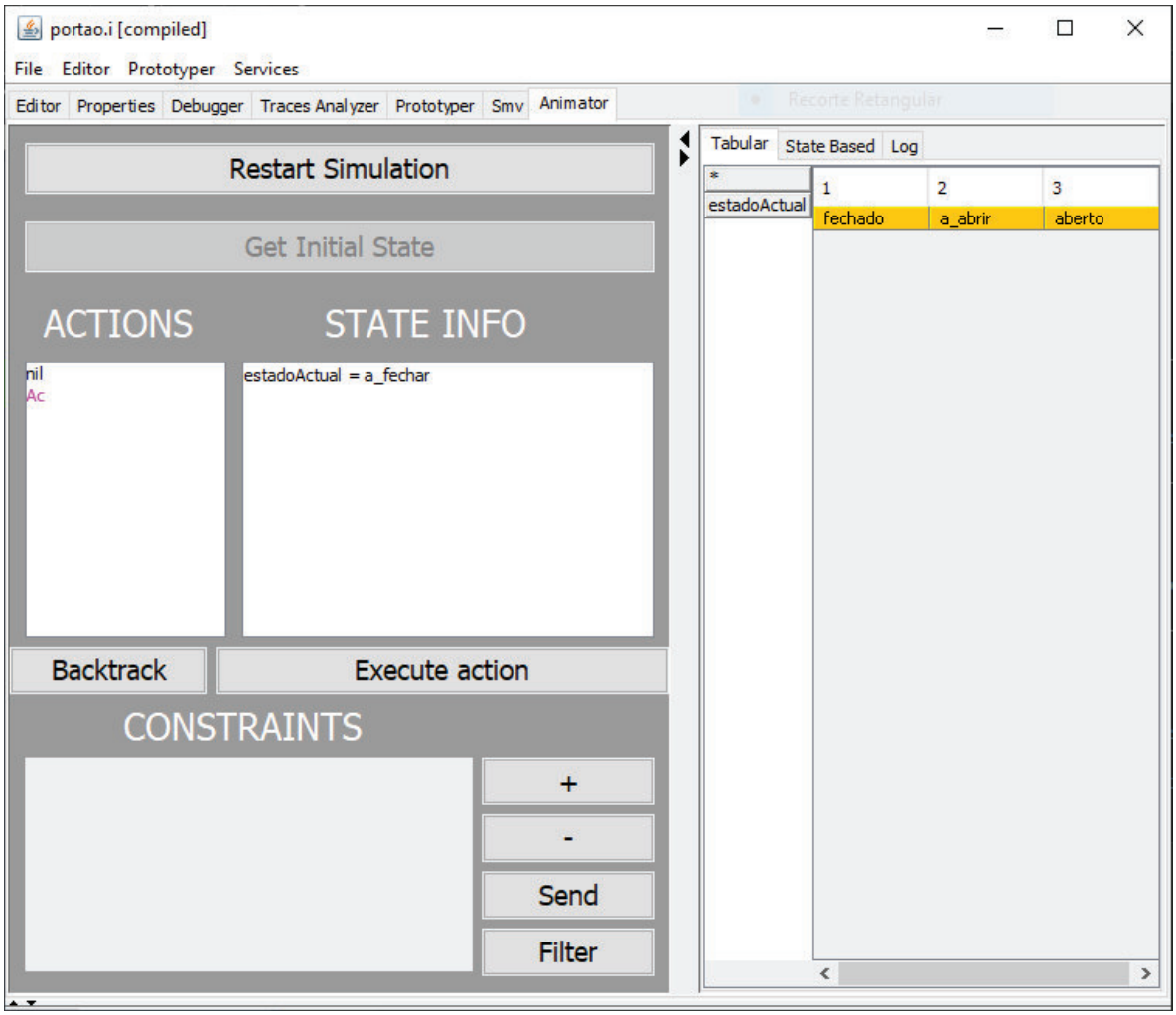


Figura 3: Exemplo de uma sequência de ações no *plugin* AniMAL

cura capturar o modo como os utilizadores interagem. A ferramenta foi desenvolvida na Carnegie Mellon University pela equipa liderada por Bonnie E. John. Neste momento, a ferramenta não se encontra em desenvolvimento ativo, mas todos os materiais relacionados, incluindo o código fonte, estão disponíveis no GitHub[1] e num *blog*[2] criado pela investigadora mencionada. A última atualização ocorreu em 2014. Apesar disso, considera-se que os princípios e conceitos que esta aplica ainda são relevantes no contexto de visualização do comportamento de sistemas modelados.

O principal objetivo da ferramenta é auxiliar no desenho de interfaces, para que efetuar tarefas comuns no programa seja intuitivo e rápido. O CogTool segue o seguinte princípio: "Tornar tarefas frequentes fáceis e tarefas menos frequentes possíveis" (John, 2012, Capítulo 1.1). Para atingir este objetivo, a ferramenta aplica modelos preditivos de com-

1 https://github.com/cogtool, visitado a 4 de julho de 2019
2 https://cogtool.wordpress.com/, visitado a 4 de julho de 2019

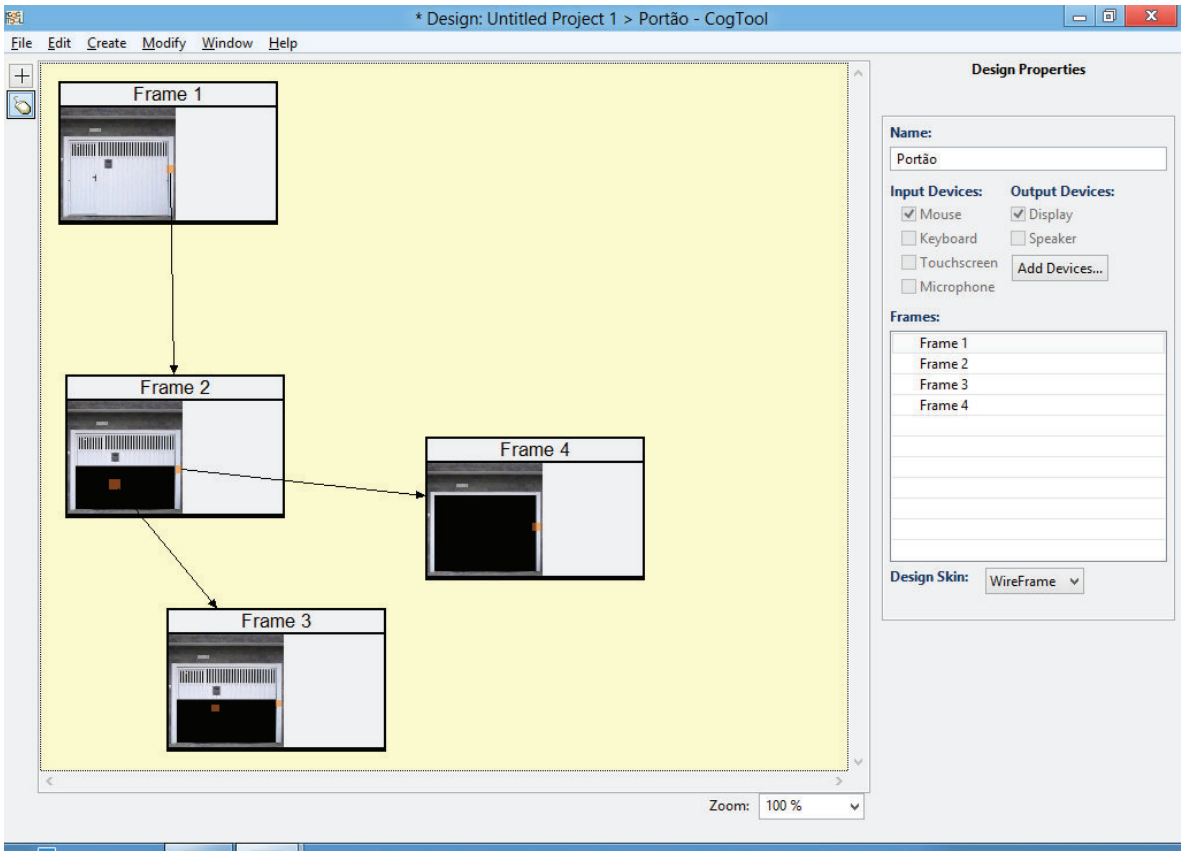


Figura 4: Interface do CogTool

portamento humano. É possível elaborar a interface e definir as sequências de ações do utilizador paras tarefas pretendidas. O modelo cognitivo incorporado calcula, então, uma estimativa do tempo que o utilizador irá demorar a efetuar as tarefas. O modelo de previsões implementado é baseado na técnica KLM (Card et al., 1980). De forma sucinta, esta técnica tem em consideração diversos fatores relacionados com a realização de uma tarefa na interface, tais como o número de teclas a pressionar no teclado, a distância que o cursor irá percorrer e o nível de perícia do utilizador, só para listar alguns. Estes fatores são depois utilizados no cálculo do tempo que o utilizador irá demorar a completar a tarefa.

A interface utiliza o conceito de *storyboards* ou *frames*, onde cada *frame* representa a interface num dado momento. Em cada um destes *frames* é possível definir uma imagem, representativa da interface a modelar, e sobrepor controlos sobre ela. A partir do momento em que os *mockups* e as transições entre eles estão definidos, é possível utilizá-los para demonstrar a execução de tarefas concretas (num estilo *capture-replay*). recorrendo ao modelo cognitivo que incorpora, a ferramenta é então capaz de calcular uma estimativa do tempo que demoraria cada uma das tarefas demonstradas. Calcula ainda uma série de indicadores

relacionados com a interacção, em particular relacionados com para onde o utilizador irá olhar ou o movimento das mãos.

Na Figura 4 observa-se uma tentativa de replicar o comportamento do exemplo do portão de garagem, descrito na secção 2.1.2. Cada *storyboard* representa um momento de execução do portão e os controlos para interação foram definidos para equivaler ao sensor e ao comando. No entanto, relembrando o objetivo do CogTool e o seu modo de funcionamento, nota-se que este modelo não se adequa à ferramenta. O principal objetivo do CogTool é auxiliar no desenho de interfaces de software através da análise de dados de interação. O portão de garagem não constitui um sistema de software nem se pode extrair dados de utilização deste. Note-se, no contexto desta dissertação, que o foco da análise do CogTool passa pela maneira como este efetua a representação visual do comportamento do sistema, e não pelo tipo de análises que se podem realizar dos modelos criados. Neste contexto, o uso do exemplo do portão adequa-se para retirar conclusões.

Um aspeto interessante desta ferramenta é que o modelo utilizado para análise é derivado do protótipo e não o contrário. Embora essa não seja o tipo de abordagem que interessa prosseguir neste momento, o conceito de *storyboards* e controlos de interação indiciam uma direção que a representação gráfica dos modelos do IVY pode tomar. Cada valor que um atributo toma pode ser associado a vários *mockups*, representativos desses mesmos valores.

2.3 PVSio-Web

O PVSio-Web (Oladimeji et al., 2014) é uma ferramenta de modelação e análise formal de interfaces, com foco na construção de protótipos. Explicar o modo de funcionamento da ferramenta e dos princípios em que assenta pode ser feito com base no nome da mesma. A sigla PVS (Owre et al., 2001) significa Prototype Verification System, um demonstrador de teoremas que assenta numa lógica de ordem superior. O PVSio (Muñoz, 2003) é uma extensão do PVS que facilita a utilização do avaliador de expressões (*ground evaluator*) do demonstrador de teoremas, adicionando um conjunto de facilidades de programação imperativa. Em particular, uma biblioteca de *input/output*. O PVSio-Web, utilizando o PVSio para comunicar com o PVS, suporta a construção e animação de protótipos, recorrendo ao avaliador de expressões para dinamicamente calcular o comportamento definido pelos modelos.

Analisando a arquitetura da ferramenta na Figura 5, verifica-se que o PVSio-Web utiliza um cliente baseado em tecnologias Web, mais especificamente em NodeJS[1], para abstrair a lógica do PVSio (Oladimeji et al., 2014, p. 3). O construtor de protótipos da interface gráfica interage com o servidor que contém os processos do PVSio e PVS através de *WebSockets*. Este modelo de funcionamento permite definir protótipos, importando imagens e

1 https://nodejs.org/en/, visitado a 19 de outubro de 2019

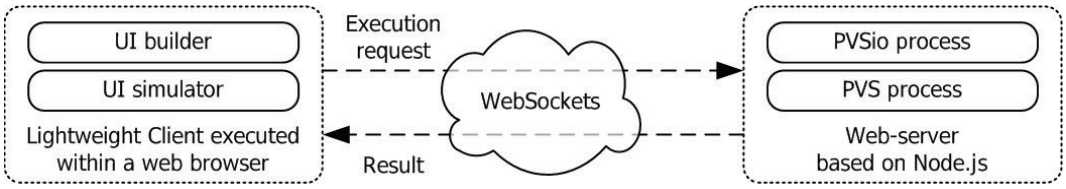


Figura 5: Arquitetura do PVSio-Web

Figura 6: Exemplo de um modelo no PVSio-Web

definindo as áreas de interatividade nessas imagens diretamente a partir da interface gráfica do PVSio-Web. Estas áreas ficam definidas por cima do protótipo como *overlays* e a cada uma são associados atributos ou ações do modelo. Assim, torna-se possível interagir com o protótipo e observar o seu comportamento. Esta interação serve, assim, para observar o comportamento do sistema, tal como modelado, e não para ilustrar sequências de interacção para posterior análise (como no CogTool).

Na figura 6 visualiza-se a representação de um dispositivo médico de infusão onde se podem distinguir cinco áreas retangulares semi-transparentes sobrepostas no ecrã e nos controlos. Estas são as áreas de interatividade onde o comportamento do modelo se irá manifestar. Quando o modelo estiver a executar, o *overlay* do ecrã irá apresentar o valor a infundir e os controlos irão permitir ajustar essa quantidade.

2.4 DISCUSSÃO DO COGTOOL E PVSIO-WEB

O modo de construção dos protótipos do PVSio-Web assemelha-se ao oferecido pelo CogTool, mas com o objetivo de modelar do comportamento de sistemas em vez da extração e análise de dados de usabilidade. Tal aproxima o PVSio-Web ao IVY em termos de objetivos. No que diz respeito à representação gráfica dos sistemas, o PVSio-Web apresenta um método que difere do CogTool. Enquanto que o conceito de *storyboards* do CogTool nos direciona para uma representação do sistema em que todos os estados são representados separadamente, no PVSio-Web essas mesmas mudanças de estado podem ocorrer numa mesma representação do sistema, com essas mudanças representadas nas áreas interativas.

Esta distinção origina duas vias possíveis para a representação gráfica dos modelos do IVY. Por um lado, há a abordagem de importar múltiplas variações do *mockup* representativas dos valores que um ou mais atributos podem tomar; por outro, pode ter-se um só *mockup* representativo do sistema, com as mudanças de estado representadas em secções ou áreas do mesmo. Esta análise auxilia na compreensão das possibilidades existentes para a representação gráfica dos modelos que se poderá implementar no *plugin*. O esclarecimento destas ideias possibilita a definição de um conjunto de requisitos.

2.5 REQUISITOS

Tal como já referido, pretende-se desenvolver um *plugin* que adicione ao IVY capacidades de prototipagem. Esta nova funcionalidade do IVY deverá permitir importar o protótipo de um sistema interativo, sob a forma de *mockups*, e estabelecer uma mapeamento entre estes e o modelo formal do sistema, descrito no IVY. Tendo em consideração estes objetivos e a análise realizada acima, foi formulado o seguinte conjunto de requisitos funcionais para o *plugin* a desenvolver:

1. O *plugin* deverá suportar a importação de um ou mais ficheiros que descrevam o protótipo da interface desejada;

A conceção do *mockup* deverá originar de ferramentas de esboço de interfaces dedicadas para tal tarefa. Optou-se por relegar esta atividade para fora do contexto do *plugin* por dois motivos: as ferramentas de esboço de interfaces permitem bastante flexibilidade no design dos *mockups* e a incorporação destes aspetos no *plugin* a desenvolver acarretaria não só um aumento da complexidade como a perda de foco do objetivo principal. O objetivo principal é a representação gráfica dos modelos descritos no IVY, não como essa mesma representação será construída. Adicionalmente, os utilizadores poderão manter o seu fluxo de trabalho habitual no desenho do *mockup*.

A importação dos *mockups* requer que estes tenham sido exportados para um ficheiro num formato apropriado. A maneira como esta informação será representada, assim como detalhes das ferramentas de criação dos *mockups* será reservada para o próximo capítulo.

A capacidade de importar de mais do que um ficheiro é influenciada pela análise efetuada ao CogTool. A importação de múltiplos *mockups* (descritos nos vários ficheiros) possibilita a representação dos diferentes estados por que o protótipo possa passar. No cenário mais simples, a animação do protótipo resumir-se-á à transição entre os vários ecrãs possíveis.

2. O ficheiro a importar deve estar num formato aberto de livre utilização;

De modo a não restringir o utilizador na forma como concebe o *mockup* do sistema, o ficheiro a importar deve ser gerado num formato aberto, ou seja, que não esteja restringido à plataforma de onde foi gerado. Tal irá permitir liberdade na escolha de ferramentas de esboço de *mockups* e na conceção dos mesmos.

3. Deverá ser possível a identificação/especificação de áreas de interatividade dentro do *mockup* importado;

A interação com o protótipo que será configurado no *plugin* terá de ser definida para permitir a execução da animação do modelo por parte do utilizador. Assim, define-se que esta interação se irá manifestar na forma de áreas interativas. Equipara-se este conceito ao apresentado anteriormente no PVSio-Web.

O nível de integração destas áreas com o ficheiro do *mockup* importado será analisado no decorrer no próximo capítulo.

4. Deverá ser possível mapear essas mesmas áreas para atributos e ações do modelo;

Relembrando as representações dos sistemas apresentadas pelo CogTool e o PVSio-Web, opta-se por enveredar por uma solução mista. As áreas de interatividade associadas a atributos demonstrarão o valores dos mesmos no momento de execução e as áreas associadas a ações irão permitir, através de uma interação definida, a sua execução. Será assim possível representar os valores dos atributos de duas maneiras: através da transição entre *mockups* importados e nas áreas de interatividade definidas.

5. O *plugin* deverá suportar vários tipos de interações com os elementos interativos do protótipo;

Para interagir com as áreas associadas a ações do modelo será necessário disponibilizar conjunto de *triggers*, que serão acionados pelo utilizador, para a execução da ação. Estes *triggers* podem ser cliques do rato ou o pressionar do teclado.

6. Deverá ser possível animar o protótipo para visualizar o comportamento do sistema modelado;

Após a definição do mapeamento entre o modelo e o(s) *mockup(s)* importados, o utilizador deverá poder iniciar a animação do modelo, à semelhança do modo de funcionamento do AniMAL e do PVSio-Web.

7. Deverá ser possível guardar o estado do animador e posteriormente restaurá-lo e resumir o trabalho.

Este requisito foca-se na facilidade de uso do *plugin*. Após efetuar o mapeamento entre os componentes do modelo e os elementos do *mockup*, a opção de gravar a configuração efetuada deverá poupar trabalho ao utilizador quando este decidir resumir o trabalho.

3

FORMATOS, PROGRAMAS DE ESBOÇO E MÉTODO DE IMPORTAÇÃO DOS MOCKUPS

Neste capítulo serão descritas as possíveis ferramentas a utilizar na criação do *mockup* para importação pelo *plugin* a desenvolver, quais os formatos a utilizar para essa importação, e as bibliotecas que suportarão a manipulação do mesmo pelo *plugin*.

3.1 FERRAMENTAS DE CRIAÇÃO DE MOCKUPS

Para proceder à importação de um ficheiro estruturado será preciso identificar um conjunto de ferramentas que permitam a criação de *mockups* do sistema pretendido, assim como a exportação destes mesmos para formatos abertos e de livre utilização.

Existe uma variedade de programas de prototipagem que cobrem áreas de aplicação distintas e diferentes fases do processo de desenvolvimento. O principal objetivo desta análise é avaliar a flexibilidade e fidelidade na criação dos *mockups* e as características dos formatos de exportação disponíveis. São de interesse as ferramentas focadas na modelação de sistemas em fases iniciais da conceção de um projeto.

Apresentam-se os pontos que servirão como base de comparação:

- Flexibilidade na conceção dos protótipos - a ferramenta permite criar protótipos de vários tipos de sistemas.
- Opções de exportação - o protótipo criado pode ser exportado para formatos abertos de livre utilização (tal como especificado no requisito 2).

Foram escolhidas as seguintes ferramentas para análise: Balsamiq Mockups, Pencil Project e Mockplus. Estas ferramentas foram consideradas por se apresentarem como ferramentas populares de construção de interfaces.

Figura 7: Representação do exemplo base de um relógio digital (Kmseteam, 2013)

3.1.1 *Exemplo Base*

Para auxiliar na comparação das ferramentas de prototipagem, a definição de um exemplo base permite estabelecer termos de comparação comuns.

Para servir de exemplo, decidiu-se representar um relógio digital. O exemplo permitirá testar a flexibilidade das ferramentas no que diz respeito às formas de o representar. A Figura 7 apresenta um exemplo do tipo de relógio referido.

O relógio suporta dois modos de funcionamento: modo de relógio para mostrar as horas e o modo de cronómetro para medir o tempo com um temporizador. É possível alternar entre estas funções utilizando botões físicos no dispositivo, como os botões laterais representados no dispositivo da Figura 7. No ecrã do relógio é possível visualizar informação sobre o modo selecionado, as horas ou o tempo decorrido, dependendo do modo selecionado, e uns indicadores do nível de bateria e nível de conectividade *wireless*.

3.1.2 *Balsamiq Mockups*

Ferramenta baseada na *framework* Adobe AIR, neste momento só dispõe de opções pagas para o seu uso, com uma versão de testes de 30 dias. Graças ao uso da framework mencionada atrás, é compatível com os sistemas operativos comuns (Mac e Windows).

Na Figura 8 apresenta-se o exemplo base, replicado utilizando as funções disponibilizadas pelo Balsamiq Mockups. Todos os elementos da interface do relógio foram construídos utilizando os *widgets* disponibilizados para construção da interfaces. Estes *widgets* equiva-

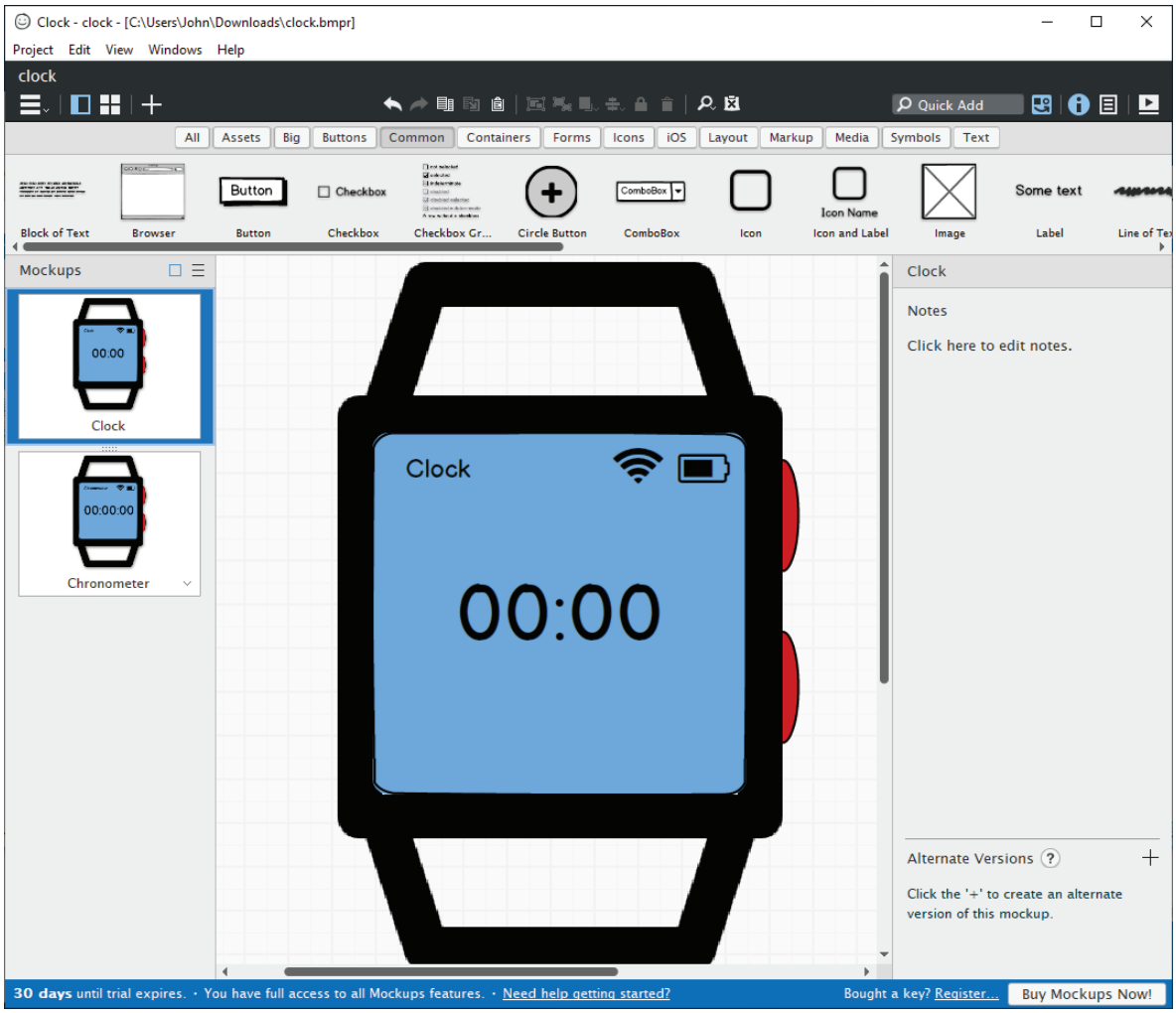


Figura 8: Exemplo base replicado no Balsamiq Mockups

lem a elementos de construção da interface, como botões, caixas de texto, ou figuras. Estes são selecionados através do painel superior representado na Figura 8. A moldura do dispositivo advém de uma imagem importada, pois a ferramenta não disponibilizou um *widget* para esta. Para representar as duas funções existentes no relógio, também se tirou proveito da funcionalidade de camadas, visível na Figura 8 no painel lateral esquerdo onde se encontram as duas camadas. A porção visível do relógio no *canvas* de visualização encontra-se no modo de relógio, enquanto que na outra camada o relógio encontra-se no modo de cronómetro. Os botões para mudar de função encontram-se na lateral do dispositivo e são de cor vermelha.

Em relação à variedade dos *widgets* em oferta, estes focam-se muito na construção de GUI (Graphical User Interfaces) de sistemas de software. O próprio website[1] salienta que

1 https://balsamiq.com/, visitado a 8 de outubro de 2019

um dos objetivos é a criação de *mockups* de baixa fidelidade, com o propósito de comunicar o design dos mesmos de forma simples e rápida. Este aspeto do programa dificulta a construção de *mockups* com um nível mais elevado de fidelidade com os *widgets* nativos.

O Balsamiq Mockups também oferece opções de exportação. Os formatos suportados são: PNG, JSON e ZIP que contém ficheiros que detalham a configuração do protótipo.

O Balsamiq Mockups, no seu todo, apresenta-se como uma opção interessante para a construção dos *mockups*. Apesar dos *widgets* em oferta serem principalmente direcionados para a construção de interfaces de software e de aspeto minimalista, a importação de imagens externas pode mitigar eventuais lacunas para a construção do *mockup*. Os formatos de exportação em oferta constituem um primeiro conjunto a considerar para importação no novo *plugin*.

3.1.3 *Pencil Project*

Ferramenta construída com base em tecnologias Web, recorrendo à framework Electron. O programa é open source e compatível com Windows, Linux e Mac OS. O código fonte encontra-se disponível no GitHub.

Na Figura 9 apresenta-se o exemplo base, definido anteriormente, replicado no Pencil. Como se pode observar no painel lateral esquerdo, a ferramenta oferece um conjunto de *widgets* para a construção do protótipo. Estes variam, desde formas básicas como texto, formas retangulares e circulares a elementos visuais associados a sistemas específicos como Android, iOS e Windows. Também há a opção de importar *widgets* adicionais na forma de coleções que podem ser instaladas através da internet (do repositório disponibilizado pela ferramenta), localmente, ou criadas pelo utilizador, utilizando a API disponibilizada (Evolus, 2012), algo não possível no Balsamiq Mockups.

Na Figura 9 apresenta-se o exemplo base do relógio, replicado no Pencil, no modo de cronómetro. O processo de construção do dispositivo foi semelhante ao exposto para o Balsamiq Mockups. Foram utilizados diversos *widgets* para compor os elementos do relógio. Destaca-se o uso de uma coleção externa, transferida da internet, para descrever o aspeto do dispositivo em si. Esta coleção continha ícones do *design* "Material" da Google (Google, 2019). Neste caso não houve necessidade, mas também há a opção de importação de imagens externas. Os dois modos do relógio também foram replicados utilizando camadas. As camadas são geridas na secção inferior da interface. O modo de relógio apresenta uma interface semelhante ao modo de cronómetro, sem o campo dos segundos na visualização e com a indicação "Clock".

O Pencil também oferece um conjunto de formatos de exportação. Tem os seguintes formatos disponíveis: PNG, HTML, PDF, SVG e ODT.

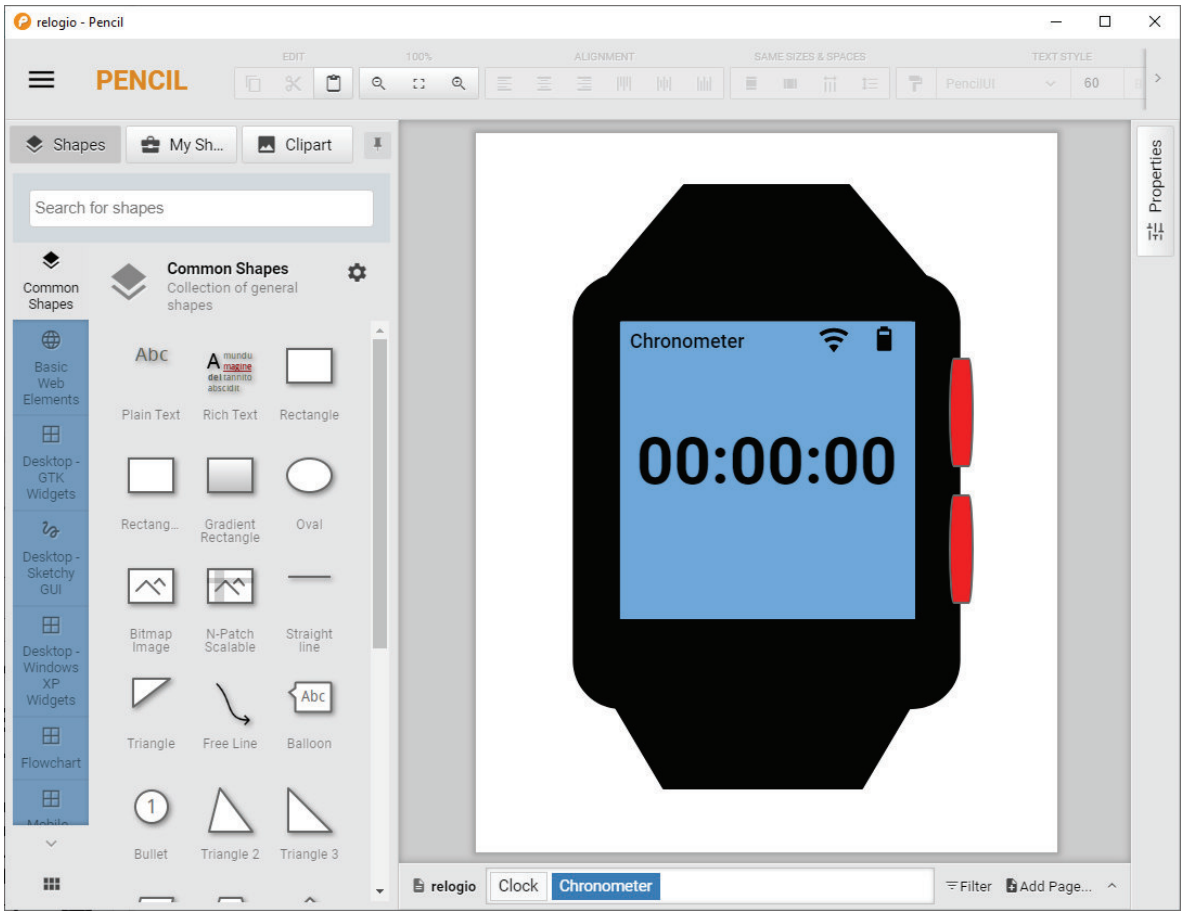


Figura 9: Exemplo base replicado no Pencil

No seu todo, o Pencil apresenta-se como uma proposta interessante para a criação dos protótipos que serão importados para o IVY. A funcionalidade de importação de coleções de *widgets* aumenta a flexibilidade disponível para a conceção de *mockups* para além do disponibilizado pelo Balsamiq Mockups. Adicionalmente, a maior oferta de formatos de exportação alarga o leque de opções para a importação no novo *plugin*.

3.1.4 *Mockplus*

Ferramenta orientada para o desenho de interfaces disponível para Windows, Mac e Android, neste momento não divulga em que tecnologias é que se baseia. Só dispõe de opções pagas, com uma oferta de 7 dias de *trial* para experimentar todas as funcionalidades. A sua utilização exige também a criação de uma conta.

A Figura 10 demonstra o protótipo replicado no Mockplus. As funcionalidades em oferta são semelhantes às ferramentas de prototipagem analisadas. O *mockup* é construído à base

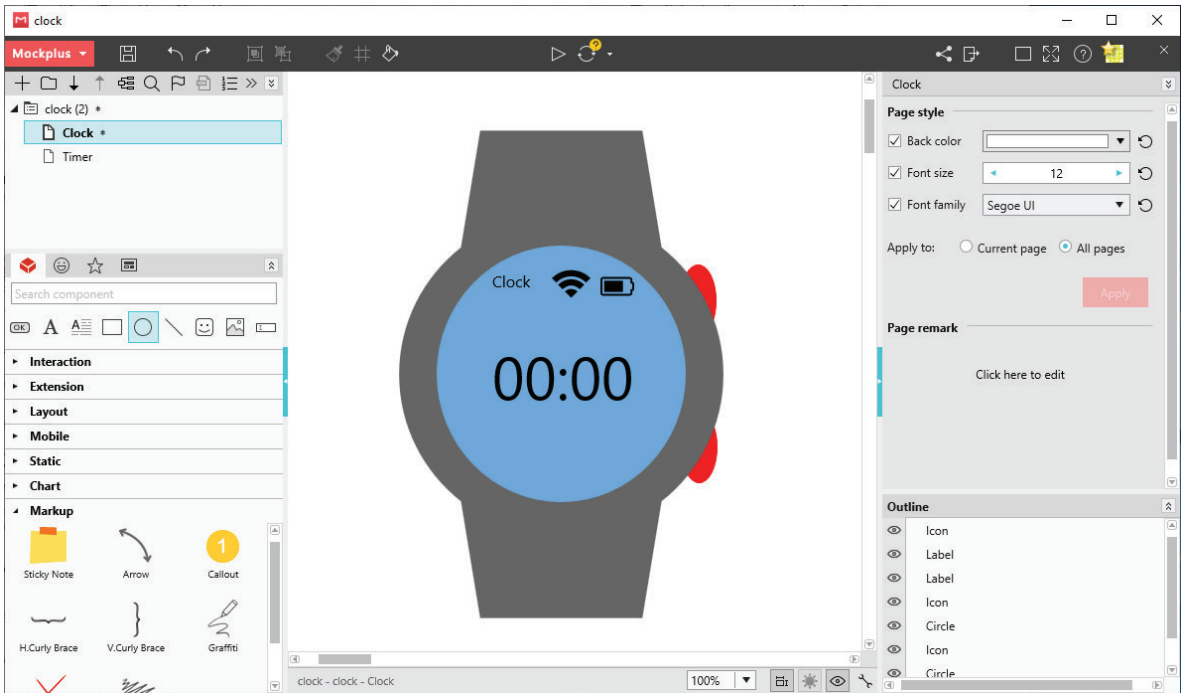


Figura 10: Exemplo base replicado no Mockplus

de *widgets*, com suporte para camadas para representar os diferentes estados do sistema. O dispositivo suporta uma forma circular na representação criada, devido ao uso do *widget* representativo de um relógio, considerado o mais próximo da visualização esperada. Os *widgets* oferecidos estão orientados para a construção de GUI de *software*. Existe, no entanto, mais variedade na oferta comparativamente ao Balsamiq Mockups graças à existência de uma coleção de *widgets* que representam items comuns (dispositivos, logótipos, ações, ...). No entanto, ao contrário das ferramentas analisadas até ao momento, não é possível importar imagens externas ou a importação de coleções de *widgets* externos como no Pencil.

O Mockplus também tem uma seleção de formatos de exportação. Pode-se exportar para: PNG, JPEG, HTML e um executável, usado para visualizar o protótipo sem depender do Mockplus. Adicionalmente, pode-se exportar para uma vista de árvore dos componentes do protótipo. Esta vista de árvore meramente lista os componentes de forma textual, não contendo nenhuma informação sobre a sua constituição visual.

O Mockplus oferece uma variedade de *widgets* que supera a oferta do Balsamiq Mockups. No entanto, a ausência da opção para importar imagens externas ou até definir um conjunto personalizado de *widgets* como o Pencil Project diminui o seu apelo, com base nos critérios definidos. Acrescentando a tudo isto o suporte à exportação de quatro formatos, sendo um deles um formato fechado do qual não se pode extrair a constituição do *mockup*, o executável.

	Pencil Project	**Balsamiq Mockups**	**Mockplus**
Construção por *widgets*	✓		
Utilização de Elementos Adicionais	Coleções de *Widgets* e Imagens	Imagens	-
Edição de Conteúdo Textual	✓		
Opções de Exportação	PNG, PDF, SVG, ODT e HTML	PNG, JSON e ZIP	PNG, JPG, HTML e árvore de componentes

Tabela 1: Características das ferramentas de prototipagem

3.1.5 *Análise*

A Tabela 1 resume a análise efetuada às ferramentas de construção de *mockups*. De todas, o Pencil é o que melhor cumpre os pontos de comparação definidos, permitindo a importação de imagens e coleções de *widgets* externas para complementar a oferta existente. Também oferece o maior número de formatos para exportação.

A flexibilidade na construção do *mockup* pode ser verificada e avaliada com o uso e comparação das ferramentas. No entanto, a especificação do formato de exportação a adotar exige uma análise mais cuidada. A menção do número de formatos que cada ferramenta de esboço de *mockups* consegue exportar constitui uma métrica preliminar, necessariamente superficial, de análise, meramente indicativa dos formatos a partir dos quais se pode iniciar uma análise mais profunda. A definição do formato é crucial para o desenvolvimento do *plugin*, e irá ditar o rumo que este irá tomar durante o desenvolvimento. A escolha do formato a suportar deverá assentar no tipo de informação que este suporta, ou seja, como a informação sobre os *mockups* é representada.

3.2 FORMATO DO MOCKUP

Como definido na secção 2.5, os requisitos para o novo *plugin* especificam que o ficheiro que descreve o *mockup* deve estar num formato aberto (i.e. independente da plataforma onde foi criado e de livre utilização) e de fácil criação. Tal especificação levou à identificação de dois tipos de solução, e dois caminhos para o desenvolvimento desta nova funcionalidade:

- Utilização de imagens *raster* como base para a definição dos protótipos.
- Utilização de formatos estruturados como base para a definição dos protótipos.

Ou seja, a diferença está entre ter o *mockup* representado numa imagem como PNG ou JPEG, ou em algum formato exportado por ferramentas de elaboração de *mockups*, nos quais é possível ter acesso à sua estrutura.

3.2.1 *Utilização de imagens* raster

A importação de imagens *raster* para o IVY deverá passar por escolher uma imagem representativa da interface do sistema modelado. Depois, será necessário definir os controlos sobre a imagem, representativos do comportamento definido no modelo, à semelhança do PVSio-Web ou CogTool, discutidos nas secções 2.2 e 2.3.

A principal vantagem deste método é que a importação de imagens é uma tarefa que se realiza facilmente. Não há necessidade de analisar ou alterar o ficheiro que será importado.

3.2.2 *Utilização de formatos estruturados*

A importação de formatos estruturados possibilita a criação do protótipo com acesso aos componentes que definem o *mockup*. Este detalhe permitirá atribuir o comportamento do modelo a elemento/controlos do *mockup* importado, como botões ou elementos textuais.

A um nível mais básico, esta abordagem permite replicar a representação de modelos com a importação de imagens *raster*. Neste cenário, certos elementos da estrutura interna do *mockup* seriam mapeados como áreas interativas. No entanto, o facto de se utilizar um formato estruturado possibilita casos de uso adicionais. O nível de integração mais profunda com o ficheiro que descreve o *mockup* importado poderá permitir uma definição mais granular do comportamento e automatização no mapeamento entre os elementos na interface e elementos no modelo.

Como a utilização de formatos estruturados para a representação gráfica do modelo consegue abranger um maior número de casos de uso que a utilização de imagens *raster*, o desenvolvimento do *plugin* irá enveredar nesta direção. Para aprofundar a análise, serão detalhados os formatos exportados pelas ferramentas de esboço de *mockups* estudadas, de modo a reduzir os formatos em consideração para um único.

3.2.3 *Análise dos formatos de exportação*

Ao longo da descrição das ferramentas de esboço de *mockups* foram mencionados diversos formatos que a exportação do *mockup* poderá adotar. Procede-se com uma analise destes formatos e a informação contida neles. Desta análise, serão derivados um conjunto de critérios que serão utilizados para definir quais destes melhor se adequa à representação do *mockup*.

Agregando toda a informação, os programas de prototipagem mencionados exportam os seguintes tipos de formato:

- SVG;
- JSON;
- HTML;
- PNG;
- JPEG;
- PDF;
- ODT.

Formatos como o PNG e JPEG encaixam na categoria de imagens *raster*, e portanto podem desde logo ser excluídos do processo de seleção. Com os formatos PDF e ODT existe a possibilidade de extração de informação. Das ferramentas analisadas, apenas o Pencil consegue exportar para estes formatos. Acresce que, mesmo assim, a exportação é feita na forma de imagens *raster* embebidas nos ficheiros PDF ou ODT. O mesmo acontece com o HTML exportado pelo Pencil, que descreve o *mockup* recorrendo a imagens deste e das suas camadas.

Depois, destacam-se os formatos ZIP e HTML. O primeiro é gerado pelo Balsamiq Mockups, enquanto o segundo é gerado tanto pelo Balsamiq como pelo Mockplus.

O ZIP gerado pelo Balsamiq contém no interior ficheiros no formato proprietário, "bmml", contendo texto XML e quaisquer imagens externas que o utilizador tenha importado para a conceção do *mockup*. O XML descreve a estrutura do *mockup*, com a peculiaridade que a representação gráfica dos *widgets* é omitida. Na prática, estes ficheiros "bmml" foram desenhados para serem importados apenas para o Balsamiq Mockups. Ao importar para a ferramenta, esta analisa o XML contido e verifica quais dos *widgets* internos a utilizar, na configuração especificada. Caso este formato viesse a ser utilizado pelo *plugin* planeado, seria necessário efetuar o mesmo processo para processar os *mockups* contidos nestes ficheiros. Tal exigiria implementar a biblioteca de *widgets* do Balsamiq Mockups (ou de aspeto e função semelhantes). Esta implementação aumentaria a complexidade do *plugin*, implicando aspetos semelhantes à incorporação da criação de *mockups* no mesmo e, acima de tudo, criaria uma dependência no Balsamiq Mockups e a eventuais alterações aos *widgets* em oferta em futuras iterações do programa. Todos estes elementos constituem aspetos que não cumprem os requisitos definidos para o *plugin* a desenvolver (especificamente os requisitos 1 e 2).

O HTML gerado pelo Mockplus exporta uma pasta com a estrutura de um projeto de uma página Web. Neste existe um `index.html` que pode ser aberto por qualquer navegador para visualizar o *mockup* com controlos interativos. Na estrutura de pastas existe um ficheiro XML denominado `Main.xml`, que descreve o *mockup*. Esta descrição consiste na menção de um conjunto de ficheiros SVG, contidos igualmente na estrutura de pastas, que no seu todo descrevem o *mockup*. Cada ficheiro SVG corresponde a um dos *widgets* utilizados na construção da interface. Apesar de ser um melhoramento em comparação com ZIP exportado pelo Balsamiq Mockups, uma das desvantagens deste formato de exportação acaba por ser a sua dependência em múltiplos ficheiros para descrever a totalidade de um só *mockup*.

No caso do formato JSON, este só é gerado pelo Balsamiq Mockups. Este lista os *widgets* utilizados no *mockup* e algumas propriedades, como o posicionamento, tamanho e id, sem qualquer tipo de descrição gráfica destes. Assim, verificam-se os mesmos problemas levantados relativamente ao formato ZIP.

Por fim, temos o formato SVG, gerado pelo Pencil. Este é exportado na forma de um único ficheiro. O SVG em si é um formato de imagens vetorial baseado em XML (Cohn et al., 2000). O ficheiro gerado pelo Pencil descreve todos os *widgets* como componentes, ou formas, dessa imagem vetorial. As camadas também se encontram especificadas no ficheiro.

Após a análise cuidada de todos os formatos, o SVG exportado pelo Pencil é o formato que se apresenta como o candidato mais provável a ser utilizado pelo *plugin*, não aparentando apresentar nenhuma das desvantagens apontadas pelos outros formatos de exportação. De modo a melhor entender o formato e a maneira como a informação dos *mockups* é representada neste, na secção seguinte este o formato será melhor descrito.

3.3 SVG

O SVG (Cohn et al., 2000) é um standard desenvolvido pela W3C, com a primeira versão lançada em 2001 que continua a ser iterado até hoje, com uma nova versão a ser preparada (versão 2.0 (David Dailey, 2018)). Este formato foi desenvolvido para solucionar um dos principais problemas dos formatos de imagens mais comuns utilizados na internet: a escalabilidade. Tomando uma imagem no formato JPEG, por exemplo, quando se aumenta a proporção desta para além do especificado, começamos a notar a degradação da qualidade da imagem. Isto deve-se ao facto que este tipo de formatos apenas guardam um conjunto fixo de pixéis. Já com o SVG, guardam-se um conjunto fixo de formas, que depois são desenhadas pelo programa de visualização. Esta solução permite manter a definição da imagem, independentemente do seu tamanho. Claro que o SVG não substitui formatos de imagens *raster*, servindo ambos propósitos distintos.

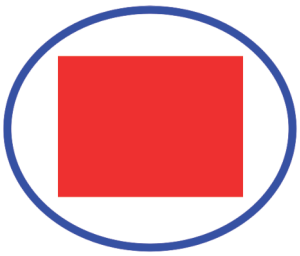


Figura 11: Figura de exemplo para demonstração da sintaxe SVG

De seguida apresenta-se um exemplo simples de uma figura criada em SVG e o seu código para demonstrar a sintaxe e características desta.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <svg height="500" version="1.1" width="500" xmlns="http://www.w3.org/2000/svg"
    >
3     <g id="layer1">
4       <rect height="99.029755" id="rect3713" style="stroke-width:0.26458332;
            fill:#ff2a2a" width="130.0238" x="40.065475" y="101.96429"/>
5       <ellipse cx="104.69941" cy="152.99106" id="path3715" rx="100.1637" ry="
            83.532738" style="fill:none;stroke-width:5.465;stroke:#0000f9;stroke
            -opacity:1;stroke-miterlimit:4;stroke-dasharray:none"/>
6   </g>
7 </svg>
```

Listagem 3.1: Código SVG da figura de exemplo

A Listagem 3.1 descreve a Figura 11. O exemplo em si é de complexidade reduzida mas permite transmitir as características essências do formato.

Como se verifica pela *tag* na linha 1, uma imagem SVG é descrita na linguagem de *markup* XML (Maler et al., 2004). Para utilizar as estruturas ou *tags* associadas ao SVG também é necessário definir o *namespace* do XML dentro do atributo `xmlns` no elemento `svg`, na linha 2. O elemento `svg` em si envolve sempre todas as estruturas utilizadas no documento e define atributos essenciais. Neste exemplo, os atributos especificados são o tamanho do *canvas* (`width` e `height`) onde serão desenhadas as formas.

Envolvendo a definição das formas encontra-se a *tag* `g`, nas linhas 3 e 6. Este é utilizada para agrupar formas, e permite separar diferentes elementos em grupos (ou camadas) distintas. No exemplo, o grupo composto pelo círculo e retângulo é denominado `layer1`.

Por último, são definidas as formas em si que compõe a Figura 11. A *tag* `rect` na linha 3 detalha o retângulo e a *tag* `ellipse` na linha 4 detalha o círculo. Para declarar as propriedades das formas, são utilizados atributos nas *tags* associadas. Observe-se que o círculo não tem preenchimento, tendo apenas definido o traço. Para atingir este efeito há três componentes:

- `style=''fill:none''` - define o preenchimento do círculo como inexistente;
- `stroke-width:5.465` - define a espessura do traço do círculo;
- `stroke:#0000f9` - define a cor do traço do círculo com representação hexadecimal.

O acesso a toda esta informação estruturada torna-se numa mais valia uma vez que pode ser extraída no planeado *plugin* para o IVY. Para além da definição de formas, também se podem definir animações, utilizando para o efeito linguagens de *scripting* como o Javascript, ou o SMIL. Nesta fase inicial não se prevê o uso desta funcionalidade mais avançada.

3.3.1 *Bibliotecas*

O *plugin* será desenvolvido em Java, visto que o próprio IVY também faz uso da mesma linguagem. Nativamente, a linguagem de programação suporta a manipulação de dados XML[1] (necessária para identificação dos elementos no SVG) e ainda oferece a possibilidade de visualização, utilizando a biblioteca Java2D[2] para o efeito. No entanto, para facilitar o trabalho de desenvolvimento, decidiu-se investigar bibliotecas de manipulação de SVGs. Definiu-se a seguinte lista de funcionalidades que a biblioteca escolhida deverá fornecer. Esta encontra-se ordenana desde mais importante a menos importante:

- Visualização do SVG - para apresentar os *mockups* na interface do programa;
- Edição do SVG - poderá ser necessário para manifestar o comportamento do *mockup* e identificação dos elementos;
- Documentação disponível - útil para expor a API em oferta e auxiliar o desenvolvimento.

Batik

O Batik (Foundation, 2016) é uma biblioteca Java que fornece um conjunto de funcionalidades relacionadas com a visualização, manipulação e animação de SVG. Suporta um número elevado de elementos, atributos e propriedades da especificação SVG 1.1. Também vem acompanhado de bastante documentação e exemplos para ajudar na utilização de todas as funcionalidades existentes.

SVG Salamander

O SVG Salamander (blackears, 2019) é uma biblioteca Java que, à semelhança do Batik, oferece um conjunto de funcionalidades para visualizar, manipular e animar SVG. O principal

1 https://docs.oracle.com/javase/tutorial/jaxp/dom/readingXML.html, visitado em 5 de julho de 2019
2 https://www.oracle.com/technetwork/articles/java/svg-141884.html, visitado em 5 de julho de 2019

	Batik SVG	SVG Salamander	JFreeSVG
Acesso árvore DOM	✓		-
Geração SVG	✓		
Adição Elementos	✓		
Edição Elementos	✓		
Remoção Elementos	✓		-
Visualização	✓		-
Eventos/Animação	✓		-
Scripting	✓	-	-
Documentação	✓	-	✓

Tabela 2: Comparação das bibliotecas SVG

elemento que distingue o Salamander é a eficiência, ou seja, a biblioteca foi desenhada para ser compacta e rápida. Ambos oferecem funcionalidades semelhantes, com o Batik a disponibilizar um leque mais vasto de opções, como por exemplo vários métodos de definir *scripting*, inexistentes no Salamander. Adicionalmente, o SVG Salamander sofreu uma recolocação de *sites*. Com este evento, grande parte da documentação disponível desapareceu, existindo neste momento apenas um tutorial básico sobre como utilizar a biblioteca.

JFreeSVG

O JFreeSVG (Limited, 2019) também surgiu como uma possível opção para lidar com SVG. Ao efetuar uma análise da biblioteca, notou-se que esta só fornece métodos para gerar SVG, não cumprindo os requisitos impostos.

Resumo

Na Tabela 2 apresentam-se um resumo das funcionalidades analisadas em cada uma das bibliotecas analisadas. Com uma análise da mesma, verifica-se que o Batik é a solução mais completa. Assim, este será selecionado para auxiliar com todas as questões relacionadas com SVG.

3.4 Conclusão

A definição do *mockup* e dos programas que o criam constitui um passo preliminar importante na investigação necessária para a conceção do programa. Neste capítulo foram

descritas várias ferramentas de criação de *mockups*: o Balsamiq Mockups, o Pencil Project e o Mockplus. Após uma análise e comparação, decidiu-se escolher o Pencil Project por apresentar a maior flexibilidade na escolha de *widgets* para desenhar o sistema, assim como na variedade de opções de exportação. Definiu-se que o formato que descreve o *mockup* deve estar num ficheiro estruturado, ou seja, que contenha uma descrição da sua estrutura acessível. Esta opção deverá permitir mais flexibilidade na conceção do protótipo no futuro *plugin*. Por último, optou-se por utilizar o SVG como o formato para descrever o *mockup*, por ser um formato de imagens vetoriais baseado em XML que detalha as várias figuras contidas.

A definição do *mockup* efetuada permite contextualizar o design do *plugin*. No próximo capítulo será descrito o *plugin*, a sua arquitetura e funcionalidades.

4

O *PLUGIN* "PROTOTYPER"

O nome dado ao novo *plugin* reflete a sua função. O "Prototyper" pretende acrescentar funcionalidades de representação gráfica do sistema a ser modelado no IVY Workbench através da importação de *mockups* representativos do mesmo. Este capítulo é dedicado à descrição do "Prototyper": o seu desenvolvimento e arquitetura, as decisões tomadas e o porquê destas e ainda uma explicação detalhada do modo de funcionamento da versão final do *plugin*.

4.1 FLUXO DE TRABALHO

O "Prototyper" serve como complemento às capacidades de análise de modelos do IVY Workbench. A Figura 12 representa um diagrama de atividades que ilustra a utilização do "Prototyper" no IVY. Espera-se que antes de se utilizar o *plugin* que o utilizador disponha do modelo do sistema compilado no IVY e do ficheiro do *mockup* no formato SVG. As atividades de elaboração do modelo e desenho do *mockup* do sistema podem ser realizadas concorrentemente. O "Prototyper" irá depois utilizar os artefactos de ambas as atividades para permitir a elaboração do protótipo. De notar que a atividade de desenho do *mockup* é realizada no Pencil. Apesar da especificação do SVG detalhar as formas e as suas propriedades, o Pencil utiliza uma estrutura padronizada. Os *widgets* podem ser compostos por uma ou mais formas, e encontram-se rodeados pelo elemento `g`, que identifica o nome do *widgets* pelo atributo `id`. O "Prototyper" depende desta estrutura para alguma da sua funcionalidade (mais detalhe na secção 4.3). A importação de ficheiros SVG gerados por outros programas é suportada, mas algumas das funcionalidades não se encontram disponíveis.

Depois de importar o SVG para o *plugin*, deve-se efetuar o mapeamento entre o *mockup* descrito e os atributos e ações do modelo e, por fim, iniciar a animação. Esta animação deverá auxiliar o utilizador na avaliação do protótipo, e se este manifesta o comportamento esperado. Caso seja necessário efetuar modificações ou ao modelo ou ao *mockup*, será necessário reiniciar o processo descrito no diagrama de atividades na Figura 12. Caso a configuração e o comportamento manifestado estejam de acordo com o esperado, pode-se proceder à gravação da configuração efetuada, que permitirá recarregar o protótipo quando

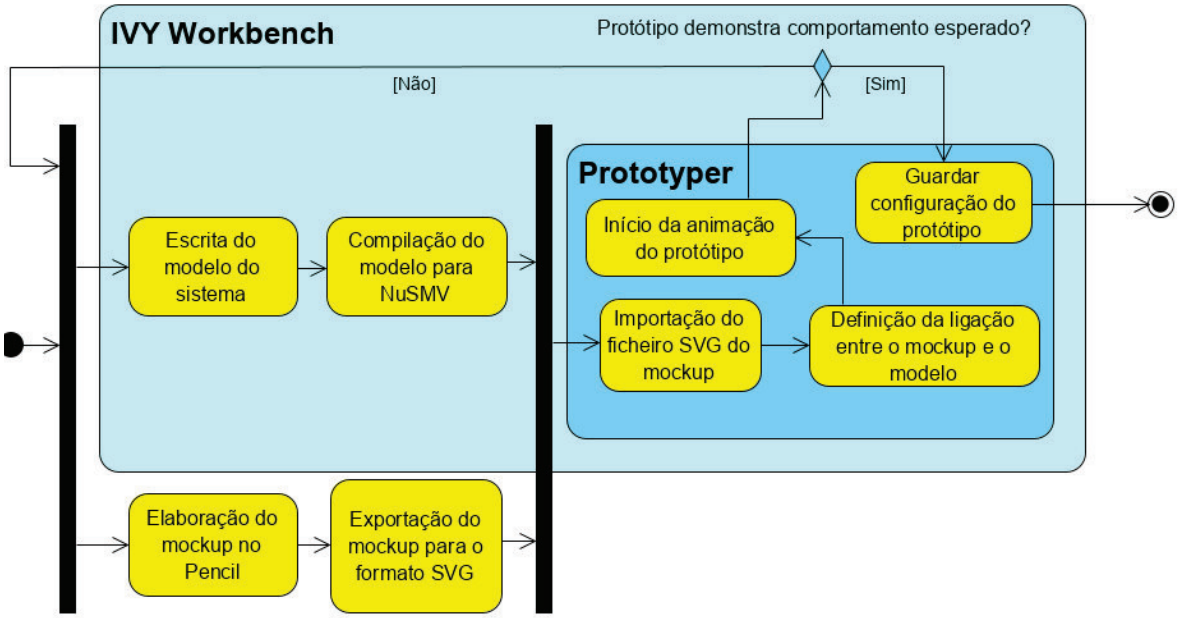


Figura 12: Diagrama de atividades que especifica o fluxo de trabalho esperado para o uso do *plugin*

necessário. A totalidade dos passos descritos constitui o fluxo de trabalho básico que se espera do uso do "Prototyper".

4.2 ARQUITETURA

Em relação à arquitetura do "Prototyper", adoptou-se uma abordagem Orientada aos Objetos, uma vez que o *plugin* foi desenvolvido em Java. Tirando proveito das capacidades do paradigma, realizou-se uma separação entre classes relacionadas com a interface e classes que efetuam todo o trabalho de fundo (não visível ao utilizador). Assim, as classes que contêm a lógica do programa foram distribuídas por dois pacotes principais. Por um lado, há o pacote denominado "Gui", constituído por classes responsáveis por atualizar a interface, apresentar os diálogos necessários e mostrar informação sobre o estado do SVG e dos seus elementos. Do lado oposto, está a infraestrutura que suporta toda a operação, denominada "Backend". As classes contidas neste pacote são responsáveis pelo processamento do SVG, e armazenamento de todas as estruturas de dados. Sempre que o utilizador executa uma ação na interface, esta efetua um pedido à "Backend", que devolve a informação necessária.

O diagrama de classes do *plugin* apresentado na Figura 13 ilustra o que foi descrito. A implementação inspira-se no padrão de *design* MVC (Krasner and Pope, 1988). As classes incorporadas no pacote "Gui" correspondem à "View", as classes no pacote "Backend" ao "Model" e a classe "MainPanel" ao "Controler", mediando os pedidos efetuados entre os

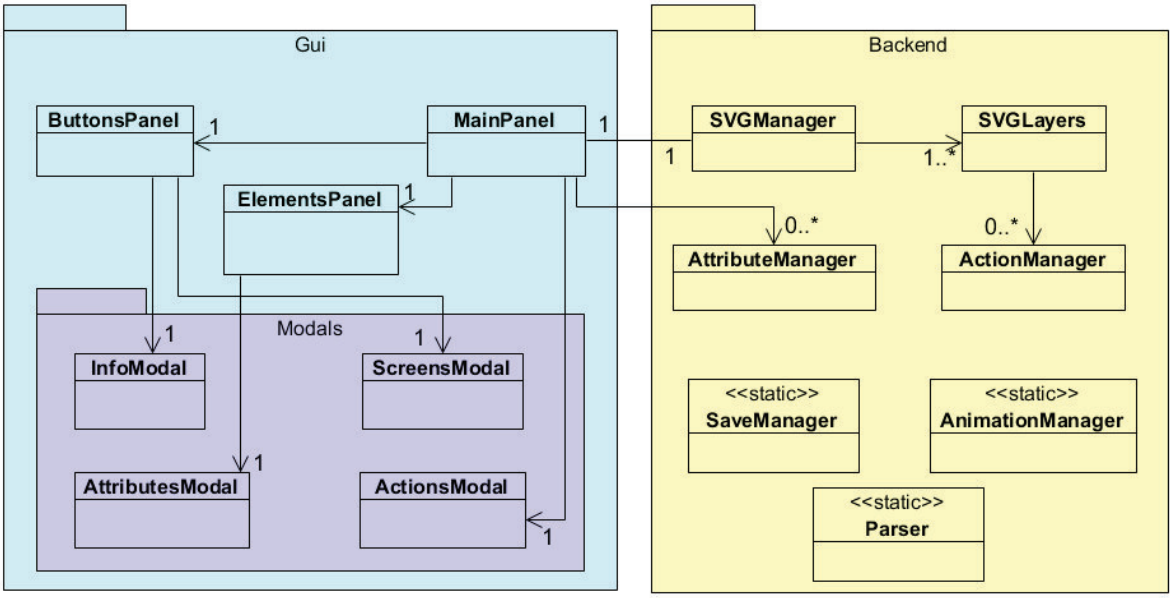


Figura 13: Estrutura do *plugin*

dois ("MainPanel" também incorpora funções de um componente "View", atualizando a visualização do SVG importado).

Para melhor demonstrar a sequência de execução típica de uma ação efetuada no *plugin* e o padrão de *design* MVC implementado, apresenta-se a Figura 14, que descreve o diagrama de sequência da importação de um SVG. Algumas das funcionalidades e conceitos que serão mencionado serão esclarecidos na secção 4.3, quando se explicar as funcionalidades fornecidas pelo *plugin* e a sua interface. Repare-se que todas as classes contidas no pacote "Gui" são identificados pela cor roxa e as classes contidas na "Backend" são identificadas pela cor verde.

Quando o utilizador importa o SVG do *mockup* para o "Prototyper" utilizando um dos botões da interface, o pedido e a informação associada são redirecionados para o "MainPanel". Este, por sua vez, envia esta informação para a "Backend", concretamente para o "SVGManager" que, como o nome indicia, é responsável por gerir os SVG importados. O "SVGManager" acede ao "Parser" para verificar os detalhes do ficheiro importado, nomeadamente se este origina de uma exportação do Pencil e quantas camadas suporta. Os dados do SVG são então enviados para "SVGLayers" que processa cada camada descrita individualmente, pedindo uma lista dos elementos interativos contidos nela ao "Parser" e armazenando-os em "ActionManager". Por fim, estes dados dos elementos interativos são devolvidos pelo "SVGLayers" até atingirem o "Main Panel". Com os novos dados recebidos, este atualiza a área de visualização com a representação do SVG importado e a tabela que contém uma lista dos elementos interativos disponíveis.

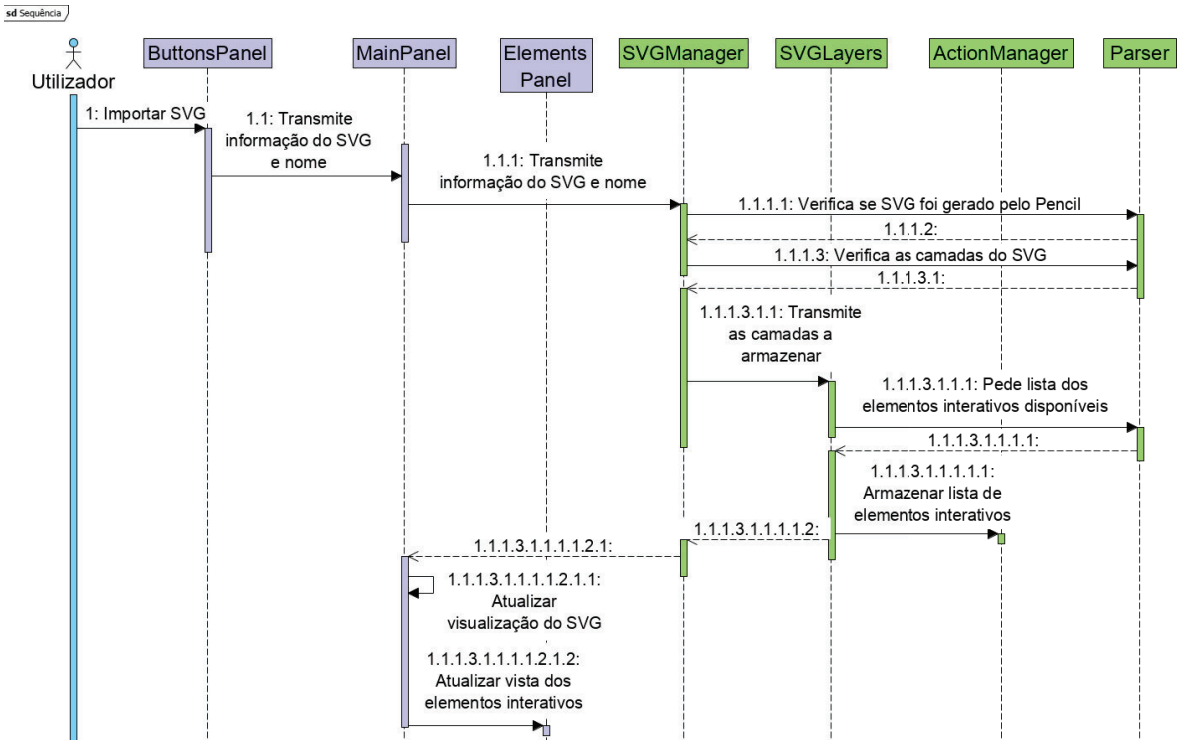


Figura 14: Diagrama de sequência que exemplifica as interações efetuadas ao importar um SVG

Por último, quanto ao modo de funcionamento, o "Prototyper" não executa num vácuo. O *plugin* efetua diversas trocas de dados para garantir as suas funcionalidades essenciais. Na Figura 15 visualiza-se os dados de *input* e *output*. Alguns dos dados de *input* são evidentes graças à informação exposta no decorrer deste documento, como a importação de protótipos em formato SVG ou a importação de um ficheiro com as configurações do utilizador. Estes são ficheiros externos ao próprio IVY, e importam-se diretamente para o *plugin*. Os outros dados de *input* advêm do IVY e das informações do modelo importado, como os atributos, ações e tipos. Já o *input* do NuSMV refere-se aos dados no momento da execução do modelo, ou seja, na animação do protótipo quando o utilizador interage com o mesmo, como os valores dos atributos e uma lista das ações possíveis. Os dados de *output* transmitidos para o NuSMV correspondem a ações que o utilizador executa. Por fim, os outros dados exportados correspondem à gravação de configurações realizadas.

A informação contida na Figura 15 demonstra uma visão geral de todas as trocas de informação necessárias para o funcionamento do *plugin*. No entanto, o diagrama constitui uma visão simplista das trocas de dados efetuadas com o IVY, não especificando os componentes envolvidos ou o método de troca. Relembrando a arquitetura da ferramenta, é possível efetuar o intercâmbio de informações entre *plugins* e serviços por dois métodos: subscrição de canais e memória partilhada.

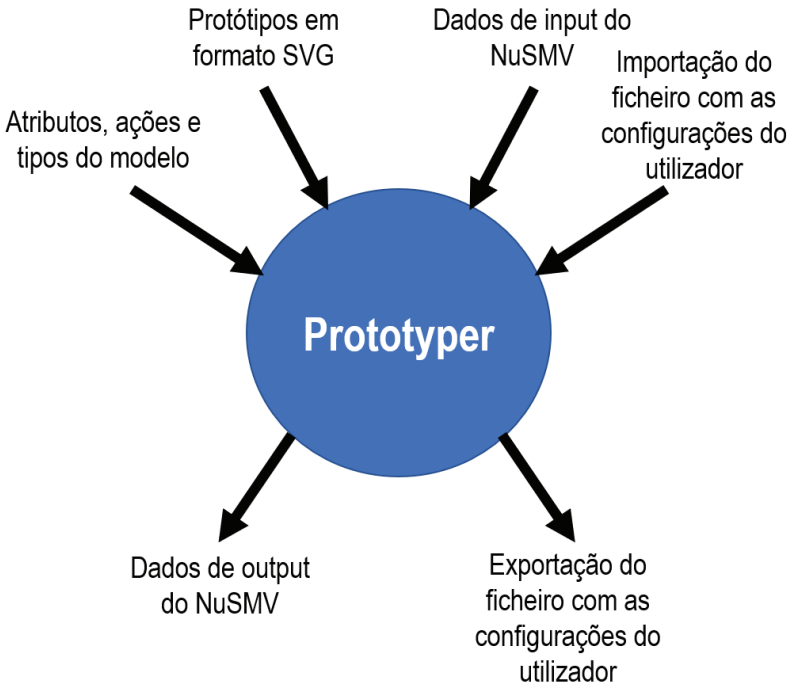


Figura 15: Dados de input e output que o "Prototyper" processa

Na Figura 16 identificam-se os canais a que o *plugin* subscreve, de onde os mesmos originam e a direção dos dados permutados. O "Prototyper" subscreve ao canal "model", oferecido pelo IVY-core, que permite verificar se um modelo foi previamente importado para o IVY. Os outros dois canais, "nusmv" e "nusmv-out-sim", são oferecidos pelo "Nusmv-service" e correspondem a dados de *input* e *output* da execução do modelo, respetivamente. O canal "nusmv" é utilizado para obter os valores dos atributos e uma listagem das ações possíveis no momento de execução e o canal "nusmv-out-sim" para enviar a ação executada pelo utilizador.

Relativamente à memória partilhada, é nesta onde se pode obter os dados do modelo. O "IVY-core", representado na Figura 16, publica os dados do modelo sempre que este for modificado ou compilado para a memória partilhada e o "Prototyper" recolhe-os. Esta troca é efetuada de forma assíncrona. O *plugin* assume que o modelo já foi definido anteriormente, como discutido no início desta secção, pelo que espera que estes dados já se encontrem colocados em memória partilhada.

4.3 FUNCIONALIDADES

A interface do novo *plugin* tem dois modos de funcionamento: o modo de edição e o modo de animação. O "Prototyper" inicia sempre no modo de edição. Neste é possível preparar

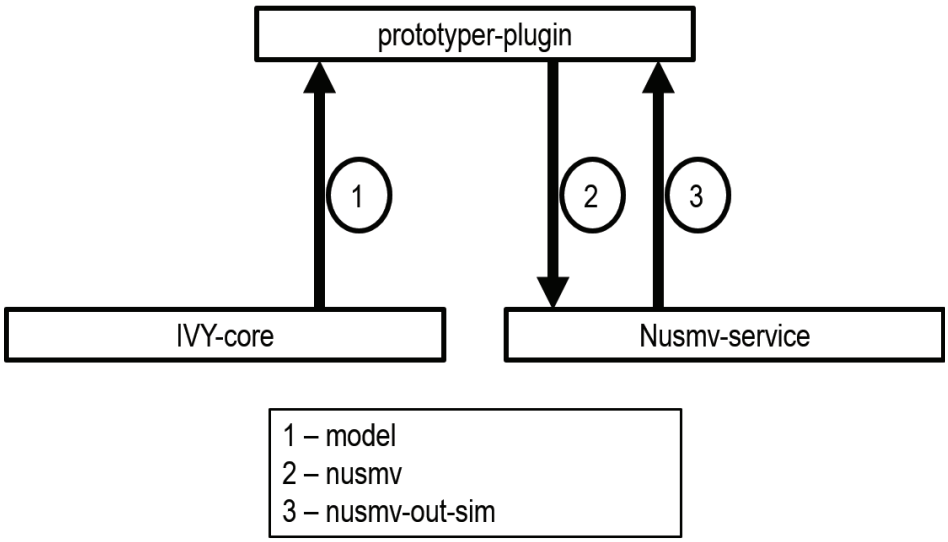


Figura 16: Esquema representativo dos canais que o "Prototyper" subscreve e o seu fluxo de informação

a animação, gerindo a ligação entre o *mockup* e o modelo e ajustar os vários parâmetros. Depois de tudo estar configurado o modo de animação encontra-se disponível, onde é possível visualizar a interação definida.

Ao selecionar o "Prototyper" dos vários separadores existentes no IVY, na parte superior da interface da Figura 17, a GUI do *plugin* será apresentada. A interface está dividida em três áreas principais, a partir das quais se pode aceder às funcionalidades disponibilizadas. A maior área, à esquerda, corresponde ao *canvas* de visualização do SVG importado. No topo encontram-se os controlos principais. Estes controlos incluem botões e *dropdowns* e são responsáveis por variadas funções:

- Importar um ficheiro SVG;
- Alternar entre os ficheiros SVG existentes (se houver mais do que um importado);
- Alternar entre as camadas do SVG selecionado;
- Eliminar SVG importado do *plugin*;
- Configurar a associação entre elementos do SVG e atributos do modelo do IVY;
- Apresentar informação sobre todos os elementos e as suas configurações;
- Iniciar a animação.

É possível importar mais do que um SVG representativo do protótipo, sendo que estes são selecionáveis pelo primeiro *dropdown* visível na Figura 17. Para alternar entre as camadas de cada SVG utiliza-se o segundo *dropdown*.

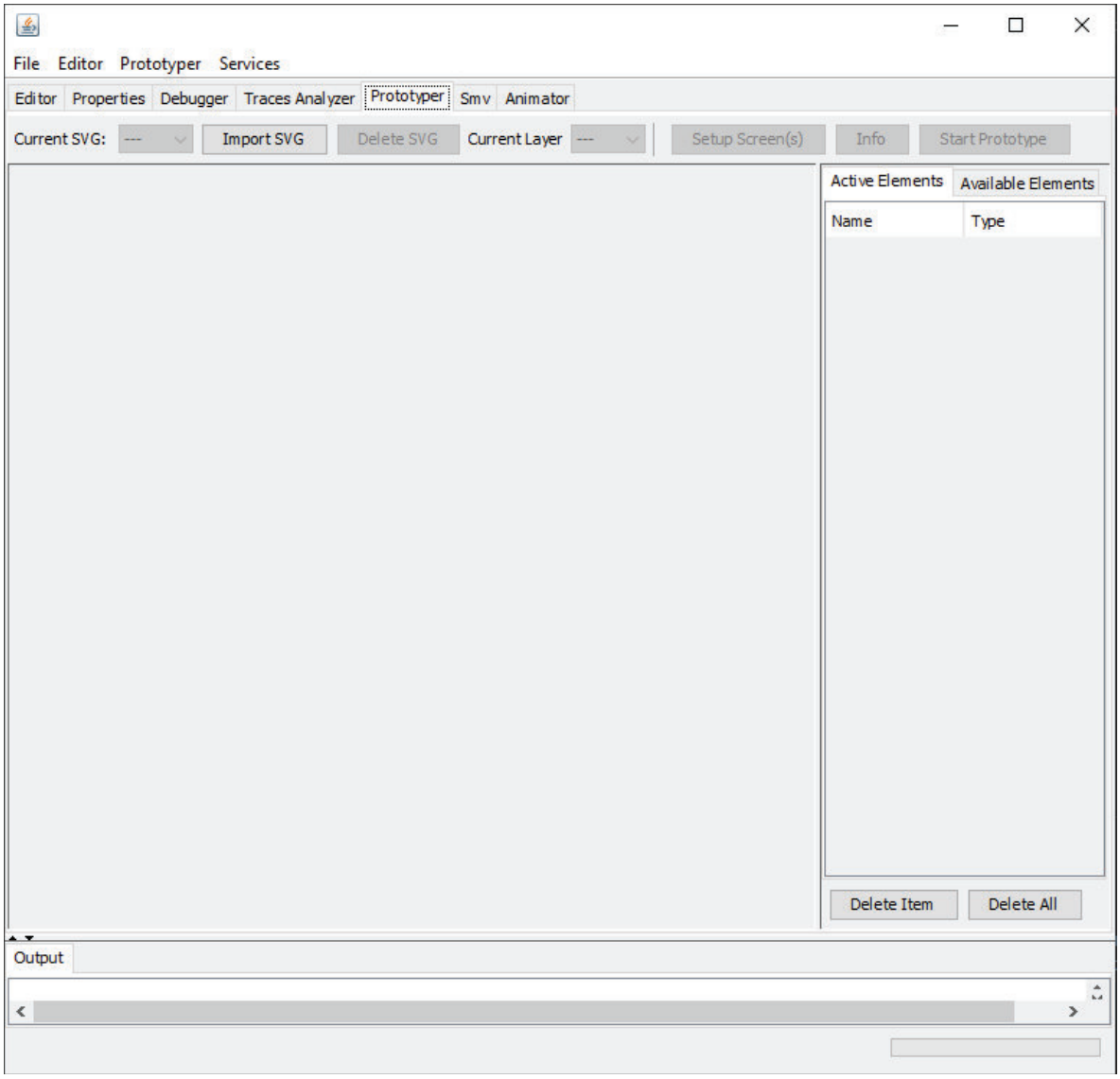


Figura 17: Interface do *plugin* Prototyper

A área à direita do *canvas* de visualização contém dois separadores, e em cada um destes existe uma tabela. Estas tabelas são denominadas "Available Elements" e "Active Elements". A primeira tabela mencionada lista todos os elementos textuais e interativos identificados no *mockup*. Já a tabela dos elementos ativos lista os elementos configurados pelo utilizador. Quando se seleciona o separador dos elementos ativos também aparecem dois botões (demonstrados na Figura 17) que permitem remover as configurações realizadas pelo utilizador de um elemento selecionado da tabela ou de todos os elementos.

O "Prototyper" cria a lista de todos os elementos interativos e textuais possíveis que existam dentro do SVG do protótipo no momento da importação. A identificação dos elementos que entram nesta lista é efetuada de forma diferente. Os SVG exportados pelo Pencil contêm neles uma identificação do tipo de forma. Por exemplo, caso o utilizador tenha utilizado os *widgets* de botões com estilo Android para construir o protótipo, estes encontrar-se-ão devidamente identificados no SVG. Esta informação é utilizada para identificar os elementos interativos que se podem associar a ações do modelo. Elementos interativos podem ser associados a ações do modelo e são reconhecidos pelo tipo. De um modo geral, só os *widgets* identificados como botões ou *checkboxes* é que são selecionados. Elementos textuais podem ser associados aos valores de um dado atributo do modelo e, à semelhança dos elementos interativos, também são reconhecidos pelo tipo. Qualquer elemento textual, como *labels* ou caixas de texto será selecionado.

Qualquer dos dois tipos de elementos podem ser configurados de duas maneiras: duplo clicando na entrada da tabela dos elementos disponíveis ("Available Elements") ou seleccionando-os diretamente no *canvas de visualização*. Quando se selecionam elementos interativos, a janela representada na Figura 18 aparecerá. Nesta pode-se definir a ação do modelo a associar ao elemento selecionado e o tipo de *trigger* para executar a ação no modo de animação. Há dois tipos de *trigger*:

- Clique do rato - a ação será executada no modo de animação ao clicar no elemento;
- Pressionar uma tecla no teclado - a ação será executada no modo de animação ao pressionar uma tecla especificada pelo utilizador.

A janela de configuração dos elementos textuais pode ser visualizada na Figura 24 na página 47, no segundo exemplo de aplicação. Nesta janela aparece uma *dropdown* com a lista dos atributos do modelo ainda não configurados e uma visualização dos valores que o atributo atualmente selecionado pode tomar, para auxiliar o utilizador na seleção.

Para distinguir entre elementos interativos e elementos textuais, a tabela dos elementos ativos contém uma segunda coluna, intitulada "Type", que indica se o elemento foi associado a uma ação ou a um atributo (esta coluna pode ser vista na Figura 17). Quando se seleciona um dos elementos de qualquer uma das tabelas, o mesmo é realçado no *canvas* de visualização para auxiliar o utilizador (este efeito pode ser visto na Figura 19). Quando um

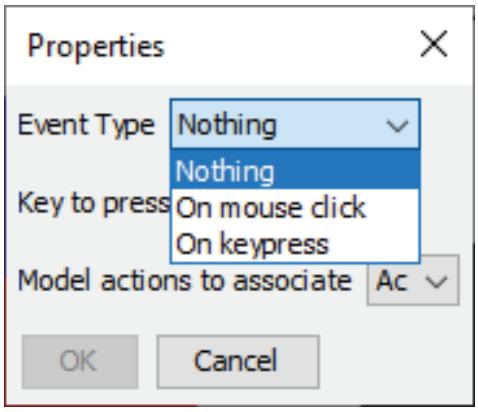


Figura 18: Janela *popup* para definir as propriedades de elementos interativos

elemento é configurado, este é movido da tabela de elementos disponíveis ("Available Elements") para a tabela dos elementos ativos ("Active Elements"). De forma idêntica, quando a configuração de um elemento é eliminada este é movido da tabela dos elementos ativos para a tabela dos elementos disponíveis.

A configuração da associação entre protótipo e atributos do modelo efetua-se de duas maneiras distintas, mas complementares:

- Associando atributos a elementos textuais no SVG (como já foi mencionado);
- Associando os valores tomados por um atributo e a camadas específicas dos ficheiros SVG importados.

A associação dos valores dos atributos aos SVG e as suas camadas significa que cada valor que o atributo possa tomar fica associado a uma camada de um SVG específico. Com os exemplos descritos no capítulo seguinte, esta funcionalidade será exemplificada. Para configurar esta associação, o utilizador pressiona o botão entitulado "Setup Screen(s)" nos controlos superiores, visíveis na Figura 17. Ao selecionar este botão, a janela apresentada na Figura 21 na página 43, relativa ao primeiro exemplo de aplicação, irá aparecer. Nesta é possível selecionar o atributo cujos valores se pretendem associar aos ecrãs do SVG. Adicionalmente, todos os valores tomados são listados com *dropdowns* para os associar cada um a uma camada de um SVG específica.

Outra funcionalidade igualmente importante no Prototyper é a possibilidade de guardar o protótipo e as associações realizadas. Esta funcionalidade alivia o trabalho necessário quando o modelo do sistema tem uma elevada complexidade, pois permite reiniciar o trabalho sem ter de realizar as mesmas associações sempre que se abra o *plugin*. Para guardar ou importar as configurações realizadas basta aceder ao menu do "Prototyper" na barra de menus do IVY. O ficheiro que contém a configuração contém a extensão ".pcf", que significa "Prototyper Configuration File". Este equivale um ficheiro ZIP que alberga os

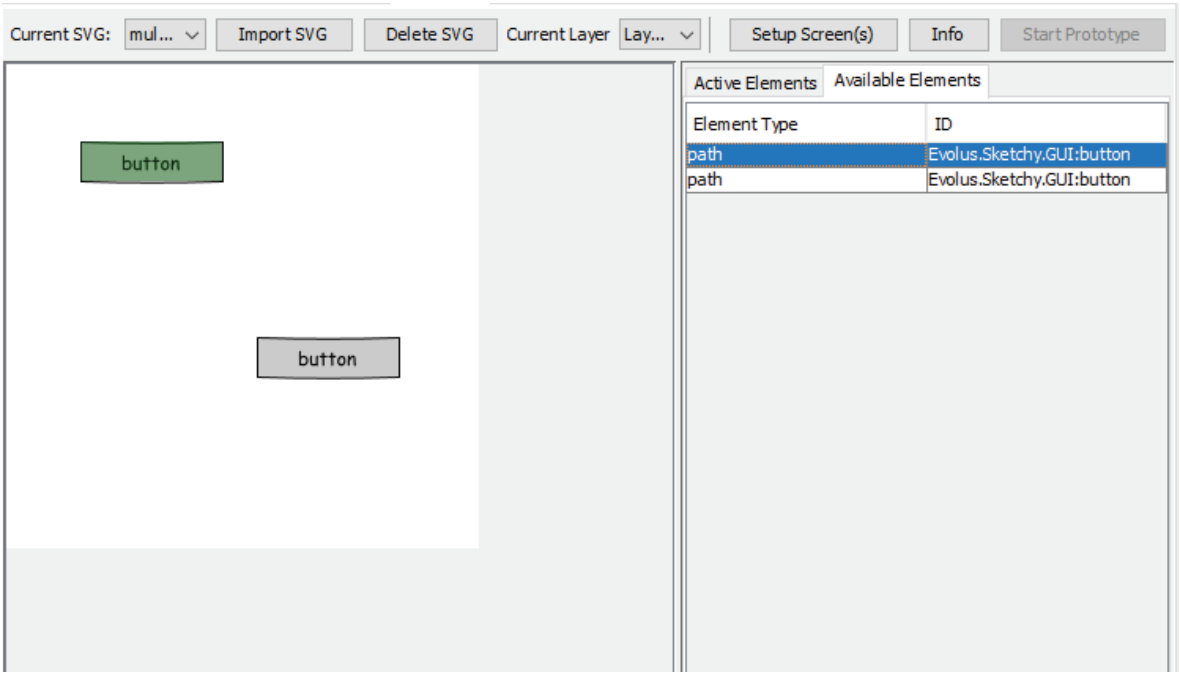


Figura 19: Exemplo do realce de um elemento selecionado na tabela de elementos disponíveis

ficheiros SVG importados e um ficheiro de texto JSON, que explicita todas as configurações efetuadas.

4.4 CONCLUSÃO

Neste capítulo foi detalhado o *plugin* "Prototyper", o fluxo de trabalho esperado, a sua arquitetura e o seu modo de funcionamento. Estas descrições permitiram compreender a forma como o *plugin* interage com o IVY, como o seu fluxo de trabalho encaixa no IVY e os diferentes modos para configurar o protótipo.

Para complementar a descrição das funcionalidades do "Prototyper", no capítulo seguinte serão apresentados três casos de estudo: o portão de uma garagem, uma bomba de infusão médica e um relógio digital. Estes exemplos irão permitir uma melhor compreensão das capacidades de prototipagem em oferta e do modo de animação.

5

EXEMPLOS DE APLICAÇÃO

Neste capítulo apresentam-se três exemplos de aplicação, que irão utilizar as funcionalidades do "Prototyper" descritas até este ponto. Cada exemplo apresentado terá um maior nível de complexidade que o anterior. No final pretende-se que as funcionalidades de construção de protótipos sejam esclarecidas.

5.1 PORTÃO DE GARAGEM

O primeiro exemplo que se irá expor descreve um portão de uma garagem controlado por um comando. O exemplo propriamente dito é relativamente simples, mas servirá para ilustrar, quer que a solução proposta não está limitada a interfaces no sentido tradicional, quer o tratamento de múltiplas camadas no SVG.

Os detalhes do exemplo e o próprio modelo descrito em MAL já foram descritos na secção 2.1.2. Resumidamente, o estado do portão é modelado por um atributo "situacao" que pode estar em um de quatro estados: a abrir, a fechar, aberto e fechado. O botão do comando é modelado pela acção "Ac", existindo ainda sensores que sinalizam quando o portão fica completamente aberto ou completamente fechado, modelados pelas acções "Ia" e "If", respetivamente. O modelo pode ser consultado na Listagem 2.1, apresentado na página 8. Aqui, o foco passará pela descrição do *mockup* criado no Pencil, em suporte à criação do protótipo no IVY, do mapeamento entre este mockup e o modelo, efetuado no "Prototyper", e do modo de animação.

Na Figura 20 encontram-se ilustrados os *mockups*, criados no Pencil, correspondentes aos vários estados em que o modelo se pode encontrar, representados por A, B, C e D. O portão no estado A tem um tamanho superior para melhor visualizar os detalhes do protótipo, que contém os mesmos controlos no resto dos estados. A imagem maior, A, representa o portão fechado. Os botões pretos representam os sensores que são utilizados para indicar quando o portão fica completamente fechado ou completamente aberto. O botão vermelho representa o controlo remoto que o utilizador usa para abrir ou fechar o portão. Os restantes *mockups* possuem os mesmos botões. O mockup, B representa o portão a abrir, o portão C aberto e o portão D a fechar.

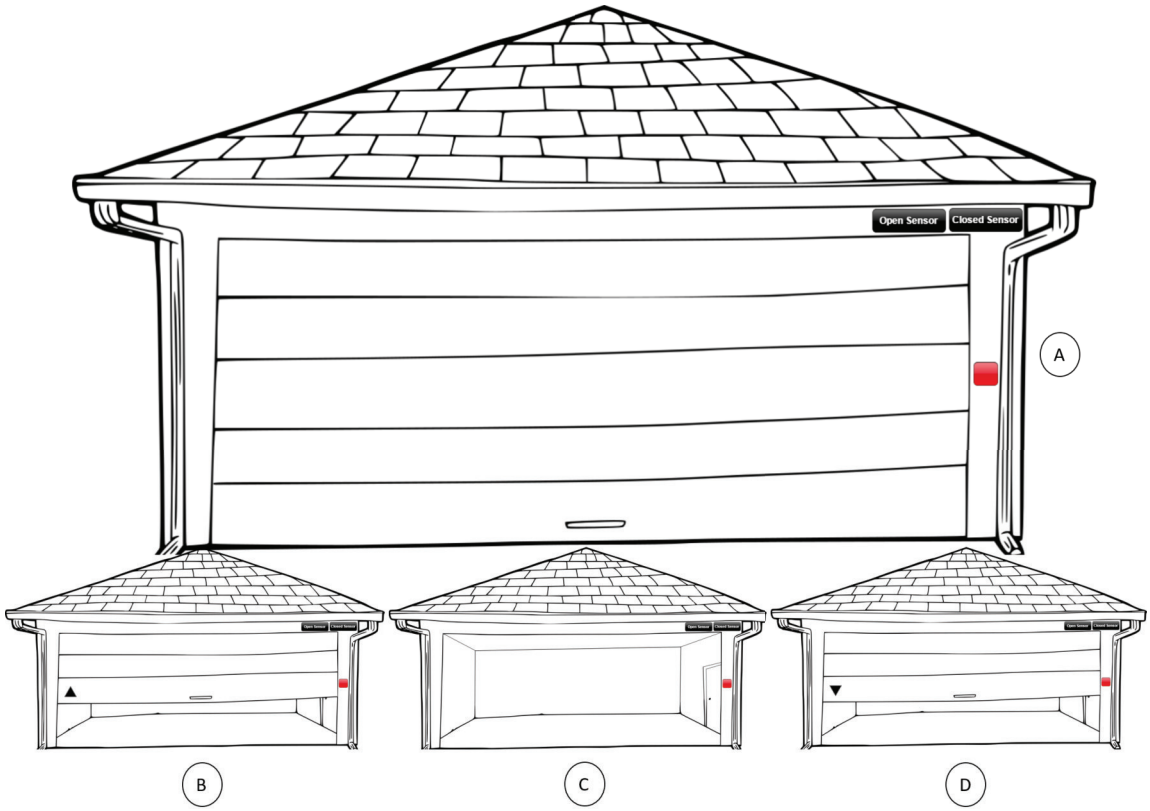


Figura 20: Protótipo do portão de garagem criado no Pencil com os seus estados representados por A, B, C, e D[1]

Após finalizar o protótipo no Pencil, este foi exportado para o formato SVG e importado para o *plugin*. Para obter o protótipo é necessário configurar as ligações entre o atributo e as ações do modelo e os elementos relevantes do protótipo. As ligações entre as ações e o SVG foram efetuadas da seguinte maneira:

- `Ac` - a ação representativa do botão do comando fica associada ao botão vermelho no protótipo e este será acionado com o clique do rato;
- `If` - a ação que representa o sensor a detetar que o portão está fechado fica associada ao botão preto esquerdo no protótipo que é acionado com o clique do rato;
- `Ia` - a ação que representa o sensor a detetar que o portão está aberto fica associada ao botão preto direito no protótipo que é acionado com o clique do rato;

O valores que o estado pode tomar serão ligados às camadas do protótipo:

- `a_abrir` - camada B

1 Imagens do portão retiradas de: https://www.canstockphoto.com.br/porta-garagem-esbo%C3%A7o-abertos-25407780.html

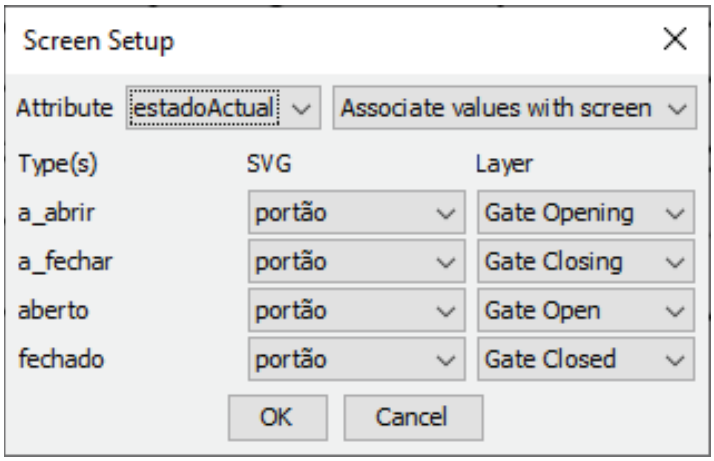


Figura 21: Janela de configuração dos valores do atributo com os ecrãs do *mockup*

- `a_fechar` - camada D
- `aberto` - camada C
- `fechado` - camada A

Este é um exemplo de configuração simples, exigindo apenas o mapeamento das ações em elementos do SVG e dos valores do atributo às camadas do mesmo. Na Figura 21 apresenta-se a configuração efetuada. Note-se que os valores dos SVG e *layers* de cada valor do atributo correspondem aos nomes dados às camadas do SVG no Pencil na elaboração do *mockup*. Respetivamente, a *layer* "Gate Opening" corresponde ao estado B da Figura 20, "Gate Closing" ao estado D, "Gate Open" ao estado C e "Gate Closed" ao estado A.

Na Figura 22 demonstra-se a interface da janela de animação com o protótipo do portão. À semelhança da interface principal do *plugin*, a secção à esquerda da interface apresenta o protótipo na camada SVG que foi configurada a aparecer no dado momento de execução. Como o `estadoActual` assume o valor de "aberto", visualiza-se o protótipo na camada C. Os elementos do protótipos associados a ações podem ser interagidos diretamente no *canvas*. Como todos os elementos foram configurados para serem acionados com um clique do rato, clicar nos elementos no *canvas* irá resultar na execução de uma ação. Caso fossem configurados para acionarem com uma tecla especifica do teclado, o pressionar da mesma resultaria na execução da ação especificada. O painel lateral direito encontra-se dividido em duas secções. A secção superior demonstra as ações que se podem executar no estado atual. A secção inferior apresenta os atributos e o valor que tomam no momento de execução. Estes painéis auxiliam o utilizador com informação adicional e, no caso da secção que lista as ações, também se pode clicar duas vezes numa ação para progredir a animação. Sendo este um exemplo de complexidade reduzida, as animações consistem na transição entre ecrãs.

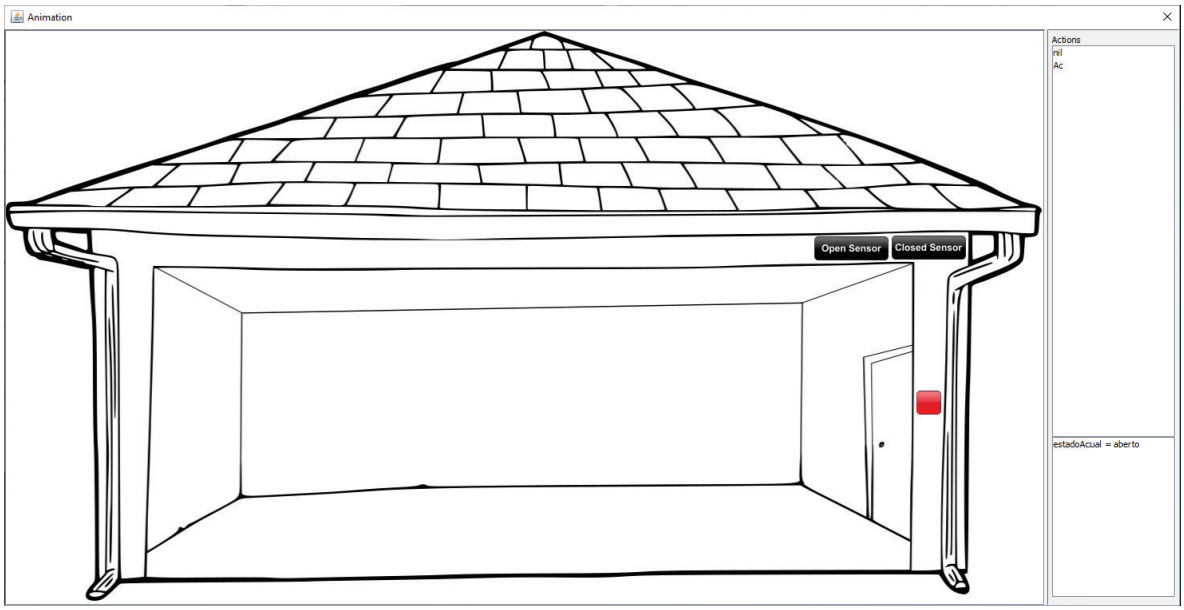


Figura 22: Janela de animação do "Prototyper"

A seguir apresenta-se um exemplo mais complexo que lida com a manipulação de conteúdo dentro do protótipo.

5.2 BOMBA DE INFUSÃO

Apresenta-se agora um exemplo de utilização do *plugin* para obtenção de um protótipo de uma bomba de infusão. Trata-se de um dispositivo responsável pela injeção intravenosa de fluídos em pacientes (por exemplo, medicamentos ou contrastes endovenosos para realização de exames). Considere-se que existe já um modelo da bomba de infusão em causa (ver Listagem 5.1). Por uma questão de facilidade de apresentação no contexto da dissertação, o modelo captura simplesmente a manipulação da quantidade de fluído a ser injetado.

O modelo tem especificados atributos, ações e axiomas. Nas linhas 8 e 9 estão declarados dois atributos:

- `on` - do tipo boolean, pode ter o valor de verdadeiro ou falso. Este atributo é utilizado para indicar se a bomba de infusão se encontra ligada, ou não.
- `value` - pode ter um valor numérico, no intervalo de 0 a 25. Este atributo guarda a quantidade de fluído a ser infundido pela bomba de infusão quando está ligada, sendo esta quantidade numa unidade de medida não especificada.

Na linha 11 são apresentadas as ações do modelo:

```
1  defines
2    MAXV = 25
3  types
4    TDisplayValue = 0..MAXV
5
6  interactor main
7    attributes
8      value: TDisplayValue
9      on: boolean
10   actions
11     onoff sup sdown lup ldown
12   axioms
13     [] !on & value = 0
14     [onoff] on' = !on & value'=0
15     per(sup) -> value<MAXV & on
16     [sup] value' = value + 1 & on'
17     per(sdown) -> value>0 & on
18     [sdown] value' = value - 1 & on'
19     per(lup) -> value <= (MAXV-10) & on
20     [lup] value' = value + 10 & on'
21     per(ldown) -> value >= 10 & on
22     [ldown] value' = value - 10 & on'
```

Listagem 5.1: Modelo da bomba de infusão em MAL

- `onoff` - liga ou desliga a bomba de infusão se estiver desligada ou ligada, respetivamente.
- `sup` - incrementa o fluído a ser infundido em 1 unidade.
- `sdown` - decrementa o fluído a ser infundido em 1 unidade.
- `lup` - incrementa o fluído a ser infundido em 10 unidades.
- `ldown` - decrementa o fluído a ser infundido em 10 unidade.

Por fim, existem os axiomas, visíveis entre as linhas 13 e 22, que definem o comportamento do modelo com base nas ações e atributos mencionados atrás. Para além de definirem o efeito das diferentes acções no estado do dispositivo (por exemplo, o axioma na linha 16 indica que a ação `sup` incrementa o valor de `value` em uma unidade e deixa o atributo `on` com valor verdadeiro), também definem algumas restrições: a bomba só pode infundir quando estiver ligada e o valor a infundir não pode ir abaixo de 0 e acima dos 25 definidos. O axioma na linha 15, por exemplo, define que o evento `sup` (incrementa o valor de 1) só é permitido se o valor actual for inferior ao máximo e se o dispositivo estiver ligado.

Na Figura 23 apresenta-se um protótipo criado no Pencil como representação visual do modelo descrito. A bomba de infusão pode estar desligada ou ligada. Estes dois estados são representados pelas duas imagens na figura. No topo encontra-se a bomba desligada e em baixo a bomba ligada. A distinção entre elas faz-se pelo facto de a iluminação do ecrã

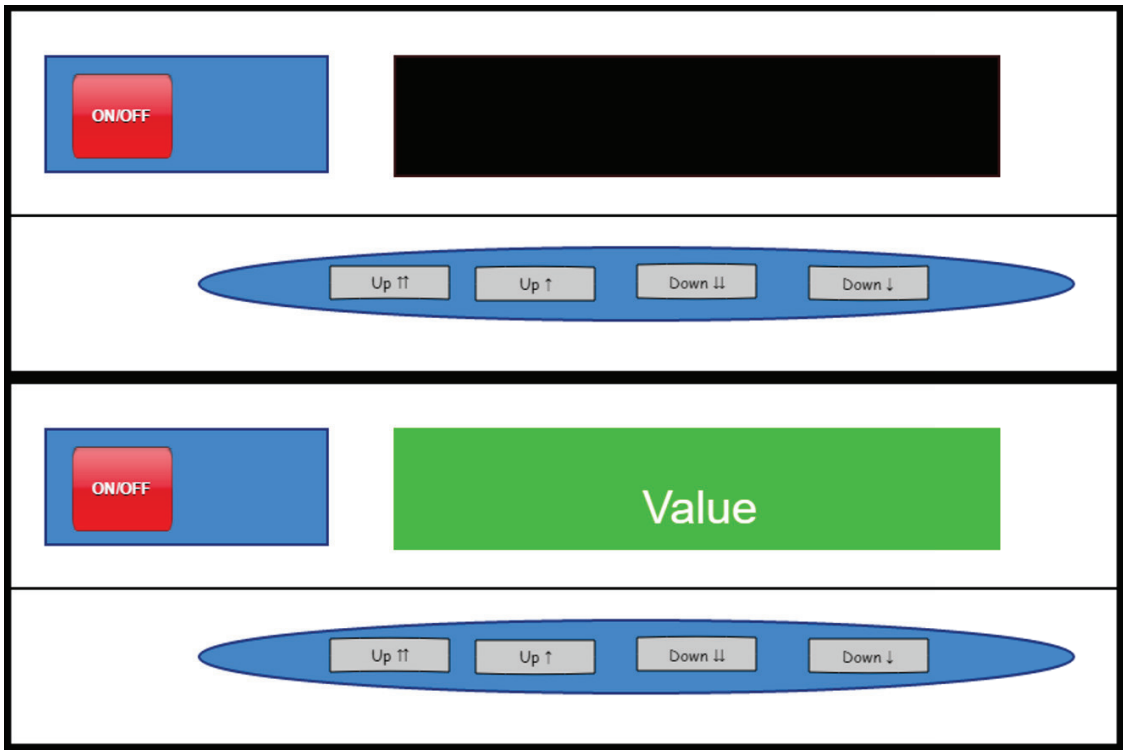


Figura 23: Protótipo da bomba de infusão criado no Pencil com as suas duas camadas

estar desligada (imagem de cima) ou ligada (imagem de baixo). No SVG exportado estas duas figuras estão descritas como duas camadas. Os botões de controlo são comuns aos dois estados, e representam todas as ações descritas anteriormente. A distinção entre botões que aumentam ou decrementam o fluído a ser injetado em 1 ou 10 unidades é realizada pelo número de setas a acompanhar o rótulo. Se o botão tiver duas setas significa que aumenta ou decrementa 10 unidades. Se tiver uma seta aumenta ou decrementa 1 unidade.

A configuração do protótipo passa assim, por associar o valor `false` da variável `on` à camada apresentada na parte superior da imagem 23, e o valor `true` à camada apresentada na parte inferior. Em ambas o botão ON/OFF é associado ao evento `onoff`. Já os botões "Up ↑↑", "Up ↑", "Down ↓↓" e "Down ↓" são associados aos eventos `lup`, `sup`, `ldown` e `sdown`, respetivamente.

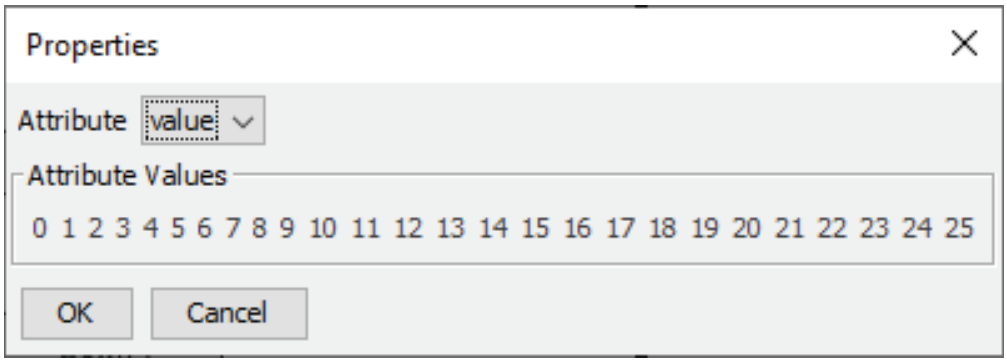


Figura 24: Janela de configuração de atributos com elementos textuais do protótipo

Neste exemplo existe uma configuração extra não demonstrada no anterior. É necessário apresentar os valores tomados pelo atributo `value` do modelo no ecrã do protótipo. Para conseguir isso é necessário associar o elemento de texto existente no ecrã do *mockup* correspondente ao estado ligado ao atributo `value`. Esta associação é realizada selecionando o elemento e texto, ou directamente no *mockup*, ou na na tabela "Available Elements", como descrito atrás na secção 4.3. Ao efetuar esta seleção, a janela apresentada na Figura 24 irá aparecer com uma *dropdown* para selecionar o atributo e uma visualização dos valores que este pode tomar, para auxiliar o utilizador na escolha.

Na Figura 25 pode-se observar o modo de animação do Prototyper em execução, com a ligação entre os elementos interativos do protótipo e o modelo já configurada. A sequência de execução inicia-se com a bomba de infusão desligada, tal como especificado no modelo. De seguida, liga-se a bomba, momento representado pelo ecrã seguinte, em que o valor a infundir se encontra a zero. Ao pressionar o botão "Up ↑" a execução progride, mudando o valor no ecrã para uma unidade a infundir. Finalmente, ao pressionar o botão "Up ↑↑" atinge-se o último ecrã demonstrado na Figura 25, com o valor de 11 unidades de líquido a infundir.

O exemplo demonstrado manifesta um comportamento mais avançado que o anterior do portão, com manipulação de elementos do protótipo importado. A seguir apresenta-se um caso de estudo que conjuga a transição entre ecrãs com a manipulação de múltiplos valores.

5.3 Relógio multi-funções

Por último, apresenta-se o exemplo de um relógio digital com duas funções. O aspeto e as funções que deve suportar já foram mencionados na secção 3.1.1. A descrição do exemplo passará por uma análise do modelo em MAL, que se apresenta a seguir.

O modelo MAL do relógio é de maior complexidade que os anteriores, apresentando um maior número de atributos e axiomas. Neste é descrito um relógio com dois modos que oferecem funcionalidades distintas. O modo de relógio apresenta as horas e minutos. O modo

Figura 25: Animação da bomba de infusão em execução

de cronómetro permite iniciar um temporizador e medir o tempo com sensibilidade até aos segundos. Sempre que se alterna entre o cronómetro e o relógio, o primeiro reinicia o temporizador a zero. O relógio mantém sempre as horas em memória, independentemente do modo selecionado. Quando o cronómetro tem o temporizador ligado e um minuto passa, o mesmo ocorre no relógio.

Introduz-se o modelo, começando pelos tipos que definem os valores a utilizar:

- `Minutes` - um valor numérico no intervalo de 0 a 59 unidades. Este tipo pode representar tanto minutos como segundos;
- `Hours` - um valor numérico no intervalo de 0 a 23 unidades. Este tipo representa as horas;
- `Mode` - um tipo enumerado utilizado para representar os modos possíveis do relógio.

De seguida, apresentam-se os atributos do modelo:

- `clkmin` - representa os minutos do relógio;

```
defines
 clockTick = (clkmin=59 & clkhour=23 -> clkmin'=0 & clkhour'=0) & (clkmin=59 &
      clkhour<23 -> clkmin'=0 & clkhour'=clkhour + 1) & (clkmin<59 -> clkmin'=
     clkmin + 1 & keep(clkhour))
types
 Minutes = 0..59
 Hours = 0..23
 Mode = {Clock, Chrono}
interactor main
 attributes
  clkmin: Minutes
  clkhour: Hours
  chmin: Minutes
  chhour: Hours
  chsec: Minutes
  [vis] dispmode: Mode
 actions
  [vis] clockbtn
  [vis] chronobtn
  tick
 axioms
  [] dispmode=Clock & clkmin=0 & clkhour=0 & chmin=0 & chhour=0 & chsec=0
  dispmode=Clock -> chhour=0 & chmin=0 & chsec=0
  [clockbtn] dispmode'=Clock & keep(clkmin,clkhour)
  [chronobtn] dispmode'=Chrono & keep(clkmin,clkhour,chmin,chhour,chsec)
  [tick]  keep(dispmode)
  dispmode=Clock -> [tick] clockTick
  dispmode=Chrono & chsec=59 & chmin=59 & chhour=23 -> [tick] chsec'=0 & chmin
      '=0 & chhour'=0 & clockTick
  dispmode=Chrono & chsec=59 & chmin=59 & chhour<23 -> [tick] chsec'=0 & chmin
      '=0 & chhour'=chhour + 1 & clockTick
  dispmode=Chrono & chsec=59 & chmin<59 -> [tick] chsec'=0 & chmin'=chmin + 1
     & keep(chhour) & clockTick
  dispmode=Chrono & chsec<59 -> [tick] chsec'=chsec+1 & keep(chmin,chhour,
     clkmin,clkhour)
```

Listagem 5.2: Modelo do relógio em MAL

- `clkhour` - representa as horas do relógio;
- `chsec` - representa os segundos do cronómetro;
- `chmin` - representa os minutos do cronómetro;
- `chhour` - representa as horas do cronómetro;
- `dispmode` - representa o modo do relógio;

Por último, só existem três ações no modelo:

- `clockbtn` - botão para mudar para o modo de relógio;
- `chronobtn` - botão para mudar para o modo de cronómetro;
- `tick` - representa o passar do tempo, em minutos no caso do relógio e em segundos no caso do cronómetro;

Os axiomas detalham o comportamento do sistema. O dispositivo inicia sempre no modo de relógio na meia noite. Destaca-se a definição de `clockTick` na linha 2, que corresponde à especificação do comportamento que o relógio deve ter quando um `tick` ocorre. Se o valor dos minutos for inferior a 59, no próximo estado este será incrementado por uma unidade, mantendo-se o valor das horas. Se o valor dos minutos for igual a 59 e o valor das horas inferior a 23, no próximo estado os minutos regressarão às 0 unidades e as horas serão incrementadas por uma unidade. Finalmente, se o valor dos minutos for igual a 59 e o valor das horas igual a 23, no próximo estado ambos os valores regressarão às zero unidades.

O cronómetro tem uma definição do comportamento semelhante, com a adição dos segundos. A definição de `clockTick` adicionada ao modelo para simplificar a representação dos axiomas, nomeadamente na secção do cronómetro. Como já se mencionou, quando um minuto passa no temporizador também passa um minuto no relógio. Tal implica que sempre que se executa um `tick` no modo de cronómetro e os segundos têm um valor de 59, um `clockTick` será executado.

Na Figura 26 apresenta-se o protótipo relógio implementado no Pencil. O *mockup* aqui utilizado é diferente do apresentado na Figura 9 (Capítulo 3). Lá, por uma questão de consistência com os *mockups* desenvolvidos nas outras ferramentas, procurou representar-se um relógio de pulso; aqui, optou-se por uma representação mais próxima de um relógio de mesa. Tal deve-se aos elementos da interface maiores, que permitem distinguir melhor os botões e ter símbolos que indicam o modo que cada botão pode mudar. Apesar disso, o mockup inclui os mesmos elementos, com a adição da seta na lateral direita, representativa do `tick` do relógio ou cronómetro. Os botões de baixo permitem mudar entre os modos

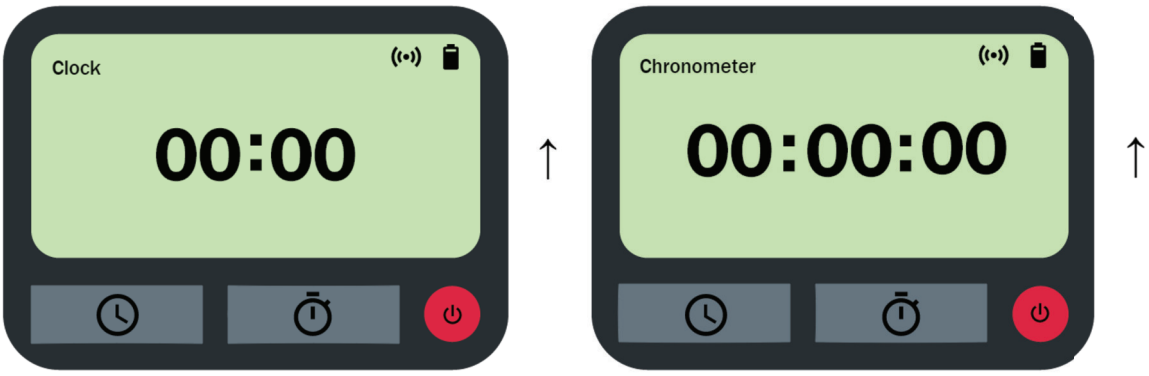


Figura 26: Protótipo do relógio criado no Pencil com as suas duas camadas

do relógio. O botão vermelho é meramente cosmético, pois no modelo não é definido que o dispositivo se possa desligar. Um detalhe relevante do protótipo é o facto dos campos das horas, minutos e segundos corresponderem a áreas de texto distintas, para se poder associar individualmente cada um destes aos atributos correspondentes. O atributo do modelo `dispmode` ficou associado aos ecrãs do protótipo. Destaca-se também a atribuição do `tick`, onde se definiu o *trigger* como o pressionar da tecla "U".

Com as ligações entre o protótipo e o modelo configuradas, pode-se executar a animação. Na Figura 27 demonstra-se um exemplo de execução. A animação inicia no modo de relógio, como especificado no modelo. Pressionando a tecla "U" cinco vezes, os minutos são incrementados por cinco. De seguida, alterna-se para o modo de cronómetro. Este inicia a zero. Como o cronómetro mede os segundos, para passar um minuto é necessário executar a ação `tick` sessenta vezes, resultando tal na imagem à direita. Ao alternar de volta para o relógio, o contador do cronómetro reinicia para zero e verifica-se que o relógio incrementou uma unidade no campo dos minutos, face à representação anterior.

Um efeito curioso na representação dos valores é que a casa das décimas não é visível quando os números são de tamanho unitário. Tal deve-se ao facto de os valores representados no protótipo serem uma réplica dos valores do modelo na altura da execução, sem qualquer tipo de processamento para acrescentar o zero nas décimas no caso dos números unitários. Esta questão representa uma mera peculiaridade estética que não retira da informação que se pode retirar da animação do protótipo. Mas a possibilidade de ter widgets para representar números, em que fosse possível formar os números, podia ser incluída no trabalho futuro,

Este caso de estudo é o mais complexo dos três apresentados e é o que melhor abrange todas as funcionalidades do *plugin*. Há associações de elementos interativos a ações com os

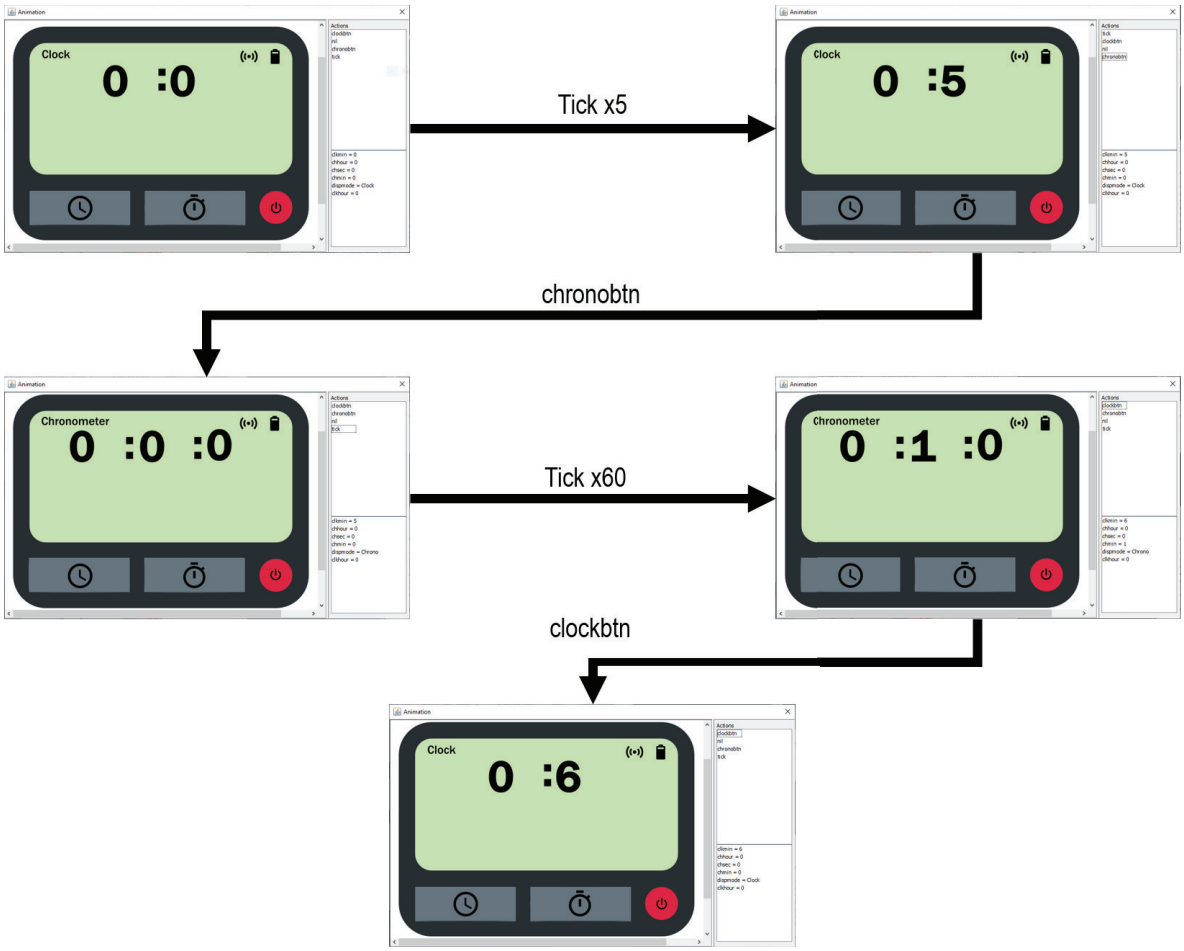


Figura 27: Exemplo de uma sequência de animação do relógio

dois *triggers* disponíveis (clique do rato e pressionar de uma tecla) e associação de atributos a ecrãs do protótipo a elementos textuais do mesmo.

5.4 ANÁLISE

Com os exemplos de aplicação descritos foi possível demonstrar o uso do "Prototyper" em casos de uso reais, com graus de complexidade variáveis. A análise dos modos como as ligações entre os componentes do modelo descrito no IVY com os elementos interativos e textuais do *mockup* se efetuam ilustra as capacidades e limitações que existem na definição e representação do protótipo.

Sendo possível a construção de protótipos com modelos de IVY complexos, o foco passa para o modo como o comportamento dos mesmos se pode definir. A transição entre ecrãs e alteração de componentes textuais no *mockup* satisfazem a necessidade de representação gráfica dos modelos, mas correspondem a maneiras básicas de a implementar. Tomando

como exemplo a bomba de infusão, foi necessário definir duas camadas para o *mockup* para representar os estados de desligado e ligado. A única diferença entre estes era a cor do ecrã e a ausência, ou não, do valor textual indicativo da quantidade de líquido a infundir. Considerando as capacidades do formato que descreve o *mockup*, o SVG, deve-se considerar acrescentar a futuras iterações do *plugin* mais maneiras de representar os atributos.

6

CONCLUSÃO

No início deste projeto foi identificada uma lacuna na comunicação do comportamento dos modelos especificados no IVY Workbench a utilizadores fora do domínio da análise formal de interfaces. Para mitigar este problema, foi desenvolvido um novo *plugin*, que permite a representação gráfica dos modelos e do seu comportamento, através de protótipos.

Neste documento foi descrito o *plugin* desenvolvido, bem como a sua concepção e desenvolvimento. O trabalho realizado deu ainda origem a um artigo aceite na conferência ICGI 2019 (*International Conference on Graphics and Interaction*) (Araújo et al., 2019).

6.1 RESUMO

O documento iniciou-se com uma descrição do próprio IVY Workbench para contextualizar a necessidade de representação dos sistemas modelados face às funcionalidades em oferta. Deu-se especial atenção ao AniMAL, que permite ao utilizador decidir os passos a tomar durante a execução. Este conceito assemelha-se ao modo de execução esperado da funcionalidade planeada para o novo *plugin* de prototipagem. No entanto, este apresentará uma vista sobre o modelo mais próxima do que será a interface final.

Para recolher requisitos para o novo *plugin* e perceber de que maneira é que a interação entre o modelo e a sua representação gráfica se iria manifestar, foram analisadas ferramentas de modelação de sistemas concorrentes ao IVY com uma componente gráfica. Esta análise permitiu distinguir dois métodos para representar os componentes dos modelos e a elaboração de um conjunto de requisitos funcionais.

A nova funcionalidade deveria permitir a importação de um ou mais ficheiros descritivos do *mockup* do sistema, modelação e análise de sistemas interactivos, tal como o IVY, mas em que a componente de representação gráfica da interface assume um papel relevante. Foram analisadas duas ferramentas: o CogTool e o PVSio-web, para fins de levantamento de requisitos.

A especificação dos requisitos instigou a necessidade de detalhar os programas que poderão ser usados para desenhar o *mockup*, o tipo de formato que o iria descrever e que

bibliotecas se utizariam para a importação e manipulação do ficheiro. Conclui-se que a ferramenta Pencil Project oferecia maior flexibilidade na conceção dos *mockups* e o formato SVG exportado foi considerado como o mais adequado para descrevê-los por apresentar um formato normalizado que pode ser gerado ou lido por outros programas, graças à sua especificação aberta e disponível a todos.

Com os requisitos e a estrutura do *mockup* definidos, prosseguiu-se para uma descrição do próprio *plugin*, entitulado "Prototyper", da sua arquitetura e das funcionalidades implementadas na versão final. O código foi estruturado de modo a haver uma separação clara entre a lógica da interface e a lógica de fundo, sendo esta última responsável pelo processamento dos SVGs importados e dados relacionados.

Por último, foram apresentados 3 casos de estudo: um portão de uma garagem, uma bomba de infusão médica e um relógio digital multi-funções. Os exemplos permitiram melhor demonstrar as funcionalidades em oferta do "Prototyper" e como a animação dos protótipos configurados se podia manifestar.

6.2 Análise

Após uma análise de todo o trabalho efetuado e descrito neste documento, considera-se que o produto final cumpre os objetivos inicialmente definidos. A representação gráfica do comportamento dos modelos possibilita a comunicação da especificação destes a pessoas não peritas em técnicas de modelação e análise formais. Os protótipos criados permitem a análise do comportamento do sistema por mais pessoas e, consequentemente, a obtenção de mais *feedback*. Este *feedback* permite iterar sobre os *designs* existentes e atingir o *design* final do sistema mais rapidamente.

Detalham-se, agora, os requisitos inicialmente definidos, com uma explicação sobre como foram implementados:

- Requisito 1 - O *plugin* deverá suportar a importação de um ou mais ficheiros que descrevam o protótipo da interface desejada.

O "Prototyper" suporta a importação de múltiplos ficheiros SVG, como descrito na secção 4.3. Adicionalmente, o SVG permite a definição de camadas num só ficheiro, incorporando a capacidade de representar os múltiplos estados do sistema num só ficheiro.

- Requisito 2 - O ficheiro a importar deve estar num formato aberto de livre utilização.

O "Prototyper" suporta a importação de ficheiros SVG. O SVG é um formato de imagens vetorial baseado em XML. Descreve gráficos a duas dimensões, incluindo formas geométricas, texto e gradientes. É um formato aberto desenvolvido pela W3C, pelo que cumpre o requisito descrito.

Um detalhe que deve ser esclarecido é que, apesar do *plugin* suportar a importação de ficheiros SVG, depende da estrutura gerada pelo Pencil. Significa isto que algumas funcionalidades essenciais, como a identificação das áreas interativas, não estão garantidas com SVG gerados por outros programas. Esta limitação é algo contrária às intenções do requisito descrito, mas planeia-se em futuras revisões do *plugin* melhorar o suporte de ficheiros SVG genéricos.

- Requisito 3 - Deverá ser possível a identificação/especificação de áreas de interatividade dentro do *mockup* importado;

O "Prototyper", no momento da importação do SVG, identifica as áreas interativas que o utilizador pode configurar. O *plugin* identifica certas formas no ficheiro que podem ser associadas a elementos de interface, como botões ou *checkboxes*. Neste momento não há suporte para a definição destas áreas manualmente por parte do utilizador, sendo esta uma funcionalidade a adicionar em trabalho futuro.

- Requisito 4 - Deverá ser possível mapear essas mesmas áreas para atributos e ações do modelo;

O "Prototyper" suporta dois tipos de áreas, interativas e textuais, que podem ser associadas a ações e atributos do modelo, respetivamente. Esta funcionalidade é descrita na secção 4.3 e nos exemplos de aplicação apresentados no capítulo 5.

- Requisito 5 - O *plugin* deverá suportar vários tipos de interações com os elementos interativos do protótipo;

É possível definir dois tipos de *trigger* para as áreas interativas: clique do rato ou pressionar de uma tecla.

- Requisito 6 - Deverá ser possível animar o protótipo para visualizar o comportamento do sistema modelado;

Após efetuar o mapeamento entre o protótipo e o modelo descrito no IVY, é possível iniciar a animação.

- Requisito 7 - Deverá ser possível guardar o estado do animador e posteriormente restaurá-lo e resumir o trabalho.

O "Prototyper" oferece a opção de guardar e restaurar o trabalho efetuado. Utiliza um ficheiro com a extensão ".pcf" que alberga informação sobre os SVG importados e o mapeamento efetuado.

Um aspeto que pode ser questionado é o limitado leque de opções, no que toca à configuração do protótipo. Neste momento, apenas se podem definir dois tipos de *trigger* para as ações associadas: clicar com o rato, ou premir uma tecla; e dois modos de mapear os atributos do modelo para o protótipo: associá-los a camadas dos ficheiros SVG, para definir que camada deve ser apresentada em cada momento, ou a elementos textuais de uma dada camada, para que o seus valores sejam representados nestes. No entanto, ficaram lançadas as bases para a evolução da abordagem para possibilidades de configuração mais avançadas.

Das duas ferramentas inicialmente analisadas, o PVSio-web é aquela cujas características mais se aproximam das do IVY. Poder-se-ia efetuar a comparação do "Prototyper" com o PVSio-Web e a sua forma de definir e interagir com os protótipos, através das áreas definidas numa imagem *raster* do sistema. No entanto, argumenta-se que mais que replicar ou competir com as funcionalidade de outra ferramenta, o trabalho focou-se em adicionar uma funcionalidade inexistente ao IVY. O "Prototyper", como está implementado, encontra-se aberto à adição de funcionalidades que permitirão novos métodos de definir a animação do protótipo. A escolha do formato SVG possibilita a futura integração de funcionalidades mais avançadas que este suporta, como a especificação de animações e *scripting*.

6.3 Trabalho Futuro

Como já foi mencionada na secção anterior, o trabalho futuro irá focar-se em adicionar às funcionalidades base do "Prototyper" descritas neste documento. Estas poderão incluir: a adição de novas formas de visualização dos valores dos atributos (alteração da cor dos elementos, parâmetros de visibilidade, ...), suporte para *widgets* que permitirão a identificação dos tipos de elementos no SVG do *mockup* e a automatização do mapeamento destes com atributos e ações do modelo, e suporte para animações descritas no próprio SVG. Adicionalmente, pretende-se suportar o SVG de uma forma mais genérica para remover a dependência da estrutura com que o Pencil Project exporta, e melhor cumprir os requisitos definidos.

Estas futuras adições ao *plugin* deverão permitir uma especificação mais rica do protótipo, facilidade de uso da funcionalidade e uma capacidade de comunicação do comportamento dos sistemas melhorada para facilitar a comunicação a um maior número de pessoas do *design* do mesmo.

BIBLIOGRAFIA

J. Araújo, R. Couto, and J.C. Campos. Gerador de protótipos de interfaces gráficas para o IVY workbench. In *ICGI 2019 – International Conference on Graphics and Interaction*. IEEE, 2019. aceite para publicação.

blackears. Svg salamander, 2019. URL https://github.com/blackears/svgSalamander.

J. C. Campos and M. D. Harrison. Interaction engineering using the ivy tool. In *ACM Symposium on Engineering Interactive Computing Systems (EICS 2009)*, pages 35–44, New York, NY, USA, 2009. ACM. doi: 10.1145/1570433.1570442.

J.C. Campos. High assurance interactive computing systems. In J. Ziegler, J. C. Campos, and L. Nigay, editors, *HCI Engineering: Charting the Way towards Methods and Tools for Advanced Interactive Systems*, pages 39–42, 2014.

José C. Campos and Michael D. Harrison. Model checking interactor specifications. *Automated Software Engineering*, 8(3):275–310, Aug 2001. ISSN 1573-7535. doi: 10.1023/A:1011265604021. URL https://doi.org/10.1023/A:1011265604021.

Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, 23(7):396–410, July 1980. ISSN 0001-0782. doi: 10.1145/358886.358895. URL http://doi.acm.org/10.1145/358886.358895.

Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An open-source 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.

E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pages 117–126. ACM, 12 1983. ISBN 0897910907.

Richard M. Cohn, David Dodds, Andrew Donoho, David A. Duce, J. Paul O. Evans, Jon Ferraiolo, S. Furman, Peter Graffagnino, Rick Graham, Lofton Henderson, Alan Hester, Bob Hopgood, Christophe Jolif, K. Faith Lawrence, Chris Lilley, Philip Andrew Mansfield,

Kevin McCluskey, Tuan Minh Nguyen, Troy Sandal, Peter Santangeli, Haroon I. Sheikh, Robert Stevahn, and Shenxue Zhou. Scalable vector graphics svg 1.0 specification. 2000.

R. Couto and J.C. Campos. IVY 2 - a model-based analysis tool. In *The 11th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS 2019*, pages 5:1–5:6. ACM, 2019. doi: 10.1145/3319499.3328228.

Jarek Foksa Daniel Holbert Paul LeBeau Robert Longson Henri Manson Ms2ger Kari Pihkala Philip Rogers David Zbarsky David Dailey, Eric Eastwood. Scalable vector graphics (svg) 2, 2018. URL https://www.w3.org/TR/2018/CR-SVG2-20181004/.

Evolus. Collection properties, 2012. URL https://pencil.evolus.vn/wiki/devguide/Tutorial/Collection_properties.html.

The Apache Software Foundation. Apache batik svg toolkit, 2016. URL https://xmlgraphics.apache.org/batik/.

Google. Material icons guide, 2019. URL https://google.github.io/material-design-icons/.

Bonnie E. John. *CogTool User Guide*. IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne NY 10532, 2 edition, 5 2012.

Kmseteam. File:suuntoambit2.jpg, 2013. URL https://commons.wikimedia.org/wiki/File:SuuntoAmbit2.JPG. Licença: CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0).

Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, August 1988. ISSN 0896-8438. URL http://dl.acm.org/citation.cfm?id=50757.50759.

Object Refinery Limited. About jfreesvg, 2019. URL http://www.jfree.org/jfreesvg/.

Eve Maler, Jean Paoli, C. M. Sperberg-McQueen, François Yergeau, and Tim Bray. Extensible markup language (XML) 1.0 (third edition). first edition of a recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-xml-20040204.

César A. Muñoz. Rapid prototyping in PVS. Technical Report NASA/CR-2003-212418 / NIA Report No. 2003-03, 2003.

Patrick Oladimeji, Paolo Masci, Paul Curzon, and Harold Thimbleby. PVSio-web: a tool for rapid prototyping device user interfaces in PVS. *Electronic Communications of the EASST*, 69, 2014. doi: 10.14279/tuj.eceasst.69.963. URL http://dx.doi.org/10.14279/tuj.eceasst.69.963.

S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calver. *PVS System Guide*. SRI International, 333 Ravenswood Avenue, Menlo Park, CA, 94025, 2 edition, 11 2001.

M. Ryan, J. Fiadeiro, and T. Maibaum. Sharing actions and attributes in modal action logic. In *Theoretical Aspects of Computer Software*, volume 526 of *LNCS*, pages 569–593. Springer, 1991. doi: 10.1007/3-540-54415-1_65.