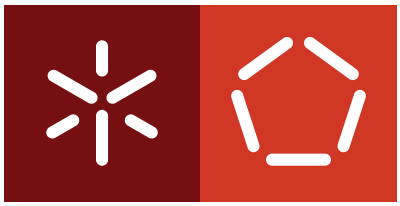


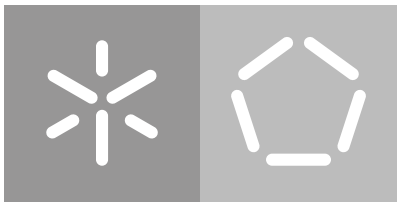
Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Miguel Bonjardim Pinto

Simulação de dispositivos médicos em Android

Simulation of medical devices in Android

Outubro 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Miguel Bonjardim Pinto

Simulação de dispositivos médicos em Android

Simulation of medical devices in Android

Dissertação de Mestrado
Mestrado em Engenharia Informática

Dissertação supervisionada por
José Creissac Campos
Paolo Masci

Outubro 2018

AGRADECIMENTOS

Tentando não me alongar muito nesta secção gostaria de começar por agradecer à Universidade do Minho, como instituição, por todas as condições disponibilizadas ao longo dos últimos 5 anos que me permitiram chegar até este ponto. Queria também agradecer a todos os docentes que contribuíram na minha formação académica ao longo dos últimos 5 anos.

De forma mais individual, começo por agradecer ao meu orientador, Dr. José Creissac Campos da Universidade do Minho, e ao meu co-orientador, Dr. Paolo Masci do INESC TEC, pela orientação desta dissertação, pelas ideias e por toda a ajuda e suporte prestado.

Queria agradecer de forma muito especial aos meus pais, Miguel e Marta Pinto, que ao longo destes 5 anos me proporcionaram todas as condições necessárias para que eu pudesse tirar partido máximo das minhas capacidades ao longo do processo de aprendizagem e pelo apoio incansável.

Ainda de forma muito especial o meu muito obrigado à minha namorada, Cláudia Catarino, por todo o apoio e compreensão em dias menos positivos e por ser uma fonte de motivação e inspiração para mim.

RESUMO

Hoje em dia, em qualquer unidade de saúde (hospitais, centros de saúde, clínicas, etc.), existem diversos equipamentos médicos, cada um com a sua complexidade. Alguns destes equipamentos são cruciais para o tratamento de pacientes e requerem que a interação das equipas médicas com os mesmos seja precisa e exata, correndo-se o risco de, ao mínimo erro, causar danos fatais ao paciente.

Todos os dias, pelo mundo fora, ocorrem erros com dispositivos médicos e de dispositivos médicos. Os erros com dispositivos médicos são causados devido a algum erro de utilização: uma sequência errada de botões, informação inserida em campos errados, falta de atenção às unidades de medida, etc. Em alguns casos, estes erros ocorrem devido à falta de interação de um profissional com certo dispositivo. Esta interação, pode ser reduzida, muitas vezes, devido a um dispositivo ter um custo elevado que não é justificável para um dispositivo disponível somente para testes. Assim, as equipas médicas apenas interagem com o dispositivo quando um paciente precisa de cuidados médicos, o que pode ser perigoso caso um médico ou enfermeiro não esteja familiarizado(a) com o dispositivo.

Por outro lado, os erros de dispositivos médicos são causados por alguma falha no próprio dispositivo, não podendo ser evitada pelas equipas médicas, e que pode dever-se a uma má programação ou construção do dispositivo.

Uma solução, para a prevenção de erros na utilização de dispositivos médicos, passa pela simulação de dispositivos médicos, de modo a que os dispositivos possam ser tanto testados para prevenir eventuais erros de software na interface de utilizador que possam existir, como usados para a formação e treino de pessoal médico para diminuir os erros provocados na interação com os dispositivos.

A presente proposta de dissertação, no âmbito do Mestrado Integrado em Engenharia Informática, visa tirar partido das aplicações móveis para a simulação de dispositivos médicos com que não podemos interagir todos os dias.

Com a criação destas simulações será possível ter inúmeras virtualizações de dispositivos médicos presentes nas unidades de saúde (centros de saúde, enfermarias, hospitais, clínicas, etc.) ao alcance de um *tablet* ou *smartphone*. Isto irá também permitir uma maior mobilidade para simulações, permitindo aos utilizadores levar a cabo um treino diário ou semanal com um dispositivo simulado. Passa também a haver uma maior mobilidade para quem desenvolve e cria protótipos de dispositivos inovadores, sendo que podemos apresentar a aplicação ao público alvo e ter um *feedback* instantâneo sobre os aspetos positivos

e negativos do protótipo, bem como novas ideias para futuras funcionalidades que possam ser adicionadas ao protótipo.

Palavras-chave: dispositivos médicos, simulação, mobilidade, tecnologia, aplicações móveis.

ABSTRACT

Nowadays, in any health unit (hospitals, health centers, clinics, etc.), there are several medical equipment, each one with its complexity. Some of these equipment are crucial for treatment of patients and require that the interaction of medical teams with them is precise and accurate, at the risk of causing fatal injuries to the patient at the slightest error.

Use errors with medical devices can be caused by human error: a wrong sequence of buttons, information entered in wrong fields, lack of attention to units of measurement, etc. Various research studies revealed that these use errors are often caused by design errors in the user interface of the device, rather than negligence or lack of training of clinical practitioners.

Simulation of medical devices is a way to facilitate early detection of latent design errors. The same simulations can also be used for training of end users, to help them identify and avoid latent design problems that might affect a device, while waiting that the manufacturer fixes the problem.

The present dissertation proposal, within the scope of the Integrated Master in Computer Science, aims to take advantage of the mobile applications for the simulation of medical devices with which we cannot interact every day. This interaction at the training level can be reduced, often because a device has a high cost and is not justifiable for a test-only device.

With the creation of these simulations it will be possible to have numerous virtualizations of medical devices present in health facilities within a tablet or smartphone. This will also allow greater mobility for simulations, allowing practitioners to carry out daily or weekly training with a simulated device. It also gives a great advantage for those who develop and create prototypes of innovative devices. They can present the application to the target audience and have instant feedback on the positive and negative aspects of the prototype, as well as new ideas for future functionalities that can be added to the prototype.

Keywords: medical devices, simulation, mobility, technology, mobile applications.

CONTEÚDO

LISTA DE FIGURAS

EXCERTOS DE CÓDIGO

1

INTRODUÇÃO

1.1 DEFINIÇÃO DO PROBLEMA

Apesar de muitas vezes não serem divulgados, os problemas na interaçãao com dispositivos médicos em hospitais, clínicas, centros de saúde, etc. são reais e põem em risco a saúde e segurança dos pacientes, visto estes problemas poderem tanto atrasar a intervencão médica em alguns instantes cruciais como levarem a intervenções incorretas.

Hoje em dia, a formação de equipas médicas é uma atividade de elevada importância na preparação das próximas gerações de médicos e enfermeiros. É, assim, de elevada importância disponibilizar os meios necessários, para possibilitar tal formação.

Apesar da intenção das instituições de saúde em prestar essa formação aos seus funcionários, certos dispositivos médicos, por serem dispendiosos e não muito utilizados, não são comprados em quantidades extra para serem disponibilizados às formações de funcionários. Ou seja, apesar das unidades de saúde possuírem os equipamentos, todos estão em uso ou de prevenção para emergências, de modo a que não é possível libertar nenhum para teste por parte de funcionários em formação.

Uma medida para combater esta indisponibilidade dos equipamentos, passa pela virtualização e prototipagem dos dispositivos. Desta forma, através de um Web site é possível proporcionar a um maior número de funcionários um modelo com o qual podem interagir de modo a ganharem alguma prática com o equipamento. Com a prática é possível evitar ou diminuir os erros de utilização por parte dos funcionários, podendo inclusive aumentar a eficiência das intervenções médicas.

O Instituto ECRI da Pensilvânia [4], uma organização independente sem fins lucrativos que estuda melhorias no atendimento ao paciente, apontou quais os principais perigos do uso de dispositivos médicos do ano de 2012. Estes perigos foram selecionados porque provocaram alguma lesão ou eventual morte de um paciente, porque ocorreram de forma sistemática, porque afetaram um grande número de pacientes ou porque foram amplamente divulgados. Segue-se uma lista de problemas detetados na interação com dispositivos médicos:

- "Fadiga do alarme" – vários dispositivos médicos, como ventiladores ou unidades de diálise, são construídos com o intuito de monitorizarem o paciente e emitirem um alerta para um qualquer perigo que o paciente possa correr. O problema destes alarmes é que as equipas médicas podem perder a sensibilidade de distinguirem diferentes sinais por apresentarem um tom parecido. Esta perda de sensibilidade pode resultar na atuação tardia ou de forma errada por parte das equipas médicas, o que pode trazer graves consequências para o paciente. Uma possível resolução para este tipo de erro, poderia passar por uma maior precisão do dispositivo com a emissão de sons mais distintos que permitisse às equipas médicas uma melhor distinção do sinal ou por uma maior familiarização das equipas com os diferentes sons do dispositivo.

- Perigos de exposição à radiação – técnicas que recorrem ao uso de radiação, como raios-X, quimioterapia, radioterapia, etc. podem ter consequências devastadores para os pacientes caso não sigam os procedimentos corretos. A utilização indevida dos dispositivos emissores de radiação, por parte das equipas médicas pode conduzir a uma excessiva exposição do paciente a radiação. Um caso mediático deste tipo de erros ocorreu na década de 80, quando o *Therac-25*, uma máquina de radioterapia que permitia a aplicação de diversas doses de radiação nos pacientes, provocou vários casos de overdoses[16]. Estes casos ocorreram devido a falhas de software que levaram à aplicação de doses excessivas de radiação, levando mesmo à morte de alguns dos pacientes. Para que tal não aconteça é estritamente necessário que as instituições de saúde mantenham níveis adequados de pessoal e procedimentos de garantia de qualidade. Terá de ser garantido que os funcionários destas unidades médicas seguem procedimentos padrão de tratamento do paciente à medida que vão documentando os mesmos. É também necessário que o pessoal médico esteja familiarizado com os dispositivos e imagens diagnósticas de modo a submeterem o paciente à mínima radiação possível.

- Erros de administração de medicação através de bombas de infusão – erros de informação, informação inserida em campos errados, ou outros erros de inserção de informação podem por em risco a vida de um paciente. Tanto médicos(as) como enfermeiros(as) ou farmacêuticos(as) podem cometer quaisquer destes erros, basta haver uma troca na ordem do medicamento, inclusive as ordens podem ser ilegíveis, ou haver um erro de dosagem na preparação de uma solução, ou até um medicamento ser administrado no paciente errado. Estes erros pode acontecer em dispositivos com diferentes formas de programação das quantidades, isto é, estarem programados com diferentes unidades de medida o que pode levar a uma confusão por parte do utilizador se este não reparar nas unidades de medida que o dispositivo usa. "Uma maneira de evitar erros é usar um sistema de redução de dose-erro", diz James P. Keller, Jr.,

> vice-presidente do Instituto ECRI. "Tem limites definidos e barreiras de segurança. Se uma enfermeira insere uma dose que exceda os limites, será emitido um alerta antes que o erro ocorra".

Um estudo [17] levado a cabo nos Estado Unidos da América e publicado em 2010, revelou alguns números alarmantes que envolvem erros com dispositivos médicos. Estimou-se que cerca de 98.000 pessoas morrem, por ano, devido a falhas médicas envolvendo o uso incorreto de um dispositivo, o que equivale a uma média de 1,5 quedas de aviões de grande porte por dia. Apesar deste elevado número de mortes, apenas 60% da população norte-americana inquirida acredita que os erros médicos causem mais de 5.000 mortes por ano. Foi também estimado que o sistema de saúde americano tem um prejuízo anual de 24 biliões de dólares, distribuídos por indemnizações, reposição de dispositivos, formação de pessoal, etc.

Esta dissertação pretende, então, possibilitar uma medida para a prevenção da ocorrência de erros médicos, através do desenvolvimento de protótipos com que as equipas médicas poderão interagir antes de interagirem com o dispositivo físico. Desta forma, as equipas médicas, através dos protótipos, podem perceber como funciona e como deve ser operado o dispositivo real.

1.2 Protótipos de dispositivos médicos

A plataforma do PVSio-web [21] possui uma coleção de dispositivos para que qualquer utilizador possa interagir com os mesmos. Uma desvantagem desta ferramenta é a sua portabilidade, pois requer a utilização de um computador, visto não ser muito prático para dispositivos móveis como *smartphones* e *tablets*. A utilização de um computador não fornece ao utilizador a mesma experiência que um dispositivo móvel que, devido ao seu tamanho mais reduzido, se assemelha ao dispositivo médico que se pretende simular permitindo uma maior perceção do dispositivo por parte do utilizador.

A pensar nesta reduzida mobilidade por parte dos computadores, idealizou-se uma solução adaptada a dispositivos mais portáteis que um utilizador comum possui hoje em dia: *smartphones* e *tablets Android*. Um estudo efetuado pela IDC [3], referido no segundo capítulo deste documento, mostra que o *Android* é o sistema operativo de dispositivos móveis mais presente no dia-a-dia da população em geral, pelo que é razoável presumir que também terá um elevado número de utilizadores dentro das equipas médicas. Também, a nível de custo, é mais acessível às instituições médicas sendo que apenas alguns dispositivos poderiam bastar para dar formação a um conjunto pessoas, por exemplo. Assim, não seria necessário o dispositivo real estar dedicado apenas a formações, visto que seria substituído pelas simulações do mesmo.

Pretende-se, então, a criação de uma ferramenta que permita a migração dos protótipos desenvolvidos na plataforma do *PVSio-web*[1]. A criação da ferramenta será realizada com base no desenvolvimento de um caso de estudo, que permitirá identificar quais as funcionalidades necessárias na ferramenta a desenvolver. Como caso de estudo irá ser utilizado o dispositivo *Stellant V2*[1], Figura **??**. Este dispositivo é utilizado na preparação de radiografias, mais especificamente nas arteriografias (radiografias ao cérebro), ecografias ou TACs, pelo facto de que permite a injeção de um contraste que aumenta a definição de certas partes do corpo humano nas radiografias, ecografias, etc. Pelo facto deste contraste ser tóxico, é necessário que seja injetada também uma solução salínica que previna os efeitos colaterais do contraste. Este dispositivo, tem como prioridade a saúde do paciente, pelo que a disponibilização deste dispositivo para formação de funcionários se torna de especial imporância.

Figura 1: Dispositivo *Stellant V2*

Sendo assim, o protoótipo presente na plataforma do PVSio-web será portado para uma aplicação Android que irá simular o comportamento do dispositivo. Será então necessário efetuar uma tradução do código que implementa o protótipo do dispositivo na plataforma para a linguagem nativa do *Android*, a linguagem *Java*. Além da tradução deste código, será também necessário desenvolver uma interface que seja semelhante à interface do protótipo com que o utilizador possa interagir, novamente de forma semelhante ao protótipo. Esta interface será composta pela imagem do dispositivo e todos os seus componentes auxiliares,

1 *Site* do PVSio-web: http://www.pvsioweb.org

implementação de botões e respetivas interações do utilizador com esses botões (toque singular, toque duplo, toque longo, etc.) e pelas respetivas animações do funcionamento do dispositivo, desde luzes de botões a movimentos do dispositivo.

Com base na aplicação *Android* construída, será desenvolvida uma *framework JavaScript* que possa ser integrada no ambiente do *PVSio-web*. O desenvolvimento da aplicação *Android* terá um papel importante para o desenvolvimento da *framework*, no sentido em que dará informações cruciais sobre como devem ser construídas as aplicações Android para que o processo de migração dos protótipos possa ser simplificado com recurso à *framework* que se pretende desenvolver.

1.3 OBJETIVOS

O principal objetivo desta dissertação passa pela criação de uma *framework JavaScript* que permita a migração de protótipos da plataforma do *PVSio-web* para aplicações *Android*, de forma simples e intuitiva.

As aplicações resultantes da migração dos protótipos devem simular o comportamento dos respetivos dispositivos físicos como acontece na plataforma do *PVSio-web*. Estas aplicações devem também ser o menos dependente possível de outros serviços, como serviços de *Bluetooth*, GPS ou Internet, a não ser que os mesmos sejam estritamente necessários para o funcionamento correto das simulações. As aplicações não devem também depender de aplicações secundárias.

1.4 CONTRIBUIÇÃO

A contribuição desta dissertação será o desenvolvimento da *framework JavaScript* que posteriormente poderá ser integrada na plataforma do *PVSio-web*. Desta forma o utilizador poderá descarregar diretamente da página do PVSio-web destinada a cada dispositivo, um projeto *Android Studio* de uma aplicação *Android* que irá simular o comportamento do respetivo dispositivo.

Apesar do foco desta dissertação ser a construção de uma *framework JavaScript* para migrar protótipos *PVSio-web* para *Android*, esta dissertação irá abordar aspetos como a construção de aplicações *Android*, as linguagens *PVS*, *Java* e *JavaScript* e a prototipagem de dispositivos médicos.

1.5 ESTRUTURA DO DOCUMENTO

No primeiro capítulo, Introdução, temos uma descrição do projeto e do problema ao qual esta dissertação tentará dar resposta.

No Capítulo 2 é explicado o caso de estudo, com alguns exemplos e estatísticas reais.

O Capítulo 3 irá abordar a construção de uma aplicação *Android* nativa para um protótipo do *Stellant V2*, sendo que no início do capítulo é realizada uma apresentação do dispositivo.

O Capítulo 4 irá descrever o trabalho realizado durante a construção da *framework* e irá apresentar exemplos da utilização da mesma.

Para finalizar, haverá uma secção de conclusão onde será avaliado o trabalho desenvolvido ao longo desta dissertação e onde serão referidos alguns aspetos que poderão ser melhorados.

2

CONTEXTO DO TRABALHO

Ao longo deste capítulo serão abordados temas relacionados com a prototipagem de interfaces de utilizador e com a tecnologia *Android*, sendo que no fim deverá ser possível compreender as vantagens da prototipagem, bem como o porquê de o *Android* ter sido escolhido para a virtualização das simulações.

2.1 PROTOTIPAGEM DE INTERFACE DE UTILIZADOR

A prototipagem de um dispositivo médico não pretende acabar com todos os erros cometidos pelas equipas médicas, mas sim permitir um aprimoramento do treino das equipas médicas na interação com os dispositivos. De igual forma a prototipagem pode também ser útil na fase de desenvolvimento dos dispositivos permitindo que os dispositivos sejam aprimorados à medida que são desenvolvidos, com base nas avaliações dos utilizadores acerca dos protótipos apresentados. A prototipagem pretende então aumentar a usabilidade dos dispositivos. Os protótipos podem ter dois pontos de vista no que toca à sua utilização. Da perspetiva do desenvolvedor os protótipos ajudam a validar a compreensão dos diferentes estados do dispositivo e das funções necessárias para completar um procedimento em segurança. De uma perspetiva clínica, os protótipos ajudam a ganhar confiança com a interface do utilizador e promovem a aprendizagem através da exploração das funcionalidades (o protótipo não está conectado ao paciente por isso podem explorar situações de "e se" em segurança). De um modo geral, a prototipagem permite: maior rapidez na interação com os dispositivos (sendo possível começar a testar desde cedo os protótipos e evitando problemas de interface no dispositivo), maior capacidade de treino e suporte, maior conformidade com os requisitos regulamentares e aumento da satisfação do utilizador. A prototipagem eé uma fase muito importante no desenvolvimento de um dispositivo meédico. Segundo a norma EN 62366-1 [9] aprovada pelo CENELEC (European Committee for Electrotechnical Standardization), a concepção de um protótipo assenta num ciclo de etapas, o que não implica que, dependendo do dispositivo, se possa dar mais atenção a etapas diferentes dependendo das necessidades.

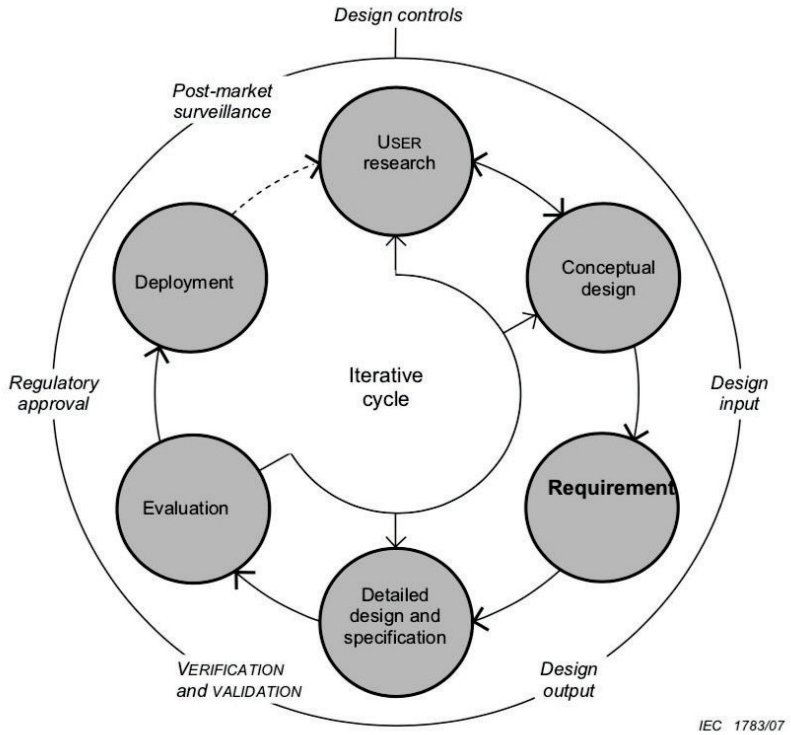


Figura 2: Ciclo de desenvolvimento da Interface do Utilizador - **Fonte:** "Medical devices - Application of usability engineering to medical devices"[9]

A Figura 2 deixa claro as diferentes etapas do desenvolvimento da interface de utilizador. As etapas de *User research* e *Conceptual design* fazem parte da fase de *Design Controls*, que consiste na definição das funções mais usadas, identificação de caraterísticas relacionadas com a segurança e identificação de perigos conhecidos e situações perigosas. A etapa de *Requirement* está integrada da fase de *Design Input* e implica a definição das principais funções operacionais, especificação e definição do plano de validação da utilização. A etapa *Detailed design and specification* faz a ligação entre as etapas de *Design Output* e *Verification and Validation* e consiste no desenvolvimento e implementação da interface de utilizador. E, por fim, a etapa *Evaluation* será dedicada à fase de *Verification and validation* onde se efetua a verificação e validação da utilização da interface e a avaliação de possíveis perigos ou atividades perigosas relacionadas com o uso da interface. Apesar de a Figura 2 apresentar um ciclo, nem sempre as diferentes etapas são totalmente sequenciais. A integração entres diferentes etapas ocorre frequentemente e muitas das vezes de forma concorrente. Isto porque podem ser aperfeiçoados os requisitos do protótipo, ao mesmo tempo que é aperfeiçoado o design do mesmo e que é feita uma primeira verificação do protótipo.

Após o protótipo, ou uma primeira versão do mesmo, ser validada e considerada sem erros, muitas vezes é fornecida uma versão deste protótipo a possíveis utilizadores para

que estes possam oferecer *feedbacks* sobre o que gostavam que fosse alterado no protótipo e as dificuldades que obtiveram durante a sua utilização, para que o protótipo possa ser melhorado antes de ser iniciada a fase de construção do dispositivo.

Este projecto assenta na construção de protótipos. Estes protótipos poderão ser úteis, quer durante a fase de *Detailed design and specification*, quer durante a fase de *Deployment*, tanto para permitir a avaliação do protótipo intermédio de modo evitar erros no dispositivo físico, como para permitir o treino dos utilizadores. Nas etapas anteriores, foram definidos os critérios de utilizador, de modo a que este protótipo e a sua *User Interface* possam ser ajustados conforme as necessidades dos utilizadores. De igual forma, previamente, foram definidas, especificadas e revistas as normas de utilização que o protótipo terá de respeitar, sendo que estas normas podem servir tanto de avaliação do desempenho do dispositivo como para prevenção de situações de perigo para os pacientes e/ou utilizadores.

2.1.1 *Porquê a prototipagem?*

"A *User Interface*, também denominada por UI ou simplesmente interface, é o meio pelo qual um utilizador controla uma aplicação de *software* ou um dispositivo de *hardware*. Uma boa *User Interface* proporciona uma experiência *user-friendly*, permitindo ao utilizador interagir com o software ou com o hardware de forma intuitiva."[20].

A prototipagem é uma tarefa de grande importância e relevo, hoje em dia, pelo simples facto de nos permitir um primeiro contacto com o produto que pretendemos criar. Ao criar um protótipo, é possível avaliarmos uma versão mais próxima da realidade do produto e determinar quais os aspetos que que estão de acordo com o pretendido e quais as partes que precisam de ser alteradas ou descartadas. Neste processo, pode ser possível encontrar falhas que, inicialmente, não eram visíveis.

Por outro lado, é também uma primeira possibilidade para a comercialização do produto final, na medida em que o protótipo pode ser apresentado a potenciais investidores ou parceiros para que estes possam também ficar com uma ideia dos benefícios que podiam ter ao usufruir do nosso produto. Sem um protótipo, o produto é apenas um conceito e pode ser difícil conseguir que um potencial cliente se comprometa com a compra de um conceito. Com um protótipo na mão, o conceito torna-se instantaneamente real e é muito mais fácil convencer os potenciais clientes.

Uma outra vantagem da prototipagem é a possibilidade de análise do processo de produção do dispositivo e ver se as etapas podem ser alteradas, combinadas ou mesmo removidas. Isso não só agiliza a produção, mas também minimiza o custo da produção real.

2.1.2 *Como prototipar?*

Para uma boa prototipagem há um conjunto de etapas pelas quais um protótipo deve passar. Inicialmente deve ser feita uma reflexão sobre o conceito do protótipo para se criar uma ideia inicial de como o protótipo deverá ser. Posteriormente, começa-se a prototipagem com um protótipo de baixa fidelidade. Quando o protótipo de baixa fidelidade estiver terminado, deve-se analisar o método de utilização do mesmo, de modo a avaliar se o dispositivo construído com base no protótipo será usável ou possuíra algum condicionamento ao seu uso que leve à alteração no protótipo de baixa fidelidade. Quando o protótipo de baixa fidelidade estiver bem definido e sem nenhuma falha aparente, passa-se para o desenvolvimento de um protótipo mais real, de alta fidelidade. Por fim, efetua-se nova revisão, desta vez do protótipo de alta fidelidade, para corrigir algum erro ou mal funcionamento que se possa detetar e efetua-se a documentação de normas e *guidelines* que servirão de apoio ao treino das equipas médicas.

2.1.3 *Protótipos de baixa fidelidade*

Um protótipo de baixa fidelidade, normalmente, assenta em desenhos (Figura 3) ou capturas de ecrã, tipicamente de usar e deitar fora. Devem ser rápidos e fáceis de executar. Este tipo de prototipagem tem um baixo custo associado, sendo também fácil de construir. O aspeto "imperfeito" que o protótipo apresenta, encoraja novas ideias e melhoramentos e/ou alterações.

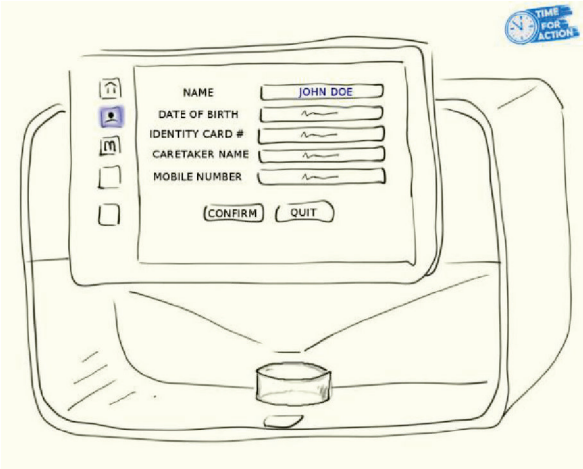


Figura 3: Protótipo de baixa fidelidade (desenho) — **Fonte**:Formal modelling as a component of user centred design[14]

2.1.4 *Protótipos de alta fidelidade*

De forma oposta aos protótipos de baixa fidelidade, os de alta fidelidade estão mais perto do aspeto final da aplicação (Figuras 4 e 5), permitindo programar o comportamento da interface. São tipicamente evolucionários e apelativos para o utilizador. Apresentam um custo elevado, visto que têm de ser desenvolvidos numa tecnologia de desenvolvimento.

Existe também algum perigo de "comprometimento prematuro" sendo que podem ser tomadas decisões demasiado cedo e, após se ter gasto tempo e recursos a desenvolver o protótipo de determinada forma, surge um problema na interface e o protótipo tem de ser alterado, possivelmente perdendo-se todo o progresso entretanto obtido.

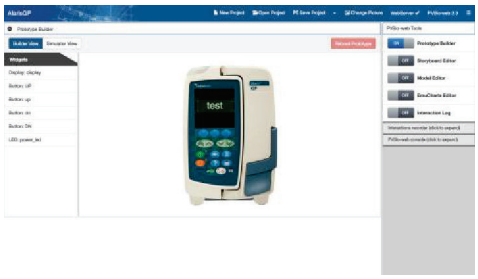


Figura 4: Protótipo de uma bomba de infusão em desenvolvimento - **Fonte:** *PVSio-web*

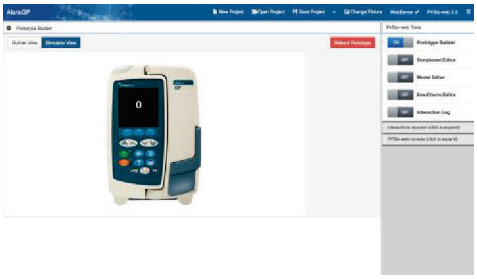


Figura 5: Protótipo de uma bomba de infusão em utilização - **Fonte:** *PVSio-web*

2.1.5 *PVSio-Web*

O *PVSio-web* é um ambiente gráfico que facilita a conceção e avaliação de sistemas interativos. Ao utilizar o *PVSio-web*, é possível gerar e avaliar protótipos interativos de alta fidelidade a partir de modelos formais, Figura 6. O *PVSio-web* foi utilizado com sucesso para analisar dispositivos médicos comerciais, críticos de segurança, para criar material de treino para desenvolvedores e utilizadores de dispositivos e também para o design de dispositivos médicos, tanto por especialistas em métodos formais como por utilizadores finais. O *PVSio-web* está disponível online com uma versão de teste e também fornece a opção de instalação de um servidor local para efeitos de produção.

Um protótipo do *PVSio-web* é composto por: uma componente comportamental, implementada através do *PVS Model*, que define o comportamento do sistema perante a interação de um utilizador (quais os comandos que podem ser executados no estado atual do sistema) e quais as alterações que essa interação irá provocar no estado do modelo, por exemplo, clicar num botão para ligar o dispositivo; e uma componente gráfica, implementada em *Ja-*

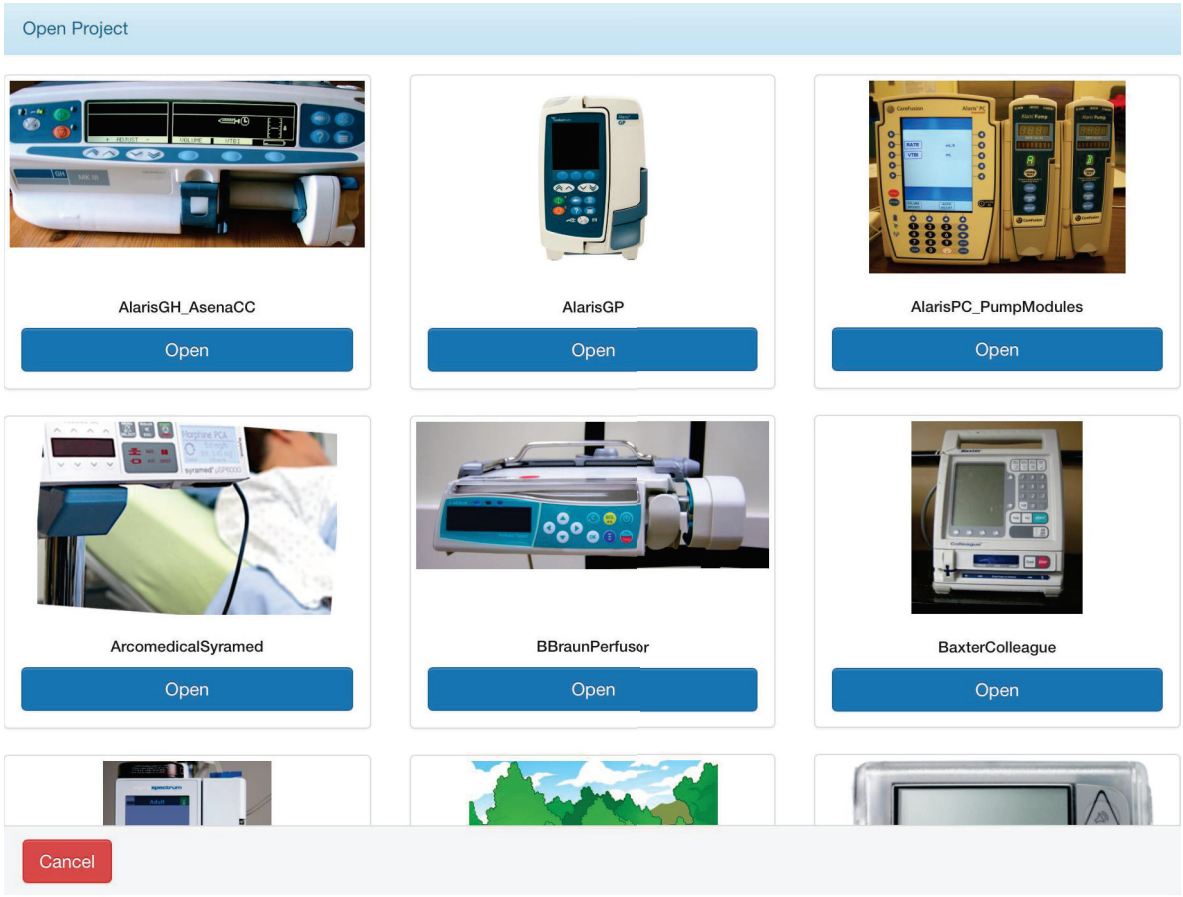


Figura 6: Protótipos disponíveis no site PVSio-web

vaScript ou *HTML5*, que é responsável pela criação da interface do utilizador, como botões, animações, etc (Figura 7).

Convém ainda saber que o *PVSio-web* assenta num modelo *MVC* (*Model-View-Controller*). O *MVC* divide uma determinada aplicação em três partes interligadas. As representações internas de dados são separadas das representações externas, ao utilizador, o que permite uma reutilização eficiente do código, bem como o desenvolvimento paralelo. Como referido, o *MVC* é constituído por três partes: *Model*, *View* e *Controller*. O *Model* contem os dados do protótipo e responde a instruções vindas do *Controller* para alterar o estado ou responder sobre algum aspeto relacionado com o seu estado atual. Define também as regras de interação com o dispositivo. A *View* é a representação da informação que utilizador vê.

Por fim, o *Controller* interliga a *View* e o *Model* e traduz as interações do utilizador com o dispositivo em comandos, remetendo esses comandos para a *View* ou para o *Model*.

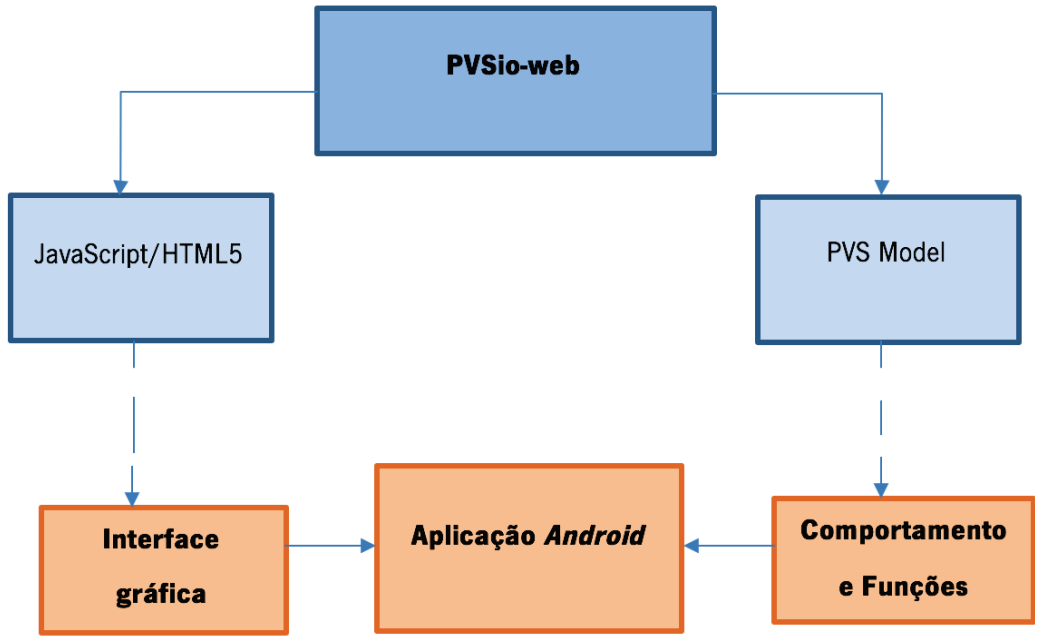


Figura 7: Esquema de adaptação de um protótipo do PVSio-web para uma aplicação *Android*

2.2 PVS

Os protótipos existentes na plataforma do *PVSio-web* funcionam com base num modelo *PVS*. Para que se possa perceber como se comporta o protótipo e se possa adaptar o mesmo comportamento para uma aplicação *Android*, é necessário ter um conhecimento mínimo da linguagem *PVS*.

2.2.1 *O que é o PVS?*

PVS é um sistema de verificação, isto é, uma linguagem de especificação integrada com ferramentas de suporte para provar um teorema. Destina-se a capturar o estado-da-arte em métodos formais mecanizados e a ser suficientemente robusto para que possa ser usado para aplicações significativas. O *PVS* é um protótipo de pesquisa: evolui e melhora à medida que desenvolvemos ou aplicamos novas capacidades.

2.2.2 *Linguagem PVS*

Para este projeto é importante perceber bem a linguagem *PVS*, de modo a que as funções do protótipo, já disponível no site do PVSio-web [21], sejam corretamente adaptadas. O documento *PVS* é iniciado pelo nome da teoria principal seguido da identificação do início da mesma, sendo que no fim deverá existir uma identificação do fim da teoria. Como é possível conferir no excerto 2.1, existem palavras-chave que denotam o início e o fim de uma `THEORY`. Dentro de uma teoria, poderão estar presentes definições de variáveis, funções, novos tipos de dados, etc.

```
1   main :THEORY
2    BEGIN
3
4    state: TYPE = [#
5      is_on: bool
6    #]
7
8    btn_on(st:state): state =
9      st WITH [btn_on = IF btn_on = TRUE
10       THEN FALSE
11       ELSE TRUE
12     ENDIF]
13
14
15   END main
```

Excertos de Código 2.1: Exemplo de início e fim de uma teoria

A criação de novos tipos de dados é composta pelo nome do novo tipo, seguido da identificação `TYPE` e do conjunto de valores que esse tipo pode adotar (`Nome: TYPE = {valor 1, ..., valor N}`). Normalmente, estas definições encontram-se no início do documento PVS onde são definidos diferentes tipos de dados (Excerto 2.2).

```
1  Mode: TYPE = {OFF, INIT, INIT_SYRINGE, INIT_COMPLETE, AUTO, MANUAL,
       READY_TO_PRIME, PRIMING, CONFIRM_PRIME, INFUSING, INFUSION_COMPLETE}
2
3  AutoloadMode: TYPE = {LOAD30, MAKE_EMPTY, FILL_VOLUME, WAIT5, FINALIZE,
       DONE}
4
5  InfuseMode: TYPE = {NIL, READY, INFUSING_CONTRAST, COMPLETE,
       INFUSING_SALINE, PAUSE, STOP}
```

Excertos de Código 2.2: Exemplo de definição de novos tipos de dados

Há ainda um caso de uma definição de um novo tipo de dados que não é abrangida pela fórmula referida acima, o *record type*. Um *record type* é definido sobre a forma de `Nome: TYPE = [# campo`$_1$`: tipo`$_1$`, ..., campo`$_n$`: tipo`$_n$` #]`, onde cada `campo`$_i$ é um campo e cada `tipo`$_i$ é o tipo do referido campo. A variável `state` é um exemplo deste caso, como podemos ver no excerto 2.3.

```
1     state: TYPE = [#
2         mode: Mode,
3         autoload_mode_saline: AutoloadMode,
4         (...)
5         syringe_saline_present: bool,
6         (...)
7         plunger_saline: Volume,
8         (...)
9         display_saline: Display,
10    #]
```

Excertos de Código 2.3: Exemplo de definição de um *record type*

Em PVS, essa definição de constantes é feita através da identificação da constante, seguida do tipo e do valor (`Nome_const: Tipo = valor`). No excerto 2.4, pode-se ver um exemplo da definição de duas constantes no ficheiro *PVS*. Ainda no excerto abaixo, está presente também o exemplo de um comentário no código, iniciada pelo símbolo %.

```
1     MAX_VOLUME: nat = 230 %-- mL
2     VOL_BUFFER: nat = 10 %-- mL
```

Excertos de Código 2.4: Exemplo de definição de variáveis

A linguagem *PVS* permite também a definição de funções, começando-se por definir o nome da função, seguindo-se os argumentos, o tipo do resultado e posteriormente o corpo da função (`Nome_funcao(Argumento1, ..., ArgumentoN) : tipo_resultado = Corpo`). Os argumentos são definidos por nome e tipo (nome: tipo). Por fim, o corpo da função irá ser composto por alterações dos valores de constantes ou varáveis, que podem ser restringidas através de condições e *if clauses* ou invocação de outras funções.

O excerto 2.5, onde estão definidas funções, revela alguns operadores lógicos presentes na linguagem *PVS* como o 'e' lógico representado pelo operador `AND`, o 'não' lógico representado pelo operador `NOT`. Um outro operador presente no exemplo anterior é o operador 'diferente de' representado por `/=`. A sintaxe da linguagem PVS permite ainda a definição de condições. As condições são definidas na seguinte forma: `COND condição`$_1$`->operação`$_1$`,`

..., condição$_n$->operação$_n$, ELSE operação$_{else}$ ENDCOND), sendo que a operação executada depende da condição que se verificar verdadeira, senão será executada a operação$_{else}$.

```
1     per_inc_contrast(st: state): bool = (mode(st) /= OFF AND NOT
          vol_contrast_confirmed(st))
2
3     click_btn_fill_saline(st: (per_btn_fill_saline)): state =
4         COND
5             per_btn_fill_saline(st)
6         -> LET st = st WITH [ vol_saline_confirmed := TRUE,
7                               btn_auto_timeout_saline := -1 ]
8               IN set_LED_state(st),
9         ELSE -> st
10    ENDCOND
```

Excertos de Código 2.5: Exemplo de definição de funções e condições

O operador *LET IN* também está contemplado na linguagem *PVS* e é utilizado da seguinte forma `LET código IN código_principal`. Este operador permite definir variáveis que serão necessárias para a execução de um excerto de código antes da sua execução (`LET x = 2, y = x * x IN x + y`). Por sua vez, operador `WITH` permite a alteração de campos de um *record type*, como é visível do exemplo anterior.

Por fim, resta apenas referenciar que a linguagem *PVS* também permite a definição de *if clauses* que são definidas por: `IF condição THEN código`$_{then}$ `ELSE código`$_{else}$ `ENDIF`, como é visível no excerto 2.6.

```
1     push_plunger_contrast(step: Volume)(st: state): state =
2         st WITH [ plunger_contrast:= IF plunger_contrast(st)-step>0
3             THEN plunger_contrast(st) - step
4             ELSE 0
5         ENDIF ]
```

Excertos de Código 2.6: Exemplo de restrição mediante uma *if clause*

Não foram referidos todos os elementos existentes na linguagem *PVS*, apenas se referiu os elementos mais genéricos e que foram necessários para compreender o ficheiro de código *PVS*, do dispositivo *Stellant V2* implementado na plataforma do *PVSio-web*.

2.3 ANDROID

Com o atual avanço tecnológico existe mais do que uma solução para construirmos uma aplicação. A primeira decisão a ser tomada é para qual sistema será desenvolvida

a aplicação. As possíveis opções passam por vários sistemas utilizados hoje em dia, como o *Android*, o *iOS* ou o *Windows Phone*. Tendo em conta que esta decisão irá influenciar todo o desenvolvimento da aplicação procurou-se saber qual o sistema mais usado e que, por consequência, podia abranger um maior número de utilizadores.

2.3.1 *Porquê Android?*

Um estudo realizado pela IDC (International Data Corporation) [3] mostra que entre 2016 e 2017 o mercado de dispositivos móveis era dominado pelo *Android* com cerca de 85% dos dispositivos mundiais a usar este sistema. O *iOS* surge como o segundo sistema mais usado, abrangendo cerca de 14% dos dispositivos a nível global. Todos os outros sistemas, incluindo *Windows Phone*, *BlackBerry*, etc. representam cerca de 1% dos sistemas usados em dispositivos móveis, Figura 8 e Figura 9. Sendo o Android, de longe, o sistema mais usado em dispositivos móveis, foi decidido que a adaptação dos protótipos do *PVSio-web* a plataformas móveis seria desenvolvida inicialmente para o sistema *Android*.

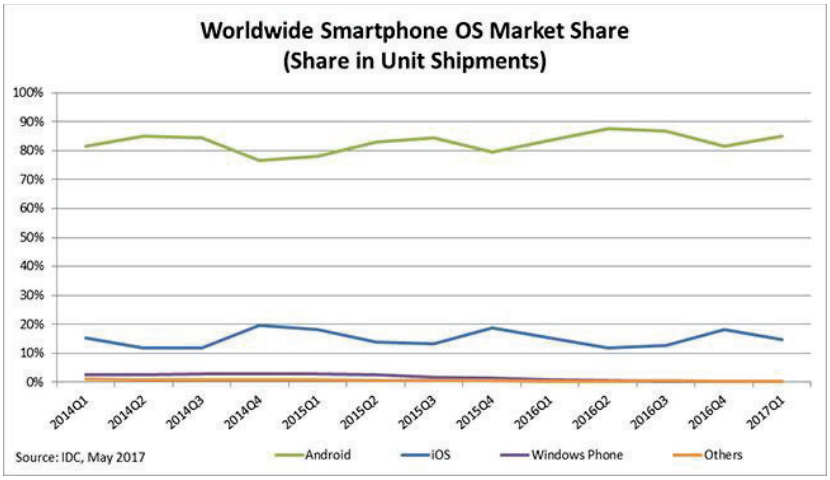


Figura 8: Abrangência do mercado por cada sistema

2.3.2 *Ferramentas existentes*

Antes de se partir para o desenvolvimento da aplicação *Android*, foi realizado um estudo de mercado relativo a soluções já existentes que pudessem agilizar a migração dos

Period	Android	iOS	Windows Phone	Others
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%
2016Q4	81.4%	18.2%	0.2%	0.2%
2017Q1	85.0%	14.7%	0.1%	0.1%

Source: IDC, May 2017

Figura 9: Abrangência pormenorizada do mercado por cada sistema

protótipos existentes na plataforma do *PVSio-web*, de modo a que não fosse necessário estar a desenvolver de raiz uma nova ferramenta.

Apesar de existirem bastantes soluções a nível de prototipagem, como o *Pencil*[1] ou o *Sketch*[2], nenhuma destas ferramentas permite o desenvolvimento funcional dos protótipos. Existem também outras soluções para o desenvolvimento de aplicações *Android* como o *Appy Pie*[3] ou a plataforma da *Outsystems*[4]. Estas aplicações poderiam ser utilizadas na etapa de construção da interface de utilizador, mas não serviriam de muito nas etapas de tradução do código *PVS* nem na definição das animações da aplicação *Android*.

Uma ferramenta que poderia ser bastante útil para a construção de uma aplicação *Android*, seria uma ferramenta que automatizasse ou, pelo menos, agilizasse a tradução do código *PVS* para *Java*, para que não houvesse necessidade de ser o utilizador a encarregar-se dessa tradução. No entanto, não foi encontrada nenhuma ferramenta que conseguisse responder a esta necessidade.

Em suma, existem soluções que poderiam ser utilizadas em determinadas etapas do desenvolvimento de protótipos de dispositivos médicos, mas sem que exista uma única solução que possa dar resposta ao problema abordado por esta dissertação.

2.3.3 *Programação em Android*

Para a construção da aplicação *Android* foi decidido usar um IDE. Um IDE permite aumentar a produtividade dos desenvolvedores e agiliza os processos de compilação e construção do executável final. Hoje em dia, existem diversas soluções no que toca aos IDE para de-

1 Pencil: https://pencil.evolus.vn
2 Sketch: https://www.sketchapp.com/
3 Appy Pie: https://www.appypie.com/
4 https://www.outsystems.com/platform/

senvolvimento em *Android*. Para escolher qual o IDE a utilizar primeiro é necessário tomar conhecimento das ofertas existentes no mercado [18]:

- Android Studio – é o IDE oficial para *Android*. Suporta as principais linguagens de programação para *Android* (*Java*, *C++* e *Katlin*) e realiza todas as tarefas necessárias à organização e gestão do processo de desenvolvimento, desde a criação da estrutura de pastas e ficheiros do projecto até à geração dos ficheiros APK. Incorpora o *Software Developer Kit* (*SDK*), mas não inclui o *Java Developer Kit* (*JDK*).

- Eclipse – o *Eclipse* tem uma origem prévia ao *Android Studio*, e eé bastante semelhante ao seu sucessor. Incorpora suporte para várias linguagens, mas ao contrário do *Android Studio* já não é suportado pela *Google* e, como tal, não incorpora o *Software Developer Kit* (*SDK*).

- Visual Studio com Xamarin - o *Visual Studio* é o IDE da *Microsoft* que suporta uma variedade de linguagens, como *C#*, *VB.net*, *JavaScript* e mais algumas utilizando extensões incorporáveis. Com o *Xamarin*, que agora vem incluído no *Visual Studio*, também é possível criar aplicações multi-plataforma usando *C#* e depois testar em vários dispositivos conectados à *cloud*. É de uso gratuito e uma boa escolha se tencionarmos criar uma aplicação de utilização em *Android* e *iOS*, sem termos de escrever o código duas vezes. Como desvantagem, tem alguns problemas em usar as bibliotecas *Java*, pelo que se perde alguns recursos valiosos.

- AIDE ou Android IDE – o fator diferenciador deste IDE é ser executado no próprio sistema *Android*. Isso significa que podemos criar aplicações utilizando *smartphones* ou *tablets* e, em seguida, testar as aplicações diretamente nesse mesmo dispositivo. Apesar de não possuir os recursos tão desenvolvidos como o *Android Studio*, possui a vantagem de podermos testar o coódigo que estamos a desenvolver em tempo real.

- B4A ou Basic for Android - é uma ferramenta de desenvolvimento de *Android* menos conhecida, desenvolvida pela *Anywhere Software*, e focada no desenvolvimento rápido de aplicações (RAD). Como o nome sugere, este é um IDE permite aos desenvolvedores criar aplicações através da linguagem de programação *BASIC*. Inclui também um editor que permite organizar a interface de utilizador das aplicações.

Com base nesta pesquisa, foi decidido optar pelo uso do *Android Studio*, tanto pelo facto de ser a ferramenta com maior suporte neste momento, como pelo facto haver alguma experiência no uso deste IDE.

2.3.4 *Android Studio*

Para o desenvolvimento da aplicação *Android* que irá simular o comportamento do dispositivo *Stellant V2*, foi decidido recorrer-se ao *Android Studio*.

De uma forma geral, o Android Studio é um IDE que tanto permite o desenvolvimento da aplicação como permite testar a mesma aplicação através da emulação de um sistema *Android*. Baseado no *IntelliJ IDEA*[5], fornece o editor de código e as ferramentas de desenvolvedor avançadas, semelhantes ao *IntelliJ IDEA*, ao que acresce um compilador baseado no *Gradle*[6]. Este IDE possibilita a compatibilidade entre várias linguagens, além do *Java*, e permite o desenvolvimento das aplicações para todas as versões de dispositivos *Android*, quer sejam *smartphones* ou *tablets*. A parte da emulação fica a cargo do *AVD Manager*, que permite criar e executar *AVD*s (*Android Virtual Devices*) onde a nossa aplicação pode ser testada de forma eficaz. O *AVD Manager* permite que sejam criados vários *AVD*s, de modo a que a aplicação que se pretende desenvolver possa ser testada em todos os ambientes que deverão suportar a aplicação a ser desenvolvida. Caso seja pretendido, o *Android Studio* incorpora o *Android Debug Bridge* que permite a instalação direta da aplicação num *smartphone* ou *tablet* conectado ao computador via USB e acompanha o processo de utilização permitindo ao utilizador efetuar o *debug* da aplicação que está a correr no dispositivo *Android*.

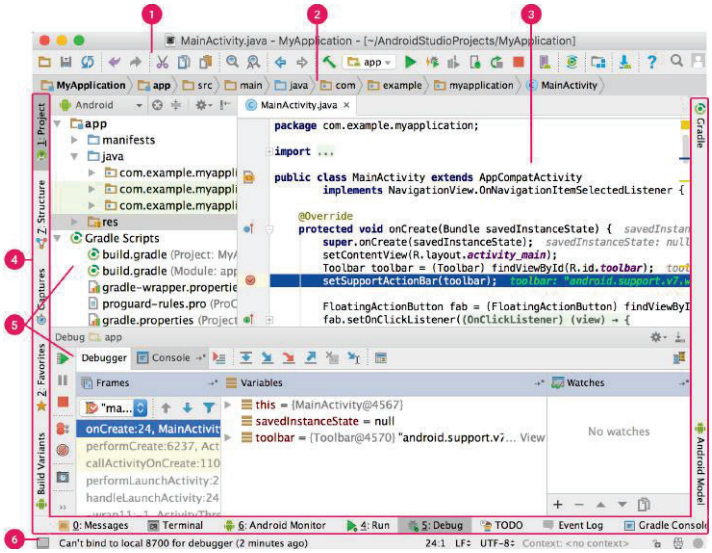


Figura 10: A janela principal do *Android Studio*

A interface de utilizador do *Android Studio*, Figura 10, engloba um editor de código, onde se cria e modifica código da aplicação, podendo alterar para um editor de *layout* ao visualizar um ficheiro de *layout*, uma janela de gestão do projeto e uma janela de ferramentas onde se poderá efetuar o acompanhamento do *debugging* da aplicação, bem como encontrar os erros detetados pelo Android Studio durante a compilação da aplicação *Android*. A interface possui ainda uma barra de ferramentas e uma barra de navegação, onde é possível

5 IntelliJ IDEA: https://www.jetbrains.com/idea/
6 Gradle: https://gradle.org/

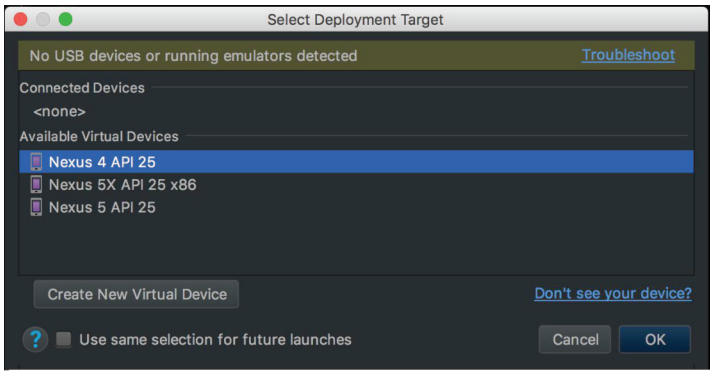


Figura 11: Janela de seleção de tipo de dispositivo do *AVD Manager*

executar diversas ações, incluindo executar aplicações e inicializar ferramentas do *Android* e onde é possível navegar pelas diferentes visões do nosso projeto, respetivamente.

Para iniciar o *AVD Manager* com uma emulação da aplicação a ser desenvolvida o utilizador precisa de clicar no botão '*Run*' na barra de tarefas (botão verde semelhante a uma seta de *play*) e definir o tipo de dispositivo que pretende iniciar com a aplicação desenvolvida, como se pode ser na Figura 11.

2.4 WORKFLOW

Neste ponto desta dissertação, já existe um trabalho de pesquisa efetuado sobre como um protótipo de um dispositivo médico deve ser construído e quais as vantagens desta prototipagem. Este trabalho de pesquisa permite estar haver uma consciencialização das adversidades que o desenvolvimento desta dissertação poderá acarretar.

Sendo assim, o plano desta dissertação passará agora pelo desenvolvimento de um protótipo *Android* que simule o desenvolvimento do *Stellant V2*, sendo que este desenvolvimento poderá passar por duas fases: uma primeira fase em que a implementação passará pela utilização de tecnologias já existentes, detalhadas posteriormente, e, caso não seja obtida uma solução viável, uma segunda fase em que o desenvolvimento irá passar pelo desenvolvimento integral da aplicação *Android*. Deste modo, tenta-se encontrar o método mais ágil de criar uma simulação *Android* de um determinado dispositivo médico para que depois se possa passar ao desenvolvimento de uma *framework JavaScript* que agilize aindamais este processo para os utilizadores, Figura 12.

2.5 PRIMEIRO PROTÓTIPO ANDROID

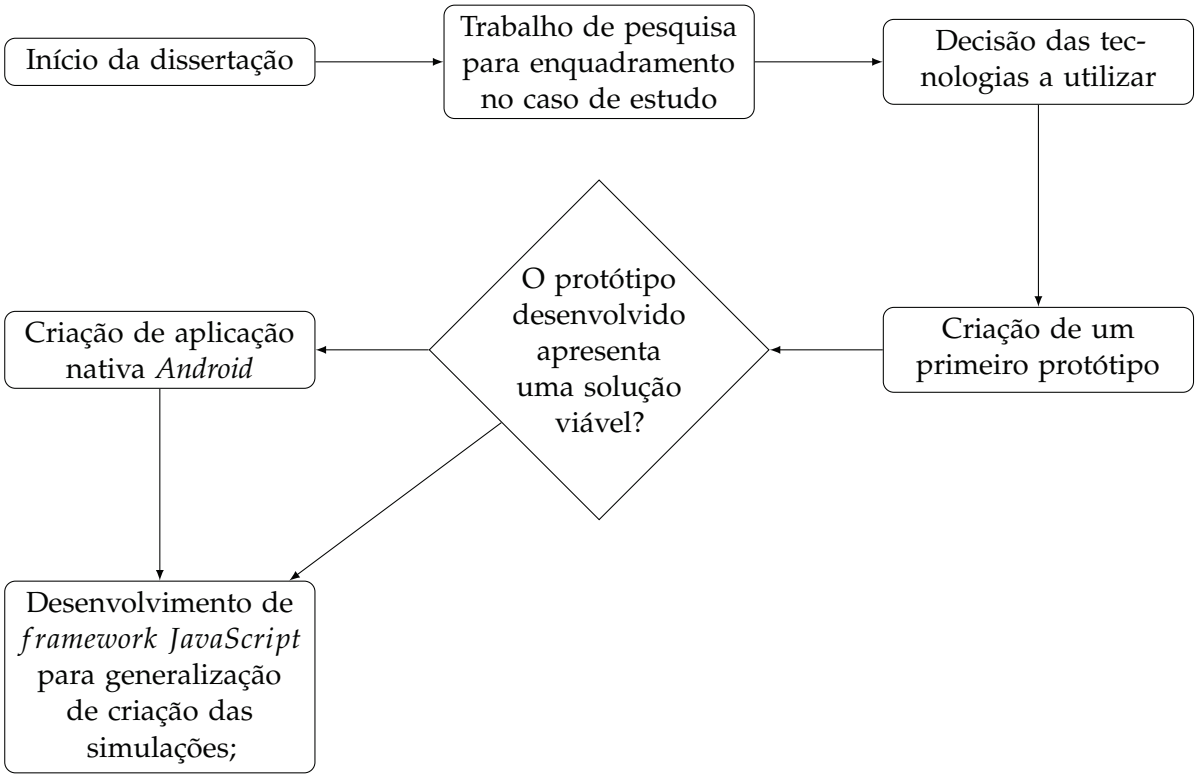


Figura 12: Processo de desenvolvimento da dissertação

De modo a podermos avaliar a adaptação do protoótipo *online* para uma aplicação *Android*, decidiu-se tirar partido das tecnologias disponíveis em *Android* e criar um primeiro protótipo de modo a analisar quais as dificuldades que poderiam existir da utilização do protótipo num *smartphone*.

Para tal, analisaram-se um conjunto de tecnologias desenvolvidas pela *Google*: o *Google Chrome*, as *Google WebView*[11] e as *Google Custom Tabs*[10].

O Google Chrome é um *browser* no qual se aceder à página da plataforma do *PVSio-web* destinada a um determinado protótipo, Figura 13 do lado esquerdo.

As *Google Custom Tabs* passam pelo acesso à mesma página, dentro de uma aplicação *Android*, evitando que o utilizador mude de página acidentalmente, Figura 13 no centro.

As *Google Custom Tabs* e as *Google WebViews* são componentes dos sistemas *Android* baseados no projeto *Chromium* da *Google* que permitem às aplicações apresentar conteúdo *Web*. Estes componentes costumam estar pré-instalados nos dispositivos *Android* e incluem um motor de *JavaScript* para que possam ser mais versáteis e compatíveis para um maior número de *sites*, Figura 13 no centro e do lado direito, respetivamente.

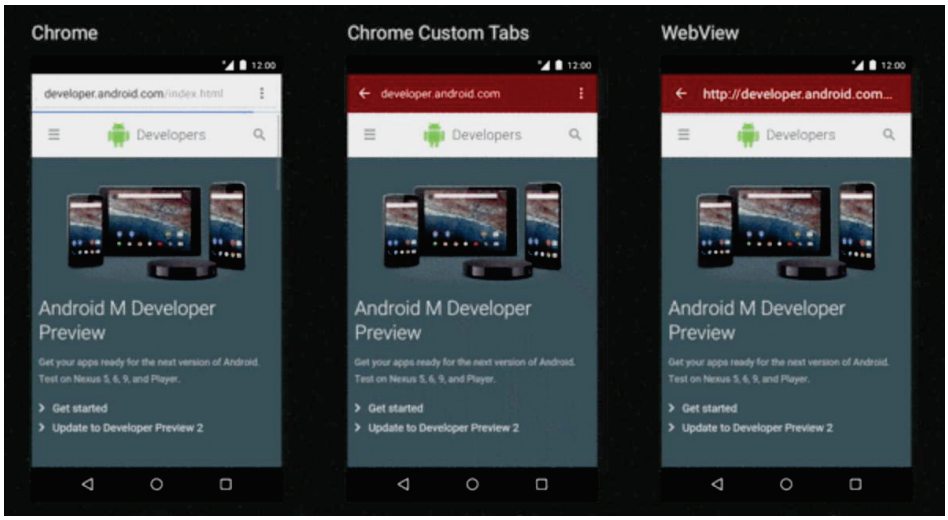


Figura 13: Diferentes alternativas para a implementação do protótipo

A utilização do *Google Chrome* foi de início eliminada das opções visto que obrigava à utilização de uma aplicação externa, o que iria contra os requisitos iniciais da aplicação.

A decisão sobre a tecnologia a utilizar foi então dividida entre as *Google Webview* ou as *Google Custom Tabs*. A grande diferença entre estas duas tecnologias é no controlo sobre as interações do utilizador. Isto é, as *Google WebView* permitem que as interações do utilizador possam ser interceptadas antes de serem executadas. Apesar das diferenças ambas as soluções assentam na apresentação de um determinado *site* com que o utilizador poderá interagir como se estivesse a usar um *browser*. No entanto, as *Google Custom Tabs* são mais direcionadas para casos em que uma aplicação nos redireciona para um URL externo sobre

o qual não possuímos qualquer controlo e não sabemos até que ponto esse URL poderá comprometer a segurança do utilizador.

Neste caso, como há um conhecimento do *site* do *PVSio-web* e este não apresenta risco para a segurança dos utilizadores, optou-se pela solução que permite um maior controlo sobre a simulação: as *Google WebView*.

A utilização das *Google WebView* é bastante simples, ver Excerto 2.7. Começa-se por definir um novo objeto *WebView*. Após a definição deste novo objeto são definidas algumas opções da *WebView*, como a possibilidade desta executar excertos de código *JavaScript*, sendo esta parte bastante importante visto que as simulações dos protótipos na plataforma do *PVSio-web* utilizam código *JavaScript*. Posteriormente, na opção *setUseWideViewPort*, define-se se é pretendido que a *WebView* seja carregada com base nos atributos definidos nos meta-dados da página *Web*, ou seja, se é pretendido que a *WebView* seja escalada como definido pelo *HTML* da página *Web*. Por sua vez, a opção *setLoadWithOverviewMode*, permite definir se é pretendido que a *WebView* seja apresentada sem *zoom*, ou seja, seja apresentada na totalidade no ecrã. Como o ecrã de um smartphone é bastante mais pequeno foi decidido acrescentar uma funcionalidade de *zoom* que permita ao utilizador efetuar uma aproximação da imagem para interagir com os botões mais pequenos. Para tal, ativam-se os *BuiltInZoomControls*, que são os controlos ativos do *Android* para efetuar *zoom in* e *zoom out* através da aproximação ou afastamento dos dedos no ecrã. As restantes funções ativam o suporte das funcionalidades de *zoom* do *Android* e desativam qualquer efeito visual extra que as mesmas impliquem. Por fim, é necessário definir o conteúdo da janela *Android*, que será a *WebView* desenvolvida, e definir qual a página *Web* que será carregada.

```
1     WebView webview = new WebView(this);
2         webview.getSettings().setJavaScriptEnabled(true);
3         webview.getSettings().setUseWideViewPort(true);
4         webview.getSettings().setLoadWithOverviewMode(true);
5         webview.getSettings().setBuiltInZoomControls(true);
6         webview.getSettings().setDisplayZoomControls(false);
7         webview.getSettings().setSupportZoom(true);
8
9         setContentView(webview);
10
11        webview.loadUrl("http://www.pvsioweb.org/demos/stellantV2/");
```

Excertos de Código 2.7: Exemplo de definição de uma *WebView* e das suas funcionalidades

As Figuras 14 e 15, mostram o protótipo *Android* construído através das *Google WebView* em execução. No entanto, esta solução tem a desvantagem de precisar de uma conexão constante a um serviço de Internet, o que vai contra os requisitos da aplicação delineados inicialmente, ver Secção 1.3.

Apesar de, neste protótipo, a única condicionante ser a utilização de um serviço de Internet, poderá haver também condicionantes na utilização de outros serviços como o serviço de *Bluetooth* ou o serviço NFC, impedindo o funcionamento correto de um protótipo mais complexo que possa vir a ser migrado da plataforma do *PVSio-web*. A dependência de um serviço de Internet pode fazer com que a aplicação não consiga ser totalmente carregada, caso a conexão não seja forte o suficiente, ou que a mesma se torne mais lenta. Já para não referir que a aplicação pode também ficar mais lenta pelo facto de a linguagem *PVS*, que é usada para definir a lógica de controlo do dispositivo na plataforma do *PVSio-web*, ser uma linguagem mais lenta a nível de processamento por natureza.

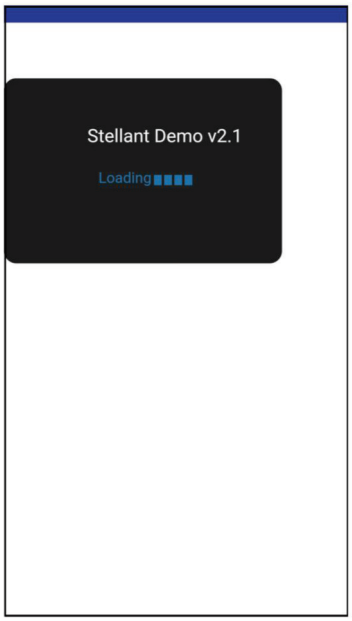


Figura 14: Início da do protótipo do *Stellant V2* através de uma *WebView*

Figura 15: Execução do protótipo do *Stellant V2* através de uma *WebView*

2.6 CONCLUSÕES

A prototipagem visa melhorar a deteção precoce de erros dos dispositivos, neste caso os dispositivos médicos, enquanto que a posterior virtualização poderá ser uma forma de aumentar o tempo de interação com os diversos dispositivos e aumentar a capacidade das equipas médicas em trabalharem com os mesmos. A tecnologia *Android* permite que os

protótipos possam chegar às equipas médicas de forma mais rápida, sendo que os dispositivos *Android* são bastante comuns hoje em dia.

O primeiro protótipo construído no *Android Studio* para o *Stellant V2*, consiste na utilização da tecnologia *WebView* desenvolvida pela *Google* para apresentar a página Web de um determinado *site*, neste caso da página do *PVSio-web* onde podemos interagir com o protótipo do *Stellant V2*.

Apesar deste protótipo ser funcional e apresentar uma simulação bastante próxima do dispositivo físico, tem uma grande condicionante: depende de um serviço de Internet constante.

Ora, sendo que o objetivo das simulações passa por estar disponível para interação em qualquer circunstância, o facto de haver uma dependência de um serviço externo, neste caso um serviço de Internet, pode inviabilizar o uso da simulação. Sendo assim, parte-se para o desenvolvimento de uma aplicação *Android* nativa para o *Stellant V2*, com o intuito de criar uma simulação que se aproxime o máximo possível do dispositivo físico e que não esteja dependente de qualquer serviço externo.

3

STELLANT V2 ANDROID APP

Apesar de o principal objetivo desta dissertação passar pela criação de uma *framework Javascript* que agilize a adaptação de protótipos da plataforma do *PVSio-web* para aplicaões *Android*, é necessário estudar as soluções que o *Android* fornece para a criação de uma aplicação que seja capaz de simular os comportamentos dos protótipos já desenvolvidos. A primeira abordagem, como referido no Capítulo 4, apesar de se apresentar como uma boa solução e que não necessitava de grandes adaptações na migração do protótipo do site do *PVSio-web* para uma aplicação *Android*, tinha a restrição de precisar de uma conexão a um ponto de Internet, o que limitava a mobilidade da aplicação. Numa segunda abordagem, será desenvolvida uma aplicação *Android* totalmente de raíz cujo funcionamento se aproxime o máximo possível do funcionamento do protótipo do *Stellant V2* presente na plataforma do *PVSio-web*.

De modo a facilitar o desenvolvimento deste aplicação, o desenvolvimento foi dividido em 3 etapas (figura **??**): tradução do código de *background*, desenvolvimento da interface do utilizador e, por fim, a aplicação de animações à interface desenvolvida.

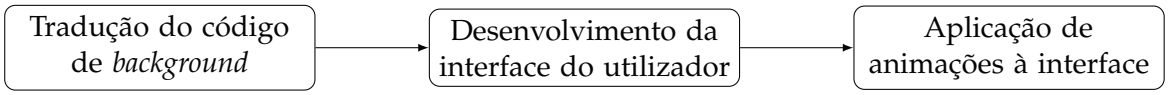


Figura 16: Processo de desenvolvimento da aplicação *Android* nativa

A primeira etapa, consiste na tradução do código *PVS*. Como o dispositivo já se encontra prototipado no *PVSio-web*, irá haver uma reutilização do modelo *PVS* através da sua tradução para *Java*. Este modelo fornece todas as funções que implementam os diversos comportamentos e que poderão ser invocadas ao longo da execução da aplicação.

Na segunda etapa, irão ser definidos todos os botões com que o utilizador poderá interagir.

A última etapa servirá para implementar algumas animações, como movimentos de objetos ou luzes intermitentes, sobre a interface desenvolvida na etapa anterior.

A criação de etapas visa um maior controlo sobre a construção da aplicação, havendo verificações e correções do trabalho desenvolvido em cada etapa.

3.1 SOBRE O DISPOSITIVO STELLANT V2

O trabalho desenvolvido até este ponto não necessitou de uma compreensão a fundo do *Stellant V2* [1], mas, antes de se começar o desenvolvimento da aplicação *Android* que visa simular o comportamento deste dispositivo médico, convém primeiro ter uma noção básica do dispositivo: saber como o dispositivo é constituído, quais as suas utilidades para os serviços de saúde e como funciona ou deve ser operado o mesmo dispositivo.

Só depois destas noções estarem consolidadas é possível passar ao desenvolvimento integral da aplicação *Android*, pois já serão conhecidos os detalhes do dispositivo, bem como as partes que irão necessitar de maior atenção no desenvolvimento da simulação *Android* do dispositivo.

3.1.1 *O que é o Stellant V2?*

O *Stellant V2* é um dispositivo de injeção de contraste indicado para o uso em tomografias computorizadas.

O *Stellant V2* é constituído pelo corpo principal do dispositivo e por uma consola auxiliar, onde é permitido inserir alguns valores para configuração do dispositivo. Para que o dispositivo possa servir o seu propósito possui duas seringas e um cabo de infusão que será, posteriormente, conectado ao paciente para que as soluções presentes nas seringas possam ser injetadas no paciente.

3.1.2 *Para que serve o Stellant V2?*

Uma tomografia computorizada, por vezes, requer que seja administrado um material de contraste ao paciente. Este material de contraste permite que a projeção raios-X obtenha imagens mais nítidas da(s) área(s) que se pretende analisar, quer sejam tecidos, órgão ou vasos sanguíneos.

A natureza do material de contraste é, normalmente, tóxica, pelo que é misturado com uma preparação salínica de modo a que a toxicidade do mesmo atinja níveis não prejudiciais para a saúde do paciente, tornando-se num contraste iodado.

O *Stellant V2* permite que a solução de contraste e a solução salínica sejam misturadas durante o processamento, ou seja, não é necessário haver uma preparação do material de contraste antes da injeção. O dispositivo está configurado para inicialmente remover todo o ar que possa existir no cano da seringa, entre o êmbolo e o orifício da seringa, sugar as quantidades previamente indicadas de ambas as soluções para dentro das seringas e por fim expelir as soluções de forma gradual, para que estas se possam misturar no cabo de infusão que estará ligado ao paciente.

3.1.3 *Como funciona o Stellant V2?*

Qualquer dispositivo médico tem um determinado modo de operação que conduz ao resultado desejado. Como tal o `Stellant V2` também possui um determinado método de operação. O método de operação consiste no conjunto ou sequência de operações/interações realizadas pelo utilizador no dispositivo.

Mas antes de analisarmos o método de operação do *Stellant V2* convém conhecermos o dispositivo primeiro.

O corpo do dispositivo é composto por 13 botões diferentes, como se pode comprovar nas Figuras 17, 18 e 19:

- Os botões *Fill A* e *Fill B* determinam a inicialização do enchimento das seringas com os líquidos devidos a partir de pequenos frascos.

Figura 17: Apresentação dos botões do *Stellant V2* - parte 1 — **Fonte**: https://radiology.bayer.com/products-and-services/ct-x-ray/devices/medrad-stellant

- O botão *Check For Air* confirma que não existem bolsas de ar entre o êmbolo e o orifício da seringa.

- O botão *Arm* bloqueia o dispositivo, preparando-o para que seja inicializada a injeção do material de contraste.
- Os botões imediatamente a seguir aos botões *Fill A* e *Fill B* ajustam o volume que se pretende injetar de cada uma das soluções.
- O botão *Move Piston* permite que o utilizador realize o enchimento das seringas de forma manual, com a ajuda dos botões marcados com setas azuis e verdes.

Figura 18: Apresentação dos botões do *Stellant V2* - parte 2 — **Fonte**: https://radiology.bayer.com/products-and-services/ct-x-ray/devices/medrad-stellant

Figura 19: Apresentação dos botões do *Stellant V2* - parte 3 — **Fonte**: https://radiology.bayer.com/products-and-services/ct-x-ray/devices/medrad-stellant

- O botão *Auto Load* define de forma automática (segundo valores previamente definidos) os volumes das soluções a serem sugados pelas seringas.
- O botão *Prime* permite que o utilizador expila uma quantidade determinada de qualquer uma das soluções de forma manual para o cabo de infusão, com a ajuda dos botões marcados com setas azuis e verdes.
- O botão *Start/Hold* inicia ou pausa a injeção das soluções no paciente.
- O botão *Abort Injection* cancela a injeção das soluções no paciente.

Para além dos controlos presentes no corpo do dispositivo, existe também uma consola auxiliar com 3 outros botões, Figuras 20 e 21:

- Botão *On* que serve para ligar o dispositivo.

- Botão *Block* que serve para bloquear o dispositivo quando este está pronto para a injeção do material de contraste. Este bloqueio impede que o utilizador acidentalmente modifique qualquer das quantidades de liquido existente nas seringas, o que poderia apresentar risco para a saúde do paciente.
- Botão da janela de segurança que aparece na inicialização do dispositivo. O utilizador deve ler os avisos apresentados na janela de segurança e pressionar o botão *Continue*.

Figura 20: Exemplo dos botões *On* e *Block* da consola auxiliar

Figura 21: Exemplo dos botões *On* e *Continue* da consola auxiliar

Além de conhecer os principais botões do dispositivo e da sua consola auxiliar, o utilizador, deve de ter conhecimento sobre o método de operação do dispositivo:

1. Ligar o dispositivo e confirmar o aviso de segurança na consola auxiliar do mesmo;
2. Inserir as seringas nos respetivos orifícios na parte superior do dispositivo;
3. Esperar que o dispositivo retire o ar das seringas;
4. Conectar os frascos com as soluções às respetivas seringas;
5. Pressionar o botão *Auto Load* para preparar o enchimento das seringas;
6. Pressionar os botões *Fill A* e *Fill B* para encher as respetivas seringas. Estes botões não requerem uma ordem específica;

7. Esperar que as seringas acabem de ser enchidas e, de seguida, conectar o cabo de infusão;
8. Pressionar botão *Check For Air*, após confirmar que não existem bolhas de ar dentro das seringas;
9. Bloquear o dispositivo na consola auxiliar;
10. Pressionar o botão *Arm*;
11. Conectar a ponta solta do cabo de infusão ao paciente e pressionar o botão *Start/Hold* para iniciar a injeção.

O protótipo *Android* que irá ser desenvolvido deverá possibilitar a execução de todas as ações referidas, de modo a que possa proporcionar aos utilizadores uma ideia realística de como o protótipo funciona.

3.2 APLICAÇÃO ANDROID

3.2.1 *Tradução do modelo PVS*

A tradução do código *PVS* foi um trabalho longo, durante o qual foram tomadas algumas decisões com o intuito de deixar a aplicação *Android* mais estável, menos propicia a erros e mais fácil de, posteriormente, replicar para outros dispositivos. Abaixo são explicadas as decisões tomadas durante a tradução e, também, alguns dos desafios encontrados ao longo desta tarefa:

- De modo a deixar o código a aplicação menos confuso, e tirando vantagem da linguagem *Java*, foi criada uma classe denominada *State* que, como o nome indica, irá representar o estado de um dispositivo e irá incluir as funções que poderão alterar o mesmo (funções traduzidas do modelo *PVS*).
- Os métodos invocados sobre uma variável do tipo `State` em vez de se alterarem o objecto `State`, criam e devolvem um novo *State*. Esta decisão visa possibilitar a criação de versões, podendo haver uma variável `last_state` que contenha o último estado da aplicação, podendo ser recuperado em caso de falha.
- A definição de novos tipos de dados foi solucionada com recurso aos tipos `Enum`. Assim de forma simples são definidos os vários tipos de dados existentes, bem como as variáveis que cada tipo pode adquirir, como visível no Excerto 3.1.

```
1  public enum Day {
2      SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
3      THURSDAY, FRIDAY, SATURDAY
4  }
```

Excertos de Código 3.1: Exemplo de definição de novos tipos de dados em *Java*

- O código do dispositivo *Stellant V2* incluía algumas variáveis constantes, que serviam de limites para outras variáveis. Essas variáveis constantes foram definidas como `final` de modo a que não fossem permitidas alterações aos valores das mesmas (Excerto 3.2).

```
1  final int MAX_VOLUME = 230 ;
2  final int VOL_BUFFER= 10 ;
```

Excertos de Código 3.2: Exemplo de definição de variáveis constantes em *Java*

- Todas os métodos foram declaradas como `public` para que possam ser invocadas a partir de outros ficheiros *Java* onde estará o código da interface do utilizador

Poderá ser consultado em anexo um conjunto de exemplos que ilustram as principais alterações entre o código *PVS* e o código *Java*.

3.2.2 *Interface de utilizador*

Ao longo desta secção será explicado como foram definidos os diversos elementos da interface do utilizador, desde botões a imagens e *displays*. Também serão apresentados todos os problemas encontrados na definição de cada elemento cada elemento.

3.2.3 *Botões*

A parte principal da interface de utilizador desta aplicação *Android* é a definição dos botões da aplicação. Como tal, é estritamente necessário que os mesmos estejam colocados de forma correta e que executem as devidas funções. O Android Studio fornece uma forma de definir botões que não foi usada, devido a imperfeições entre dispositivos com tamanhos

de ecrã diferentes: um botão que estivesse definido de forma perfeita num ecrã de 5', ficava completamente desfigurado num ecrã de 5.5'. Como podemos ver nas figuras 22 e 23, apesar de haver um ligeiro aumento da imagem no ecrã de 5.5', o aumento do tamanho de botão é desproporcionado em relação ao aumento do tamanho do ecrã, sendo que ambos os botões partilham da mesma definição a nível de posição e tamanho.

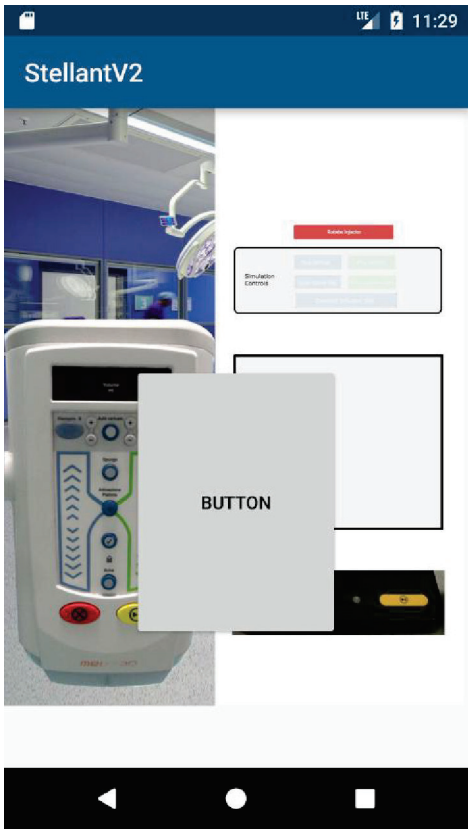


Figura 22: Botão num ecrã de 5'

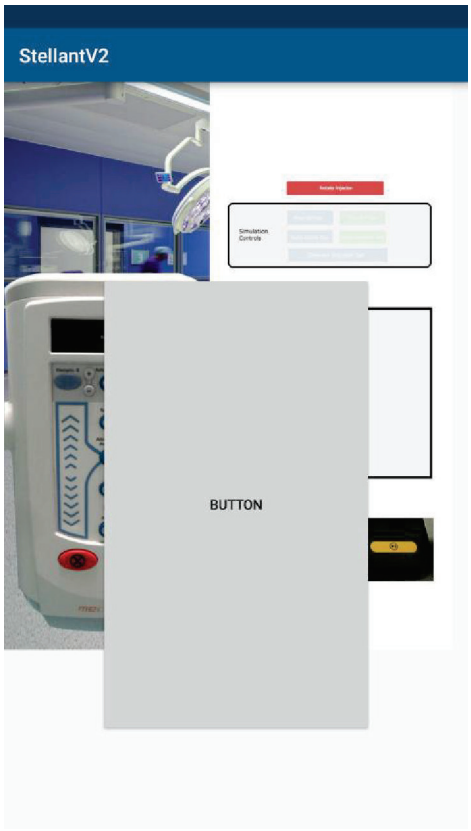


Figura 23: Botão num ecrã de 5.5'

Como esta aplicação *Android* terá sempre uma imagem de fundo (a imagem do dispositivo), optou-se pela utilização da biblioteca *ClickableAreas*[13]. Esta biblioteca *Android* permite a definição de áreas retangulares numa imagem, simulando botões, com as quais se pode interagir através de toques. Sendo esta biblioteca uma solução viável para a definição de botões foi preterida à solução disponibilizada pelo *Android Studio*. As *ClickableAreas* são muito semelhantes aos *Image map* que se podem definir na linguagem *JavaScript*.

Para utilizar esta biblioteca, é preciso criar uma `ClickableAreasImage` que será associada à `ImageView` que contém a imagem de fundo do dispositivo, linha 2 (no Excerto 3.3 a `ImageView` chama-se `background`), criar uma lista de `ClickableArea` onde serão adicionadas todas as áreas que vão ser criadas, linha 5 ,e por fim, associa-se a lista das `ClickableArea` criadas à `ClickableAreasImage` criada previamente, linha 14. Uma `ClickableArea` é definida pelas coordenadas X e Y da posição da imagem no ecrã, por esta ordem, a lar-

gura e altura da imagem, por esta ordem novamente, e por um objecto que é passado como argumento à função que recebe e atua sobre os cliques nas `ClickableArea`, linha 11.

```
1  // Create your image
2    ClickableAreasImage clickableAreasImage = new ClickableAreasImage(new
         PhotoViewAttacher(background), this);
3
4    // Initialize your clickable area list
5    List<ClickableArea> clickableAreas = new ArrayList<>();
6
7    // Define your clickable areas
8    // parameter values (pixels): (x coordinate, y coordinate, width, height)
         and assign an object to it
9
10   //inc/dec saline
11   clickableAreas.add(new ClickableArea(174, 607, 23, 23, (int) 1));
12
13   // Set your clickable areas to the image
14   clickableAreasImage.setClickableAreas(clickableAreas);
```

Excertos de Código 3.3: Exemplo da definição das *ClickableAreas*

A função `onClickableAreaTouched` é executada quando uma `ClickableArea` é pressionada. O objeto `item` corresponde ao objeto que tinha sido definido juntamente com a `ClickableArea`. Mediante o objeto recebido como argumento, a função `onClickableAreaTouched` irá executar determinados excertos de código (Excerto 3.4).

```
1   @Override
2    public void onClickableAreaTouched(Object item) {
3        switch ((int) item){
4            case 1:
5                if(!state.mode.equals(State.Mode.OFF))
6                state = state.press_inc_saline(state);
7                break;
8            case 2:
9                if(!state.mode.equals(State.Mode.OFF))
10               state = state.press_dec_saline(state);
11               break;
12           (...)
13        }
14   }
```

Excertos de Código 3.4: Definição da função `onClickableAreaTouched`

3.2.4 *Imagens*

De forma semelhante ao que acontece com os botões, também a disposição das imagens é afetada pelos diferentes tamanhos de ecrãs. Como a representação do dispositivo é composta por várias imagens (figuras 24, 25, 26 e 27), estas podem ficar desposicionadas devido às diferentes escalas definidas para as margens das diferentes imagens.

Figura 24: Imagem do dispositivo *Stellant V2*

Figura 25: Imagem dos botões da consola externa do dispositivo

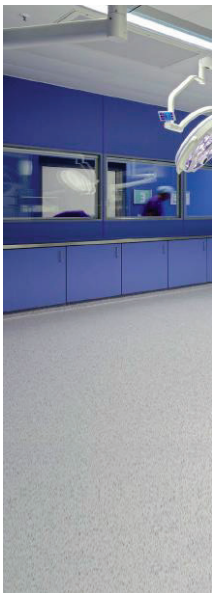


Figura 26: Background a ser usado na aplicação *Android*

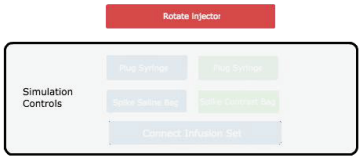


Figura 27: Controlos externos ao dispositivo

De modo a contornar este problema relativo à posição das imagens, foi criada uma função que recebendo um conjunto da imagens e a sua posição relativa, desenha essas imagens,

pela sua ordem, de modo a criar uma nova imagem onde todas as outras estão contidas. Primeiro é necessário criar um `Bitmap` para cada uma das imagens que, posteriormente, serão inseridos num *array* de objetos `Image`. O objeto `Image` foi criado com o propósito de poder agrupar a imagem e a sua posição inicial de desenho (cada imagem tem uma determinada posição em que tem de ser desenhada). De seguida é calculada a largura e altura do ecrã e, caso ultrapassem certos limites, são reajustadas de modo a que a imagem final fique de acordo com o pretendido. Por fim, é invocada a função `combineAllImageIntoOne` à qual são passados como argumentos o *array* de objetos `Image`, e a altura e largura final da imagem, excerto 3.5.

```
1  (...)
2  ImageView background1 = (ImageView) findViewById(R.id.background);
3
4    Bitmap bm1 = BitmapFactory.decodeResource(getResources(), R.drawable.all);
5    Bitmap bm2 = BitmapFactory.decodeResource(getResources(), R.drawable.
         buttons_10);
6    Bitmap bm3 = BitmapFactory.decodeResource(getResources(), R.drawable.
         console_led_off);
7    Bitmap bm4 = BitmapFactory.decodeResource(getResources(), R.drawable.
         empty_console);
8
9    ArrayList<Image> a=new ArrayList<Image>();
10   a.add(new Image(bm1,0,0));
11   a.add(new Image(bm2,0,0));
12   a.add(new Image(bm3,0,0));
13   a.add(new Image(bm4,0,0));
14
15   Double h = 0.0 + background1.getHeight();
16   Double w  = 0.0 + background1.getWidth();
17
18   if (h>1600){h*=0.8;w*=0.77;}
19
20   Bitmap result = combineImageIntoOne(a,h.intValue(),w.intValue());
21               background1.setImageBitmap(result);
22   (...)
```

Excertos de Código 3.5: Exemplo de utilização da função `combineAllImageIntoOne`

Por sua vez, a função `combineAllImageIntoOne`, excerto 3.6, cria uma nova imagem com as medidas pretendidas e desenha as imagens inseridas no *array* passado como argumento, pela ordem em que foram inseridas no mesmo, com a ajuda de um `Canvas`[1]. O *Canvas* é um dos 4 componentes de desenho necessários para desenhar algo em *Android*, neste caso o componente que possui as funções de "desenho"(além do `Canvas` é ainda necessário uma

1 Android Canvas: https://developer.android.com/reference/android/graphics/Canvas

`Bitmap` para guardar os pixels desenhados, uma primitiva de desenho que, neste caso são as imagens a serem desenhadas, e uma variável que define as cores e estilos de desenho, que neste caso é nula por não ser preciso alterações às cores ou estilos das imagens). A função `combinaAllImageIntoOne` também faz um reajustamento da altura e largura do `Canvas`, consoante uma escala pré-definida.

```
1  private Bitmap combineImageIntoOne(ArrayList<Image> bitmap, int height, int
       width) {
2
3          Double a = width * canvasScale;
4          Double b = height * canvasScale;
5
6          Bitmap temp = Bitmap.createBitmap(a.intValue(), b.intValue(), Bitmap.
               Config.ARGB_8888);
7          Canvas canvas = new Canvas(temp);
8          for (int i = 0; i < bitmap.size(); i++) {
9              canvas.drawBitmap(bitmap.get(i).image, bitmap.get(i).left, bitmap.
                   get(i).top, null);
10         }
11         return temp;
12     }
```

Excertos de Código 3.6: Definição da função `combineAllImageIntoOne`

3.2.5 *Displays*

Para que a interface do utilizador fique concluída é necessário também definir os *displays*. Um *display* funciona como uma caixa de texto, permitindo a representação de texto no ecrã. Na programação da aplicação *Android* um *display* é definido através de uma `TextArea` e, de forma semelhante aos elementos já referidos, também apresenta problemas de posicionamento em ecrãs de tamanhos diferentes, figuras 28 e 29.

Para corrigir o posicionamento deficiente entre tamanhos de ecrãs diferentes, foram criados diferentes *layouts* consoante as medidas mínimas de diferentes dispositivos. Deste modo, é possível definir diferentes *layouts* consoante o tamanho do ecrã do dispositivo. Um *layout* é um ficheiro em *XML*, que define os objetos que são criados juntamente com o início da aplicação *Android*, desde `ImageViews`, `TextAreas`, etc. Desta forma as `TextAreas` ficam definidas de igual forma para qualquer dispositivo, sendo que cada dispositivo irá usar um *layout* otimizado para o seu tamanho de ecrã.

Durante a implementação dos *displays* surgiu um segundo problema. A funcionalidade de efetuar *zoom* sobre imagens com um duplo toque no ecrã apenas efetuava o *zoom* sobre a imagem e não sobre toda a interface do utilizador, ou seja, a imagem e os seus botões eram

Figura 28: Exemplo de `TextAreas` em dispositivos de 5.5'

Figura 29: Exemplo de `TextAreas` em dispositivos de 5'

afetados pela funcionalidade de *zoom* mas os *displays* ficavam na mesma posição (figura 30) o que deixava os *displays* mal posicionados.

Para evitar esta imperfeição na interface do utilizador foi desenvolvida uma função que ignora os cliques duplos. Isto é possível ignorando os cliques com menos de 250 milissegundos de intervalo. Como podemos ver na função do Excerto 3.7, caso se trate de uma ação de pressionar um ponto do ecrã, é calculado o intervalo entre o primeiro e o segundo clique. Caso esse intervalo seja superior a 250 milissegundos são considerados dois cliques singulares no ecrã, senão o segundo clique é ignorado.

Figura 30: Problema de *zoom* sobre as imagens

```
1   @Override
2     public boolean dispatchTouchEvent(MotionEvent ev) {
3         timestamp2 = System.currentTimeMillis();
4
5 //  botão pressionado com mais de 250 milissegundos
6         if(timestamp2-timestamp1 >250 && ev.getAction()==MotionEvent.
              ACTION_DOWN){
7
8             timestamp1=timestamp2;
9             return super.dispatchTouchEvent(ev);
10        }
11 //  botão largado
12        else if (ev.getAction()==MotionEvent.ACTION_UP){
13            return super.dispatchTouchEvent(ev);
14        }
15 //  botão pressionado com menos de 250 milissegundos
16        else {
17            return true;
18        }
19    }
```

Excertos de Código 3.7: Função que desabilita a funcionalidade de *zoom* ao duplo toque no ecrã

3.3 ANIMAÇÕES

Para que a simulação dos dispositivos numa aplicação *Android* seja mais fiel ao seu protótipo na plataforma do *PVSio-web*, é necessário implementar animações tanto nos movimentos das seringas como na animação de botões, de modo a que os utilizadores fiquem com uma ideia mais realística de como o dispositivo físico irá funcionar.

3.3.1 *Representação de movimentos*

O dispositivo *Stellant V2* durante o seu processo de utilização necessita que sejam inseridas duas seringas (uma para a solução de contraste e outra para a solução salina), às quais são conectados os respetivos frascos com as soluções devidas. Quando a solução é sugada pela seringa, o êmbolo da mesma movimenta-se consoante a quantidade de liquido sugado. Posteriormente, quando a solução é expelida há nova movimentação do êmbolo da seringa (que é o responsável por a solução ser expelida).

Para a representação de movimentos foram consideradas duas técnicas: *sprites* ou *SVG*. A representação por *sprites* assenta na movimentação dos *sprites* no ecrã dando a ideia de

movimento. Os *sprites*, neste caso, são várias imagens que no seu conjunto formam apenas uma, como a seringa e o seu êmbolo (figuras 31 e 32). A representação por *SVG* assenta na definição das imagens num ficheiro *XML* e posterior renderização por parte da aplicação *Android*.

A representação com recurso a *sprites* foi a abordagem escolhida, por ser aquela que está há mais tempo aperfeiçoada e que requer menor capacidade de processamento (não necessita de estar constantemente a renderizar a imagem), tornando a aplicação mais rápida.

Figura 31: *Sprite* do cano da seringa

Figura 32: *Sprite* do êmbolo da seringa

Para se criar a imagem final da seringa é utilizada a função *combineImageIntoOne*, já referida anteriormente, que recebendo várias imagens desenha uma de cada vez, sendo que a segunda será sobreposta à primeira e assim sucessivamente. A construção da imagem final é constituída por duas etapas. Na primeira etapa, são combinadas as imagens do êmbolo e do cano da seringa, por esta ordem, de modo a criar o *sprite* da seringa e posteriormente o *sprite* criado é combinado com as restantes imagens de fundo, do dispositivo, da consola e dos botões externos, Excerto 3.8.

Neste mesmo excerto pode-se perceber como foi implementado o movimento do êmbolo da seringa, sendo que sempre que há uma alteração do valor da volume de liquido, a posição do êmbolo, representada pela variável `plunger_pos`, seraá alterada (Figuras 33 e 34).Esta alteração será, então, proporcional à alteração do volume do líquido na seringa correspondente, que será expresso também na variável `plunger_pos`.

```
1  (...)
2  if(state.syringe_contrast_present) {
3
4    Bitmap bm_con_syr_data = BitmapFactory.decodeResource(getResources(), R.
       drawable.syringe_data_green);
5        Bitmap bm_con_syr_plun = BitmapFactory.decodeResource(getResources(),
           R.drawable.syringe_pluger);
6
7        ArrayList<Image> syringe_con_images = new ArrayList<Image>();
8        syringe_con_images.add(new Image(bm_con_syr_plun, 0, plunger_pos));
9        syringe_con_images.add(new Image(bm_con_syr_data, 0, 0));
10
11       Bitmap result_con_syringe = combineImageIntoOne(syringe_con_images,
           1000, 700);
12       (...)
13       ArrayList<Image> a = new ArrayList<Image>();
14       (...)
15       a.add(new Image(result_con_syringe,850,270));
16       (...)
17
18   Bitmap result = combineImageIntoOne(a, background.getHeight(), background.
       getWidth());
19       background.setImageBitmap(result);
20       }
21   (...)
```

Excertos de Código 3.8: Exemplo da implementação do movimento do êmbolo da seringa

Um problema que pode vir a ocorrer, durante a execução da simulação, com as animações é o facto de estas poderem ”saltar” de uma posição para outra. Considerando um exemplo concreto, se ocorrer uma sucção muito rápida do líquido de um frasco por parte de uma seringa, o êmbolo que marcava um volume de 20 unidades pode passar a marcar 100 unidades de forma quase instantânea. Ou seja, o êmbolo que estava na posição A (20 unidades) ”salta” para a posição B (100 unidades). Este ”salto”deve-se ao facto de a posição dos *sprites* estar a ser atualizada apenas quando os valores dos volumes das seringas são atualizados.

Figura 33: Seringa com o êmbolo completamente dentro, sem liquido no cano

Figura 34: Seringa com o êmbolo recuado, com liquido no cano

3.3.2 *Representação de luzes intermitentes*

A representação de luzes intermitentes, assenta tanto na utilização de *sprites* como também na utilização de *displays* . Adicionalmente, é criada uma função que torna os *sprites* ou os *displays* visíveis ou invisíveis, sendo ativada entre intervalos de tempo constantes previamente definidos.

No excerto 3.9, podemos verificar que a função `blink` começa por criar um `Handler`[2] e uma `Thread`, que serão responsáveis pelo escalonamento do objeto `Runnable`[3]. Uma vez iniciada a nova `Thread`, esta começa por adormecer durante 700 milissegundos. Quando a `Thread` for "acordada" são executadas as operações definidas no objeto `Runnable` associado à `Thread`. Neste exemplo, é alterada a visibilidade do objeto `textView1` da interface do utilizador e, no fim, caso seja necessário é executada de novo a função `blink` ou o objeto `textArea1` fica sempre visível até ordem em contrário.

2 Android Handler: https://developer.android.com/reference/android/os/Handler
3 Android Runnable: https://developer.android.com/reference/java/lang/Runnable

```
1   private void blink(){
2         final Handler handler = new Handler();
3         new Thread(new Runnable() {
4
5
6             @Override
7             public void run() {
8
9                 int timeToBlink = 700;    //in miliseconds
10                try{Thread.sleep(timeToBlink);}catch (Exception e) {}
11
12                handler.post(new Runnable() {
13
14                    @Override
15                    public void run() {
16                        TextView txt = (TextView) findViewById(R.id.textView1)
                              ;
17                        if(txt.getVisibility() == View.VISIBLE){
18                            txt.setVisibility(View.INVISIBLE);
19                        }else{
20                            txt.setVisibility(View.VISIBLE);
21                        }
22                        if(state.display_saline.equals(State.Display.DISP_INIT
                              ))
23                            blink();
24                        else
25                            txt.setVisibility(View.VISIBLE);
26                    }
27                });
28            }
29        }).start();
30    }
```

Excertos de Código 3.9: Exemplo da implementação da intermitência das luzes do dispositivo

Esta representação tanto pode ser utilizada para alterar a visibilidade de dos *displays* como de *sprites* dos *LED*'s sobrepostos aos botões dos dispositivos.

3.4 CONCLUSÕES

Com base na informação presente neste último capítulo, deve ser possível para um utilizador comum perceber como funciona o *Stellant V2*, bem como reconhecer os principais botões referidos. Deve também ser possível compreender quais os diferentes passos tomados na construção da aplicação *Android*, bem como obter uma ideia geral de como a aplicação iraá funcionar.

A aplicação *Android* desenvolvida tem um comportamento muito semelhante ao protótipo existente na plataforma do *PvSio-web*. Esta semelhança deve-se, em grande parte, ao facto de a parte da lógica de controlo da aplicação *Android* ser a mesma usada no protótipo já existente, devido a ser derivada da tradução do modelo *PVS*. As partes da interface de utilizador e das animações foram construídas de raíz, mas baseadas no protótipo já desenvolvido na plataforma do *PVSio-web*.

A aplicação *Android* é, então, composta pela junção da parte da lógica de controlo, da interface de utilizador e das animações. A aplicação é completamente autónoma e consegue executar todas as ações que são possíveis de executar no protótipo da plataforma do *PVSio-web*.

O único ponto menos positivo na aplicação *Android* é uma ligeira deficiência que poderá ocorrer nas animações das seringas, mas que não põe em causa o bom funcionamento deste protótipo.

4

FRAMEWORK JAVASCRIPT

Após a primeira parte dos objetivos propostos para esta dissertação estar concluída, avançou-se para a criação de uma *framework* em *JavaScript* que possa ser integrada no ambiente do *PVSio*-web e que seja genérica ao ponto de poder ser utilizada para o desenvolvimento ou migração de protótipos previamente desenvolvidos para aplicações *Android*.

O objetivo desta *framework* é possibilitar a criação mais rápida de simulações de dispositivos, para que estas simulações possam servir de teste para a deteção precoce de erros a nível do desenvolvimento e funcionamento do dispositivo. Convém também salientar que a *framework* foi desenvolvida para que pudesse ser operada por um utilizador com apenas conhecimentos mínimos de programação, simplificando o código necessário para criar uma aplicação *Android*.

De seguida irá ser explicada a estrutura da framework, bem como o seu funcionamento e como deverá ser utilizada.

4.1 ESTRUTURA DA FRAMEWORK

A *framework* desenvolvida é constituída por três partes principais: um projeto padrão compatível com *Android Studio*, um conjunto de ficheiros padrão que serão modificados e completados consoante o dispositivo que se pretender simular e um *script* que contém uma biblioteca de funções disponíveis para serem invocadas pelo utilizador, Figura 35.

O projeto padrão é uma diretoria composta por ficheiros estáticos de uma aplicação *Android*, ou seja, este conjunto de ficheiros manter-se-à sem alterações ou será alterado apenas quando houver uma nova compilação do projeto pelo *Android Studio* que atualizará este conjunto ficheiros.

Existe também um segundo conjunto de ficheiros padrão que serão utilizados pela *framework*. Estes ficheiros foram criados a partir dos ficheiros que constituem parte da aplicação *Android* e estão adaptados para que seja necessário apenas completá-los de acordo com as especificações de cada protótipo que se pretende migrar para uma aplicação *Android*.

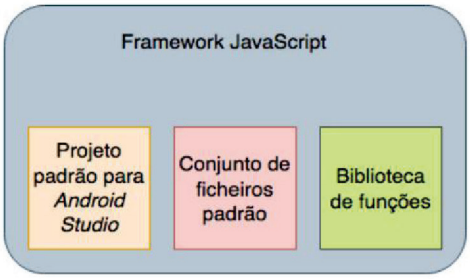


Figura 35: Exemplo da estrutura da *framework*

Por fim, o *script*, escrito em *JavaScript*, divide-se em duas partes: a primeira parte é composta por um conjunto de variáveis que irão guardar a informação sobre os diversos componentes da interface da simulação, como botões, imagens e *displays*. Como podemos ver no excerto 4.1 existe um conjunto de variáveis, cada uma com a função de agrupar um conjunto de elementos. Por exemplo, a variável `buttons` agrupa a informação de todos os botões definidos através da *framework*, a variável `displays` agrupa a informação de todos os *displays*, a variável `images` agrupa a informação de todas as imagens. Por outro lado, as variáveis `gALLvariables` e `functions` não agrupam a informação sobre elementos da interface da aplicação, mas sim informação sobre variáveis e funções necessárias para os processos de *background* da simulação, como ligar o dispositivo, inserir e encher as seringas, por exemplo. Existem ainda variáveis que possuem algumas outras informações sobre a simulação, como as variáveis `appName`, `orientation` ou `canvasScale`, que correspondem ao nome que o utilizador pretende dar à aplicação *Android*, à orientação do ecrã da simulação e à escala que deverá ser usada para desenhar as imagens no ecrã, respectivamente.

Todas estas variáveis serão agrupadas dentro de uma outra, a variável `device`. Esta variável irá conter toda a informação sobre a constituição da aplicação *Android* que será construída através da *framework*.

A segunda parte do *script* corresponde à definição das funções a serem invocadas pelo utilizador e que serão explicadas mais à frente neste documento.

```
1 var device = {
2  appName: "Template",
3  orientation: 1,
4  canvasScale: 2.7,
5  gALLvariables: [],
6  buttons: [],
7  displays: [],
8  images: [],
9  functions: []
10 }
```

Excertos de Código 4.1: Exemplo das variáveis do *script*

4.2 COMO FUNCIONA?

O método de funcionamento desta*framework* é bastante simples, ver Figura 36. O utilizador começa por definir os elementos da interface do utilizador da aplicação que pretende criar, sendo que consoante os elementos definidos pelo utilizador a variável `device` começa a ser composta.

O utilizador poderá também definir os nomes das funções que compõem a lógica de controlo da aplicação. Devido à falta da existência de um tradutor não é possível traduzir o modelo *PVS* para a linguagem *Java* de forma automática, pelo que posteriormente terá de ser o utilizador a completar o ficheiro `State.java` que implementa a lógica de controlo, com a tradução do correspondente ficheiro *PVS* que poderá ser obtido no repositório do *PVSio-web*.

Com base na informação presente na variável `device` a *framework* irá completar os ficheiros padrão que constituem parte da aplicação *Android* com o auxílio de uma biblioteca externa, o *Handlebars* [12]. Estes ficheiros serão colocados dentro do projeto padrão previamente criado, no local devido para os ficheiros *Java* que constituem a aplicação. Desta forma, é criado um novo projeto *Android Studio* com as especificações definidas pelo utilizador. Este projeto é criado numa diretoria especificada pelo utilizador como argumento da função *createJava*. Por fim, a *framework* analisa se na diretoria onde opera existe algum ficheiro de imagem que o utilizador terá indicado como pertencente à simulação e copia as imagens para a diretoria correspondente dentro do projeto *Android Studio*.

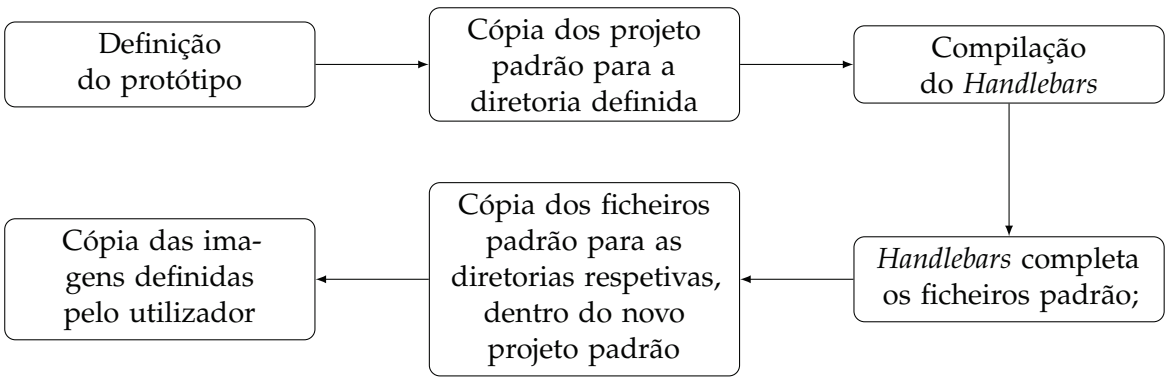


Figura 36: Método de desenvolvimento da *framework*

4.2.1 *Handlebars*

O *Handlebars* é uma ferramenta que permite construir padrões semânticos, isto é, permite que certas expressões ou *tags* de um ou mais *template* possam ser substituídas pelos valores correspondentes às mesmas *tags* num objeto previamente como contexto.Esta biblioteca permite escrever compiladores de uma forma simples com base em *pattern matching*.

As *tags* são nomes de atributos presentes no contexto que será usado na compilação do *template Handlebars* e estão delimitadas por dupla chaveta ({{...}}).

Os *templates* são ficheiros de texto que serão compilados pelo *Handlebars* e não necessitam de estar escritos numa linguagem específica, uma vez que são lidos de modo genérico como ficheiros de texto pelo compilador.

O funcionamento deo *Handlebars*simples: o compilador *Handlebars* recebe um objeto de formato *JSON* que usará como contexto e no qual estarão definidos vários pares chave-valor. Estes pares possuem informação sobre o valor pelo qual uma determinada tag deverá ser substituída, sendo que a *tag* será identificada pelas chaves dos pares. De seguida o compilador recebe também o ficheiro a compilar, no qual existem as diversas *tags* e procede à substituição das mesmas.

Na Figura 37 podemos encontrar duas *tags* pertencentes a um *template*: a *tag* `title` e a *tag* `body`. Enquanto que na Figura 38 podemos encontrar um exemplo da definição de um contexto que será posteriormente enviado para o compilador do *Handlebars*.

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

Figura 37: *Template* a ser compilado pelo *Handlebars*

```
var context = {title: "My New Post", body: "This is my first post!"};
```

Figura 38: Definição do contexto usado para substituição das *tags*

O ficheiro final, como esperado, é resultado da substituição das *tags* do *template* pelos valores associados às respetivas *tags* no contexto. Como se pode ver na Figura 39:

Para além da definição das *tags* o *Handlebars* permite também a definição de outras expressões. Estas novas expressões permitem a definição de *if clauses* e ciclos durante o processo de substituição de *tags*. Como os nomes indicam as *if clauses* permitem a escrita de certo excerto de texto apenas no caso de o valor da *tag* associada à *if clause* ser `true`, enquanto que os ciclos permitem que todos os elementos de um *array* associado como valor

```
<div class="entry">
  <h1>My New Post</h1>
  <div class="body">
    This is my first post!
  </div>
</div>
```

Figura 39: Ficheiro final após compilação pelo *Handlebars*

de uma *tag* possam ser utilizados, sequencialmente, na substituição das *tags* do ficheiro padrão, Figuras 40, 41 e 42. Os ciclos são inicializados pela expressão {{`#each ...`}} e as *if clause* são inicializadas pela expressão {{`#if ...`}}.

Após o desenvolvimento da aplicação *Android* que simula o comportamento do *Stellant V2* foi possível determinar quais as partes dos ficheiros da aplicação que necessitariam de ser adaptadas para que a própria aplicação pudesse ser adaptada para qualquer outro protótipo de dispositivo. Foram então criados *templates* desses ficheiros *Java* que, com a ferramenta *Handlebars*, serão completados com base na definição de um contexto definido pelo utilizador da *framework*.

Após o desenvolvimento da aplicação *Android* que simula o comportamento do *Stellant V2* foi possível determinar quais as partes dos ficheiros da aplicação que necessitariam de ser adaptadas para que a própria aplicação pudesse ser adaptada para qualquer outro protótipo de dispositivo. Foram então criados *templates* desses ficheiros *Java* que, com a ferramenta *Handlebars*, serão completados com base na definição de um contexto definido pelo utilizador da *framework*.

```
{{#each nav}}
  <a href="{{url}}">
    {{#if test}}
      {{title}}
    {{^}}
      Empty
    {{/if}}
  </a>
{{~/each}}
```

Figura 40: *Template* a ser compilado pelo *Handlebars*

```
{
  nav: [
    {url: 'foo', test: true, title: 'bar'},
    {url: 'bar'}
  ]
}
```

Figura 41: Definição do contexto usado para substituição das *tags*

```
<a href="foo">
    bar
</a>
<a href="bar">
    Empty
</a>
```

Figura 42: Ficheiro final após compilação pelo *Handlebars*

4.2.2 *Funções*

Como já referido nesta dissertação, a *framework JavaScript* fornece diversas funções que devem ser invocadas pelo utilizador consoante a simulação que pretender criar. Estas funções permitem a definição de vários parâmetros do projeto *Android Studio*, que será criado pela mesma *framework*.

As funções podem ser agrupadas em três grupos: funções que definem elementos da interface, como botões, *displays* (elementos de apresentação de texto, como os valores do volume das seringas no *Stellant V2*), ou imagens; funções que definem elementos do código de *background*; e funções que efetuam a geração do projeto *Android Studio*. No Anexo B pode ser consultada a documentação de todas as funções.

Funções de definição da interface

Este conjunto de funções serve para definir os elementos da interface da simulação *Android* como botões, *displays* ou imagens. Refere-se às funções `createButton`, `createDisplay` e `createImage` (Excerto4.2) sendo que todas têm uma estrutura bastante semelhante.

```
1    createButton: function (name, coords, other){...}
2    createDisplay: function (name, coords, other){...}
3    createImage: function (name,coords,other){...}
```

Excertos de Código 4.2: Assinatura das funções `createButton`, `createDisplay` e `createImage`

Todas estas funções recebem três argumentos: o argumento `name` corresponde ao nome que o elemento irá adquirir na simulação *Android*, o argumento `coord` corresponde à posição e medidas do elemento no ecrã da aplicação (com exceção da função `createImage` que não necessita do tamanho da imagem) e, por fim, o argumento `other` que poderá conter informações diferentes mediante a função em que se insira (Excerto 4.3):

- uma função `createButton`, contém a informação sobre uma função que altera o estado da simulação e que será invocada quando houver um toque no botão.

- uma função `createDisplay`, contém a informação sobre o texto a apresentar no *display*, bem como o tamanho, cor e fonte deste mesmo texto. Contém ainda informação sobre a visibilidade do *display* no início da aplicação.
- uma função *createImage* o argumento `other` não fornece qualquer informação útil para a *framework*.

```
createButton(
  "connect_infusion_set",
  { left :620 , top :370 , width :217 , height :28} ,
  {function:"connect_infusion_set",display:false}
  );

createDisplay( "tv2",
  {left:280, top:624, width:100, height:50},
  {startText:"---", textsize:25, color:"#0000ff", font:"fonts/abc.ttf",
    visible:false}
    );

createImage( "all",
  {left:0, top:0},
  {visible:true}
  );
```

Excertos de Código 4.3: xemplo de estrutura dos objetos passados como argumentos às funções

Em cada uma destas funções a informação fornecida pelos argumentos é armazenada num novo objeto ao qual é associado automaticamente um número de identificação e posteriormente é inserido numa das variáveis referidas anteriormente:

- caso se trate de um botão é inserido na variável `buttons`.
- caso se trate de um *display* é inserido na variável `displays`.
- caso se trate de uma imagem é inserido na variável `images`.

Funções de definição da lógica de controlo

Este conjunto de funções define as funções e variáveis presentes no ficheiro `State.java` e que serão vitais para o bom funcionamento da simulação do dispositivo. As funções pertencentes a este conjunto são as funções `createFunction` e `createGVariable`, Excerto 4.4:

```
1     createFunction: function (name,type){...}
2
3     createGVariable: function (name, type, value){...}
```

Excertos de Código 4.4: Assintura das funções `createFunction` e `createGVariable`

- A função *createFunction* tem como objetivo definir de forma básicas os métodos que serão necessáris para o bom funcionamento da simulação dos dispositivo correspondente. Para tal a função deve receber como parâmetros o nome do método e o tipo da variável que o mesmo irá devolver, para que essas informações sejam associadas a um objeto `func` que irá ser criado e incluido no objeto `functions` que contém as informações de todas as funções definidas pela *framework*.

- A função *createGVariable* permite a definição de variáveis globais ao dispositivo que se pretende simular. Esta função recebe como parâmetros o nome da variável (`name`), o tipo da variável (`type`) e o valor inicial da mesma (`value`).

  Tendo em conta estes parâmetros a função `createGVariable` começa por verificar se o objeto `gALLvariables`, que contém a informação de todas as variáveis globais definidas através da *framework*, está devidamente inicializado. De seguida é analisado o tipo da variável, isto porque, caso a variável seja uma `String` é colocada num objeto à parte das restantes variáveis. A razão das variáveis do tipo `String` serem colocadas à parte das restantes é pelo facto das mesmas necessitarem de uma diferente geração do código da atribuição (o valor a ser atribuído a uma variável do tipo `String` tem de estar embutido em aspas). Desta forma as variáveis do tipo `String` serão colocadas no objeto `gstringvariables`, enquanto que as restantes serão colocadas no objeto `gvariables`. Ambos os objetos estão contidos no objeto `gALLvariables`. Com a separação das variáveis em 2 objetos é possível implementar 2 formas diferentes de substituição de *tags* nos *templates* do *Handlebars* (uma para as variáveis do tipo `String` e outra para as restantes).

  Após a análise do tipo da variável, a função prossegue o seu caminho de forma idêntica para os dois ramos. Primeiro é criado um objeto que irá conter as informações acerca da variável global. De seguida define-se o nome, o tipo e o valor inicial da variável, consoante os valores definidos como parâmetros da função. Por fim o objeto criado é adicionado ao `gstringvariables` ou `gvariables`, consoante o tipo da variável.

Funções de geração do projeto

A geração do projeto *Android Studio* assenta apenas numa única função: a função `createJava`, Excerto 4.5. Esta função é talvez a função mais importante desta *framework*, sendo que tem como objetivo criar um projeto *Android Studio* com base no projeto padrão previamente definido, mas adicionando a esse protejo padrão os ficheiros relativos à simulação *Android* que o utilizador pretende criar. Esta função para além de efetuar uma cópia do projeto padrão e de gerar os ficheiros com a informação da simulação *Android* copia também as imagens que o utilizador definir através da função *createImage* para o seu devido local dentro do novo projeto criado, sendo que as imagens apenas tenham de estar situadas na mesma diretoria da *framework*.

```
1     createJava: function (path){...}
```

Excertos de Código 4.5: Assinatura da função `createJava`

Esta função, que recebe como argumento o caminho completo da diretoria onde o utilizador pretende criar o seu projeto *Android Studio*, começa por efetuar uma cópia de todos os ficheiros e diretorias contidos dentro do projeto definido como padrão para a nova localização, indicada pelo utilizador, através de uma função auxiliar. De seguida, os ficheiros *Handlebars* são compilados e, após a substituição das *tags*, escritos na sua diretoria de destino, dentro do novo projeto *Android Studio*. Por fim, para todos os objetos contidos no objeto `images` é verificado se existe algum ficheiro com o mesmo nome da imagem no formato `.jpg` ou `.png` (formatos de imagens mais comuns) e, caso exista é copiado para o novo projeto *Android Studio*.

Outras funções

Além das funções já enumeradas existem mais algumas funções auxiliares que completam esta *framework*:

- A função *setAPPName* é uma função que permite apenas definir o nome que pretendemos dar à simulação *Android* a ser criada. O nome que se pretende atribuir à simulação é passado como parâmetro (`name`) à função *setAPPName*.
- A função *setCanvasScale* permite a definição da variável `canvasScale` usada posteriormente na aplicação *Android* que será criada. Esta variável permite que as imagens sejam escaláveis consoante o seu valor definido. O valor da variável deve ser definido tendo em conta o tamanho de ecrã do dispositivo ou dispositivos alvo da aplicação *Android* e o tamanho das imagens que irão ser usadas como fundo da aplicação *Android*.

- A função *setOrientation*, como o nome sugere, permite a definição da orientação de visualização da aplicação *Android*. Apenas existem dois modos de orientação no *Android*: *portrait* (visualização com o dispositivo ao alto) e *landscape* (visualização com o dispositivo deitado). Como tal a função recebe como parâmetro a orientação pretendida (`PORTRAIT` ou `LANDSCAPE`) e converte o parâmetro recebido para 0 ou 1, que são os valores utilizados pelo *Android* para definir a orientação do ecrã como *portrait* ou *landscape*, respetivamente.

4.3 Método de utilização

A utilização da *framework* desenvolvida é dividida em duas partes: a primeira parte corresponde ao desenvolvimento do *script* em *JavaScript* onde o utilizador define os principais componentes da aplicação *Android*; na segunda parte o utilizador tem de executar o *script* previamente criado, através do qual será gerado o projeto que poderá ser aberto com *Android Studio*. Esta segunda parte irá necessitar da instalação do *NodeJS*[1] que irá interpretar o código do *script* e executar os respetivos comandos.

O utilizador terá de começar o seu *script* com a importação da biblioteca das funções que irá usar para definir os objetos da aplicação *Android*. A importação da biblioteca é feita com recurso a uma simples linha de código como se pode ver no Excerto 4.6, linha 2.

Após a importação da biblioteca o utilizador deve definir os componentes que deseja incluir na sua aplicação *Android*, conforme ilustrado nas linhas 5 a 32 do Excerto **??**, com a opção de definir também o nome da aplicação, o valor da variável `canvasScale` e a orientação da aplicação. Como as função são importadas da biblioteca que foi nomeada `AndroidPinter`, quando são invocadas tem de referenciar a biblioteca de onde são originárias, sendo preciso escrever `AndroidPrinter.` antes do nome da função.

Quando todos os componentes da simulação *Android* estiverem definidos, o utilizador invoca, por fim, a função `createJava` que irá criar o novo projeto compatível com *Android Studio*.

A segunda parte da interação com a *framework* desenvolvida corresponde à execução do *script* criado pelo utilizador. A execução é efetuada a partir da aplicação "Terminal"nos sistemas *MacOS* ou *Linux* ou da aplicação "Linha de Comandos"nos sistemas *Windows* e o utilizador deve primeiro navegar até à diretoria da framework e digitar o código presente no excerto 4.7 , sendo que em vez de digitar `<scriptName>` deverá digitar o nome do ficheiro criado pelo mesmo:

```
1 //importação da biblioteca de funções
2 var AndroidPinter = require('./AndroidPinter.Paolo');
```

1 *NodeJS* em https://nodejs.org/en/

```
3
4 //definição do nome da aplicação Android
5 AndroidPinter.setAPPName("AppTemplate");
6
7 //definição da variável canvasScale
8 AndroidPinter.setCanvasScale(2.8);
9
10 //definição da orientação da aplicação Android
11 AndroidPinter.setOrientation("PORTRAIT");
12
13 //definição de uma variável global
14 AndroidPinter.createGVariable("globalVar1","int",230);
15
16 //definição de um botão
17 AndroidPinter.createButton("inc_saline",
18   { left:174, top:607, width:23, height:23},
19   {function:"press_inc_saline",display:false});
20
21 //definição de um display
22 AndroidPinter.createDisplay("tv1",
23   {left:120, top:624, width:100, height:50},
24   {startText:"---", textsize:25, color:"#2DFF1B", font:"fonts/abc.ttf",visible
       :false});
25
26 //definição de uma imagem
27 AndroidPinter.createImage("all",
28   {left:0, top:0},
29   {visible:true});
30
31 //definição de uma função
32 module.createFunction("inc_saline", "State");
33
34 //criação do novo projeto para Android Studio
35 AndroidPinter.createJava("/Users/andrepinto/Desktop");
```

Excertos de Código 4.6: Exemplo de como utilizar a *framewrok* desenvolvida

```
1   node <scriptName>.js
```

Excertos de Código 4.7: Comando de execução do *NodeJS* no terminal

De modo a concluir a aplicação, o utilizador deverá abrir o projeto *Android Studio* no IDE e completar o ficheiro `State.java` com a tradução do modelo *PVS*, visto não haver um tradutor que possa ser utilizado pela *framework*.

4.4 EXEMPLOS

Como já referido nesta dissertação, um dos objetivos da *framework* desenvolvida era possibilitar a migração de protótipos do *PVSio-web* para aplicações *Android* de forma simplificada, mesmo que estes dispositivos fossem totalmente diferentes.

Para testar a utilização da *framework*, o dispositivo *Stellant V2* foi migrado para uma aplicação *Android* para que pudesse ser comparado com a aplicação nativa previamente criada.

De modo a avaliar também a versatilidade desta *framework* foi escolhido um segundo dispositivo para ser migrado, o *Radical-7*. O protótipo deste dispositivo disponível na plataforma do *PVSio-web* será, então, migrado para uma aplicação *Android*, através da utilização da *framework*.

Stellant V2

A simulação do dispositivo *Stellant V2* foi a mais fácil de desenvolver, visto já haver algum conhecimento adquirido acerca do dispositivo durante o desenvolvimento da aplicação *Android* nativa, sendo que todos os objetos definidos com o auxílio da *framework* foram baseados nas definições dos seus homólogos na aplicação *Android* nativa.

O script que constrói o projeto da aplicação *Android* do *Stellant V2* foi escrito com base nas funções da *framework* previamente explicadas, sendo possível consultar um exemplo do mesmo no Excerto 4.8.

Com este *script* é agora possível criar um projeto para o *Android Studio* que permita criar uma aplicação para o *Stellant V2*. Após o projeto estar criado abre-se o mesmo a partir do *Android Studio* e poderemos executar o projeto num emulador e pré-visualizar o resultado final, a aplicação que simula o comportamento do *Stellant V2* (Figura 43). Antes de inicializar o emulador poderá ser necessário completar as funções existentes no ficheiro `State.java`, visto que os métodos foram criados, mas não definidos.

A construção do *script* ilustrado no Excerto 4.8 foi bastante rápida, cerca de 10 minutos, sendo que um utilizador que não estivesse tão familiarizado com a *framework* poderia levar mais 5 a 10 minutos para a construção do mesmo (podendo este tempo ser minimizado com recurso a *copy&paste* e depois apenas alterar os valores de cada função invocada). De qualquer forma, a construção e execução do *script* seria mais rápida que a construção da aplicação nativa que demoraria no mínimo entre 30 a 35 minutos.

Estendendo-se a comparação da aplicação construída através da *framework*, ao nível da sua utilização, com a aplicação *Android* nativa e a aplicação construída com as *Google WebView*, conclui-se que há uma grande semelhança no aspeto e comportamento entre a

aplicação construída através da *framework* e a aplicação nativa. A aplicação construída com as *Google WebView* tem um aspeto mais uniforme, mas o comportamento apresenta-se mais lento, devido à necessidade de estabelecer uma conexão a um serviço de Internet.

```
1  var AndroidPrinter = require('./AndroidPrinter.Paolo');
2
3  AndroidPrinter.setAPPName("StellantV2");
4  AndroidPrinter.setCanvasScale(2.8);
5  AndroidPrinter.setOrientation("PORTRAIT");
6  (...)
7  modulAndroidPrinter.createGVariable("MAX_VOLUME","int",230);
8  (...)
9  AndroidPrinter.createButton("btn_fdown_saline",{ left:115, top:876, width:65,
       height:65}, {function:"press_btn_fDOWN_saline",display:false});
10 (...)
11 AndroidPrinter.createDisplay("tv1", {left:120, top:624, width:100, height:50},
        {startText:"---", textsize:25, color:"#2DFF1B", font:"fonts/abc.ttf",
       visible:false});
12 (...)
13 AndroidPrinter.createImage("all", {left:0, top:0}, {visible:true});
14 (...)
15 AndroidPrinter.createFunction("inc_saline", "State");
16 (...)
17 AndroidPrinter.createJava("/Users/andrepinto/Desktop");
```

Excertos de Código 4.8: Exemplo do código *JavaScript* da *framewrok* para migração do *Stellant V2*

Figura 43: Exemplo da aplicação alusiva ao *Stellant V2* construída a partir da *framework* desenvolvida

Radical-7

Por sua vez, o dispositivo *Radical-7* necessitou de uma prévia investigação, de modo a haver uma familiarização com as funcionalidades que o dispositivo possui. Desta forma, é possível ter uma ideia geral do dispositivo, do seu objetivo e do seu funcionamento.

O *Radical-7* é um dispositivo que é utilizado para controlar alguns parâmetros de um paciente, tais como a sua pulsação, a saturação de oxigénio, o índice de perfusão, os índices de hemoglobina, entre outros. O *Radical-7* permite aos utilizadores, neste caso às equipas médicas, selecionar quais os parâmetros mais importantes e que, por consequência, necessitam de uma monitorização mais próxima, permitindo ao dispositivo alertar as equipas médicas através de alarmes audio-visuais quando algum desses parâmetros atinge valores anormais e pode constituir perigo para a saúde do paciente.

O protótipo deste dispositivo presente na plataforma do *PVSio-web* simula todas as funcionalidades do dispositivo.

```
1  var AndroidPrinter = require('./AndroidPrinter.Paolo');
2
3  AndroidPrinter.setAPPName("Radical7");
4  AndroidPrinter.setCanvasScale(2.7);
5  AndroidPrinter.setOrientation("LANDSCAPE");
6
7  AndroidPrinter.createGVariable("id","String", "Radical7");
8  (...)
9  AndroidPrinter.createDisplay(
10   "disp1",
11   {top:220, left:820, width:100, height:50},
12   {startText:"99",textsize:25,visible:"false",color:"#ffffff"});
13 (...)
14 AndroidPrinter.createImage(
15   "radical_7",
16   {top:0, left:0},
17   {visible:"true"});
18 (...)
19 AndroidPrinter.createFunction ("per_on", "boolean");
20 (...)
21 AndroidPrinter.createButton (
22   "btn_on",
23   {top:330, left:1040, width:50, height:50},
24   {function:"btn_on"});
25
26 AndroidPrinter.createJava("/Users/andrepinto/Desktop");
```

Excertos de Código 4.9: Exemplo do código *JavaScript* da *framewrok* para migração do *Radical-7*

No Excerto 4.9 pode-se analisar o código do *script* usado para a construção do projeto da aplicação *Android* do dispositivo *Radical-7*. Todos os objetos foram baseados, e devidamente ajustados, no ficheiro *PVS* existente no site do PVSio-Web[21] e que define o protótipo existente na plataforma.

Utilizando o *script* apresentado, foi criado um projeto *Android Studio* para a aplicação do *Radical-7*. Antes de ser inicializada a aplicação é necessário completar os métodos do ficheiro `State.java` com a tradução do modelo *PVS* do protótipo do *Radical-7*. Uma primeira inicialização da aplicação será igual à Figura 44.

O *script* do *Radical-7* foi mais rápido de construir que o *script* do *Stellant V2*, devido ao protótipo do *Radical-7* possuir uma interface mais simples, tendo a sua construção demora entre 5 a 7 minutos. Apesar de continuar a existir uma poupança de tempo na utilização da *framework*, há uma menor poupança de tempo, sendo que uma aplicação nativa para o *Radical-7* demoraria à volta de 15 minutos a ser construída. Esta estatística pode indiciar que a utilização da *framework* será tão mais eficiente, a nível de simplificação do processo de construção da aplicação *Android*, quanto maior a complexidade do protótipo que se pretende migrar.

Figura 44: Exemplo da aplicação alusiva ao *Radical-7* construída a partir da *framework* desenvolvida

4.5 CONCLUSÕES

A framework desenvolvida possibilitou a migração de dois protótipos diferentes da plataforma do *PVSio-web* para aplicações *Android*. Também é possível integrar esta *framework* como um componente do site do *PVSio-web*, sendo que partes da *framework* foram construídas a pensar na posterior integração nessa plataforma. Apesar desta *framework* estar funcional tem ainda espaço para melhoramentos.

5

CONCLUSÃO

Como referido no início deste documento, o objetivo desta dissertação passa pela criação de uma *framework* que agilize o processo de migração de protótipos presentes na plataforma do *PVSio-web* para aplicações *Android*. As aplicações *Android* podem ser instaladas em dispositivos móveis possibilitando a formação de equipas médicas de instituições de saúde. Sem que seja necessário as instituições adquiriram dispositivos extra para que possam ser utilizados nas formações, podem aumentar o número de formações oferecidas pelas às equipas médicas e aumentar a familiarização das equipas com os diversos dispositivos, como forma de prevenir a ocorrência de erros médicos na utilização desses mesmos dispositivos.

5.1 AVALIAÇÃO DOS PROTÓTIPOS CONSTRUÍDOS

Ao longo desta dissertação foram apresentados dois protótipos que utilizaram ferramentas diferentes: um primeiro protótipo desenvolvido através das *Google WebView* e um segundo desenvolvido numa aplicação *Android* nativa, isto é, criada de raiz.

O protótipo desenvolvido a partir de uma aplicação *Android* nativa, apresenta a grande vantagem de ser auto-suficiente. Apesar de apresentar algumas deficiências no que toca ao aspeto das animações do dispositivo. Contudo, se considerarmos que o aspeto gráfico apesar de não ser perfeito consegue dar a entender ao utilizador como o dispositivo físico irá funcionar, pois só afeta a movimentação do êmbolo das seringas que a maior parte das vezes não é controlada pelo utilizador, então este protótipo apresenta-se como uma solução bastante eficaz na simulação do dispositivo *Stellant V2*. No entanto, para uma simulação de um dispositivo diferente, estas animações podem provocar algum mal entendido por parte dos utilizadores, pelo que deve ser procurada uma solução que permita melhorar as animações da aplicação.

Por outro lado, o protótipo desenvolvido através das *Google WebView*, como executa um protótipo já desenvolvido e validado que está disponível na plataforma do *PVSio-web* não apresenta falhas a nível da sua interface e respetivas animações. No entanto, como já

referido, está dependente de uma conexão a um serviço de Internet forte e estável. Dependendo do ambiente em que é utilizado, o serviço de Internet disponível pode não ser suficiente para que a página apresentada pela aplicação possa ser totalmente carregada, principalmente em instituições de saúde como hospitais em que há um grande número de utilizadores a utilizar o mesmo serviço. O facto de este protótipo estar dependente de um serviço de Internet e ser baseado na linguagem PVS (linguagem usada para a definição dos protótipos na plataforma do *PVSio-web*), leva a que a aplicação fique mais lenta, o que também pode ser uma condicionante para o protótipo.

5.2 CONCLUSÕES

Sendo o principal objetivo desta dissertação a construção de uma *framework JavaScript* que simplificasse o processo de construção de aplicações *Android* a partir de protótipos existentes na plataforma do *PVSio-web*, pode-se afirmar que esta dissertação cumpre o seu objetivo principal.

No Capítulo 4 foram apresentados dois protótipos construídos com a *framework* desenvolvida. Estes protótipos, apesar de necessitarem de ser completados com as traduções dos modelos *PVS*, apresentam grande parte da estrutura da aplicação *Android* já definida, o que facilita o trabalho do utilizador visto que não tem de construir uma aplicação *Android* totalmente de raiz.

A framework encontra-se pronta para ser integrada no ambiente do *PVSio-web* e poderá funcionar, com alguma adaptação, como uma forma de descarregar um projeto *Android Studio* a partir da página de cada um dos protótipos presentes na plataforma do *PVSio-web*.

5.3 TRABALHO FUTURO

Apesar de os resultados obtidos nos protótipos construídos ao longo desta dissertação serem já satisfatórios, existe ainda espaço para melhorar no que toca à *framework JavaScript* ou às aplicações *Android* criadas com as migrações dos protótipos.

- Ao nível da *framework* podia-se desenvolver um complemento que possibilitasse a tradução do código *PVS* dos protótipos para o ficheiro `State.java` das aplicações *Android*. Esta medida também faria com que não fosse necessário o utilizador definir as todas as funções fundamentais para o funcionamento do dispositivo de forma manual, como acontece de momento. O tradutor poderia funcionar de forma autónoma, com a possibilidade de ser invocado por outras ferramentas, sendo que nessa invocação

seria indicado qual o ficheiro que o tradutor deveria utilizar para produzir o ficheiro `State.java` completo.

- Ainda na *framework JavaScript* desenvolvida, poderia ser alargado o conjunto de tipos de variáveis que podem ser definidas. Neste momento só um conjunto reduzido de tipos se encontra corretamente abrangido pela *framework*, como é o caso dos tipos `int`, `float`, `double` ou `String`. O objetivo seria a possibilidade de serem definidos tanto novos tipos de objetos com definições mais específicas, mas também possibilitar a definição de conjuntos como `Arrays`, `Sets` ou `Maps`. Apesar de os tipos de dados abrangidos pela *framework* serem suficientes para as simulações do *Stellant V2* e do *Radical-7*, podem não ser suficientes para simulações de dispositivos mais complexos e que possuam novos tipos de variáveis.

- Com o intuito de abranger um maior número de dispositivos, esta *framework* poderia ser adaptada para a construção de aplicações para o sistema operativo *iOS*. Com a possibilidade de criar também uma aplicação *iOS*, garantia-se que cerca de 98% dos dispositivos móveis em todo o mundo seriam capazes de correr as simulações dos protótipos. Assim, a formação das equipas médicas com recurso a dispositivos móveis ficava ainda mais facilitada.

- No que toca às aplicações *Android* poderia haver um melhoramento das animações. A técnica de *sprites*, apesar de servir o seu propoósito, não é tão fluida quanto o desejado. Por vezes, a movimentação do êmbolo das seringas parece "saltar" de uma posição para outra, sendo que do ponto de vista gráfico poderia haver um melhoramento das animações. Este melhoramento poderia passar por um melhoramento de uma tecnologia já existente que se tornasse mais vantajoso para a animação das simulações ou pela criação de uma nova tecnologia.

- De modo a procurar saber qual a melhor forma de simular o comportamento de uma aplicação *Android* (se através das *Google WebView* ou através de uma aplicação nativa) e se a utilização da *framework* de facto compensa, deveria ser realizado um estudo junto de potenciais utilizadores, em vários ambientes, de modo a que os próprios utilizadores pudessem confirmar qual a aplicação que apresentaria melhor aspeto e comportamento face a um dispositivo real.

BIBLIOGRAFIA

[1] Bayer. Medrad® stellant® ct injection system with certegra® workstation. URL https://www.radiologysolutions.bayer.com/products/ct/injection/stellant.

[2] José Creissac Campos. *Prototipagem, Concepção Centrada no Utilzador*. 2017.

[3] International Data Corporation. Smartphone os market share, 2017 q1, 2017. URL https://www.idc.com/promo/smartphone-market-share/os.

[4] Mark Crane. 12 worst medical technology dangers. URL https://www.medscape.com/features/slideshow/tech-dangers.

[5] Serviço Médico de Imagem Computorizada (SMIC). Tomografia computorizada. URL http://www.smic.pt/exam/exames/tomografia-computorizada/.

[6] Google Developers. Android developers, . URL https://developer.android.com.

[7] Google Developers. Avd manager, . URL https://developer.android.com/studio/run/managing-avds.

[8] Google Developers. Android studio, . URL https://developer.android.com/studio/intro/.

[9] European Committee for Electrotechnical Standardization. Medical devices — application of usability engineering to medical devices. PDF, dec 2007.

[10] Google. Chrome custom tabs, . URL https://developer.chrome.com/multidevice/android/customtabs.

[11] Google. Webview for android, . URL https://developer.chrome.com/multidevice/webview/overview.

[12] Handlebars. Handlebarsjs. URL http://handlebarsjs.com.

[13] Lukas Lechner. Clickableareas. URL https://github.com/Lukle/ClickableAreasImages.

[14] José Creissac Campos; Michael D. Harrison; Paolo Masci. Formal modelling as a component of user centred design. URL https://haslab.uminho.pt/masci/files/fmis18rev2-camera-ready.pdf.

[15] Masimo. Radical-7® pulse co-oximeter®. URL http://www.masimo.com/products/continuous/radical-7/.

[16] Clark S. Turner Nancy G. Leveson. An investigation of the therac-25 accidents, July 1993. URL https://web.stanford.edu/class/cs240/old/sp2014/readings/therac-25.pdf.

[17] Tim Reeves. Our medical error index, jun 2010. URL http://www.humanfactorsmd.com/our-medical-error-index/.

[18] Adam Sinicki. Best android developer tools, jun 2017. URL https://www.androidauthority.com/best-android-developer-tools-671650/.

[19] Judy Crow; Sam Owre; John Rushby; Natarajan Shankar; Mandayam Srivas. A tutorial introduction to pvs. Presented at Wift'95: Workshop on Industria-Strength Formal, jun 1995. URL https://www.cs.umd.edu/~mvz/handouts/wift-tutorial.pdf.

[20] TechTerms. User interface definition. URL https://techterms.com/definition/user_interface.

[21] Patrick Oladimeji; Paolo Masci; Paul Curzon; Harold Thimbleby. Pvsio-web: a tool for rapid prototyping device user interfaces in pvs. URL https://journal.ub.tu-berlin.de/eceasst/article/view/963.

[22] Steve Upton. Four key uses of prototyping, feb 2010. URL https://www.moldmakingtechnology.com/articles/why-is-prototyping-important.

A

TRADUÇÃO PVS-*JAVA*

A.1 CÓDIGO DO MODELO PVS DO STELLANT V2

```
1  main: THEORY
2 BEGIN
3
4  Mode: TYPE = { OFF, INIT, INIT_SYRINGE, INIT_COMPLETE, AUTO, MANUAL,
5              READY_TO_PRIME, PRIMING, CONFIRM_PRIME, INFUSING,
                   INFUSION_COMPLETE }
6  AutoloadMode: TYPE = { LOAD30, MAKE_EMPTY, FILL_VOLUME, WAIT5, FINALIZE,
       DONE }
7
8  InfuseMode: TYPE = { NIL, READY, INFUSING_CONTRAST, INFUSING_SALINE,
       COMPLETE, PAUSE, STOP }
9
10 ConsoleScreen: TYPE = { CONSOLE_INIT, CONSOLE_SECURITY, CONSOLE_PROTOCOL }
11 Protocol: TYPE = { TOTAL_BODY_BARG }
12
13 ConsoleLED: TYPE = { ORANGE, GREEN, LED_OFF }
14
15 MAX_VOLUME: nat = 230 %-- mL
16 VOL_BUFFER: nat = 10 %-- mL
17 Volume: TYPE = upto(MAX_VOLUME) %-- mL
18 MAX_RATE: nat = 200 %-- mL/sec
19 Rate: TYPE = { x: nonneg_real | x < MAX_RATE } %-- mL/sec
20 Time: TYPE = nonneg_real %-- sec
21 LED: TYPE = { BLINKING, DARK, LIGHT, BLINK3 }
22 ConsoleButton: TYPE = { PRESSED, IDLE }
23
24 PlungerLevel: upto(MAX_VOLUME)
25 % plunger speed
26 FAST: Volume = 10
27 SLOW: Volume = 1
28
29 Display: TYPE = { DISP_OFF, DISP_INIT, MIRROR_PLUNGER_LEVEL,
       MIRROR_TARGET_VOLUME }
```

```
30
31  DEFAULT_VOLUME_SALINE: Volume = 224
32  DEFAULT_VOLUME_CONTRAST: Volume = 224
33  AUTOLOAD_STEP: Volume = 6
34  PRIME_VOLUME_SALINE: Volume = 4
35  PRIME_VOLUME_CONTRAST: Volume = 1
36
37  tick_step: posreal = 250 %-- millis
38  BTN_ACC_TIMEOUT: posreal = 750 %-- millis
39  BTN_MANUAL_TIMEOUT: posreal = 3000 %-- millis
40  BTN_AUTO_TIMEOUT: posreal = 8000 %-- millis
41
42  ConsoleCMD: TYPE = { LOCK, ENGAGE, DISENGAGE }
43  ConsoleDLG: TYPE = { NIL, ASK_CONFIRM_AIR_CHECK, VOLUME_WARNING,
        MSG_INFUSION_COMPLETE }
44
45  state: TYPE = [#
46    mode: Mode,
47    autoload_mode_saline: AutoloadMode,
48    autoload_mode_contrast: AutoloadMode,
49    syringe_saline_present: bool,
50    syringe_contrast_present: bool,
51    plunger_saline: Volume,
52    plunger_contrast: Volume,
53    display_saline: Display,
54    display_contrast: Display,
55    vol_saline: Volume,
56    vol_contrast: Volume,
57    vol_saline_confirmed: bool,
58    vol_contrast_confirmed: bool,
59    lock_LED: LED,
60    infusion_contrast_LED: LED,
61    infusion_saline_LED: LED,
62    btn_fill_saline: LED,
63    btn_fill_contrast: LED,
64    btn_auto: LED,
65    btn_manual: LED,
66    btn_prime: LED,
67    btn_confirm: LED,
68    btn_engage: LED,
69    btn_manual_timeout: nonneg_real,
70    btn_auto_timeout_saline: real, %-- -1 disables the timeout
71    btn_auto_timeout_contrast: real, %-- -1 disables the timeout
72    timeout_autoload_saline: nonneg_real,
73    timeout_autoload_contrast: nonneg_real,
74    prime_confirmed: bool,
75    prime_volume_saline: Volume,
```

```
76      prime_volume_contrast: Volume,
77      prime_warning: bool, %-- this is used to handle the warning given by the
            injector when trying to arm without activating the air-in-line check
            button first
78      lock_warning: bool,
79      armed: bool,
80      vol_saline_infused: Volume,
81      vol_contrast_infused: Volume,
82      infuse_mode: InfuseMode,
83      injector_rotated: bool,
84      %-- console
85      console_btn_timeout: nonneg_real, % millis
86      console_btn_ACC: ConsoleButton,
87      console_LED_ACC: ConsoleLED,
88      console_screen: ConsoleScreen,
89      console_protocol: Protocol,
90      console_vol_saline: Volume, % mL
91      console_vol_contrast: Volume, % mL
92      console_rate_saline: Rate, % mL/sec
93      console_rate_contrast: Rate, % mL/sec
94      console_time_saline: Time, % sec
95      console_time_contrast: Time, % sec
96      console_cmd: ConsoleCMD,
97      console_dlg: ConsoleDLG,
98      console_locked: bool,
99      %-- bags
100     bag_saline_present: bool,
101     bag_contrast_present: bool,
102     fluid_in_saline_syringe: bool,
103     fluid_in_contrast_syringe: bool,
104     %-- infusion set
105     infusion_set_present: bool
106   #]
107
108   init(x: real): state = (#
109     mode := OFF,
110     autoload_mode_saline := LOAD30,
111     autoload_mode_contrast := LOAD30,
112     syringe_saline_present := FALSE,
113     syringe_contrast_present := FALSE,
114     plunger_saline := DEFAULT_VOLUME_SALINE,
115     plunger_contrast := DEFAULT_VOLUME_SALINE,
116     display_saline := DISP_OFF,
117     display_contrast := DISP_OFF,
118     vol_saline := 0,
119     vol_contrast := 0,
120     vol_saline_confirmed := FALSE,
```

```
121     vol_contrast_confirmed := FALSE,
122     lock_LED := DARK,
123     infusion_contrast_LED := DARK,
124     infusion_saline_LED := DARK,
125     btn_fill_saline := DARK,
126     btn_fill_contrast := DARK,
127     btn_auto := DARK,
128     btn_manual := DARK,
129     btn_prime := DARK,
130     btn_confirm := DARK,
131     btn_engage := DARK,
132     btn_manual_timeout := 0,
133     btn_auto_timeout_saline := 0,
134     btn_auto_timeout_contrast := 0,
135     timeout_autoload_saline := 0,
136     timeout_autoload_contrast := 0,
137     prime_confirmed := FALSE,
138     prime_volume_saline := 0,
139     prime_volume_contrast := 0,
140     prime_warning := FALSE,
141     lock_warning := FALSE,
142     armed := FALSE,
143     vol_saline_infused := 0,
144     vol_contrast_infused := 0,
145     infuse_mode := NIL,
146     injector_rotated := FALSE,
147     console_btn_timeout := 0,
148     console_btn_ACC := IDLE,
149     console_LED_ACC := ORANGE,
150     console_screen := CONSOLE_INIT,
151     console_protocol := TOTAL_BODY_BARG,
152     console_vol_saline := 45,
153     console_vol_contrast := 100,
154     console_rate_saline := 4,
155     console_rate_contrast := 4,
156     console_time_saline := 45/4,
157     console_time_contrast := 100/4,
158     console_cmd := LOCK,
159     console_dlg := NIL,
160     console_locked := FALSE,
161     bag_saline_present := FALSE,
162     bag_contrast_present := FALSE,
163     fluid_in_saline_syringe := FALSE,
164     fluid_in_contrast_syringe := FALSE,
165     infusion_set_present := FALSE
166   #)
167
```

```
168  step: Volume = 5
169  inc(x: Volume): Volume =
170   COND
171    x + step <= MAX_VOLUME -> x + step,
172    ELSE -> MAX_VOLUME
173   ENDCOND
174  dec(x: Volume): Volume =
175   COND
176    x - step >= 0 -> x - step,
177    ELSE -> 0
178   ENDCOND
179
180  %-- contrast
181  per_inc_contrast(st: state): bool = (mode(st) /= OFF AND NOT
         vol_contrast_confirmed(st))
182  inc_contrast(st: (per_inc_contrast)): state = st WITH [ vol_contrast := inc(
         vol_contrast(st)) ]
183  click_inc_contrast(st: state): state =
184    COND
185      per_inc_contrast(st) -> inc_contrast(st),
186      ELSE -> st
187    ENDCOND
188  press_inc_contrast(st: state): state = click_inc_contrast(st)
189  release_inc_contrast(st: state): state = st
190
191  per_dec_contrast(st: state): bool = (mode(st) /= OFF AND NOT
         vol_contrast_confirmed(st))
192  dec_contrast(st: (per_inc_contrast)): state = st WITH [ vol_contrast := dec(
         vol_contrast(st)) ]
193  click_dec_contrast(st: state): state =
194    COND
195      per_dec_contrast(st) -> dec_contrast(st),
196      ELSE -> st
197    ENDCOND
198  press_dec_contrast(st: state): state = click_dec_contrast(st)
199  release_dec_contrast(st: state): state = st
200
201  %--saline
202  per_inc_saline(st: state): bool = (mode(st) /= OFF AND NOT
         vol_saline_confirmed(st))
203  inc_saline(st: (per_inc_saline)): state = st WITH [ vol_saline := inc(
         vol_saline(st)) ]
204  click_inc_saline(st: state): state =
205    COND
206      per_inc_saline(st) -> inc_saline(st),
207      ELSE -> st
208    ENDCOND
```

```
209  press_inc_saline(st: state): state = click_inc_saline(st)
210  release_inc_saline(st: state): state = st
211
212  per_dec_saline(st: state): bool = (mode(st) /= OFF AND NOT
         vol_saline_confirmed(st))
213  dec_saline(st: (per_inc_saline)): state = st WITH [ vol_saline := dec(
         vol_saline(st)) ]
214  click_dec_saline(st: state): state =
215    COND
216      per_dec_saline(st) -> dec_saline(st),
217      ELSE -> st
218    ENDCOND
219  press_dec_saline(st: state): state = click_dec_saline(st)
220  release_dec_saline(st: state): state = st
221
222  volumes_confirmed?(st: state): bool = vol_saline_confirmed(st) AND
         vol_contrast_confirmed(st)
223  set_LED_state(st: state): state =
224    st WITH [
225      lock_LED := IF mode(st) = OFF THEN DARK
226                  ELSE IF lock_warning(st) THEN BLINK3
227                     ELSIF console_locked(st) THEN LIGHT
228           ELSE DARK ENDIF ENDIF,
229      infusion_contrast_LED := COND
230                                  armed(st) AND infuse_mode(st) = READY ->
                                        BLINKING,
231         armed(st) AND infuse_mode(st) = INFUSING_CONTRAST -> LIGHT,
232               ELSE -> DARK ENDCOND,
233      infusion_saline_LED := COND
234                                  armed(st) AND infuse_mode(st) = READY ->
                                        BLINKING,
235         armed(st) AND infuse_mode(st) = INFUSING_SALINE -> LIGHT,
236               ELSE -> DARK ENDCOND,
237      btn_fill_contrast := IF mode(st) = AUTO
238                             THEN COND vol_contrast_confirmed(st) AND
                                    autoload_mode_contrast(st) /= DONE -> LIGHT,
239                 NOT vol_contrast_confirmed(st) AND btn_auto_timeout_contrast(
                        st) > 0 -> BLINKING,
240                                ELSE -> DARK ENDCOND
241         ELSE DARK ENDIF,
242      btn_fill_saline := IF mode(st) = AUTO
243                           THEN COND vol_saline_confirmed(st) AND
                                  autoload_mode_saline(st) /= DONE -> LIGHT,
244                 NOT vol_saline_confirmed(st) AND btn_auto_timeout_saline(st)
                        > 0 -> BLINKING,
245                                ELSE -> DARK ENDCOND
246             ELSE DARK ENDIF,
```

```
247       btn_confirm := IF mode(st) = OFF THEN DARK
248                      ELSE IF prime_warning(st) THEN BLINK3
249              ELSIF prime_confirmed(st) THEN LIGHT
250         ELSE DARK ENDIF ENDIF,
251       btn_manual := IF mode(st) = MANUAL THEN LIGHT ELSE DARK ENDIF,
252       %-- the following three LEDS are always dark? if that's the case, we can
              remove them from the model
253       btn_auto := DARK,
254       btn_prime := DARK,
255       btn_engage := DARK ]
256
257
258
259   %-- auto load
260   per_btn_auto(st: state): bool = (mode(st) = INIT_COMPLETE OR mode(st) =
          CONFIRM_PRIME OR mode(st) = READY_TO_PRIME)
261                                     AND (bag_saline_present(st) AND
                                            bag_contrast_present(st))
262   click_btn_auto(st: (per_btn_auto)): state =
263    COND
264     per_btn_auto(st)
265       -> LET st = st WITH [ mode := AUTO,
266                             vol_saline := IF console_vol_saline(st) +
                                               PRIME_VOLUME_SALINE < MAX_VOLUME THEN
                                               console_vol_saline(st) + PRIME_VOLUME_SALINE ELSE
                                               MAX_VOLUME ENDIF,
267           vol_contrast := IF console_vol_contrast(st) + PRIME_VOLUME_CONTRAST
                  < MAX_VOLUME THEN console_vol_contrast(st) +
                  PRIME_VOLUME_CONTRAST ELSE MAX_VOLUME ENDIF,
268                  vol_saline_confirmed := FALSE,
269                  vol_contrast_confirmed := FALSE,
270                               display_saline := MIRROR_TARGET_VOLUME,
271           display_contrast := MIRROR_TARGET_VOLUME,
272           autoload_mode_contrast := IF plunger_contrast(st) > 0 THEN
                  FILL_VOLUME ELSE LOAD30 ENDIF,
273           autoload_mode_saline := IF plunger_saline(st) > 0 THEN FILL_VOLUME
                  ELSE LOAD30 ENDIF,
274           prime_volume_saline := 0,
275           prime_volume_contrast := 0,
276           %prime_confirmed := FALSE,
277           btn_auto_timeout_saline := BTN_AUTO_TIMEOUT,
278           btn_auto_timeout_contrast := BTN_AUTO_TIMEOUT ]
279     IN set_LED_state(st),
280     ELSE -> st
281    ENDCOND
282
```

```
283  per_btn_manual(st: state): bool = (mode(st) = INIT_COMPLETE OR mode(st) =
         CONFIRM_PRIME OR mode(st) = READY_TO_PRIME)
284  click_btn_manual(st: (per_btn_manual)): state =
285   COND
286    per_btn_manual(st)
287      -> LET st = st WITH [ mode := MANUAL,
288                            vol_saline := plunger_saline(st),
289           vol_contrast := plunger_contrast(st),
290                 vol_saline_confirmed := FALSE,
291                 vol_contrast_confirmed := FALSE,
292           btn_manual_timeout := BTN_MANUAL_TIMEOUT ]
293    IN set_LED_state(st),
294    ELSE -> st
295   ENDCOND
296
297  per_btn_fill_saline(st: state): bool = (mode(st) = AUTO)
298  click_btn_fill_saline(st: (per_btn_fill_saline)): state =
299   COND
300    per_btn_fill_saline(st)
301     -> LET st = st WITH [ vol_saline_confirmed := TRUE,
302                           btn_auto_timeout_saline := -1 ] %-- disables
                                 timeout for saline auto button
303           IN set_LED_state(st),
304    ELSE -> st
305   ENDCOND
306
307  per_btn_fill_contrast(st: state): bool = (mode(st) = AUTO)
308  click_btn_fill_contrast(st: (per_btn_fill_contrast)): state =
309   COND
310    per_btn_fill_contrast(st)
311      -> LET st = st WITH [ vol_contrast_confirmed := TRUE,
312                            btn_auto_timeout_contrast := -1 ] %-- disables
                                 timeout for contrast auto button
313           IN set_LED_state(st),
314    ELSE -> st
315   ENDCOND
316
317  per_btn_prime(st: state): bool = (mode(st) = READY_TO_PRIME OR mode(st) =
         CONFIRM_PRIME)
318  click_btn_prime(st: (per_btn_prime)): state =
319   COND
320    per_btn_prime(st)
321      -> LET st = st WITH [ prime_volume_saline := PRIME_VOLUME_SALINE,
322                            prime_volume_contrast := PRIME_VOLUME_CONTRAST,
323           mode := PRIMING ]
324           IN set_LED_state(st),
325    ELSE -> st
```

```
326    ENDCOND
327
328   per_btn_console_lock(st: state): bool = per_btn_prime(st)
329   click_btn_console_lock(st: (per_btn_console_lock)): state =
330    COND
331     per_btn_console_lock(st) ->
332       LET st = st WITH [ console_locked := TRUE,
333                    console_cmd := ENGAGE ] %-- this is the next command that
                         can be sent from the console (a cyclic pattern LOCK ->
                         ENGAGE -> DISENGAGE is followed)
334        IN set_LED_state(st),
335     ELSE -> st
336    ENDCOND
337
338   per_btn_console_engage(st: state): bool = (console_cmd(st) = ENGAGE)
339   click_btn_console_engage(st: (per_btn_console_engage)): state =
340    COND
341     per_btn_console_engage(st) AND prime_confirmed(st) ->
342       COND
343         console_vol_saline(st) > plunger_saline(st)
344     OR console_vol_contrast(st) > plunger_contrast(st)
345           -> LET st = st WITH [ console_dlg := VOLUME_WARNING ]
346          IN set_LED_state(st),
347   ELSE -> LET st = st WITH [ console_cmd := DISENGAGE, %-- this is the next
       command that can be sent from the console (a cyclic pattern LOCK ->
       ENGAGE -> DISENGAGE is followed)
348                              armed := TRUE,
349           infuse_mode := READY ]
350      IN set_LED_state(st)
351       ENDCOND,
352     per_btn_console_engage(st) AND NOT prime_confirmed(st) ->
353       LET st = st WITH [ console_dlg := ASK_CONFIRM_AIR_CHECK ]
354        IN set_LED_state(st),
355     ELSE -> st
356    ENDCOND
357
358   per_btn_engage(st: state): bool =
359           mode(st) /= OFF AND mode(st) /= INIT AND mode(st) /= INIT_SYRINGE
360       AND mode(st) /= INIT_COMPLETE AND mode(st) /= AUTO
361       AND (console_cmd(st) = ENGAGE OR console_cmd(st) = LOCK)
362   click_btn_engage(st: (per_btn_engage)): state =
363     COND
364      per_btn_engage(st)
365        -> COND
366             console_cmd(st) = ENGAGE -> click_btn_console_engage(st), %-- the
                 behavior of the button on the injector mirrors the behavior of
                  the corresponding button on the console
```

```
367             console_cmd(st) = LOCK -> IF prime_confirmed(st) = FALSE
368                                       THEN st WITH [ btn_confirm := BLINK3,
                                                         prime_warning := TRUE ]
369                                  ELSE st WITH [ lock_LED := BLINK3, lock_warning := TRUE ]
                                         ENDIF,
370           ELSE -> st
371     ENDCOND,
372   ELSE -> st
373    ENDCOND
374
375   per_btn_confirm(st: state): bool = per_btn_engage(st)
376   click_btn_confirm(st: (per_btn_confirm)): state =
377    COND
378     per_btn_confirm(st)
379       -> LET st = st WITH [ prime_confirmed := NOT prime_confirmed(st) ]
380           IN set_LED_state(st),
381     ELSE -> st
382    ENDCOND
383
384
385   per_btn_console_disengage(st: state): bool = mode(st) /= INFUSING AND (
          console_cmd(st) = DISENGAGE)
386   click_btn_console_disengage(st: (per_btn_console_disengage)): state =
387    COND
388     per_btn_console_disengage(st) ->
389       LET st = st WITH [ mode := CONFIRM_PRIME,
390                          console_locked := FALSE,
391                    console_cmd := LOCK, %-- this is the next command that can
                           be sent from the console (a cyclic pattern LOCK ->
                           ENGAGE -> DISENGAGE is followed)
392                    armed := FALSE,
393        infuse_mode := NIL ]
394       IN set_LED_state(st),
395     ELSE -> st
396    ENDCOND
397
398 %-- these utility functions automatically stop pulling the plunger when the
       target volume has been reached
399   per_empty_saline(st: state): bool = (mode(st) = INIT_SYRINGE)
400   empty_saline(step: Volume)(st: state): state =
401     COND
402       per_empty_saline(st) ->
403         st WITH [ plunger_saline := IF plunger_saline(st) - step > 0
404                                     THEN plunger_saline(st) - step
405             ELSE 0 ENDIF ],
406       ELSE -> st
407     ENDCOND
```

```
408   per_empty_contrast(st: state): bool = (mode(st) = INIT_SYRINGE)
409   empty_contrast(step: Volume)(st: state): state =
410     COND
411       per_empty_contrast(st) ->
412         st WITH [ plunger_contrast := IF plunger_contrast(st) - step > 0
413                                   THEN plunger_contrast(st) - step
414                 ELSE 0 ENDIF ],
415       ELSE -> st
416     ENDCOND
417
418   per_auto(st: state): bool = (mode(st) = AUTO)
419   auto_plunger_saline(step: Volume)(st: (per_auto)): state =
420     LET st = st WITH [ display_saline := MIRROR_PLUNGER_LEVEL ] IN
421     COND
422       per_auto(st) ->
423        COND
424   autoload_mode_saline(st) = LOAD30 ->
425     IF plunger_saline(st) + AUTOLOAD_STEP <= 30 THEN st WITH [ plunger_saline
          := plunger_saline(st) + AUTOLOAD_STEP ]
426           ELSE st WITH [ autoload_mode_saline := MAKE_EMPTY ] ENDIF,
427         autoload_mode_saline(st) = MAKE_EMPTY ->
428           IF plunger_saline(st) - AUTOLOAD_STEP >= 0 THEN st WITH [
                plunger_saline := plunger_saline(st) - AUTOLOAD_STEP ]
429     ELSE st WITH [ autoload_mode_saline := FILL_VOLUME ] ENDIF,
430         autoload_mode_saline(st) = FILL_VOLUME ->
431           LET target_vol: Volume = IF vol_saline(st) + VOL_BUFFER < MAX_VOLUME
                THEN vol_saline(st) + VOL_BUFFER ELSE MAX_VOLUME ENDIF
432            IN IF plunger_saline(st) + step <= target_vol THEN st WITH [
                plunger_saline := plunger_saline(st) + step ]
433         ELSE st WITH [ plunger_saline := target_vol, autoload_mode_saline :=
              WAIT5, timeout_autoload_saline := 5 ] ENDIF,
434   autoload_mode_saline(st) = WAIT5 ->
435     LET timeout = IF timeout_autoload_saline(st) - 1 > 0 THEN
          timeout_autoload_saline(st) - 1 ELSE 0 ENDIF
436      IN st WITH [ timeout_autoload_saline := timeout, autoload_mode_saline :=
           IF timeout > 0 THEN autoload_mode_saline(st) ELSE FINALIZE ENDIF ],
437         autoload_mode_saline(st) = FINALIZE ->
438           LET st = st WITH [ plunger_saline := IF plunger_saline(st) - step >=
                vol_saline(st) THEN plunger_saline(st) - step ELSE vol_saline(
               st) ENDIF ]
439      IN st WITH [ autoload_mode_saline := IF plunger_saline(st) = vol_saline(
           st) THEN DONE ELSE FINALIZE ENDIF ],
440   autoload_mode_saline(st) = DONE ->
441     st
442        ENDCOND,
443       ELSE -> st
444     ENDCOND
```

```
445  auto_plunger_contrast(step: Volume)(st: (per_auto)): state =
446    LET st = st WITH [ display_contrast := MIRROR_PLUNGER_LEVEL ] IN
447    COND
448      autoload_mode_contrast(st) = LOAD30 ->
449        IF plunger_contrast(st) + AUTOLOAD_STEP <= 30 THEN st WITH [
               plunger_contrast := plunger_contrast(st) + AUTOLOAD_STEP ]
450        ELSE st WITH [ autoload_mode_contrast := MAKE_EMPTY ] ENDIF,
451      autoload_mode_contrast(st) = MAKE_EMPTY ->
452        IF plunger_contrast(st) - AUTOLOAD_STEP >= 0 THEN st WITH [
               plunger_contrast := plunger_contrast(st) - AUTOLOAD_STEP ]
453  ELSE st WITH [ autoload_mode_contrast := FILL_VOLUME ] ENDIF,
454      autoload_mode_contrast(st) = FILL_VOLUME ->
455        LET target_vol: Volume = IF vol_contrast(st) + VOL_BUFFER < MAX_VOLUME
               THEN vol_contrast(st) + VOL_BUFFER ELSE MAX_VOLUME ENDIF
456         IN IF plunger_contrast(st) + step <= target_vol THEN st WITH [
               plunger_contrast := plunger_contrast(st) + step ]
457      ELSE st WITH [ plunger_contrast := target_vol, autoload_mode_contrast :=
             WAIT5, timeout_autoload_contrast := 5 ] ENDIF,
458      autoload_mode_contrast(st) = WAIT5 ->
459    LET timeout = IF timeout_autoload_contrast(st) - 1 > 0 THEN
          timeout_autoload_contrast(st) - 1 ELSE 0 ENDIF
460     IN st WITH [ timeout_autoload_contrast := timeout, autoload_mode_contrast
           := IF timeout > 0 THEN autoload_mode_contrast(st) ELSE FINALIZE
          ENDIF ],
461      autoload_mode_contrast(st) = FINALIZE ->
462        LET st = st WITH [ plunger_contrast := IF plunger_contrast(st) - step
               >= vol_contrast(st) THEN plunger_contrast(st) - step ELSE
               vol_contrast(st) ENDIF ]
463   IN st WITH [ autoload_mode_contrast := IF plunger_contrast(st) =
         vol_contrast(st) THEN DONE ELSE FINALIZE ENDIF ],
464      ELSE -> st
465    ENDCOND
466
467  per_prime_syringes(st: state): bool = (mode(st) = PRIMING)
468  prime_syringes(st: (per_prime_syringes)): state =
469    COND
470      mode(st) = PRIMING ->
471        LET st = st WITH [ plunger_saline := IF prime_volume_saline(st) > 0
472                                          THEN IF plunger_saline(st) - 1 > 0 THEN
                                                    plunger_saline(st) - 1 ELSE 0
                                                 ENDIF
473                ELSE plunger_saline(st) ENDIF,
474          prime_volume_saline := IF prime_volume_saline(st) - 1 > 0 THEN
               prime_volume_saline(st) - 1 ELSE 0 ENDIF ],
475      st = st WITH [ plunger_contrast := IF prime_volume_contrast(st) > 0
476                                 THEN IF plunger_contrast(st) - 1 > 0 THEN
                                         plunger_contrast(st) - 1 ELSE 0 ENDIF
```

```
477                 ELSE plunger_contrast(st) ENDIF,
478         prime_volume_contrast := IF prime_volume_contrast(st) - 1 > 0 THEN
                prime_volume_contrast(st) - 1 ELSE 0 ENDIF ]
479    IN st WITH [ mode := IF prime_volume_saline(st) = 0 AND
           prime_volume_contrast(st) = 0
480                       THEN CONFIRM_PRIME ELSE PRIMING ENDIF ],
481       ELSE -> st
482     ENDCOND
483
484  pull_plunger_saline(step: Volume)(st: state): state =
485    st WITH [ plunger_saline := IF plunger_saline(st) + step <= MAX_VOLUME
           THEN plunger_saline(st) + step ELSE MAX_VOLUME ENDIF ]
486  pull_plunger_contrast(step: Volume)(st: state): state =
487    st WITH [ plunger_contrast := IF plunger_contrast(st) + step <= MAX_VOLUME
            THEN plunger_contrast(st) + step ELSE MAX_VOLUME ENDIF ]
488  push_plunger_saline(step: Volume)(st: state): state =
489    st WITH [ plunger_saline := IF plunger_saline(st) - step >= 0 THEN
           plunger_saline(st) - step ELSE 0 ENDIF ]
490  push_plunger_contrast(step: Volume)(st: state): state =
491    st WITH [ plunger_contrast := IF plunger_contrast(st) - step > 0 THEN
           plunger_contrast(st) - step ELSE 0 ENDIF ]
492
493  %-- this is for the automatic mode
494  per_tick(st: state): bool = (mode(st) = INIT OR mode(st) = INIT_SYRINGE OR
       mode(st) = MANUAL
495                                OR mode(st) = READY_TO_PRIME %OR mode(st) =
                                       CONFIRM_PRIME
496          OR mode(st) = PRIMING OR mode(st) = INFUSING)
497  tick(st: (per_tick)): state =
498    %-- clear temporary warnings
499    LET st = IF prime_warning(st)
500              THEN st WITH [ prime_warning := FALSE ]
501              ELSE st ENDIF,
502  st = IF lock_warning(st)
503       THEN st WITH [ lock_warning := FALSE ]
504       ELSE st ENDIF
505     IN
506    COND
507     mode(st) = INIT ->
508       LET st = IF syringe_saline_present(st) OR syringe_contrast_present(st)
509                 THEN st WITH [ mode := INIT_SYRINGE ] ELSE st ENDIF
510  IN set_LED_state(st),
511     mode(st) = INIT_SYRINGE ->
512       LET st = IF syringe_saline_present(st) THEN empty_saline(FAST)(st) ELSE
              st ENDIF,
513           st = IF syringe_contrast_present(st) THEN empty_contrast(FAST)(st)
                 ELSE st ENDIF,
```

```
514     st = IF plunger_saline(st) = vol_saline(st) AND plunger_contrast(st) =
            vol_contrast(st)
515           THEN st WITH [ mode := INIT_COMPLETE ] ELSE st ENDIF
516   IN set_LED_state(st),
517     mode(st) = MANUAL ->
518       LET st = st WITH [ btn_manual_timeout := IF btn_manual_timeout(st) -
              tick_step > 0
519                                THEN btn_manual_timeout(st) - tick_step
520             ELSE 0 ENDIF],
521     st = IF btn_manual_timeout(st) = 0 THEN st WITH [ mode := READY_TO_PRIME
            ] ELSE st ENDIF
522   IN set_LED_state(st),
523     mode(st) = AUTO ->
524                LET %-- saline
525              st = IF btn_auto_timeout_saline(st) /= 0
526                               THEN LET st = IF vol_saline_confirmed(st)
527                                             THEN auto_plunger_saline(FAST)(st)
                                                   ELSE st ENDIF
528                          IN st WITH [ btn_auto_timeout_saline := COND
                                 btn_auto_timeout_saline(st) > 0
529                     -> IF btn_auto_timeout_saline(st) - tick_step > 0
530                                                THEN btn_auto_timeout_saline(
                                                       st) - tick_step
531                                              ELSE 0 ENDIF,
532                  ELSE -> btn_auto_timeout_saline(st) ENDCOND ]
533                          ELSE %-- timeout expired for auto_saline
534                        st WITH [ vol_saline := plunger_saline(st),
535                                              display_saline :=
                                                   MIRROR_PLUNGER_LEVEL ] ENDIF,
536       %-- contrast
537                           st = IF btn_auto_timeout_contrast(st) /= 0
538                                THEN LET st = IF vol_contrast_confirmed(st)
539                                              THEN auto_plunger_contrast(FAST)(
                                                     st) ELSE st ENDIF
540                          IN st WITH [ btn_auto_timeout_contrast := COND
                                 btn_auto_timeout_contrast(st) > 0
541                                    -> IF btn_auto_timeout_contrast(st) -
                                           tick_step > 0
542                                                 THEN btn_auto_timeout_contrast(
                                                        st) - tick_step
543                                     ELSE 0 ENDIF,
544                           ELSE -> btn_auto_timeout_contrast(st) ENDCOND ]
545                          ELSE %-- timeout expired for auto_contrast
546                        st WITH [ vol_contrast := plunger_contrast(st),
547                                              display_contrast :=
                                                   MIRROR_PLUNGER_LEVEL ] ENDIF,
548       %-- we decide here the next operating mode
```

```
549         st = COND
550                                  btn_auto_timeout_saline(st) <= 0 AND
                                        btn_auto_timeout_contrast(st) <= 0
551           AND autoload_mode_saline(st) = DONE AND autoload_mode_contrast(st) =
                  DONE
552                 AND plunger_saline(st) > 0 AND plunger_contrast(st) > 0
553          -> st WITH [ mode := READY_TO_PRIME,
554                       display_saline := MIRROR_PLUNGER_LEVEL,
555                 display_contrast := MIRROR_PLUNGER_LEVEL],
556
557          (btn_auto_timeout_saline(st) = 0 AND plunger_saline(st) = 0 AND
                 autoload_mode_contrast(st) = DONE)
558                 OR (btn_auto_timeout_contrast(st) = 0 AND plunger_contrast(st)
                        = 0 AND autoload_mode_saline(st) = DONE)
559            OR (btn_auto_timeout_saline(st) = 0 AND btn_auto_timeout_contrast(st
                  ) = 0)
560          -> st WITH [ mode := INIT_COMPLETE,
561                    display_saline := MIRROR_PLUNGER_LEVEL,
562                 display_contrast := MIRROR_PLUNGER_LEVEL],
563                ELSE -> st ENDCOND
564          IN set_LED_state(st),
565       mode(st) = PRIMING ->
566         LET st = prime_syringes(st)
567          IN set_LED_state(st),
568       mode(st) = INFUSING ->
569         LET st = COND
570                    vol_contrast_infused(st) < console_vol_contrast(st) AND
                          plunger_contrast(st) > 0
571         -> st WITH [ vol_contrast_infused := vol_contrast_infused(st) + 1,
572                      plunger_contrast := IF plunger_contrast(st) - 1 > 0
573                                          THEN plunger_contrast(st) - 1
574                   ELSE 0 ENDIF,
575          infuse_mode := INFUSING_CONTRAST ],
576       NOT (vol_contrast_infused(st) < console_vol_contrast(st) AND
             plunger_contrast(st) > 0)
577         AND (vol_saline_infused(st) < console_vol_saline(st) AND plunger_saline
                (st) > 0)
578         -> st WITH [ vol_saline_infused := vol_saline_infused(st) + 1,
579                      plunger_saline := IF plunger_saline(st) - 1 > 0
580                                        THEN plunger_saline(st) - 1
581                ELSE 0 ENDIF,
582          infuse_mode := INFUSING_SALINE ],
583       vol_contrast_infused(st) = console_vol_contrast(st)
584        AND vol_saline_infused(st) = console_vol_saline(st)
585         -> st WITH [ mode := INFUSION_COMPLETE,
586                      infuse_mode := COMPLETE,
587          console_dlg := MSG_INFUSION_COMPLETE ],
```

```
588       ELSE -> st
589                   ENDCOND
590          IN set_LED_state(st),
591       ELSE -> set_LED_state(st)
592      ENDCOND
593
594   %-- manual saline
595   per_sUP_saline(st: state): bool = (mode(st) = MANUAL)
596   sUP_saline(st: (per_sUP_saline)): state =
597    LET st = push_plunger_saline(SLOW)(st)
598     IN st WITH [ vol_saline := plunger_saline(st),
599            vol_contrast := plunger_contrast(st),
600       btn_manual_timeout := BTN_MANUAL_TIMEOUT ]
601   click_btn_sUP_saline(st: state): state =
602    COND
603      per_sUP_saline(st) -> sUP_saline(st),
604      ELSE -> st
605    ENDCOND
606   press_btn_sUP_saline(st: state): state = click_btn_sUP_saline(st)
607   release_btn_sUP_saline(st: state): state = st
608
609   per_fUP_saline(st: state): bool = (mode(st) = MANUAL)
610   fUP_saline(st: (per_fUP_saline)): state =
611    LET st = push_plunger_saline(FAST)(st)
612     IN st WITH [ vol_saline := plunger_saline(st),
613      vol_contrast := plunger_contrast(st),
614      btn_manual_timeout := BTN_MANUAL_TIMEOUT,
615      prime_confirmed := FALSE ]
616
617   click_btn_fUP_saline(st: state): state =
618    COND
619      per_fUP_saline(st) -> fUP_saline(st),
620      ELSE -> st
621    ENDCOND
622   press_btn_fUP_saline(st: state): state = click_btn_fUP_saline(st)
623   release_btn_fUP_saline(st: state): state = st
624
625   per_sDOWN_saline(st: state): bool = (mode(st) = MANUAL)
626   sDOWN_saline(st: (per_sDOWN_saline)): state =
627    LET st = pull_plunger_saline(SLOW)(st)
628     IN st WITH [ vol_saline := plunger_saline(st),
629      vol_contrast := plunger_contrast(st),
630      btn_manual_timeout := BTN_MANUAL_TIMEOUT,
631      prime_confirmed := FALSE ]
632
633   click_btn_sDOWN_saline(st: state): state =
634    COND
```

```
635       per_sDOWN_saline(st) -> sDOWN_saline(st),
636       ELSE -> st
637     ENDCOND
638   press_btn_sDOWN_saline(st: state): state = click_btn_sDOWN_saline(st)
639   release_btn_sDOWN_saline(st: state): state = st
640
641   per_fDOWN_saline(st: state): bool = (mode(st) = MANUAL)
642   fDOWN_saline(st: (per_fDOWN_saline)): state =
643     LET st = pull_plunger_saline(FAST)(st)
644      IN st WITH [ vol_saline := plunger_saline(st),
645       vol_contrast := plunger_contrast(st),
646       btn_manual_timeout := BTN_MANUAL_TIMEOUT,
647       prime_confirmed := FALSE ]
648
649   click_btn_fDOWN_saline(st: state): state =
650     COND
651       per_fDOWN_saline(st) -> fDOWN_saline(st),
652       ELSE -> st
653     ENDCOND
654   press_btn_fDOWN_saline(st: state): state = click_btn_fDOWN_saline(st)
655   release_btn_fDOWN_saline(st: state): state = st
656
657   %-- manual contrast
658   per_sUP_contrast(st: state): bool = (mode(st) = MANUAL)
659   sUP_contrast(st: (per_sUP_contrast)): state =
660     LET st = push_plunger_contrast(SLOW)(st)
661      IN st WITH [ vol_saline := plunger_saline(st),
662       vol_contrast := plunger_contrast(st),
663       btn_manual_timeout := BTN_MANUAL_TIMEOUT ]
664   click_btn_sUP_contrast(st: state): state =
665     COND
666       per_sUP_contrast(st) -> sUP_contrast(st),
667       ELSE -> st
668     ENDCOND
669   press_btn_sUP_contrast(st: state): state = click_btn_sUP_contrast(st)
670   release_btn_sUP_contrast(st: state): state = st
671
672   per_fUP_contrast(st: state): bool = (mode(st) = MANUAL)
673   fUP_contrast(st: (per_fUP_contrast)): state =
674     LET st = push_plunger_contrast(FAST)(st)
675      IN st WITH [ vol_saline := plunger_saline(st),
676       vol_contrast := plunger_contrast(st),
677       btn_manual_timeout := BTN_MANUAL_TIMEOUT ]
678   click_btn_fUP_contrast(st: state): state =
679     COND
680       per_fUP_contrast(st) -> fUP_contrast(st),
681       ELSE -> st
```

```
682   ENDCOND
683  press_btn_fUP_contrast(st: state): state = click_btn_fUP_contrast(st)
684  release_btn_fUP_contrast(st: state): state = st
685
686  per_sDOWN_contrast(st: state): bool = (mode(st) = MANUAL)
687  sDOWN_contrast(st: (per_sDOWN_contrast)): state =
688   LET st = pull_plunger_contrast(SLOW)(st),
689       st = st WITH [ vol_saline := plunger_saline(st),
690            vol_contrast := plunger_contrast(st),
691          btn_manual_timeout := BTN_MANUAL_TIMEOUT,
692          prime_confirmed := FALSE ]
693    IN set_LED_state(st)
694  click_btn_sDOWN_contrast(st: state): state =
695   COND
696     per_sDOWN_contrast(st) -> sDOWN_contrast(st),
697     ELSE -> st
698   ENDCOND
699  press_btn_sDOWN_contrast(st: state): state = click_btn_sDOWN_contrast(st)
700  release_btn_sDOWN_contrast(st: state): state = st
701
702  per_fDOWN_contrast(st: state): bool = (mode(st) = MANUAL)
703  fDOWN_contrast(st: (per_fDOWN_contrast)): state =
704   LET st = pull_plunger_contrast(FAST)(st),
705       st = st WITH [ vol_saline := plunger_saline(st),
706          vol_contrast := plunger_contrast(st),
707          btn_manual_timeout := BTN_MANUAL_TIMEOUT,
708          prime_confirmed := FALSE ]
709    IN set_LED_state(st)
710  click_btn_fDOWN_contrast(st: state): state =
711   COND
712     per_fDOWN_contrast(st) -> fDOWN_contrast(st),
713     ELSE -> st
714   ENDCOND
715  press_btn_fDOWN_contrast(st: state): state = click_btn_fDOWN_contrast(st)
716  release_btn_fDOWN_contrast(st: state): state = st
717
718  %-- the device does not have these buttons, but these definitions are useful
         to keep the model more compact
719  per_btn_on(st: state): bool = (mode(st) = OFF)
720  click_btn_on(st: (per_btn_on)): state =
721   COND
722     per_btn_on(st) ->
723       st WITH [ mode := INIT, display_contrast := DISP_INIT, display_saline
              := DISP_INIT ],
724     ELSE -> st
725   ENDCOND
726  %--
```

```
727
728  plug_syringe_saline(st: state): state =
729    COND console_screen(st) = CONSOLE_PROTOCOL ->
730         LET st = st WITH [ syringe_saline_present := TRUE ]
731          IN IF per_tick(st) THEN tick(st) ELSE st ENDIF,
732   ELSE -> st ENDCOND
733  plug_syringe_contrast(st: state): state =
734    COND console_screen(st) = CONSOLE_PROTOCOL ->
735         LET st = st WITH [ syringe_contrast_present := TRUE ]
736          IN IF per_tick(st) THEN tick(st) ELSE st ENDIF,
737   ELSE -> st ENDCOND
738  unplug_syringe_saline(st: state): state =
739    COND console_screen(st) = CONSOLE_PROTOCOL ->
740         LET st = st WITH [ syringe_saline_present := FALSE ]
741          IN IF per_tick(st) THEN tick(st) ELSE st ENDIF,
742   ELSE -> st ENDCOND
743  unplug_syringe_contrast(st: state): state =
744    COND console_screen(st) = CONSOLE_PROTOCOL ->
745         LET st = st WITH [ syringe_contrast_present := FALSE ]
746          IN IF per_tick(st) THEN tick(st) ELSE st ENDIF,
747   ELSE -> st ENDCOND
748
749  plug_bag_saline(st: state): state =
750   COND syringe_saline_present(st) AND plunger_saline(st) = 0 -> st WITH [
        bag_saline_present := TRUE, fluid_in_saline_syringe := TRUE ],
751     ELSE -> st ENDCOND
752  plug_bag_contrast(st: state): state =
753   COND syringe_contrast_present(st) AND plunger_contrast(st) = 0 -> st WITH [
         bag_contrast_present := TRUE, fluid_in_contrast_syringe := TRUE ],
754     ELSE -> st ENDCOND
755  unplug_bag_saline(st: state): state =
756   COND syringe_saline_present(st) -> st WITH [ bag_saline_present := FALSE ],
757     ELSE -> st ENDCOND
758  unplug_bag_contrast(st: state): state =
759   COND syringe_contrast_present(st) -> st WITH [ bag_contrast_present :=
        FALSE ],
760     ELSE -> st ENDCOND
761
762  connect_infusion_set(st: state): state =
763   COND (mode(st) = READY_TO_PRIME OR mode(st) = CONFIRM_PRIME OR (mode(st) =
        MANUAL AND plunger_contrast(st) > 0 AND plunger_saline(st) > 0)) AND
        bag_contrast_present(st) AND bag_saline_present(st) ->
764    st WITH [ infusion_set_present := TRUE,
765     bag_contrast_present := FALSE,
766    bag_saline_present := FALSE ],
767     ELSE -> st ENDCOND
768
```

```
769  restart_simulation(st: state): state = init(0);
770
771  %-- console commands
772  per_press_btn_ACC(st: state): bool = (console_screen(st) = CONSOLE_INIT)
773  press_btn_ACC(st: (per_press_btn_ACC)): state =
774   COND
775     console_btn_ACC(st) /= PRESSED ->
776       st WITH [ console_btn_ACC := PRESSED,
777                 console_btn_timeout := BTN_ACC_TIMEOUT ],
778     console_btn_ACC(st) = PRESSED AND console_btn_timeout(st) > 0  ->
779       st WITH [ console_btn_timeout := IF console_btn_timeout(st) - tick_step
              > 0
780                                  THEN console_btn_timeout(st) - tick_step
781           ELSE 0 ENDIF ],
782     %-- ON
783     console_btn_ACC(st) = PRESSED AND console_btn_timeout(st) = 0 AND mode(st
          ) = OFF ->
784       LET st = st WITH [ console_screen := CONSOLE_SECURITY, console_LED_ACC
             := GREEN, console_btn_ACC := IDLE ]
785        IN click_btn_on(st),
786     %-- OFF
787     console_btn_ACC(st) = PRESSED AND console_btn_timeout(st) = 0 AND mode(st
          ) /= OFF AND console_screen(st) /= CONSOLE_SECURITY AND mode(st) /=
          INFUSING ->
788       init(0) WITH [
789         injector_rotated := injector_rotated(st)
790       ],
791     ELSE -> st
792   ENDCOND
793  per_release_btn_ACC(st: state): bool = (console_btn_ACC(st) = PRESSED)
794  release_btn_ACC(st: (per_release_btn_ACC)): state =
795   COND
796     per_release_btn_ACC(st) -> st WITH [ console_btn_ACC := IDLE,
          console_btn_timeout := 0 ],
797     ELSE -> st
798   ENDCOND
799
800  %-- quick workaround for Andre's demo
801  click_btn_ACC(st: state): state =
802   COND
803     %-- ON
804     mode(st) = OFF ->
805       LET st = st WITH [ console_screen := CONSOLE_SECURITY, console_LED_ACC
             := GREEN, console_btn_ACC := IDLE ]
806        IN click_btn_on(st),
807     %-- OFF
```

```
808       mode(st) /= OFF AND console_screen(st) /= CONSOLE_SECURITY AND mode(st)
              /= INFUSING ->
809         init(0) WITH [
810           injector_rotated := injector_rotated(st)
811         ],
812       ELSE -> st
813     ENDCOND
814
815
816   per_click_btn_confirm_security(st: state): bool = (console_screen(st) =
          CONSOLE_SECURITY AND mode(st) = INIT)
817   click_btn_confirm_security(st: (per_click_btn_confirm_security)): state =
818     COND
819       per_click_btn_confirm_security(st) ->
820         LET st = st WITH [ console_screen := CONSOLE_PROTOCOL ],
821             st = st WITH [ display_saline := MIRROR_PLUNGER_LEVEL,
                    display_contrast := MIRROR_PLUNGER_LEVEL ]
822   IN tick(st),
823       ELSE -> st
824     ENDCOND
825
826   per_confirm_air_check(st: state): bool = (per_btn_confirm(st) AND
          console_dlg(st) = ASK_CONFIRM_AIR_CHECK)
827   click_btn_confirm_air_check_ok(st: (per_confirm_air_check)): state =
828     COND
829       per_confirm_air_check(st) ->
830         LET st = click_btn_confirm(st),
831             st = click_btn_engage(st)
832   IN st WITH [ console_dlg := IF console_dlg(st) = ASK_CONFIRM_AIR_CHECK THEN
          NIL ELSE console_dlg(st) ENDIF ],
833       ELSE -> st
834     ENDCOND
835
836   click_btn_confirm_air_check_fail(st: (per_confirm_air_check)): state =
837     COND
838       per_confirm_air_check(st) -> st WITH [ console_dlg := NIL ],
839       ELSE -> st
840     ENDCOND
841
842   per_volume_warning(st: state): bool = (console_dlg(st) = VOLUME_WARNING)
843   click_btn_confirm_volume_warning_ok(st: (per_volume_warning)): state =
844     COND
845       per_volume_warning(st) ->
846         LET st = st WITH [
847                     console_vol_contrast := plunger_contrast(st),
848                     vol_contrast := plunger_contrast(st),
```

```
849               console_time_contrast := IF console_rate_contrast(st) > 0 THEN
                      plunger_contrast(st) / console_rate_contrast(st) ELSE 0
                      ENDIF,
850               console_vol_saline := plunger_saline(st),
851               vol_saline := plunger_saline(st),
852               console_time_saline := IF console_rate_saline(st) > 0 THEN
                      plunger_saline(st) / console_rate_saline(st) ELSE 0 ENDIF,
853               console_locked := FALSE,
854               console_cmd := LOCK, %-- this is the next command that can be
                      sent from the console (a cyclic pattern LOCK -> ENGAGE ->
                      DISENGAGE is followed)
855               console_dlg := NIL ]
856        IN set_LED_state(st),
857     ELSE -> st
858   ENDCOND
859
860  click_btn_confirm_volume_warning_fail(st: (per_volume_warning)): state =
861   COND
862     per_volume_warning(st) -> LET st = st WITH [ console_cmd := ENGAGE, %--
          the button on the console stays at 'engage'
863                                                 armed := FALSE,
864               infuse_mode := NIL,
865                                                 console_dlg := NIL ]
866        IN set_LED_state(st),
867     ELSE -> st
868   ENDCOND
869
870  per_start(st: state): bool = (mode(st) = CONFIRM_PRIME OR mode(st) =
       READY_TO_PRIME) AND armed(st)
871  click_btn_start(st: (per_start)): state =
872    COND
873      per_start(st)
874        -> LET st = st WITH [ mode := INFUSING,
875                              vol_saline_infused := 0,
876             vol_contrast_infused := 0 ] IN set_LED_state(st),
877      ELSE -> st
878    ENDCOND
879
880  click_btn_console_start(st: (per_start)): state = click_btn_start(st) %--
       the console start button mirrors the start button on the injector
881
882
883  rotate_injector(st: state): state = st WITH [ injector_rotated := NOT
       injector_rotated(st) ]
884
885  %---
886  init_precache: state =
```

```
887     LET cache = click_btn_confirm_security(init(0) WITH [ console_screen :=
            CONSOLE_SECURITY, mode := INIT ])
888      IN init(0)
889
890  END main
```

A.2 CÓDIGO JAVA DA APLICAÇÃO ANDROID DO STELLANT V2

```
1   package at.lukle.clickableareas;
2
3
4  /**
5   * Created by andrepinto on 26/09/17.
6   */
7
8  public class State {
9
10
11      final int MAX_VOLUME = 230 ;
12      final int VOL_BUFFER= 10 ;
13      final int MAX_RATE = 200 ;
14
15      int PlungerLevel = 0; // upto(MAX_VOLUME)
16
17      // plunger speed
18      final int FAST = 10;
19      final int SLOW = 1;
20
21      final int DEFAULT_VOLUME_SALINE = 224;
22      final int DEFAULT_VOLUME_CONTRAST = 224;
23      final int AUTOLOAD_STEP = 6;
24      final int PRIME_VOLUME_SALINE = 3;
25      final int PRIME_VOLUME_CONTRAST = 1;
26
27      final float tick_step = 250; // posreal
28      final float BTN_ACC_TIMEOUT = 750; // posreal
29      final float BTN_MANUAL_TIMEOUT = 3000; //posreal
30      final float BTN_AUTO_TIMEOUT = 8000; // posreal
31
32      final int step = 5;
33
34      public enum Mode{ OFF,
35          INIT, INIT_SYRINGE,
36          INIT_COMPLETE, AUTO,
37          MANUAL, READY_TO_PRIME,
38          PRIMING, CONFIRM_PRIME,
39          INFUSING, INFUSION_COMPLETE}
40
41      public enum Protocol {TOTAL_BODY_BARG}
42
43      public enum LED {BLINKING, DARK, LIGHT, BLINK3}
44
45
```

```
46      public enum InfuseMode {
47          NIL, READY, INFUSING_CONTRAST,
48          INFUSING_SALINE, COMPLETE,
49          PAUSE, STOP
50      }
51
52      public enum  Display {
53          DISP_OFF,DISP_INIT,
54          MIRROR_PLUNGER_LEVEL,
55          MIRROR_TARGET_VOLUME
56      }
57
58      public enum ConsoleScreen {CONSOLE_INIT, CONSOLE_SECURITY,
            CONSOLE_PROTOCOL}
59
60      public enum  ConsoleLED {ORANGE, GREEN, LED_OFF}
61
62      public enum ConsoleDLG {
63          NIL, ASK_CONFIRM_AIR_CHECK,
64          VOLUME_WARNING, MSG_INFUSION_COMPLETE
65      }
66
67      public enum ConsoleCMD {LOCK, ENGAGE, DISENGAGE}
68
69      public enum ConsoleButton {PRESSED, IDLE}
70
71      public enum AutoloadMode {
72          LOAD30, MAKE_EMPTY, FILL_VOLUME,
73          WAIT5, FINALIZE, DONE
74      }
75
76
77      Mode mode;
78      AutoloadMode autoload_mode_saline;
79      AutoloadMode autoload_mode_contrast ;
80      boolean syringe_saline_present;
81      boolean syringe_contrast_present;
82      int plunger_saline; //upto max_volume
83      int plunger_contrast; //upto max_volume
84      Display display_saline;
85      Display display_contrast ;
86      int vol_saline; //upto max_volume
87      int vol_contrast; //upto max_volume
88      boolean vol_saline_confirmed;
89      boolean vol_contrast_confirmed;
90      LED lock_LED;
91      LED infusion_contrast_LED;
```

```
92      LED infusion_saline_LED;
93      LED btn_fill_saline ;
94      LED btn_fill_contrast ;
95      LED btn_auto ;
96      LED btn_manual ;
97      LED btn_prime ;
98      LED btn_confirm ;
99      LED btn_engage ;
100     float btn_manual_timeout;//nnegreal
101     float btn_auto_timeout_saline; //%-- -1 disables the timeout
102     float btn_auto_timeout_contrast ; //%-- -1 disables the timeout
103     float timeout_autoload_saline ; //nonneg_real
104     float timeout_autoload_contrast ; //nonneg_real
105     boolean prime_confirmed ;
106     int prime_volume_saline;//upto max_volume
107     int prime_volume_contrast ;//upto max_volume
108     boolean prime_warning ;// %-- this is used to handle the warning given by
            the injector when trying to arm without activating the air-in-line
            check button first
109     boolean lock_warning ;
110     boolean armed ;
111     int vol_saline_infused ;//upto max_volume
112     int vol_contrast_infused ;//upto max_volume
113     InfuseMode infuse_mode ;
114     boolean injector_rotated ;
115
116
117     //console
118     float console_btn_timeout ; //nonneg_real
119     ConsoleButton console_btn_ACC ;
120     ConsoleLED console_LED_ACC ;
121     ConsoleScreen console_screen ;
122     Protocol console_protocol ;
123     int console_vol_saline ; // upto max_volume
124     int console_vol_contrast ; //upto max_volume
125     float console_rate_saline ; // mL/sec  x: nonneg_real | x < MAX_RATE
126     float console_rate_contrast ; // mL/sec  x: nonneg_real | x < MAX_RATE
127     float console_time_saline ; //nnegreal
128     float console_time_contrast ; //nnegreal
129     ConsoleCMD console_cmd ;
130     ConsoleDLG console_dlg ;
131     boolean console_locked ;
132
133     //bags
134     boolean bag_saline_present ;
135     boolean bag_contrast_present ;
136     boolean fluid_in_saline_syringe ;
```

```
137     boolean fluid_in_contrast_syringe ;
138
139     //infusion set
140     boolean infusion_set_present;
141
142     /**
143      * empty constructor
144      */
145     public State(){
146         mode = Mode.OFF;
147         autoload_mode_saline = AutoloadMode.LOAD30;
148         autoload_mode_contrast = AutoloadMode.LOAD30;
149         syringe_saline_present=false;
150         syringe_contrast_present=false;
151         plunger_saline = DEFAULT_VOLUME_SALINE;
152         plunger_contrast = DEFAULT_VOLUME_SALINE;
153         display_saline = Display.DISP_OFF;
154         display_contrast = Display.DISP_OFF;
155         vol_saline = 0;
156         vol_contrast = 0;
157         vol_saline_confirmed = false;
158         vol_contrast_confirmed = false;
159         lock_LED = LED.DARK;
160         infusion_contrast_LED = LED.DARK;
161         infusion_saline_LED = LED.DARK;
162         btn_fill_saline = LED.DARK;
163         btn_fill_contrast = LED.DARK;
164         btn_auto = LED.DARK;
165         btn_manual = LED.DARK;
166         btn_prime = LED.DARK;
167         btn_confirm = LED.DARK;
168         btn_engage = LED.DARK;
169         btn_manual_timeout = 0;
170         btn_auto_timeout_saline = 0;
171         btn_auto_timeout_contrast = 0;
172         timeout_autoload_saline = 0;
173         timeout_autoload_contrast = 0;
174         prime_confirmed = false;
175         prime_volume_saline = 0;
176         prime_volume_contrast = 0;
177         prime_warning = false;
178         lock_warning = false;
179
180         armed = false;
181         vol_saline_infused = 0;
182         vol_contrast_infused = 0;
183         infuse_mode = InfuseMode.NIL;
```

```
184         injector_rotated = false;
185         console_btn_timeout = 0;
186         console_btn_ACC = ConsoleButton.IDLE;
187         console_LED_ACC = ConsoleLED.LED_OFF;
188         console_screen = ConsoleScreen.CONSOLE_INIT;
189         console_protocol = Protocol.TOTAL_BODY_BARG;
190         console_vol_saline = 45;
191         console_vol_contrast = 100;
192         console_rate_saline = 4;
193         console_rate_contrast = 4;
194         console_time_saline = 45/4;
195         console_time_contrast = 100/4;
196         console_cmd = ConsoleCMD.LOCK;
197         console_dlg = ConsoleDLG.NIL;
198         console_locked = false;
199         bag_saline_present = false;
200         bag_contrast_present = false;
201         fluid_in_saline_syringe = false;
202         fluid_in_contrast_syringe = false;
203         infusion_set_present = false;
204     }
205 
206 
207     /**
208      * basic init function
209      */
210     public void init(float x){
211         mode = Mode.OFF;
212         autoload_mode_saline = AutoloadMode.LOAD30;
213         autoload_mode_contrast = AutoloadMode.LOAD30;
214         syringe_saline_present=false;
215         syringe_contrast_present=false;
216         plunger_saline = DEFAULT_VOLUME_SALINE;
217         plunger_contrast = DEFAULT_VOLUME_SALINE;
218         display_saline = Display.DISP_OFF;
219         display_contrast = Display.DISP_OFF;
220         vol_saline = 0;
221         vol_contrast = 0;
222         vol_saline_confirmed = false;
223         vol_contrast_confirmed = false;
224         lock_LED = LED.DARK;
225         infusion_contrast_LED = LED.DARK;
226         infusion_saline_LED = LED.DARK;
227         btn_fill_saline = LED.DARK;
228         btn_fill_contrast = LED.DARK;
229         btn_auto = LED.DARK;
230         btn_manual = LED.DARK;
```

```
231         btn_prime = LED.DARK;
232         btn_confirm = LED.DARK;
233         btn_engage = LED.DARK;
234         btn_manual_timeout = 0;
235         btn_auto_timeout_saline = 0;
236         btn_auto_timeout_contrast = 0;
237         timeout_autoload_saline = 0;
238         timeout_autoload_contrast = 0;
239         prime_confirmed = false;
240         prime_volume_saline = 0;
241         prime_volume_contrast = 0;
242         prime_warning = false;
243         lock_warning = false;
244 
245         armed = false;
246         vol_saline_infused = 0;
247         vol_contrast_infused = 0;
248         infuse_mode = InfuseMode.NIL;
249         injector_rotated = false;
250         console_btn_timeout = 0;
251         console_btn_ACC = ConsoleButton.IDLE;
252         console_LED_ACC = ConsoleLED.ORANGE;
253         console_screen = ConsoleScreen.CONSOLE_INIT;
254         console_protocol = Protocol.TOTAL_BODY_BARG;
255         console_vol_saline = 45;
256         console_vol_contrast = 100;
257         console_rate_saline = 4;
258         console_rate_contrast = 4;
259         console_time_saline = 45/4;
260         console_time_contrast = 100/4;
261         console_cmd = ConsoleCMD.LOCK;
262         console_dlg = ConsoleDLG.NIL;
263         console_locked = false;
264         bag_saline_present = false;
265         bag_contrast_present = false;
266         fluid_in_saline_syringe = false;
267         fluid_in_contrast_syringe = false;
268         infusion_set_present = false;
269     }
270 
271 
272     /**
273      * basic init function
274      */
275     public void init(){
276         mode = Mode.OFF;
277         autoload_mode_saline = AutoloadMode.LOAD30;
```

```
278         autoload_mode_contrast = AutoloadMode.LOAD30;
279         syringe_saline_present=false;
280         syringe_contrast_present=false;
281         plunger_saline = DEFAULT_VOLUME_SALINE;
282         plunger_contrast = DEFAULT_VOLUME_SALINE;
283         display_saline = Display.DISP_OFF;
284         display_contrast = Display.DISP_OFF;
285         vol_saline = 0;
286         vol_contrast = 0;
287         vol_saline_confirmed = false;
288         vol_contrast_confirmed = false;
289         lock_LED = LED.DARK;
290         infusion_contrast_LED = LED.DARK;
291         infusion_saline_LED = LED.DARK;
292         btn_fill_saline = LED.DARK;
293         btn_fill_contrast = LED.DARK;
294         btn_auto = LED.DARK;
295         btn_manual = LED.DARK;
296         btn_prime = LED.DARK;
297         btn_confirm = LED.DARK;
298         btn_engage = LED.DARK;
299         btn_manual_timeout = 0;
300         btn_auto_timeout_saline = 0;
301         btn_auto_timeout_contrast = 0;
302         timeout_autoload_saline = 0;
303         timeout_autoload_contrast = 0;
304         prime_confirmed = false;
305         prime_volume_saline = 0;
306         prime_volume_contrast = 0;
307         prime_warning = false;
308         lock_warning = false;
309 
310         armed = false;
311         vol_saline_infused = 0;
312         vol_contrast_infused = 0;
313         infuse_mode = InfuseMode.NIL;
314         injector_rotated = false;
315         console_btn_timeout = 0;
316         console_btn_ACC = ConsoleButton.IDLE;
317         console_LED_ACC = ConsoleLED.ORANGE;
318         console_screen = ConsoleScreen.CONSOLE_INIT;
319         console_protocol = Protocol.TOTAL_BODY_BARG;
320         console_vol_saline = 45;
321         console_vol_contrast = 100;
322         console_rate_saline = 4;
323         console_rate_contrast = 4;
324         console_time_saline = 45/4;
```

```
325         console_time_contrast = 100/4;
326         console_cmd = ConsoleCMD.LOCK;
327         console_dlg = ConsoleDLG.NIL;
328         console_locked = false;
329         bag_saline_present = false;
330         bag_contrast_present = false;
331         fluid_in_saline_syringe = false;
332         fluid_in_contrast_syringe = false;
333         infusion_set_present = false;
334     }
335 
336 
337 
338     /**
339      * constant incrementation function
340      * @param x
341      * @return int
342      */
343     public int inc(int x){
344         if (x+step >= MAX_VOLUME){
345             return x+step;
346         }
347         else{
348             return MAX_VOLUME;
349         }
350     }
351 
352     /**
353      * constant decrementation function
354      * @param x
355      * @return int
356      */
357     public int dec(int x){
358         if (x-step >= 0){
359             return x-step;
360         }
361         else{
362             return 0;
363         }
364     }
365 
366 
367 //    contrast
368 
369     /**
370      * verifies is device isn't OFF for contrast solution
371      * @param st
```

```
372         * @return boolean
373         */
374        public boolean per_inc_contrast(State st){
375            return (!st.mode.equals(Mode.OFF) && !(st.vol_contrast_confirmed));
376        }
377
378        /**
379         * invocation of incrementation function if device isn't OFF for contrast
                solution
380         * @param st
381         * @return State
382         */
383        public State inc_contrast(State st){
384            if (per_inc_contrast(st)){
385                st.vol_contrast = inc(st.vol_contrast);
386            }
387
388            return st;
389        }
390
391        /**
392         * execution of a click on incrementation button behaviour for contrast
                solution
393         * @param st
394         * @return State
395         */
396        public State click_inc_contrast(State st){
397            if (per_inc_contrast(st)){
398                return inc_contrast(st);
399            }
400            else{
401                return st;
402            }
403        }
404
405        /**
406         * execution of press incrementation button behaviour for contrast
                solution
407         * @param st
408         * @return State
409         */
410        public State press_inc_contrast(State st){
411            return click_inc_contrast(st);
412        }
413
414        /**
```

```
415      * execution of release incrementation button behaviour for contrast
             solution
416      * @param st
417      * @return State
418      */
419     public State release_inc_contrast(State st){
420         return st;
421     }
422
423
424     /**
425      * verifies is device isn't OFF for contrast solution
426      * @param st
427      * @return boolean
428      */
429     public boolean per_dec_contrast(State st){
430         return (st.mode.equals(Mode.OFF) && !(st.vol_contrast_confirmed));
431     }
432
433     /**
434      * invocation of decrementation function if device isn't OFF for contrast
             solution
435      * @param st
436      * @return State
437      */
438     public State dec_contrast(State st){
439         if (per_dec_contrast(st)){
440             st.vol_contrast=dec(st.vol_contrast);
441         }
442         return st;
443     }
444
445     /**
446      * execution of a click on decrementation button behaviour for contrast
             solution
447      * @param st
448      * @return State
449      */
450     public State click_dec_contrast(State st){
451         if (per_dec_contrast(st)){
452             return dec_contrast(st);
453         }
454         else{
455             return st;
456         }
457     }
458
```

```
459     /**
460      * execution of press decrementation button behaviour for contrast
             solution
461      * @param st
462      * @return State
463      */
464     public State press_dec_contrast(State st){
465         return click_dec_contrast(st);
466     }
467 
468     /**
469      * execution of release decrementation button behaviour for contrast
             solution
470      * @param st
471      * @return State
472      */
473     public State release_dec_contrast(State st){
474         return st;
475     }
476 
477 
478 //    saline
479 
480     /**
481      * verifies is device isn't OFF for saline solution
482      * @param st
483      * @return boolean
484      */
485     public boolean per_inc_saline(State st){
486         return (!st.mode.equals(Mode.OFF) && !st.vol_saline_confirmed);
487     }
488     /**
489      * invocation of incrementation function if device isn't OFF for saline
             solution
490      * @param st
491      * @return State
492      */
493     public State inc_saline(State st){
494         if(per_inc_saline(st)){
495             st.vol_saline = inc(st.vol_saline);
496         }
497         return  st;
498     }
499 
500     /**
501      * execution of a click on incrementation button behaviour for saline
             solution
```

```
502      * @param st
503      * @return State
504      */
505     public State click_inc_saline(State st) {
506         if(per_inc_saline(st)) {
507             return inc_saline(st);
508         } else {
509             return st;
510         }
511     }
512 
513     /**
514      * execution of press incrementation button behaviour for saline solution
515      * @param st
516      * @return State
517      */
518     public State press_inc_saline(State st){
519         return click_inc_saline(st);
520     }
521 
522     /**
523      * execution of release incrementation button behaviour for saline
           solution
524      * @param st
525      * @return State
526      */
527     public State release_inc_saline(State st){
528         return st;
529     }
530 
531     /**
532      * verifies is device isn't OFF for saline solution
533      * @param st
534      * @return boolean
535      */
536     public boolean per_dec_saline(State st){
537         return (!st.mode.equals(Mode.OFF) && !st.vol_saline_confirmed);
538     }
539 
540     /**
541      * invocation of decrementation function if device isn't OFF for saline
           solution
542      * @param st
543      * @return State
544      */
545     public State dec_saline(State st ){
546         if(per_inc_saline(st)){
```

```
547             st.vol_saline = dec(st.vol_saline);
548         }
549         return  st;
550     }
551 
552     /**
553      * execution of a click on decrementation button behaviour for saline
             solution
554      * @param st
555      * @return State
556      */
557     public State click_dec_saline(State st){
558         if(per_dec_saline(st)) {
559             return dec_saline(st);
560         } else {
561             return st;
562         }
563     }
564 
565     /**
566      * execution of press incrementation button behaviour for saline solution
567      * @param st
568      * @return State
569      */
570     public State press_dec_saline(State st){
571         return click_dec_saline(st);
572     }
573 
574     /**
575      * execution of release decrementation button behaviour for saline
             solution
576      * @param st
577      * @return State
578      */
579     public State release_dec_saline(State st){
580         return st;
581     }
582 
583     /**
584      * verifies is volumes are confimed/correct
585      * @param st
586      * @return boolean
587      */
588     public boolean volumes_confirmed(State st){
589         return (st.vol_saline_confirmed && st.vol_contrast_confirmed);
590     }
591 
```

```
592     /**
593      * set different LEDs of the device regarding the state of the device
594      * @param st
595      * @return State
596      */
597     public State set_LED_state(State st){
598         if(st.mode.equals(Mode.OFF)){
599             st.lock_LED = State.LED.DARK;
600         }
601         else if (st.lock_warning) {
602             st.lock_LED = State.LED.BLINK3;
603         }
604         else if (st.console_locked) {
605             st.lock_LED = State.LED.LIGHT;
606         }
607 
608         else {
609             st.lock_LED = State.LED.DARK;
610         }
611 
612         // set infusion_contrast_LED
613 
614         if (st.armed && st.infuse_mode.equals(State.InfuseMode.READY)){
615             st.infusion_contrast_LED = State.LED.BLINKING;
616         }
617         else if (st.armed && st.infusion_contrast_LED.equals(State.InfuseMode.
                INFUSING_CONTRAST)){
618             st.infusion_contrast_LED = State.LED.LIGHT;
619         }
620         else {
621             st.infusion_contrast_LED = State.LED.DARK;
622         }
623 
624         // infusion_saline_LED
625         if (st.armed && st.infuse_mode.equals(State.InfuseMode.READY)){
626             st.infusion_saline_LED = State.LED.BLINKING;
627         }
628         else if (st.armed && st.infusion_contrast_LED.equals(State.InfuseMode.
                INFUSING_SALINE)){
629             st.infusion_saline_LED = State.LED.LIGHT;
630         }
631         else {
632             st.infusion_saline_LED = State.LED.DARK;
633         }
634 
635 
636         //set btn_fill_contrast & btn_fill_saline
```

```
637
638         if(st.mode.equals(State.Mode.AUTO)){
639
640
641
642             if (st.vol_contrast_confirmed && !st.autoload_mode_contrast.equals
                    (State.AutoloadMode.DONE)){
643                 st.btn_fill_contrast = State.LED.LIGHT;
644             }
645             else if(!st.vol_contrast_confirmed && st.btn_auto_timeout_contrast
                     > 0){
646                 st.btn_fill_contrast = State.LED.BLINKING;
647             }
648             else {
649                 st.btn_fill_contrast = State.LED.DARK;
650             }
651             //-----------------------------------------------
652             if (st.vol_saline_confirmed && !st.autoload_mode_saline.equals(
                    State.AutoloadMode.DONE)){
653                 st.btn_fill_saline = State.LED.LIGHT;
654             }
655             else if(!st.vol_saline_confirmed && st.btn_auto_timeout_saline >
                    0){
656                 st.btn_fill_saline = State.LED.BLINKING;
657             }
658             else {
659                 st.btn_fill_saline = State.LED.DARK;
660             }
661
662
663         }
664         else {
665             st.btn_fill_contrast = State.LED.DARK;
666             st.btn_fill_saline = State.LED.DARK;
667         }
668
669
670
671         if (st.mode.equals(Mode.OFF)){
672             st.btn_confirm = State.LED.DARK;
673         }
674         else if (st.prime_warning){
675             st.btn_confirm = State.LED.BLINK3;
676         }
677         else if (st.prime_confirmed){
678             st.btn_confirm = State.LED.LIGHT;
679         }
```

```
680         else {
681             st.btn_confirm = State.LED.DARK;
682         }
683 
684 
685         if (st.mode.equals(State.Mode.MANUAL)){
686             st.btn_manual = State.LED.LIGHT;
687         }
688         else st.btn_manual = State.LED.DARK;
689 
690         st.btn_auto = State.LED.DARK;
691         st.btn_engage = State.LED.DARK;
692         st.btn_prime = State.LED.DARK;
693 
694         return st;
695     }
696 
697     public boolean per_btn_auto(State st){
698 
699         return (st.mode.equals(State.Mode.INIT_COMPLETE)||
700                 st.mode.equals(State.Mode.CONFIRM_PRIME)||
701                 st.mode.equals(State.Mode.READY_TO_PRIME)    &&  st.
                        bag_saline_present && st.bag_contrast_present);
702     }
703 
704     public State click_btn_auto (State st){
705 
706         if (per_btn_auto(st)){
707             st.mode = State.Mode.AUTO;
708             st.vol_saline = st.console_vol_saline;
709             st.vol_contrast = st.console_vol_contrast;
710             st.vol_saline_confirmed = false;
711             st.vol_contrast_confirmed = false;
712             st.display_saline = State.Display.MIRROR_TARGET_VOLUME;
713             st.display_contrast = State.Display.MIRROR_TARGET_VOLUME;
714 
715             if(st.plunger_contrast>0){
716                 st.autoload_mode_contrast = State.AutoloadMode.FILL_VOLUME;
717             }
718             else {
719                 st.autoload_mode_contrast = State.AutoloadMode.LOAD30;
720             }
721 
722             if(st.plunger_saline>0){
723                 st.autoload_mode_saline = State.AutoloadMode.FILL_VOLUME;
724             }
725             else {
```

```
726                 st.autoload_mode_saline = State.AutoloadMode.LOAD30;
727             }
728 
729             st.prime_volume_saline = 0;
730             st.prime_volume_contrast = 0;
731             st.prime_confirmed = false; // ??????????????????????
732             st.btn_auto_timeout_contrast = BTN_AUTO_TIMEOUT;
733             st.btn_auto_timeout_saline = BTN_AUTO_TIMEOUT;
734 
735             return set_LED_state(st);
736         }
737 
738         else{
739             return st;
740         }
741     }
742 
743     public boolean per_btn_manual(State st){
744         return (st.mode.equals(State.Mode.INIT_COMPLETE)||
745                 st.mode.equals(State.Mode.CONFIRM_PRIME)||
746                 st.mode.equals(State.Mode.READY_TO_PRIME));
747     }
748 
749     public State click_btn_manual(State st){
750         if (per_btn_manual(st)){
751             st.mode = State.Mode.MANUAL;
752             st.vol_saline = st.plunger_saline;
753             st.vol_contrast = st.plunger_contrast;
754             st.vol_saline_confirmed = false;
755             st.vol_contrast_confirmed = false;
756             st.btn_manual_timeout = BTN_MANUAL_TIMEOUT;
757             return set_LED_state(st);
758         }
759         else {
760             return st;
761         }
762     }
763 
764     public boolean per_btn_fill_saline(State st){
765         return st.mode.equals(State.Mode.AUTO);
766     }
767 
768     public State click_btn_fill_saline (State st){
769         if(per_btn_fill_saline(st)){
770             st.vol_saline_confirmed = true;
771             st.btn_auto_timeout_saline  = -1;
772             return set_LED_state(st);
```

```
773         }
774         else {
775             return st;
776         }
777     } //disables timeout for saline auto button
778 
779     public boolean per_btn_fill_contrast(State st){
780         return st.mode.equals(State.Mode.AUTO);
781     }
782 
783     public State click_btn_fill_contrast (State st){
784         if(per_btn_fill_contrast(st)){
785             st.vol_contrast_confirmed = true;
786             st.btn_auto_timeout_contrast  = -1; //disables timeout for
                    contrast auto button
787             return set_LED_state(st);
788         }
789         else {
790             return st;
791         }
792     }
793 
794     public boolean per_btn_prime(State st){
795         return (st.mode.equals(State.Mode.READY_TO_PRIME) || st.mode.equals(
                State.Mode.CONFIRM_PRIME));
796     }
797 
798     public State click_btn_prime(State st){
799 
800         if (per_btn_prime(st)){
801             st.prime_volume_saline = PRIME_VOLUME_SALINE;
802             st.prime_volume_contrast = PRIME_VOLUME_CONTRAST;
803             st.mode = State.Mode.PRIMING;
804             return set_LED_state(st);
805         }
806         else {
807             return st;
808         }
809     }
810 
811     public boolean per_btn_console_lock(State st) {
812         return per_btn_prime(st);
813     }
814 
815     public State click_btn_console_lock(State st){
816 
817         if(per_btn_console_lock(st)){
```

```
818             st.console_locked = true;
819             st.console_cmd = State.ConsoleCMD.ENGAGE;
820             return set_LED_state(st);
821         }
822         else{
823             return st;
824         }
825     }
826 
827     public boolean per_btn_console_engage(State st){
828         return st.console_cmd.equals(State.ConsoleCMD.ENGAGE);
829     }
830 
831     public State click_btn_console_engage(State st){
832 
833         if(per_btn_console_engage(st) && st.prime_confirmed){
834             if(st.console_vol_saline > st.plunger_saline  || st.
                    console_vol_contrast > st.plunger_contrast){
835                 st.console_dlg = ConsoleDLG.VOLUME_WARNING;
836             return set_LED_state(st);
837             }
838             else{
839                 st.console_cmd = ConsoleCMD.DISENGAGE;
840                 st.armed = true;
841                 st.infuse_mode = InfuseMode.READY;
842                 return set_LED_state(st);
843             }
844         }
845         else if (per_btn_console_engage(st) && !st.prime_confirmed){
846             st.console_dlg = ConsoleDLG.ASK_CONFIRM_AIR_CHECK;
847             return set_LED_state(st);
848         }
849         else{
850             return st;
851         }
852     }
853 
854     public boolean per_btn_engage(State st){
855         return (!st.mode.equals(Mode.OFF) && !st.mode.equals(Mode.INIT) && !st
                .mode.equals(Mode.INIT_SYRINGE)
856                 && st.mode.equals(Mode.INIT_COMPLETE) && !st.mode.equals(Mode.
                        AUTO)
857                 && (st.console_cmd.equals(ConsoleCMD.ENGAGE) || st.console_cmd
                        .equals(ConsoleCMD.LOCK)));
858     }
859 
860     public boolean per_btn_confirm(State st){
```

```
861             return per_btn_engage(st);
862     }
863 
864     public State click_btn_confirm(State st) {
865 
866         if (per_btn_confirm(st)){
867             st.prime_confirmed = !st.prime_confirmed;
868             return set_LED_state(st);
869         }
870         else{
871             return st;
872         }
873     }
874 
875     public boolean per_btn_console_disengage(State st){
876         return (!st.mode.equals(Mode.INFUSING) && st.console_cmd.equals(
                ConsoleCMD.DISENGAGE));
877     }
878 
879     public State click_btn_console_disengage(State st){
880         if(per_btn_console_disengage(st)){
881             st.mode = Mode.CONFIRM_PRIME;
882             st.console_locked = false;
883             st.console_cmd = ConsoleCMD.LOCK;
884             st.armed = false;
885             st.infuse_mode = InfuseMode.NIL;
886             return set_LED_state(st);
887         }
888         else{
889             return st;
890         }
891     }
892 
893 //    these utility functions automatically stop pulling the plunger when the
    target volume has been reached
894     public boolean per_empty_saline(State st){
895         return (st.mode.equals(Mode.INIT_SYRINGE));
896     }
897 
898     public State empty_saline(int step,State st){
899         if(per_empty_saline(st)){
900             if(st.plunger_saline - step > 0){
901                 st.plunger_saline = st.plunger_saline - step;
902             }
903             else {
904                 st.plunger_saline = 0;
905             }
```

```
906         }
907         return st;
908     }
909 
910     public boolean per_empty_contrast(State st){
911         return st.mode.equals(Mode.INIT_SYRINGE);
912     }
913 
914     public State empty_contrast(int step,State st){
915         if(per_empty_contrast(st)){
916             if(st.plunger_contrast-step > 0){
917                 st.plunger_contrast= st.plunger_contrast - step;
918             }
919             else{
920                 st.plunger_contrast = 0;
921             }
922         }
923         return st;
924     }
925 
926     public boolean per_auto(State st){
927        return st.mode.equals(Mode.AUTO);
928    }
929 
930     public State auto_plunger_saline(int step, State st){
931         if (per_auto(st)){
932             if(st.autoload_mode_saline.equals(AutoloadMode.LOAD30)){
933                 st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
934                 if(st.plunger_saline+AUTOLOAD_STEP <=30){
935                     st.plunger_saline = st.plunger_saline + AUTOLOAD_STEP;
936                 }
937                 else{
938                     st.autoload_mode_saline = AutoloadMode.MAKE_EMPTY;
939                 }
940             }
941             else if(st.autoload_mode_saline.equals(AutoloadMode.MAKE_EMPTY)){
942                 st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
943                 if(st.plunger_saline - AUTOLOAD_STEP >= 0){
944                     st.plunger_saline = st.plunger_saline - AUTOLOAD_STEP;
945                 }
946                 else{
947                     st.autoload_mode_saline = AutoloadMode.FILL_VOLUME;
948                 }
949             }
950             else if(st.autoload_mode_saline.equals(AutoloadMode.FILL_VOLUME)){
951                 st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
952                 int target_vol;
```

```
953                 if(st.vol_saline + VOL_BUFFER < MAX_VOLUME){
954                     target_vol = st.vol_saline + VOL_BUFFER;
955                 }
956                 else{
957                     target_vol = MAX_VOLUME;
958                 }
959                 if(st.plunger_saline + step <= target_vol){
960                     st.plunger_saline = st.plunger_saline + step;
961                 }
962                 else{
963                     st.plunger_saline = target_vol;
964                     st.autoload_mode_saline = AutoloadMode.WAIT5;
965                     st.timeout_autoload_saline = 5;
966                 }
967             }
968             else if (st.autoload_mode_saline.equals(AutoloadMode.WAIT5)){
969                 float timeout;
970                 st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
971                 if(st.timeout_autoload_saline-1 > 0){
972                     timeout = timeout_autoload_saline - 1;
973                 }
974                 else {
975                     timeout = 0;
976                 }
977                 st.timeout_autoload_saline = timeout;
978                 if(timeout < 0){
979                     st.autoload_mode_saline = AutoloadMode.FINALIZE;
980                 }
981             }
982             else if (st.autoload_mode_saline.equals(AutoloadMode.FINALIZE)){
983                 st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
984                 if(st.plunger_saline - step >= st.vol_saline){
985                     st.plunger_saline = st.plunger_saline - step;
986                 }
987                 else {
988                     st.plunger_saline = st.vol_saline;
989                 }
990                 if(st.plunger_saline == st.vol_saline){
991                     st.autoload_mode_saline = AutoloadMode.DONE;
992                 }
993                 else {
994                     st.autoload_mode_saline = AutoloadMode.FINALIZE;
995 
996                 }
997             }
998             else if (autoload_mode_saline.equals(AutoloadMode.DONE)){
999                 st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
```

```
1000            }
1001        }
1002        return st;
1003    }
1004
1005    public State auto_plunger_contrast(int step, State st){
1006        if (per_auto(st)){
1007            if(st.autoload_mode_contrast.equals(AutoloadMode.LOAD30)){
1008                st.display_contrast = Display.MIRROR_PLUNGER_LEVEL;
1009                if(st.plunger_contrast + AUTOLOAD_STEP <= 30){
1010                    st.plunger_contrast = st.plunger_contrast + AUTOLOAD_STEP;
1011                }
1012                else{
1013                    st.autoload_mode_contrast = AutoloadMode.MAKE_EMPTY;
1014                }
1015            }
1016            else if(st.autoload_mode_contrast.equals(AutoloadMode.MAKE_EMPTY))
                    {
1017                st.display_contrast = Display.MIRROR_PLUNGER_LEVEL;
1018                if(st.plunger_contrast - AUTOLOAD_STEP >= 0){
1019                    st.plunger_contrast = st.plunger_contrast - AUTOLOAD_STEP;
1020                }
1021                else{
1022                    st.autoload_mode_contrast = AutoloadMode.FILL_VOLUME;
1023                }
1024            }
1025            else if(st.autoload_mode_contrast.equals(AutoloadMode.FILL_VOLUME)
                    ){
1026                st.display_contrast = Display.MIRROR_PLUNGER_LEVEL;
1027                int target_vol;
1028                if(st.vol_contrast + VOL_BUFFER < MAX_VOLUME){
1029                    target_vol = st.vol_contrast + VOL_BUFFER;
1030                }
1031                else{
1032                    target_vol = MAX_VOLUME;
1033                }
1034                if(st.plunger_contrast + step <= target_vol){
1035                    st.plunger_contrast = st.plunger_contrast+step;
1036                }
1037                else{
1038                    st.plunger_contrast = target_vol;
1039                    st.autoload_mode_contrast = AutoloadMode.WAIT5;
1040                    st.timeout_autoload_contrast = 5;
1041                }
1042            }
1043            else if(st.autoload_mode_contrast.equals(AutoloadMode.WAIT5)){
1044                st.display_contrast = Display.MIRROR_PLUNGER_LEVEL;
```

```
1045                float timeout;
1046                if(st.timeout_autoload_contrast-1>0){
1047                    timeout = st.timeout_autoload_contrast - 1;
1048                }
1049                else {
1050                    timeout = 0;
1051                }
1052                st.timeout_autoload_contrast = timeout;
1053                if(timeout<0){
1054                    st.autoload_mode_contrast = AutoloadMode.FINALIZE;
1055                }
1056            }
1057            else if(st.autoload_mode_contrast.equals(AutoloadMode.FINALIZE)){
1058                if(st.plunger_contrast - step >= st.vol_contrast){
1059                    st.plunger_contrast = st.plunger_contrast - step;
1060                }
1061                else{
1062                    st.plunger_contrast = st. vol_contrast;
1063                }
1064                if(st.plunger_contrast == st.vol_contrast){
1065                    st.autoload_mode_contrast = AutoloadMode.DONE;
1066                }
1067            }
1068        }
1069        return st;
1070    }
1071
1072    public boolean per_prime_syringes(State st){
1073        return st.mode.equals(Mode.PRIMING);
1074    }
1075
1076    public State prime_sytinges (State st){
1077        if(per_prime_syringes(st)){
1078            if(st.prime_volume_saline>0){
1079                if(st.plunger_saline -1 > 0){
1080                    st.plunger_saline = st.plunger_saline - 1;
1081                }
1082                else{
1083                    st.plunger_saline = 0;
1084                }
1085            }
1086            if(prime_volume_saline-1>0){
1087                st.prime_volume_saline = st.prime_volume_saline - 1;
1088            }
1089            else{
1090                st.prime_volume_saline = 0;
1091            }
```

```
1092            if(st.prime_volume_contrast>0){
1093                if(st.plunger_contrast -1 > 0){
1094                    st.plunger_contrast = st.plunger_contrast - 1;
1095                }
1096                else{
1097                    st.plunger_contrast = 0;
1098                }
1099            }
1100            if(prime_volume_contrast-1>0){
1101                st.prime_volume_contrast = st.prime_volume_contrast - 1;
1102            }
1103            else{
1104                st.prime_volume_contrast = 0;
1105            }
1106
1107            if(st.prime_volume_saline == 0 && st.prime_volume_contrast == 0){
1108                st.mode = Mode.CONFIRM_PRIME;
1109            }
1110            else{
1111                st.mode = Mode.PRIMING;
1112            }
1113
1114        }
1115        return  st;
1116    }
1117
1118    public State pull_plunger_saline(int step, State st){
1119        if(st.plunger_saline + step <= MAX_VOLUME){
1120            st.plunger_saline = st.plunger_saline + step;
1121        }
1122        else {
1123            st.plunger_saline = MAX_VOLUME;
1124        }
1125        return st;
1126    }
1127
1128    public State pull_plunger_contrast(int step,State st){
1129        if(st.plunger_contrast + step <= MAX_VOLUME){
1130            st.plunger_contrast = st.plunger_contrast + step;
1131        }
1132        else {
1133            st.plunger_contrast = MAX_VOLUME;
1134        }
1135        return st;
1136    }
1137
1138    public State push_plunger_saline(int step,State st){
```

```
1139         if(st.plunger_saline - step >= 0){
1140             st.plunger_saline = st.plunger_saline - step;
1141         }
1142         else{
1143             st.plunger_saline = 0;
1144         }
1145         return st;
1146     }
1147 
1148     public State push_plunger_contrast(int step, State st){
1149         if(st.plunger_contrast - step > 0){
1150             st.plunger_contrast = st.plunger_contrast -step;
1151         }
1152         else {
1153             st.plunger_contrast = 0;
1154         }
1155         return st;
1156     }
1157 //      this is for the automatic mode
1158 
1159     public boolean per_tick(State st ){
1160         return (st.mode.equals(Mode.INIT) || st.mode.equals(Mode.INIT_SYRINGE)
                 || st.mode.equals(Mode.MANUAL)
1161             || st.mode.equals(Mode.READY_TO_PRIME) || st.mode.equals(Mode.
                    CONFIRM_PRIME)
1162             || st.mode.equals(Mode.PRIMING) || st.mode.equals(Mode.INFUSING));
1163     }
1164 
1165     public State tick(State st){
1166 //        clear temporary warnings
1167         if(per_tick(st)){
1168             if (st.prime_warning){
1169                 st.prime_warning = false;
1170             }
1171             if(st.lock_warning){
1172                 st.lock_warning = false;
1173             }
1174 
1175 
1176             if(st.mode.equals(Mode.INIT)){
1177                 if(st.syringe_saline_present || st.syringe_contrast_present){
1178                     st.mode= Mode.INIT_SYRINGE;
1179                 }
1180                 return set_LED_state(st);
1181             }
1182             else if (st.mode.equals(Mode.INIT_SYRINGE)){
1183                 if(st.syringe_saline_present){
```

```
1184                    st = empty_saline(FAST,st);
1185                }
1186                if(st.syringe_contrast_present){
1187                    st = empty_contrast(FAST,st);
1188                }
1189                if(st.plunger_saline == st.vol_saline && st.plunger_contrast
                        == st.vol_contrast){
1190                    st.mode = Mode.INIT_COMPLETE;
1191                }
1192                return set_LED_state(st);
1193            }
1194            else if(st.mode.equals(Mode.MANUAL)){
1195                if(st.btn_manual_timeout - st.tick_step > 0){
1196                    st.btn_manual_timeout = st.btn_manual_timeout - st.
                            tick_step;
1197                }
1198                else {
1199                    st.btn_manual_timeout = 0;
1200                }
1201                if (st.btn_manual_timeout == 0){
1202                    st.mode = Mode.READY_TO_PRIME;
1203                }
1204                return set_LED_state(st);
1205            }
1206            else if (st.mode.equals(Mode.AUTO)){
1207 //                 saline
1208                if(st.btn_auto_timeout_saline != 0) {
1209                    if(st.vol_saline_confirmed){
1210                        st = auto_plunger_saline(FAST,st);
1211                    }
1212                    if(st.btn_auto_timeout_saline>0){
1213                        if(st.btn_auto_timeout_saline - st.tick_step >0){
1214                            st.btn_auto_timeout_saline = st.
                                    btn_auto_timeout_saline - st.tick_step;
1215                        }
1216                        else{
1217                            st.btn_auto_timeout_saline = 0;
1218                        }
1219                    }
1220                }
1221                else{
1222                    st.vol_saline = st.plunger_saline;
1223                    st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
1224                }
1225 //                contrast
1226                if(st.btn_auto_timeout_contrast != 0) {
1227                    if(st.vol_contrast_confirmed){
```

```
1228                        st = auto_plunger_contrast(FAST,st);
1229                    }
1230                    if(st.btn_auto_timeout_contrast > 0){
1231                        if(st.btn_auto_timeout_contrast - st.tick_step >0){
1232                            st.btn_auto_timeout_contrast = st.
                                    btn_auto_timeout_contrast - st.tick_step;
1233                        }
1234                        else{
1235                            st.btn_auto_timeout_contrast = 0;
1236                        }
1237                    }
1238                }
1239                else{
1240                    st.vol_contrast = st.plunger_contrast;
1241                    st.display_contrast = Display.MIRROR_PLUNGER_LEVEL;
1242                }
1243 //                 next operating mode
1244                if(st.btn_auto_timeout_contrast <= 0 && st.
                    btn_auto_timeout_contrast <=0
1245                        && st.autoload_mode_saline.equals(AutoloadMode.DONE)
                            && st.autoload_mode_contrast.equals(AutoloadMode.
                            DONE)
1246                        && st.plunger_saline > 0 && st.plunger_contrast > 0){
1247                    st.mode = Mode.READY_TO_PRIME;
1248                    st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
1249                    st.display_contrast = Display.MIRROR_PLUNGER_LEVEL;
1250                }
1251                else if((st.btn_auto_timeout_saline == 0 && st.plunger_saline
                    == 0 && st.autoload_mode_contrast.equals(AutoloadMode.DONE
                    ))
1252                        ||(st.btn_auto_timeout_contrast == 0 && st.
                            plunger_contrast == 0 && st.autoload_mode_saline.
                            equals(AutoloadMode.DONE))
1253                        ||(st.btn_auto_timeout_saline == 0 && st.
                            btn_auto_timeout_contrast == 0)){
1254                    st.mode = Mode.INIT_COMPLETE;
1255                    st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
1256                    st.display_contrast = Display.MIRROR_PLUNGER_LEVEL;
1257                }
1258                return set_LED_state(st);
1259            }
1260            else if(st.mode.equals(Mode.INFUSING)){
1261                if(st.vol_contrast_infused < st.console_vol_contrast && st.
                    plunger_contrast > 0){
1262                    st.vol_contrast_infused = st.vol_contrast_infused + 1;
1263                    if(st.plunger_contrast - 1 > 0){
1264                        st.plunger_contrast = st.plunger_contrast - 1;
```

```
1265                    }
1266                    else {
1267                        st.plunger_contrast = 0;
1268                    }
1269                    st.infuse_mode = InfuseMode.INFUSING_CONTRAST;
1270                }
1271                else if(!(st.vol_contrast_infused < st.console_vol_contrast &&
                         st.plunger_contrast > 0)
1272                        &&(st.vol_saline_infused < st.console_vol_contrast &&
                            st.plunger_saline> 0)){
1273                    st.vol_saline_infused = st.vol_saline_infused + 1;
1274                    if(st.plunger_saline -1 > 0){
1275                        st.plunger_saline = st.plunger_saline -1;
1276                    }
1277                    else{
1278                        st.plunger_saline = 0;
1279                    }
1280                    st.infuse_mode = InfuseMode.INFUSING_SALINE;
1281                }
1282                else if(st.vol_contrast_infused == st.console_vol_contrast &&
                        st.vol_saline_infused == st.console_vol_saline){
1283                    st.mode =  Mode.INFUSION_COMPLETE;
1284                    st.infuse_mode = InfuseMode.COMPLETE;
1285                    st.console_dlg = ConsoleDLG.MSG_INFUSION_COMPLETE;
1286                }
1287                return set_LED_state(st);
1288            }
1289            else{
1290                return set_LED_state(st);
1291            }
1292        }
1293
1294        return st;
1295    }
1296
1297 //  manual saline
1298    public boolean per_sUP_saline(State st){
1299        return st.mode.equals(Mode.MANUAL);
1300    }
1301
1302    public State sUP_saline(State st){
1303        st = push_plunger_saline(SLOW,st);
1304        st.vol_saline = st.plunger_saline;
1305        st.vol_contrast = st.plunger_contrast;
1306        st.btn_manual_timeout= BTN_MANUAL_TIMEOUT;
1307        return st;
1308    }
```

```
1309
1310    public State click_btn_sUP_saline(State st){
1311        if(per_sUP_saline(st)){
1312            return sUP_saline(st);
1313        }
1314        else{
1315            return st;
1316        }
1317    }
1318
1319    public State press_btn_sUP_saline(State st){
1320        return click_btn_sUP_saline(st);
1321    }
1322
1323    public State release_btn_sUP_saline(State st){
1324        return st;
1325    }
1326
1327    public boolean per_fUP_saline(State st){
1328        return st.mode.equals(Mode.MANUAL);
1329    }
1330
1331    public State fUP_saline(State st){
1332        if(per_fUP_saline(st)){
1333            st = push_plunger_saline(FAST,st);
1334            st.vol_saline = st.plunger_saline;
1335            st.vol_contrast = st.plunger_contrast;
1336            st.btn_manual_timeout = BTN_MANUAL_TIMEOUT;
1337            st.prime_confirmed = false;
1338        }
1339        return st;
1340    }
1341
1342    public State click_btn_fUP_saline(State st){
1343        if(per_fUP_saline(st)){
1344             return fUP_saline(st);
1345        }
1346        else{
1347            return st;
1348        }
1349    }
1350
1351    public State press_btn_fUP_saline(State st){
1352        return  click_btn_fUP_saline(st);
1353    }
1354
1355    public State release_btn_fUP_saline(State st){
```

```
1356        return st;
1357    }
1358
1359    public boolean per_sDOWN_saline(State st){
1360        return st.mode.equals(Mode.MANUAL);
1361    }
1362
1363    public  State sDOWN_saline(State st){
1364        if(per_sDOWN_saline(st)){
1365            st = pull_plunger_saline(SLOW,st);
1366            st.vol_contrast = st.plunger_contrast;
1367            st.vol_saline = st.plunger_saline;
1368            st.btn_manual_timeout = BTN_MANUAL_TIMEOUT;
1369            st.prime_confirmed = false;
1370        }
1371        return st;
1372    }
1373
1374    public State click_btn_sDOWN_saline ( State st){
1375        if (per_sDOWN_saline(st)){
1376            return sDOWN_saline(st);
1377        }
1378        else {
1379            return st;
1380        }
1381    }
1382
1383    public State press_btn_sDOWN_saline (State st){
1384        return click_btn_sDOWN_saline(st);
1385    }
1386
1387    public State release_btn_sDOWN_saline ( State st){
1388        return st;
1389    }
1390
1391    public boolean per_fDOWN_saline(State st){
1392        return st.mode.equals(Mode.MANUAL);
1393    }
1394
1395    public State fDOWN_saline(State st){
1396        if (per_fDOWN_saline(st)){
1397            st = pull_plunger_saline(FAST,st);
1398            st.vol_saline = st.plunger_saline;
1399            st.vol_contrast = st.plunger_contrast;
1400            st.btn_manual_timeout = BTN_MANUAL_TIMEOUT;
1401            st.prime_confirmed = false;
1402        }
```

```
1403         return st;
1404     }
1405 
1406     public State click_btn_fDOWN_saline(State st){
1407         if(per_fDOWN_saline(st)){
1408             return fDOWN_saline(st);
1409         }
1410         else {
1411             return st;
1412         }
1413     }
1414 
1415     public State press_btn_fDOWN_saline(State st){
1416         return click_btn_fDOWN_saline(st);
1417     }
1418 
1419     public State release_btn_fDOWN_saline(State st){
1420         return st;
1421     }
1422 
1423 //  manual contrast
1424     public boolean per_sUP_contrast(State st){
1425         return st.mode.equals(Mode.MANUAL);
1426     }
1427 
1428     public State sUP_contrast(State st){
1429         if(per_sUP_contrast(st)){
1430             st = push_plunger_contrast(SLOW,st);
1431             st.vol_saline = st.plunger_saline;
1432             st.vol_contrast = st.plunger_contrast;
1433             st.btn_manual_timeout = BTN_MANUAL_TIMEOUT;
1434         }
1435         return  st;
1436     }
1437 
1438     public State click_btn_sUP_contrast(State st){
1439         if(per_sUP_contrast(st)){
1440             return sUP_contrast(st);
1441         }
1442         else {
1443             return st;
1444         }
1445     }
1446 
1447     public State press_btn_sUP_contrast(State st){
1448         return click_btn_sUP_contrast(st);
1449     }
```

```
1450
1451    public State release_btn_sUP_contrast(State st){
1452        return st;
1453    }
1454
1455    public boolean per_fUP_contrast(State st){
1456        return st.mode.equals(Mode.MANUAL);
1457    }
1458
1459    public State fUP_contrast(State st){
1460        if(per_fUP_contrast(st)){
1461            st = push_plunger_contrast(FAST,st);
1462            st.vol_saline = st.plunger_saline;
1463            st.vol_contrast = st.plunger_contrast;
1464            st.btn_manual_timeout = BTN_MANUAL_TIMEOUT;
1465        }
1466        return st;
1467    }
1468
1469    public State click_btn_fUP_contrast(State st){
1470        if(per_fUP_contrast(st)){
1471            return fUP_contrast(st);
1472        }
1473        else{
1474            return st;
1475        }
1476    }
1477
1478    public State press_btn_fUP_contrast(State st){
1479        return click_btn_fUP_contrast(st);
1480    }
1481
1482    public State release_btn_fUP_contrast(State st){
1483        return st;
1484    }
1485
1486    public boolean per_sDOWN_contrast(State st){
1487        return st.mode.equals(Mode.MANUAL);
1488    }
1489
1490    public State sDOWN_contrast(State st){
1491        if(per_sDOWN_contrast(st)){
1492            st = pull_plunger_contrast(SLOW,st);
1493            st.vol_saline = st.plunger_saline;
1494            st.vol_contrast = st.plunger_contrast;
1495            st.btn_manual_timeout = BTN_MANUAL_TIMEOUT;
1496            st.prime_confirmed = false;
```

```
1497        }
1498        return set_LED_state(st);
1499    }
1500
1501    public State click_btn_sDOWN_contrast(State st){
1502        if(per_sDOWN_contrast(st)){
1503            return sDOWN_contrast(st);
1504        }
1505        else{
1506            return st;
1507        }
1508    }
1509
1510    public State press_btn_sDOWN_contrast(State st){
1511        return click_btn_sDOWN_contrast(st);
1512    }
1513
1514    public State release_btn_sDOWN_contrast(State st){
1515        return st;
1516    }
1517
1518    public boolean per_fDOWN_contrast(State st){
1519        return st.mode.equals(st);
1520    }
1521
1522    public  State fDOWN_contrast(State st){
1523        if(per_fDOWN_contrast(st)){
1524            st = pull_plunger_contrast(FAST,st);
1525            st.vol_saline = st.plunger_saline;
1526            st.vol_contrast = st.plunger_contrast;
1527            st.btn_manual_timeout = BTN_MANUAL_TIMEOUT;
1528            st.prime_confirmed = false;
1529        }
1530        return set_LED_state(st);
1531    }
1532
1533    public State click_btn_fDOWN_contrast(State st){
1534        if(per_fDOWN_contrast(st)){
1535            return click_btn_fDOWN_contrast(st);
1536        }
1537        else{
1538            return st;
1539        }
1540    }
1541
1542    public State press_btn_fDOWN_contrast(State st){
1543        return click_btn_fDOWN_contrast(st);
```

```
1544     }
1545
1546     public State release_btn_fDOWN_contrast(State st){
1547         return st;
1548     }
1549
1550 //  the device does not have these buttons, but these definitions are useful
         to keep the model more compact
1551
1552     public boolean per_btn_on(State st){
1553         return st.mode.equals(Mode.OFF);
1554     }
1555
1556     public State click_btn_on(State st){
1557         if(per_btn_on(st)){
1558             st.mode = Mode.INIT;
1559             st.display_contrast = Display.DISP_INIT;
1560             st.display_saline = Display.DISP_INIT;
1561         }
1562         return st;
1563     }
1564
1565     public State plug_syringe_saline (State st){
1566         if(st.console_screen.equals(ConsoleScreen.CONSOLE_PROTOCOL)){
1567             st.syringe_saline_present = true;
1568             if(per_tick(st)){
1569                 return tick(st);
1570             }
1571             else{
1572                 return st;
1573             }
1574         }
1575         else {
1576             return st;
1577         }
1578     }
1579     public State plug_syringe_contrast(State st){
1580         if(st.console_screen.equals(ConsoleScreen.CONSOLE_PROTOCOL)){
1581             st.syringe_contrast_present = true;
1582             if(per_tick(st)){
1583                 return tick(st);
1584             }
1585             else{
1586                 return st;
1587             }
1588         }
1589         else {
```

```
1590            return st;
1591        }
1592    }
1593    public State unplug_syringe_saline(State st){
1594        if(st.console_screen.equals(ConsoleScreen.CONSOLE_PROTOCOL)){
1595            st.syringe_saline_present = false;
1596            if(per_tick(st)){
1597                return tick(st);
1598            }
1599            else{
1600                return st;
1601            }
1602        }
1603        else{
1604            return st;
1605        }
1606    }
1607    public State unplug_syringe_contrast(State st){
1608        if(st.console_screen.equals(ConsoleScreen.CONSOLE_PROTOCOL)){
1609            st.syringe_contrast_present = false;
1610            if(per_tick(st)){
1611                return tick(st);
1612            }
1613            else{
1614                return st;
1615            }
1616        }
1617        else return st;
1618    }
1619
1620    public State plug_bag_saline(State st){
1621        if(st.syringe_saline_present && st.plunger_saline == 214){
1622            st.bag_saline_present = true;
1623            st.fluid_in_saline_syringe = true;
1624            return st;
1625        }
1626        else {
1627            return st;
1628        }
1629    }
1630    public State plug_bag_contrast(State st){
1631        if(st.syringe_contrast_present && st.plunger_contrast == 214){
1632            st.bag_contrast_present = true;
1633            st.fluid_in_contrast_syringe = true;
1634            return st;
1635        }
1636        else{
```

```
1637             return st;
1638         }
1639     }
1640     public State unplug_bag_saline(State st){
1641         if(st.syringe_saline_present){
1642             st.bag_saline_present = false;
1643             return st;
1644         }
1645         else {
1646             return st;
1647         }
1648     }
1649     public State unplug_bag_contrast(State st){
1650         if(st.syringe_contrast_present){
1651             st.bag_contrast_present = false;
1652             return st;
1653         }
1654         else {
1655             return st;
1656         }
1657     }
1658
1659     public State connect_infusion_set(State st){
1660         if(st.mode.equals(Mode.READY_TO_PRIME) ||
1661                 st.mode.equals(Mode.CONFIRM_PRIME)||
1662                 (st.mode.equals(Mode.MANUAL) && st.plunger_contrast > 0 && st.
                        plunger_saline > 0)){
1663             st.infusion_set_present = true;
1664             st.bag_contrast_present = false;
1665             st.bag_saline_present = false;
1666             return st;
1667         }
1668         else {
1669             return st;
1670         }
1671     }
1672
1673     public State restart_simulation(State st){
1674         st.init(0);
1675         return st;
1676     }
1677
1678
1679 //  console commands
1680     public boolean per_press_btn_ACC(State st){
1681         return st.console_screen.equals(ConsoleScreen.CONSOLE_INIT);
1682     }
```

```
1683     public State press_btn_ACC(State st){
1684         if(per_press_btn_ACC(st)){
1685             if(!st.console_btn_ACC.equals(ConsoleButton.PRESSED)){
1686                 st.console_btn_ACC = ConsoleButton.PRESSED;
1687                 st.console_btn_timeout = BTN_ACC_TIMEOUT;
1688                 return st;
1689             }
1690             else if(st.console_btn_ACC.equals(ConsoleButton.PRESSED) && st.
                    console_btn_timeout > 0){
1691                 if(st.console_btn_timeout - tick_step > 0){
1692                     st.console_btn_timeout = st.console_btn_timeout -
                            tick_step;
1693                 }
1694                 else {
1695                     st.console_btn_timeout = 0;
1696                 }
1697                 return st;
1698             }
1699             else if(st.console_btn_ACC.equals(ConsoleButton.PRESSED) && st.
                    console_btn_timeout == 0 && st.mode.equals(Mode.OFF)){
1700                 st.console_screen = ConsoleScreen.CONSOLE_SECURITY;
1701                 st.console_LED_ACC = ConsoleLED.GREEN;
1702                 st.console_btn_ACC = ConsoleButton.IDLE;
1703                 return click_btn_on(st);
1704             }
1705             else if(st.console_btn_ACC.equals(ConsoleButton.PRESSED) && st.
                    console_btn_timeout == 0
1706                     && !st.mode.equals(Mode.OFF) && !st.mode.equals(Mode.
                        INFUSING)){
1707                 boolean temp = st.injector_rotated;
1708                 st.init(0);
1709                 st.injector_rotated = temp;
1710                 return st;
1711             }
1712             else{
1713                 return st;
1714             }
1715         }
1716         else return st;
1717     }
1718     public boolean per_release_btn_ACC(State st){
1719         return st.console_btn_ACC.equals(ConsoleButton.PRESSED);
1720     }
1721 
1722     public State release_btn_ACC(State st){
1723         if(per_release_btn_ACC(st)){
1724             st.console_btn_ACC = ConsoleButton.IDLE;
```

```
1725            st.console_btn_timeout = 0;
1726            return st;
1727        }
1728        else {
1729            return st;
1730        }
1731    }
1732
1733    public boolean per_click_btn_confirm_security(State st){
1734        return (st.console_screen.equals(ConsoleScreen.CONSOLE_SECURITY) && st
                .mode.equals(Mode.INIT));
1735    }
1736    public State click_btn_confirm_security(State st){
1737        if(per_click_btn_confirm_security(st)){
1738            st.console_screen = ConsoleScreen.CONSOLE_PROTOCOL;
1739            st.display_saline = Display.MIRROR_PLUNGER_LEVEL;
1740            st.display_contrast = Display.MIRROR_PLUNGER_LEVEL;
1741            return tick(st);
1742        }
1743        else {
1744            return st;
1745        }
1746    }
1747
1748    public boolean per_confirm_air_check(State st){
1749        return (per_btn_confirm(st) && st.console_dlg.equals(ConsoleDLG.
                ASK_CONFIRM_AIR_CHECK));
1750    }
1751    public State click_btn_confirm_air_check(State st){
1752        if (per_confirm_air_check(st)){
1753            st = click_btn_confirm(st);
1754            st = click_btn_console_engage(st);
1755            if(st.console_dlg.equals(ConsoleDLG.ASK_CONFIRM_AIR_CHECK)){
1756                st.console_dlg = ConsoleDLG.NIL;
1757            }
1758            return st;
1759        }
1760        else {
1761            return st;
1762        }
1763    }
1764    public State click_btn_confirm_air_check_fail(State st){
1765        if (per_confirm_air_check(st)){
1766            st.console_dlg = ConsoleDLG.NIL;
1767            return st;
1768        }
1769        else {
```

```
1770            return st;
1771        }
1772    }
1773
1774    public boolean per_volume_warning(State st){
1775        return st.console_dlg.equals(ConsoleDLG.VOLUME_WARNING);
1776    }
1777    public State click_btn_confirm_volume_warning_ok (State st){
1778        if( per_volume_warning(st)){
1779            st.console_vol_contrast = st.plunger_contrast;
1780            st.vol_contrast = st.plunger_contrast;
1781            if(st.console_rate_contrast > 0 ){
1782                st.console_time_contrast = st.plunger_contrast / st.
                        console_rate_contrast;
1783            }
1784            else {
1785                st.console_time_contrast = 0;
1786            }
1787            st.vol_saline = st.plunger_saline;
1788            st.vol_saline = st.plunger_saline;
1789            if(st.console_rate_saline > 0){
1790                st.console_time_saline = st.plunger_saline / st.
                        console_rate_saline;
1791            }
1792            else{
1793                st.console_time_saline = 0;
1794            }
1795            st.console_locked = false;
1796            st.console_cmd = ConsoleCMD.LOCK;
1797            st.console_dlg = ConsoleDLG.NIL;
1798            return set_LED_state(st);
1799        }
1800        return st;
1801    }
1802    public State click_btn_confirm_volume_warning_fail(State st){
1803        if(per_volume_warning(st)){
1804            st.console_cmd = ConsoleCMD.ENGAGE;
1805            st.armed = false;
1806            st.infuse_mode = InfuseMode.NIL;
1807            st.console_dlg = ConsoleDLG.NIL;
1808            return set_LED_state(st);
1809        }
1810        else{
1811            return st;
1812        }
1813    }
1814
```

```
1815     public boolean per_start(State st){
1816         return (st.mode.equals(Mode.CONFIRM_PRIME)|| st.mode.equals(Mode.
                 READY_TO_PRIME)) && st.armed;
1817     }
1818     public State click_btn_start(State st){
1819         if(per_start(st)){
1820             st.mode = Mode.INFUSING;
1821             st.vol_saline_infused = 0;
1822             st.vol_contrast_infused = 0;
1823             return set_LED_state(st);
1824         }
1825         else{
1826             return st;
1827         }
1828     }
1829 
1830     public State click_btn_console_start(State st){
1831         return click_btn_start(st);
1832     }
1833 
1834     public State rotate_injector(State st){
1835         st.injector_rotated = !st.injector_rotated;
1836         return st;
1837     }
1838 
1839     public State init_precache(){
1840         State cache = new State();
1841         cache.init(0);
1842         console_screen = ConsoleScreen.CONSOLE_SECURITY;
1843         cache.mode = Mode.INIT;
1844         cache = click_btn_confirm_security(cache);
1845         cache.init(0);
1846         return cache;
1847     }
1848 
1849 }
```

B

DOCUMENTAÇÃO DAS FUNÇÕES DA *FRAMEWORK*

createButton method allows a definitiion of a device button, as well as his associated function that should be executed at the button pressing.

Parameters:

Name	Type	Description
name	String	the button name.
measures	var	variable that contains information about certain measures like width, height and left and top margins.
other	var	variable that contains optional and required informaation like name of associated function and and information about ui changes after button pressiong.

createDisplay(name, measures, other)

createDisplay method allows a definitiion of a device display where can be represented any kind of text.

Parameters:

Name	Type	Description
name	String	the display name.
measures	var	variable that contains information about certain measures like width, height and left and top margins.
other	var	variable that contains either optional or required information like starting text of the display color and font of the displayed text and visibility of the display.

createFunction(name, type)

createFunction method allows a definitiion of background code methods signature.

Parameters:

Name	Type	Description
name	String	method name.
type	String	method return type.

createGVariable(name, type, value)

createGVariable method allows a declaration of a global variable to be used by the background code of the simulation.

Parameters:

Name	Type	Description
name	String	the variable name.
type	String	the variable type.
value	undifined	the initial value of the variable

createImage(name, measures, other)

createImage method allows a definitiion of a image or set of images that will be part of the user interface. In the Android app images will be disposed in layers and at the bottom layer will be the fist image defined and at the top layer will be the last.

Parameters:

Name	Type	Description
name	String	the image name.
measures	var	variable that contains information about certain measures like top and left margins.
other	var	variable that contains either optional or required information like information about visibility of the image.

createJava(path)

createJava method allows the creation of JAVA and XML files, that will be used to code the Android app. Also build an Android Studio project with the new files created and any images that the user may want to set as part of the device interface.

Parameters:

Name	Type	Description
path	String	directory location where the new Android Studio should be created.

setAPPName(name)

setAPPName method allows definition of Android application name.

Parameters:

Name	Type	Description
name	String	Android application name

setCanvasScale(scale)

setCanvasScale method allows definition of canvasScale variable value.

Parameters:

Name	Type	Description
scale	double	canvasScale value

setOrientation(ori)

setOrientation method allows definition of Android application orientation.

Parameters:

Name	Type	Description
ori	String	Android application orientation

C

SCRIPTS DE CONSTRUÇÃO DE DISPOSITIVOS

C.1 STELLANT V2

```
var module = require('./module');

module.setAPPName("StellantV2");
module.setCanvasScale(2.8);
module.setOrientation("PORTRAIT");

module.createGVariable("MAX_VOLUME","int",230);
module.createGVariable("VOL_BUFFER","int",10);
module.createGVariable("MAX_RATE","int",200);
module.createGVariable("PlungerLevel","int",0);
module.createGVariable("FAST","int",10);
module.createGVariable("SLOW","int",1);
module.createGVariable("DEFAULT_VOLUME_SALINE","int",224);
module.createGVariable("DEFAULT_VOLUME_CONTRAST","int", 224);
module.createGVariable("AUTOLOAD_STEP","int",6);
module.createGVariable("PRIME_VOLUME_SALINE","int",3);
module.createGVariable("PRIME_VOLUME_CONTRAST","int",1);
module.createGVariable("tick_step","float",250);
module.createGVariable("BTN_ACC_TIMEOUT","float",750);
module.createGVariable("BTN_MANUAL_TIMEOUT","float",3000);
module.createGVariable("BTN_AUTO_TIMEOUT","float",8000);
module.createGVariable("step","int",5);


module.createButton("inc_saline",{ left:174, top:607, width:23, height:23},
    {function:"press_inc_saline",display:false});
module.createButton("dec_saline",{ left:174, top:642, width:23, height:23},
    {function:"press_dec_saline",display:false});
module.createButton("inc_contrast",{ left:260, top:607, width:23, height
    :23}, {function:"press_inc_contrast",display:false});
module.createButton("dec_contrast",{ left:260, top:642, width:23, height
    :23}, {function:"press_dec_contrast",display:false});
```

```
29  module.createButton("btn_auto",{ left:214, top:625, width:33, height:33}, {
        function:"click_btn_auto",display:false});
30  module.createButton("btn_manual",{ left:214, top:765, width:33, height:33},
        {function:"click_btn_manual",display:false});
31  module.createButton("btn_fup_saline",{ left:115, top:680, width:65, height
        :80}, {function:"press_btn_fUP_saline",display:false});
32  module.createButton("btn_sup_saline",{ left:115, top:760, width:65, height
        :58}, {function:"press_btn_sUP_saline",display:false});
33  module.createButton("btn_sdown_saline",{ left:115, top:818, width:65, height
        :58}, {function:"press_btn_sDOWN_saline",display:false});
34  module.createButton("btn_fdown_saline",{ left:115, top:876, width:65, height
        :65}, {function:"press_btn_fDOWN_saline",display:false});
35  module.createButton("btn_fup_contrast",{ left:275, top:680, width:65, height
        :80}, {function:"press_btn_fUP_contrast",display:false});
36  module.createButton("btn_sup_contrast",{ left:275, top:760, width:65, height
        :58}, {function:"press_btn_sUP_contrast",display:false});
37  module.createButton("btn_sdown_contrast",{ left:275, top:818, width:65,
        height:58}, {function:"press_btn_sDOWN_contrast",display:false});
38  module.createButton("btn_fdown_contrast",{ left:275, top:876, width:65,
        height:65}, {function:"press_btn_fDOWN_contrast",display:false});
39  module.createButton("btn_fill_saline",{ left:110, top:620, width:60, height
        :34}, {function:"click_btn_fill_saline",display:false});
40  module.createButton("btn_fill_contrast",{ left:295, top:620, width:60,
        height:34}, {function:"click_btn_fill_contrast",display:false});
41  module.createButton("btn_prime",{ left:215, top:697, width:33, height:3}, {
        function:"click_btn_prime",display:false});
42  module.createButton("btn_confirm",{ left:215, top:836, width:33, height:33},
         {function:"click_btn_confirm",display:false});
43  module.createButton("btn_engage",{ left:215, top:913, width:33, height:33},
        {function:"click_btn_console_engage",display:false});
44  module.createButton("btn_stop",{ left:120, top:970, width:93, height:46}, {
        function:null,display:false});
45  module.createButton("btn_start",{ left:242, top:970, width:93, height:46}, {
        function:"click_btn_start",display:false});
46  module.createButton("btn_on",{ left:690, top:962, width:23, height:23}, {
        function:"press_btn_ACC",display:true});
47  module.createButton("confirm_security_btn",{ left:655, top:778, width:130,
        height:35}, {function:"click_btn_confirm_security",display:true});
48  module.createButton("rotate_injector",{ left:620, top:235, width:210, height
        :28}, {function:"rotate_injector",display:false});
49  module.createButton("plug_syringe_saline",{ left:620, top:290, width:100,
        height:28}, {function:"plug_syringe_saline",display:false});
50  module.createButton("plug_syringe_contrast",{ left:738, top:290, width:100,
        height:28}, {function:"plug_syringe_contrast",display:false});
51  module.createButton("spike_saline_bag",{ left:620, top:330, width:100,
        height:27}, {function:"plug_bag_saline",display:false});
```

```
52  module.createButton("spike_contrast_bag",{ left:738, top:365, width:100,
        height:27}, {function:"displayplug_bag_contrast",display:false});
53  module.createButton("connect_infusion_set",{ left:620, top:370, width:217,
        height:28}, {function:"connect_infusion_set",display:false});//
54
55
56  module.createDisplay("tv1", {left:120, top:624, width:100, height:50}, {
        startText:"---", textsize:25, color:"#2DFF1B", font:"fonts/abc.ttf",
        visible:false});
57  module.createDisplay("tv2", {left:280, top:624, width:100, height:50}, {
        startText:"---", textsize:25, color:"#0000ff", font:"fonts/abc.ttf",
        visible:false});
58
59
60  module.createImage("all", {left:0, top:0}, {visible:true});
61  module.createImage("buttons_10", {left:0, top:0}, {visible:true});
62  module.createImage("console_led_off", {left:0, top:0}, {visible:true});
63  module.createImage("empty_console", {left:0, top:0}, {visible:true});
64
65  module.createFunction("inc_saline", "State");
66  module.createFunction("dec_saline", "State");
67  module.createFunction("inc_contrast", "State");
68  module.createFunction("dec_contrast", "State");
69  module.createFunction("btn_auto", "State");
70  module.createFunction("btn_fup_saline", "State");
71  module.createFunction("btn_sup_saline", "State");
72  module.createFunction("btn_sdown_saline", "State");
73  module.createFunction("btn_fdown_saline", "State");
74  module.createFunction("btn_fup_contrast", "State");
75  module.createFunction("btn_sup_contrast", "State");
76  module.createFunction("btn_sdown_contrast", "State");
77  module.createFunction("btn_fdown_contrast", "State");
78  module.createFunction("btn_fill_saline", "State");
79  module.createFunction("btn_fill_contrast", "State");
80  module.createFunction("btn_prime", "State");
81  module.createFunction("btn_confirm", "State");
82  module.createFunction("btn_engage", "State");
83  module.createFunction("btn_stop", "State");
84  module.createFunction("btn_start", "State");
85  module.createFunction("btn_on", "State");
86  module.createFunction("confirm_security_btn", "State");
87  module.createFunction("rotate_injector", "State");
88  module.createFunction("plug_syringe_saline", "State");
89  module.createFunction("plug_syringe_contrast", "State");
90  module.createFunction("spike_saline_bag", "State");
91  module.createFunction("spike_contrast_bag", "State");
92  module.createFunction("connect_infusion_set", "State");
```

```
93
94
95   module.createJava("/Users/andrepinto/Desktop");
```

Excertos de Código C.1: Código *JavaScript* da *framewrok* para criação do projeto para a aplicação *Stellant V2*

C.2 RADICAL-7

```
1  var module = require('./module');
2
3  module.setAPPName("Radical7");
4
5
6  module.setCanvasScale(2.7);
7  module.setOrientation("LANDSCAPE");
8
9  module.createGVariable("id","String", "Radical7");
10 module.createGVariable("spo2","float", "99");
11 module.createGVariable("spo2_max","float", "0");
12 module.createGVariable("spo2_min","float", "88");
13 module.createGVariable("spo2_label","String", "SpO2");
14 module.createGVariable("spo2_alarm","Alarm", "off");
15 module.createGVariable("spo2_fail","boolean", "false");
16 module.createGVariable("rra","float", "99");
17 module.createGVariable("rra_max","float", "0");
18 module.createGVariable("rra_min","float", "88");
19 module.createGVariable("rra_label","String", "RRa");
20 module.createGVariable("rra_alarm","Alarm", "off");
21 module.createGVariable("rra_fail","boolean", "false");
22 module.createGVariable("isOn","boolean", "true");
23
24 module.createDisplay(
25   "disp1",
26   {top:220, left:820, width:100, height:50},
27   {startText:"99",textsize:25,visible:"false",color:"#ffffff"});
28
29 module.createDisplay(
30   "disp2",
31   {top:340, left:820, width:100, height:50},
32   {startText:"99",textsize:25,visible:"false",color:"#ffffff"});
33
34 module.createImage(
35   "radical_7",
36   {top:0, left:0},
```

```
37   {visible:"true"});
38
39 module.createFunction ("per_on", "boolean");
40 module.createFunction ("on", "State");
41 module.createFunction ("click_btn_on", "State");
42 module.createFunction ("click_btn_mute", "State");
43 module.createFunction ("check_spo2", "State");
44 module.createFunction ("check_rra", "State");
45 module.createFunction ("check_vitals", "State");
46 module.createFunction ("tick", "State");
47 module.createFunction ("spo2_sensor_data", "State");
48 module.createFunction ("rra_sensor_data", "State");
49 module.createFunction ("init", "State");
50
51
52
53 module.createButton (
54   "btn_on",
55   {top:330, left:1040, width:50, height:50},
56   {function:"btn_on"});
57
58 module.createJava("/Users/andrepinto/Desktop");
```

Excertos de Código C.2: Código *JavaScript* da *framewrok* para criação do projeto para a aplicação *Radical-7*