Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Departamento de Engenharia Informática e de Sistemas

Whiteboard: Ferramenta de desenho colaborativo integrada numa aplicação RCS

Mestrado de Informática e Sistemas

Desenvolvimento de Software

Autor

Diogo Parreira

Orientadores

Doutora Teresa Rocha

ISEC

Mestre Tiago Leitão

WIT Software

Coimbra, Março de 2015

Agradecimentos

As minhas primeiras palavras de agradecimento vão para todos os professores do DEIS e do ISEC, em especial à Professora Doutora Teresa Rocha pelo apoio e disponibilidade prestada durante a Licenciatura, Mestrado, e decorrer do estágio.

À WIT Software e ao meu orientador de estágio da WIT, Mestre Tiago Leitão por me possibilitarem a experiencia enriquecedora que foi este estágio de mestrado, e por investirem na minha formação.

Aos meus amigos e minha família, em especial aos meus pais por me terem proporcionado a oportunidade de apostar na minha formação académica e apoio que me deram ao longo de todo este percurso. Este agradecimento estende-se também, à Neide Batista pela motivação e apoio diário.

Resumo

A comunicação visual através de esquemas e desenhos, hoje essencial na nossa sociedade, já remonta aos tempos da Pré-História em que as pinturas rupestres eram um veículo utilizado para a transmissão de ideias e conceitos.

Uma das vantagens da comunicação visual é a sua universalidade. Na verdade, as representações da informação sob a forma de desenhos são facilmente interpretadas e compreendidas de igual forma até pelos diferentes povos, como é o exemplo dos sinais de trânsito. Por outro lado, a simplicidade com que um determinado conceito é apresentado, torna-o mas fácil de apreender/memorizar sem necessidade de recorrer a um modelo mental elaborado. Com efeito, os desenhos permitem remover a sobrecarga de memorizar um conceito explicado por palavras, atribuindo um papel mais importante à memória visual. Tal como dizia Confúcio, uma imagem vale mais que mil palavras.

É do senso comum que colocar ideias e pensamentos sob a forma de um desenho ou esquema, suscita a inspiração e estimula a criatividade, o que pode constituir um primeiro passo na criação de novos produtos ou projetos. Embora as empresas sejam entidades que podem beneficiar deste tipo de comunicação, na realidade atual, em que a distância entre os colaboradores é cada vez maior, torna-se difícil a troca de ideias e conceitos através de desenhos.

A constatação deste facto levou à proposta do presente estágio por parte da WIT Software S.A., empresa especializada no desenvolvimento de aplicações e serviços para operadores móveis e empresas de telecomunicações. Com efeito, o objetivo deste estágio é desenvolver uma ferramenta de desenho colaborativo, tendo as empresas como público-alvo. Esta ferramenta deverá ser integrada numa aplicação Rich Communication Services (RCS) para dispositivos Android, a qual permita que utilizadores do serviço RCS partilhem e colaborem, em tempo real, em sessões de desenho colaborativo. Os desenhos poderão ser modificados por qualquer utilizador da sessão, sendo sincronizados, em tempo real, por todos os outros utilizadores dentro da mesma sessão. Criar-se-á, deste modo, uma sessão de desenho dinâmica, em que várias pessoas poderão discutir ideias independentemente da distância entre elas.

Palavras Chave: Desenho colaborativo, Whiteboard, RCS

Abstract

Visual communication through drawings and diagrams, so important in today's society, dates back to prehistoric times when the rupestrian paintings were a vehicle used for the transmission of ideas and concepts.

One of the advantages of visual communication is its universality. In fact, the representations of data in the form of drawings, are interpreted and understood in the same way by different people all over the world, as is the case of traffic signs. On the other hand, the simplicity by which a given concept is presented, makes it easy to learn / memorize without the need for a developed mental model. As a matter of fact, the drawings avoid the mental effort of memorizing a concept explained through words, by attributing a more important role to visual memory. As Confucius said, a picture is worth a thousand words.

It is common sense that placing ideas and thoughts in the form of drawings or diagrams induces inspiration and stimulates creativity, which can be a first step in the conception of new products or projects. Although companies are entities that can benefit from this type of communication, in the present context, where distance between employees is increasing, it is difficult to exchange ideas and concepts through drawings.

The realization of this fact led to the proposal of the present internship by WIT Software S.A., a software company that creates applications and services for telecommunication and media companies. In effect, the goal of this internship is to develop a collaborative drawing tool with companies as the target audience. This tool should be integrated into an application Rich Communication Suite (RCS) for Android devices, which allow users of RCS service to share and collaborate in real time, in collaborative drawing sessions. Drawings can be modified by any user in the session, being synchronized, in real time, by all other users in the same session. Therefore, a dynamic design session will be created, allowing multiple people to discuss ideas regardless of the distance between them.

Índice

Índice de Figuras

Índice de Tabelas

Índice de Gráficos

Acrónimos

GSM – Groupe Spéciale Mobile

GSMA – GSM Association

IP – Internet Protocol

ISEC – Instituto Superior de Engenharia Informática

JSON - JavaScript Object Notation

LTE – Long-Term Evolution

OTT – Over-the-top

RCS – Rich Communication Services

TV – Televisão

VoIP – Voice over IP

VoLTE – Voice over LTE

WebRTC – Web Real-Time Communication

IDE – Integrated Development Environment

SDK – Software Development Kit

1. Introdução

O presente documento visa descrever o trabalho de estágio do aluno Diogo Parreira, realizado no âmbito da unidade curricular *Estágio ou Projeto Industrial* do curso de Mestrado em Informática e Sistemas do Instituto Superior de Engenharia de Coimbra, na empresa WIT Software S.A..

Seguidamente proceder-se-á ao enquadramento do estágio e à apresentação dos seus objetivos, bem como à descrição da estrutura do documento.

1.1. Enquadramento do estágio

A comunicação visual através de esquemas e desenhos é algo essencial na sociedade dos nossos dias e que remonta aos tempos pré-históricos em que já se usavam pinturas rupestres para transmitir ideias e conceitos. Relativamente à forma verbal, este tipo de comunicação tem a vantagem de ser universal, já que é interpretado e entendido da mesma forma por diferentes pessoas nos mais variados pontos do planeta. Por outro lado, tem a capacidade de simplificar as ideias ou os conceitos a transmitir, tornando-os mais fáceis de absorver sem necessidade de um modelo mental elaborado. Com efeito, os desenhos permitem remover a sobrecarga de memorizar um conceito explicado por palavras, atribuindo um papel mais importante à memória visual. Tal como dizia Confúcio, uma imagem vale mais que mil palavras.

As empresas são entidades que podem beneficiar deste tipo de comunicação, pois é cada vez maior a necessidade que sentem de transmitir e trocar ideias e conceitos interactivamente, sendo que nem sempre as palavras são o meio mais adequado de os descrever. Aliás, é sabido que a representação de ideias através de desenhos ou esquemas estimula a inspiração e a criatividade. Por outro lado, cada vez mais as empresas têm que trabalhar e interagir com pessoas que estão bastante distantes.

Desta constatação, surgiu a ideia do presente estágio por parte da WIT Software, de criar uma ferramenta de desenho colaborativo que permita a partilha de desenhos entre os vários colaboradores de uma empresa, em tempo real, independentemente da distância entre eles.

A WIT Software é uma empresa especializada no desenvolvimento de *software* com mais de 13 anos de experiência na criação de aplicações e serviços em várias áreas, nomeadamente nas áreas de comunicações sobre IP, Tecnologia RCS, VoIP, VoLTE, sistemas e serviços baseados

em IMS, WebRTC, OTT-TV, sistemas TV para *multi-screen*, Mobile Banking e Mobile Payments.

A visão da empresa baseia-se no objetivo de criar de um novo estilo de vida no que toca a comunicações *mobile*, um estilo que torne as pessoas mais produtivas, mais inspiradas e mais ligadas entre si.

Atualmente a WIT Software está dividida em duas secções: a secção de produtos, na qual existem alguns produtos de marca branca[1], como o RCS Suite, o WebRTC Gateway e o OTT TV Suite; e a secção de serviços, que consiste no desenvolvimento de soluções à medida para clientes. Embora exista esta divisão na empresa esta não se reflete diretamente nas equipas, em muitos dos projetos é a mesma equipa que faz o desenvolvimento tanto do produto como das soluções à medida baseadas nestes produtos.

Como todos os produtos da WIT Software, a ferramenta de desenho colaborativo a desenvolver está também em sintonia com a visão da empresa. Com efeito, esta aplicação destina-se a ser usada para partilhar e criar ideias, à distância, e assim permitir aumentar a produtividade dos seus utilizadores, tendo o meio empresarial como principal alvo.

1.2. Objetivos

Com o presente trabalho de estágio pretende-se o desenvolvimento de uma aplicação de desenho colaborativo para *tablets* e *smartphones* Android, a qual deverá ser integrada num dos produtos da WIT Software, o RCS Suite.

O RCS Suite é um produto para operadoras móveis, totalmente compatível com a especificação Joyn Blackbird [2] da GSMA que inclui um conjunto de aplicações RCS para *smartphones*, *tablets*, PC, *browsers* de internet e também uma aplicação servidor RCS. Este produto representa a proposta da indústria de telecomunicações para mensagens, voz e vídeo sobre IP, com o objetivo de combater aplicações OTT como WhatsApp, Viber, Line, Skype, WeChat e muitas outras que têm um grande impacto nas receitas das operadoras. O RCS Suite engloba uma série de funcionalidades que passam pela sincronização de contactos, chamadas sobre IP, mensagens de *chat* 1-para-1 e em grupo, partilha de ficheiros, partilha de localização, e muitas outras funcionalidades que fazem desta aplicação uma aplicação prática de utilizar.

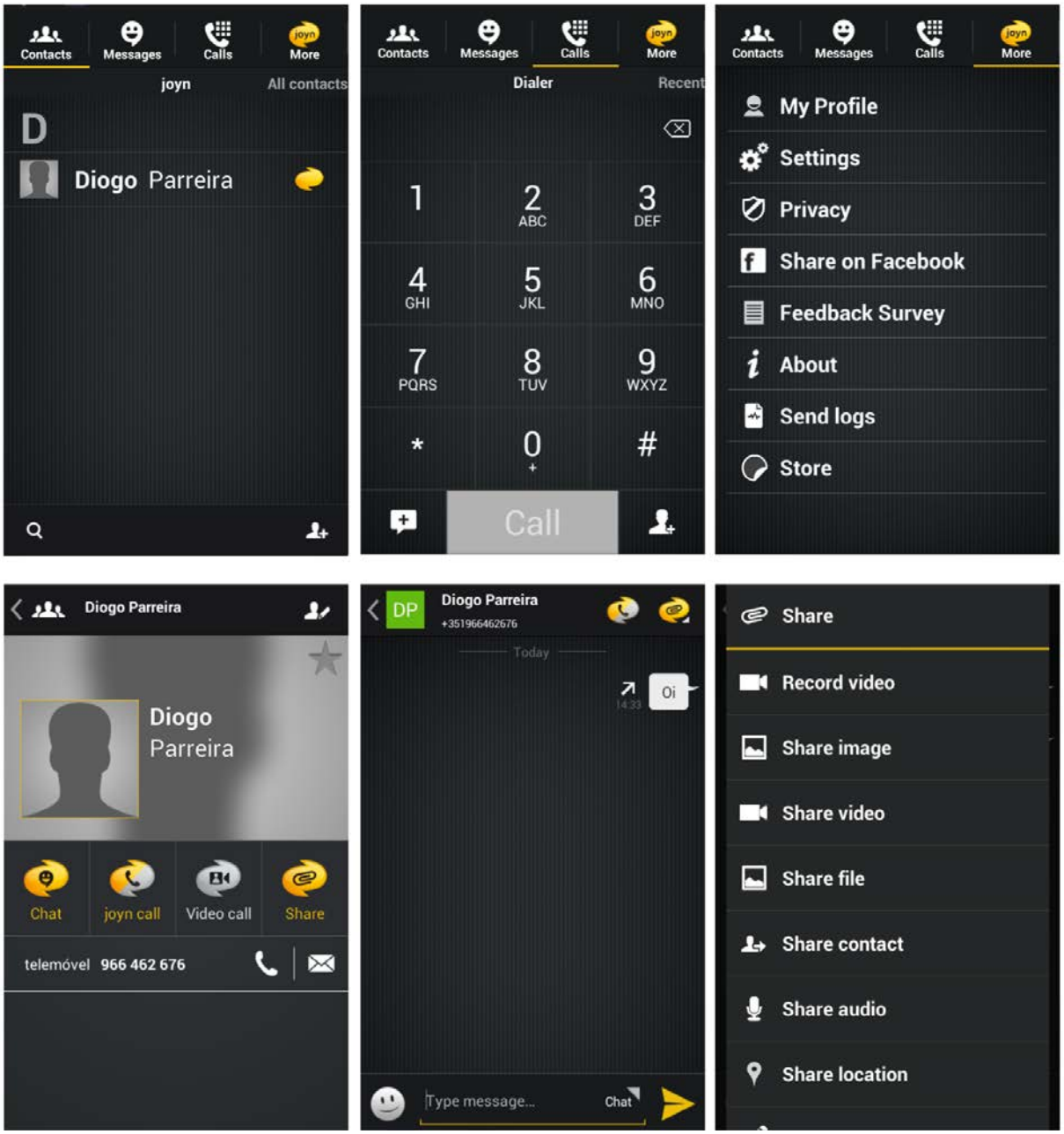


Figura 1 Alguns ecrãs da aplicação RCS Suite da WIT Software

A aplicação de desenho colaborativo a desenvolver deverá ser integrada no sistema de *chat* e partilha de ficheiros (ecrãs inferior central e inferior direito da Figura 1), de modo a que o utilizador para além de poder partilhar texto, voz, ficheiros, etc., possa também facilmente partilhar e trocar ideias de forma fácil, dinâmica, e intuitiva. Desta forma, esta nova funcionalidade acrescentará valor ao RCS Suite pois tornará o utilizador mais produtivo, ao reduzir barreiras no que toca à criatividade, inspirando a criação e desenvolvimento de novas ideias.

1.3. Estrutura do documento

O presente relatório encontra-se dividido nos seguintes capítulos:

1. Introdução;
2. Metodologia de trabalho;
3. Estado da arte;
4. Desenvolvimento da aplicação;
5. Conclusões.

No primeiro capítulo, **Introdução,** é feito o enquadramento do trabalho a desenvolver e são apresentados os objetivos a atingir com o mesmo.

No segundo capítulo, **Metodologia e Planeamento**, é explicada de forma detalhada a metodologia de trabalho, bem como a calendarização definida para o estágio.

O capítulo seguinte, **Estado da arte,** contém uma análise de ferramentas de desenho colaborativo já existentes no mercado, que inclui uma análise das funcionalidades que estas proporcionam, bem como para quais plataformas estão disponíveis. Também foi efetuada uma pequena análise da utilidade que cada ferramenta tem tendo em conta a sua facilidade de utilização, e a partilha do desenho com outros utilizadores.

O quarto capítulo, **Desenvolvimento da aplicação**, contém uma secção das ferramentas utilizadas durante o estágio, inclui a análise de requisitos e a arquitetura da nova aplicação, apresenta uma descrição do progresso de desenvolvimento da mesma ao longo do estágio, bem como dos testes efetuados e das decisões tomadas. No fim tem uma secção que dá a conhecer a aplicação desenvolvida durante o estágio.

No quinto capítulo, **Conclusões**, é feita uma análise à forma como decorreu o estágio, e são apresentadas as principais conclusões a retirar do mesmo. São igualmente apresentadas as perspetivas de trabalho futuro.

2. Metodologia e Planeamento

O presente capítulo apresenta a metodologia de desenvolvimento de *software* utilizada durante o estágio, e o planeamento definido na fase inicial do mesmo relativamente ao plano de trabalhos a desenvolver durante o primeiro e o segundo semestre em que este decorreu.

A primeira secção explica o funcionamento da metodologia adotada e justifica o porquê da adoção desta. A segunda secção faz uma descrição da planificação definida e como está relacionada com a metodologia de desenvolvimento de *software* adotada.

Por fim é feita uma descrição dos desvios do realizado em relação ao planeado, e devida justificação.

2.1. Metodologia base utilizada na WIT Software

Na WIT Software alguns dos projetos utilizam uma metodologia ágil baseada em Scrum adaptada às necessidades da empresa e dos clientes, a qual foi a base também para este projeto. Esta metodologia segue os padrões comuns do Scrum, em que existe o *product backlog* com todas as *user stories*, das quais são selecionadas as mais prioritárias, e são colocadas e detalhadas no *Sprint backlog* do *sprint* a ser executado. Depois do *sprint* começar, existe um "daily meeting" todos os dias. Usualmente, os *sprints* têm entre 2 e 4 semanas.

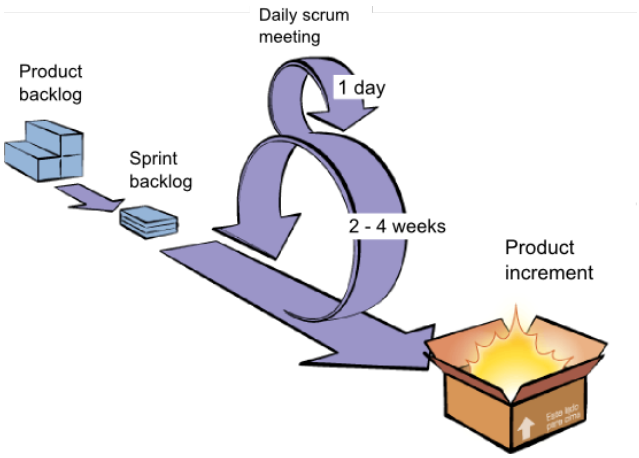


Figura 2 Modo de funcionamento do Scrum em alguns dos projetos na WIT Software

A metodologia original utilizada nos projetos da empresa foca-se em desenvolver e entregar o produto em várias iterações rápidas.

2.2. Metodologia adaptada à realidade do estágio

Neste estágio foi utilizada a mesma metodologia ágil que costuma ser utilizada na WIT Software, mas com algumas adaptações de forma a ajustar-se à realidade do mesmo. Esta mudança tem o objetivo de ajustar e flexibilizar o decorrer do estágio visto o ambiente e o fator de aprendizagem serem muito diferentes em relação a um projeto normal. No entanto, mantêm-se grande parte dos conceitos a características que fazem parte do Scrum aplicado pela empresa.

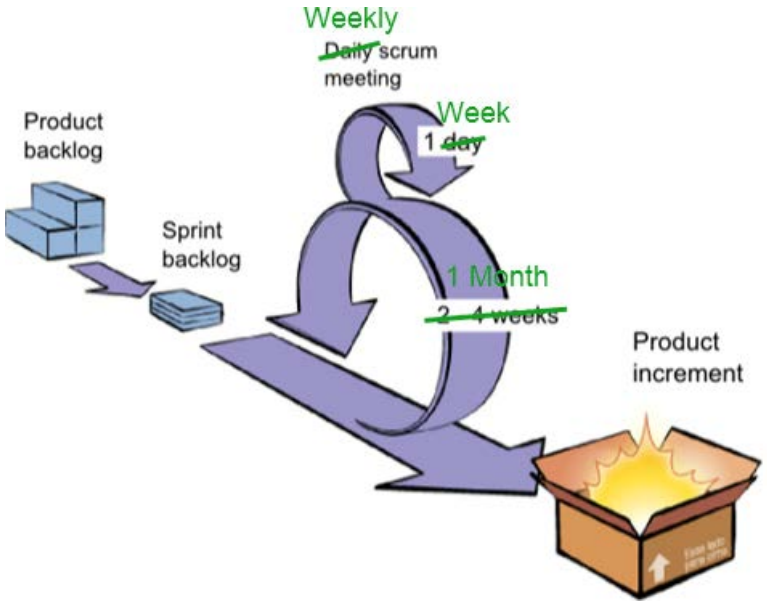


Figura 3 Funcionamento do processo do Scrum adaptado ao estágio

As mudanças principais são em relação ao "daily scrum meeting" que é realizado apenas uma vez por semana, e os *sprints* ficam definidos para durarem 4 semanas.

A equipa é constituída por:

- Tiago Leitão – Scrum Master;
- Rui Gil – Product Owner técnico;
- Pedro Pereira – Product Owner de requisitos;
- Diogo Parreira;
- Filipe Santos – Representante da equipa de testes.

No fim de cada *sprint* a aplicação foi testada pela equipa de testes (Software Quality Assurance) da WIT Software.

2.3. Planeamento do estágio

No início do estágio foi feito um planeamento inicial constituído por várias tarefas a realizar no decorrer do estágio.

As tarefas inicialmente definidas são as seguintes:

Tarefa 1 – Estudos preliminares - Leitura de documentação sobre tecnologias e protocolos de comunicação utilizadas no projeto RCS Suite, aprofundamento dos conhecimentos de desenvolvimento de aplicações para a plataforma Android.

Tarefa 2 - Estado da arte – Análise de ferramentas similares já existentes no mercado.

Tarefa 3 – Levantamento de requisitos – Obtenção, análise e especificação dos requisitos definidos pelos Product Owner após análise do estado da arte.

Tarefa 4 – Familiarização com a aplicação base – Familiarização com a aplicação RCS Suite Android e a sua organização, estrutura e API.

Tarefa 5 – Arquitetura da aplicação – Planificação da arquitetura da aplicação/módulo de desenho colaborativo da aplicação RCS Suite.

Tarefa 6 – Implementação – Implementação e desenvolvimento da aplicação segundo os requisitos.

Tarefa 7 – Relatório – Elaboração do relatório de estágio.

As tarefas foram também escalonadas pelo tempo definido para o estágio. A divisão das tarefas foi feito por mês de trabalho, e está assim detalhado na seguinte tabela.

	1	2	3	4	5	6	7
Tarefa 1							
Tarefa 2							
Tarefa 3							
Tarefa 4							
Tarefa 5							
Tarefa 6							
Tarefa 7							

- Tarefa planeada para o mês

Tabela 1 Tabela de planeamento das tarefas para o decorrer do estágio

2.4. Desvios do planeamento

Durante o primeiro mês de estágio surgiu a ideia de integrar a aplicação a desenvolver como sendo uma aplicação *plugin* em relação à aplicação RCS Suite, deste modo o planeamento inicial foi alterado, a fase de familiarização com a aplicação RCS Suite e integração com a aplicação passou para depois da implementação e desenvolvimento da aplicação, e foi criada uma nova tarefa (Tarefa 8) que consistiu em integrar a aplicação desenvolvida com a aplicação RCS Suite.

O sistema de *plugins* da aplicação RCS Suite serve para múltiplos propósitos, no entanto neste caso tinha como objetivo a criação de módulos removíveis de forma a não sobrecarregar a aplicação base e prejudicar assim a experiencia do utilizador.

No fim, esta integração por *plugin* acabou por não se realizar devido a atrasos no desenvolvimento da tecnologia de *plugins* por parte da equipa de desenvolvimento da aplicação RCS Suite Android, e consequentemente foi integrada num formato permanente.

Para além da mudança descrita anteriormente existiram outros ajustes devido à natureza dinâmica de um projeto desenvolvido com uma metodologia ágil.

A seguinte tabela apresenta a calendarização efetiva das tarefas.

	1	2	3	4	5	6	7
Tarefa 1							
Tarefa 2							
Tarefa 3							
Tarefa 5							
Tarefa 6							
Tarefa 4							
Tarefa 8							
Tarefa 7							

- Tarefa planeada para o mês

Tabela 2 Tabela da calendarização efetiva do estágio

Nas secções 4.2 e 4.3 é apresentada a divisão das tarefas e *user stories* por *sprints*, sendo possível ver um planeamento mais detalhado das tarefas e das datas correspondentes.

3. Estado da Arte

Aplicações de desenho colaborativo são ferramentas bastante recentes, mas hoje em dia já existe um vasto leque de oferta das mesmas. Estas podem usar uma multitude de tecnologias tanto para a transmissão de informação pela internet como para a impressão do desenho no ecrã. Para além disso este tipo de aplicações estão disponíveis para grande parte das plataformas mais conhecidas, como Android, iOS, *browsers* de internet, e Windows. E, embora muitas destas aplicações apenas suportem uma destas plataformas, algumas estão disponíveis para várias se não para todas.

Uma aplicação deste tipo pode disponibilizar uma série de funcionalidades, das quais umas são mais importantes que outras. Portanto, uma análise aos produtos já existentes pode beneficiar bastante um novo produto que esteja a ser criado. O objetivo desta aplicação é ser integrada no produto RCS Suite da WIT Software, caso venha a ser uma ferramenta útil a nível empresarial. Nesse sentido, foi feita uma análise do Estado da Arte para se inferir o que distingue cada uma destas aplicações e juntar o que cada uma tem de melhor, para resultar numa aplicação de excelência e com utilidade.

3.1. Seleção de ferramentas

Começou por procurar-se as ferramentas de desenho colaborativo que se enquadravam no mesmo contexto da ferramenta que se pretendia desenvolver. Da pesquisa efetuada resultaram as seguintes ferramentas:

- Google Drawings[3];
- Scribblar[4];
- CoSketch.com[5];
- FlockDraw[6];
- GroupBoard[7];
- Mighty Meeting[8];
- Chalkboard [9];
- Whiteboard: Collaborative Draw[10];
- SyncSpace Shared Whiteboard[11];
- Shared Board[12];
- Lucid Chart[13];
- Concept Board[14].

Estas foram as ferramentas consideradas relevantes para o enquadramento da aplicação a desenvolver com as aplicações existentes atualmente no mercado, todas elas têm características similares à ferramenta que se pretende criar e, portanto, foram consideradas relevantes para a análise que foi efetuada.

3.2. Seleção das funcionalidades importantes

Numa curta análise inicial foram escolhidas pela equipa as funcionalidades consideradas como as mais importantes em ferramentas deste género, bem como outras funcionalidades com potencial de poder acrescentar valor.

Com efeito, a lista seguinte apresenta apenas aquelas que foram identificadas como as mais relevantes:

- Desenho livre;
- Inserção de formas (retângulos, círculos, triângulos, etc.);
- Disponibilizar múltiplas cores;
- Ter uma borracha para poder apagar o desenho;
- Mudar a grossura da "caneta" de desenho livre e da borracha;
- Inserção de texto;
- Salvar o estado do desenho para se poder continuar mais tarde;
- Importar imagens para o desenho.

Outras funcionalidades consideradas extras, mas que podem ser úteis ao utilizador, são as seguintes:

- Salvar a imagem;
- Poder fazer *undo/redo* às ações efetuadas no desenho;
- *Chat* integrado (texto/áudio).

Para além destas, foram também consideradas de interesse algumas características da aplicação que podem influenciar a experiência do utilizador e, consequentemente, o valor que ele dá à aplicação. Nomeadamente:

- A necessidade de ter ferramentas extras como Java ou Flash para funcionarem;
- A necessidade de criar/ter uma conta para poder colaborar no desenho;
- A quantidade de pessoas permitidas a colaborar ao mesmo tempo na ferramenta;
- As plataformas para qual a aplicação está disponível;

- O preço da aplicação.

Existem muitas outras funcionalidades que poderiam ser consideradas relevantes, como partilha nas redes sociais. No entanto para esta primeira iteração teve-se mais em atenção as necessidades mais imediatas para o público-alvo definido.

Na característica das plataformas para as quais uma aplicação está disponível existe uma particularidade que é a aplicação estar disponível para *browser* de internet e não estar para dispositivos Android e iOS. No entanto, estes dispositivos têm também *browsers*. Neste caso, foram omitidas estas situações pois grande parte das aplicações não estão preparadas para o tamanho dos ecrãs destes dispositivos. Por outro lado, muitas também precisam de *plugins* para o *browser*, como o Flash e Java para funcionar, os quais não estão disponíveis ou não são comuns para estas plataformas.

3.3. Análise das ferramentas

Considerando as funcionalidades por ordem de importância, foi feita uma análise aprofundada a cada uma das ferramentas de forma a obter o máximo de informação possível acerca das suas características e funcionalidades.

3.3.1. Google Drawings

Google Drawings é uma ferramenta do Google que em junção com o editor de texto, o editor de folhas de cálculo e o editor de apresentações, formam o conjunto das ferramentas de colaboração *online* disponíveis no Google Drive que podem ser utilizadas através de *browsers* de internet.

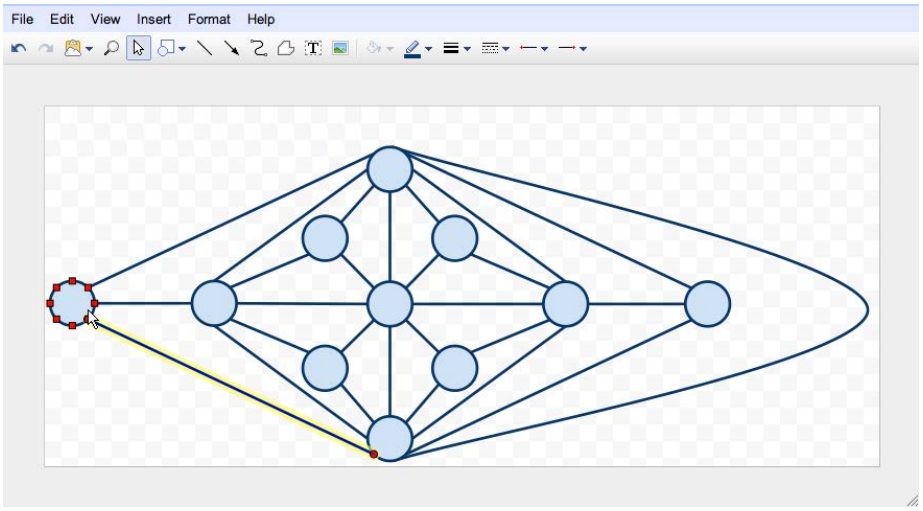


Figura 4 *Printscreen* da aplicação Google Drawings

Google Drawings disponibiliza todas as funcionalidades consideradas relevantes. Para além disso, tem algumas funcionalidades consideradas extra tais como o *undo/redo* de operações, e uma ferramenta de *chat* por texto integrado.

Em termos de características:

- Pontos positivos:
  - Não é necessário Java ou Flash;
  - É de utilização gratuita;
  - Permite até 50 pessoas a colaborar em simultâneo.
- Pontos negativos:
  - Apenas está disponível para *browsers* de internet;
  - É necessário ter uma conta Google para se poder editar o desenho.

3.3.2. Scribblar

Scribblar é uma ferramenta para *browsers* de internet dedicada à colaboração na elaboração de desenhos, diagramas e texto, o qual permite outras funcionalidades tais como a integração de fórmulas matemáticas com o Worfram Alpha, incluindo também um editor LaTeX.

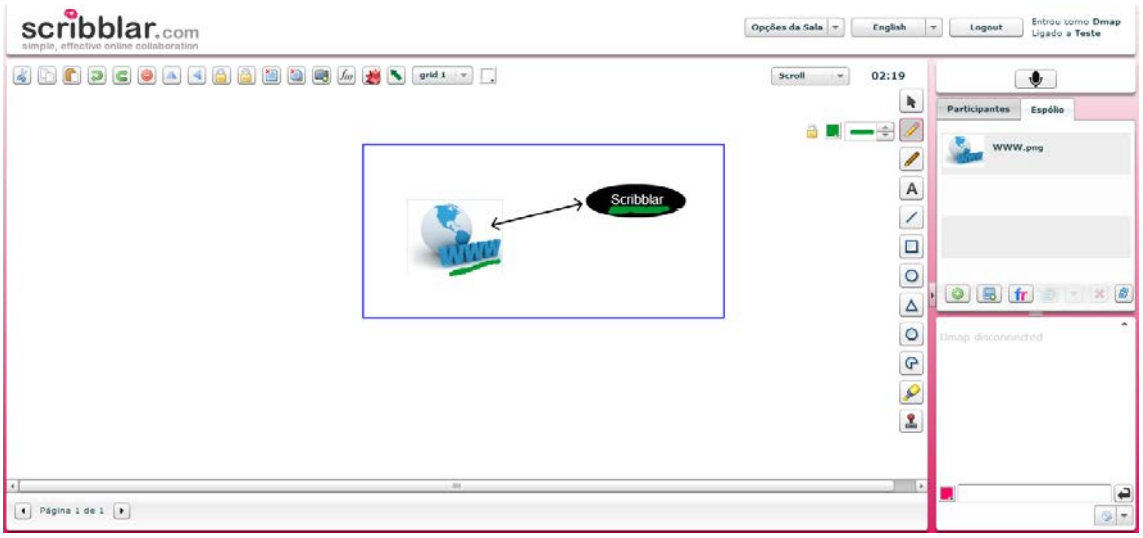


Figura 5 *Printscreen* da aplicação Scribblar

Scribblar disponibiliza todas as funcionalidades consideradas importantes e, para além disso, tem algumas funcionalidades consideradas extra como o *undo/redo* de operações, e uma ferramenta de *chat* integrado que funciona tanto para texto como para áudio.

Em termos de características:

- Pontos positivos:
  - Permite até 50 pessoas a colaborar em simultâneo.
- Pontos negativos:
  - É necessário Flash para executar a aplicação;
  - É grátis apenas para 2 utilizadores, no entanto permite até 50 na versão paga;
  - Apenas está disponível para *browsers* de internet;
  - É necessário ter uma conta para se poder editar o desenho.

3.3.3. CoSketch.com

CoSketch.com também é uma ferramenta apenas para *browsers* de internet dedicada à colaboração na elaboração de desenhos e que facilita a criação de *mockups* de interfaces de aplicações.

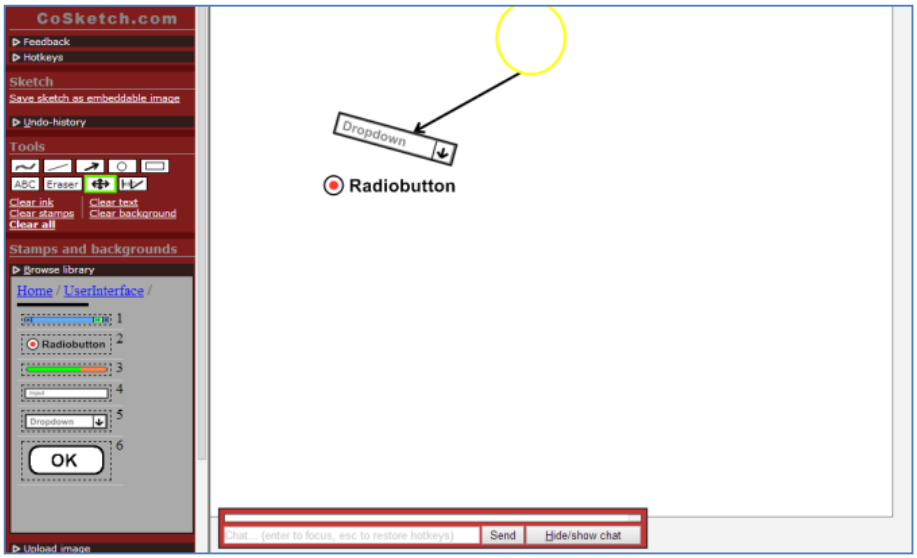


Figura 6 *Printscreen* da aplicação CoSketch.com

CoSketch.com disponibiliza grande parte das funcionalidades consideradas importantes. No entanto, não permite salvar o estado do desenho para se continuar mais tarde.

Em termos de características:

- Pontos positivos:
  - Não é necessário qualquer extra para executar a aplicação já que esta funciona apenas com JavaScript;
  - Não requer uma conta para poder criar ou participar num desenho colaborativo;
  - É de utilização gratuita.
- Pontos negativos:
  - Apenas está disponível para *browsers* de internet.

3.3.4. FlockDraw

FlockDraw é outra ferramenta de desenho colaborativo que pode ser usada através de um site na internet ou, alternativamente, uma aplicação para iPhone e iPad. Esta ferramenta é significativamente mais simples do que a grande maioria das ferramentas analisadas pois foca-se apenas nas funcionalidades mais básicas para oferecer a experiencia pretendida, ou seja, facilitar a colaboração em desenhos à distância.

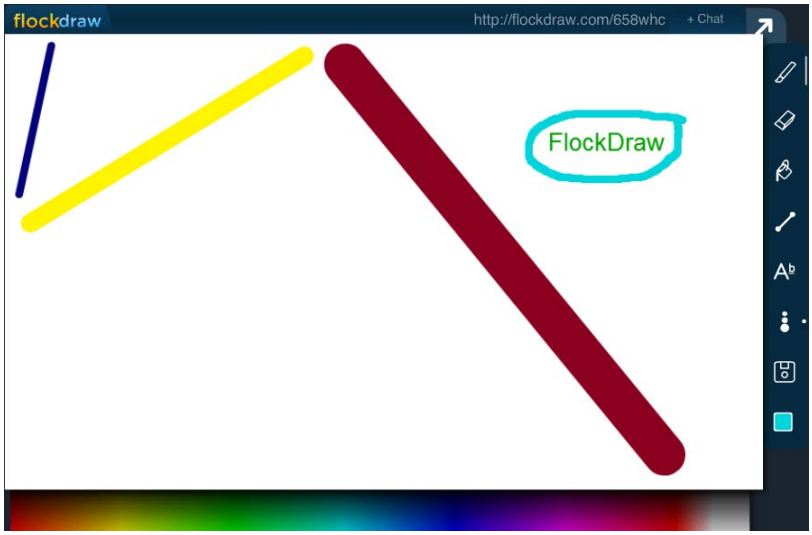


Figura 7 Printscreen da aplicação FlockDraw

Visto que a aplicação se foca apenas na área de desenho, esta não tem algumas das funcionalidades consideradas importantes tais como, a inserção de formas para facilitar a criação de diagramas, não se poder salvar o estado do desenho para se poder continuar mais tarde, e não se poder importar imagens para o desenho. Um extra que esta aplicação oferece é a possibilidade de se comunicar com os participantes do desenho através de um *chat*.

Em termos de características:

- Pontos positivos:
  - Não requer uma conta para se poder criar ou participar num desenho colaborativo;
  - A ferramenta permite até 10 pessoas a colaborar em simultâneo no mesmo desenho;
  - Para além de estar disponível para *browsers* de internet, a aplicação também está disponível para dispositivos iPhone e iPad;
  - É de utilização gratuita.
- Pontos negativos:
  - É necessário Flash para executar a aplicação nos *browsers* de internet.

3.3.5. GroupBoard

GroupBoard é uma ferramenta para desenho colaborativo que está disponível tanto para *browsers* de *internet* como para dispositivos Android, iPhone e iPad. Esta disponibiliza uma versão *embed* que permite colocá-la em qualquer website facilmente e, assim, juntar-se com outras ferramentas como Assembla[15], ou JIRA[16], para suportar o desenvolvimento de um projeto.

Esta ferramenta tem um plano de utilização gratuito que permite até 5 pessoas em simultâneo a utilizá-la, e tem um segundo plano que é pago, mas que suporta até 25 pessoas em simultâneo.

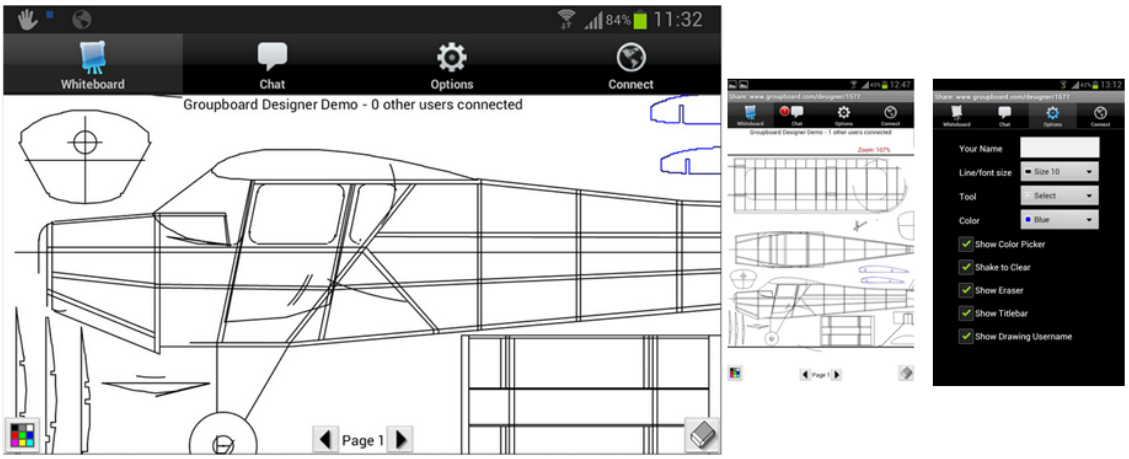


Figura 8 *Printscreens* da aplicação Groupboard (Tablet à esquerda e smartphone à direita)

GroupBoard disponibiliza todas as funcionalidades consideradas importantes, e ainda permite fazer *undo*/*redo* do estado do desenho, salvar a imagem final, e contem um *chat* de texto para os participantes da sessão de desenho colaborativo poderem comunicar.

Em termos de características:

- Pontos positivos:
  - Não é necessário qualquer extra para executar a aplicação já que esta funciona apenas com JavaScript;
  - Não requer uma conta para poder criar ou participar num desenho colaborativo;
  - É grátis até 5 utilizadores;
  - Permite até 25 utilizadores embora seja com a versão paga;
  - Está disponível para *browsers* de *internet*, dispositivos Android, iPhone e iPad.

3.3.6. Mighty Meeting

Mighty Meeting é outra ferramenta que permite desenho colaborativo. No entanto, foca-se especialmente em partilhar apresentações PowerPoint e fazer desenhos e anotações sobre estas. Como ferramenta de desenho colaborativo, esta é mais simples do que a grande maioria pois tem um objetivo um pouco diferente e, portanto, não tem algumas das ferramentas mais necessárias para a criação de diagramas ou desenhos mais elaborados.

De referir que esta ferramenta é utilizada por muitas empresas multinacionais, como por exemplo Verizon, Disney, Volvo, entre outras. O que prova que é considerada útil apesar da sua simplicidade.

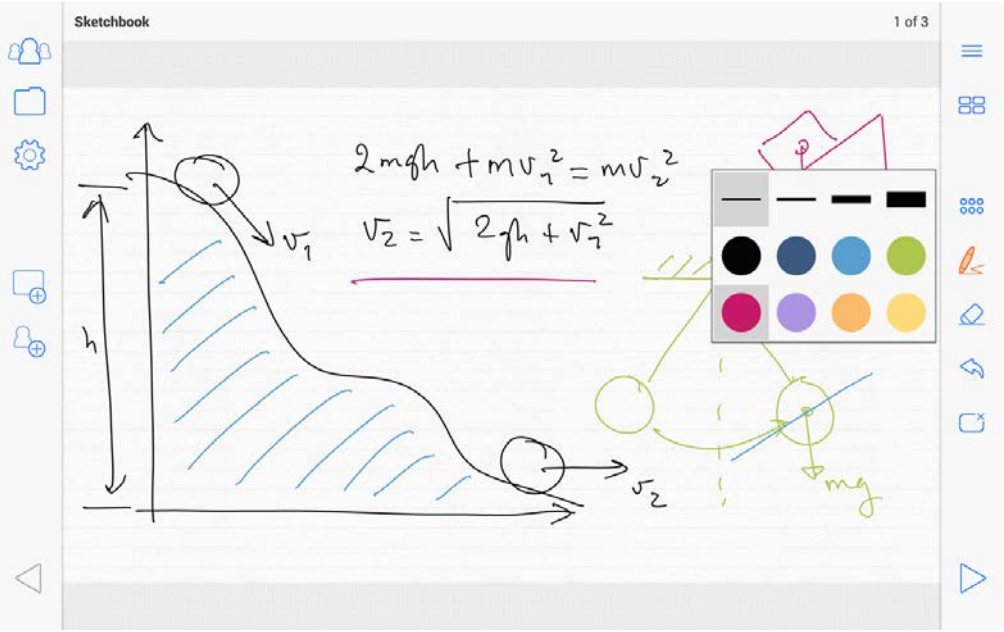


Figura 9 *Printscreen* da aplicação Mighty Meeting

Das funcionalidades consideradas importantes, esta aplicação não disponibiliza as de criação de formas, inserção de texto e importação de imagens. No entanto, permite importar ficheiros PDF e PowerPoint.

Em termos de características:

- Pontos positivos:
  - Na versão de browser não é necessário qualquer extra para executar a aplicação;
  - Está disponívem para *browsers* de *internet*, dispositivos Android, iPhone e iPad.
- Pontos negativos:
  - É necessário ter uma conta para se poder editar o desenho;
  - É grátis apenas para 2 utilizadores em simultâneo.

3.3.7. Chalkboard

Chalkboard é uma ferramenta de desenho colaborativo muito simples que está disponível apenas para Android. Esta aplicação está limitada a funcionar entre dispositivos ligados à mesma rede *WI-FI*.

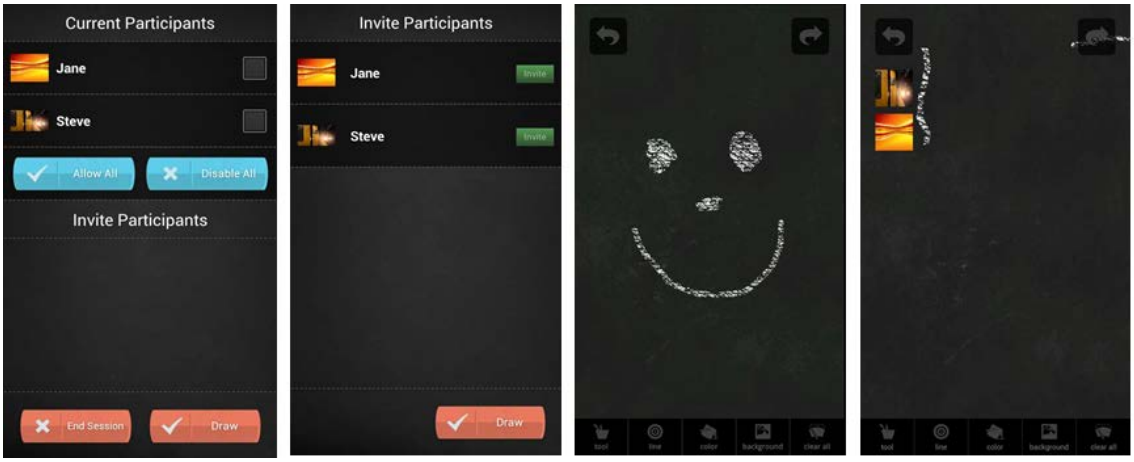


Figura 10 *Printscreens* da aplicação Chalkboard

Chalkboard permite desenho livre, no entanto, não oferece algumas das funcionalidades consideradas importantes tais como a inserção de formas ou texto e salvar o estado do desenho para continuar mais tarde. Em relação a funcionalidades extra, esta aplicação permitir fazer o *undo*/*redo* das operações do utilizador.

Em termos de características:

- Pontos positivos:
  - Não é necessário qualquer tipo de conta para se utilizar esta aplicação;
  - É uma aplicação grátis.
- Pontos negativos:
  - Apenas permite estarem ligados dois dispositivos em simultâneo;
  - Apenas está disponível para dispositivos Android.

3.3.8. Whiteboard: Collaborative Draw

Esta aplicação de desenho colaborativo em tempo real está disponível para Android, iPhone e iPad, tendo já mais de 7 milhões de *downloads* na App Store[17], o que a torna um caso de sucesso dentro das aplicações deste tipo.

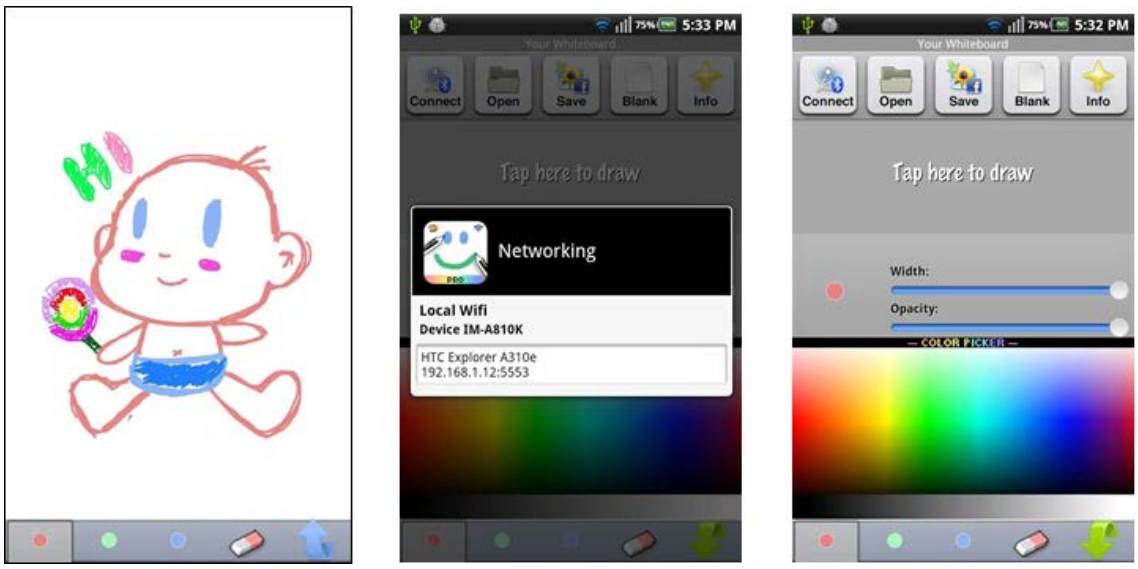


Figura 11 *Printscreens* da aplicação Whiteboard: Collaborative Draw

Esta aplicação está feita especialmente para a criação de desenhos, e não para criação de diagramas ou outro tipo de esquemas. Consequentemente, não permite a inserção de formas ou texto. Das funcionalidades consideradas importantes também não permite salvar o estado do desenho para se continuar mais tarde.

Em termos de características:

- Pontos positivos:
  - Não é necessário qualquer tipo de conta para se utilizar esta aplicação;
  - É uma aplicação grátis;
  - Está disponível para dispositivos Android, iPhone e iPad.
- Pontos negativos:
  - Apenas permite estarem ligados dois dispositivos em simultâneo.

3.3.9. SyncSpace Shared Whiteboard

A aplicação SyncSpace Shared Whiteboard está disponível para *browsers* de internet, e dispositivos Android, iPhone e iPad. No entanto, a versão para *browsers* não permite a edição do desenho mas, apenas, a sua visualização.

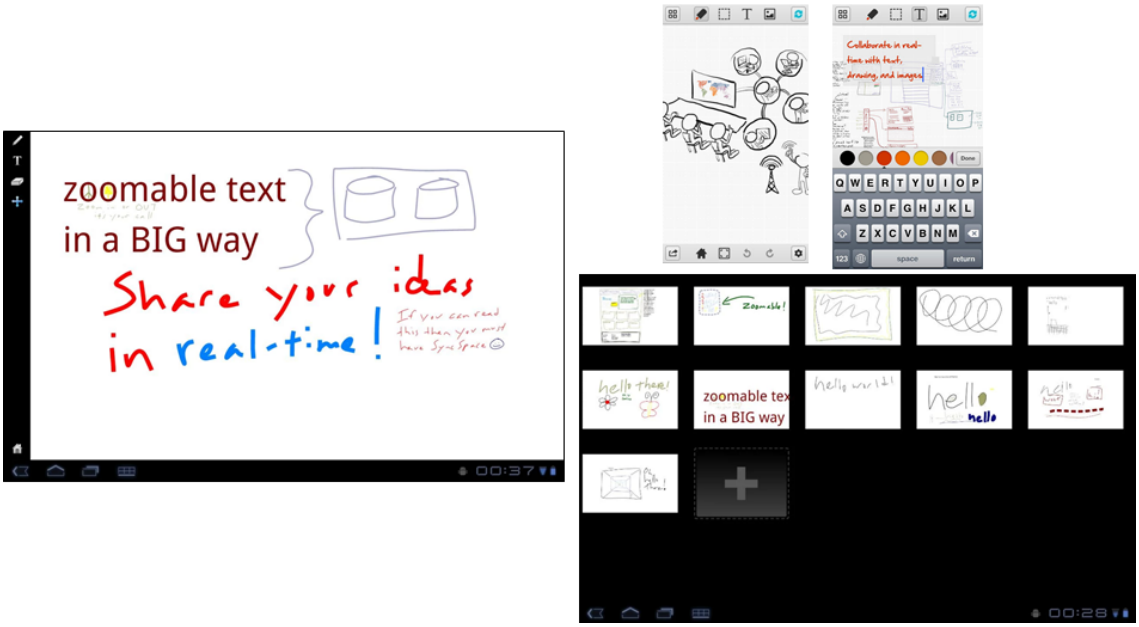


Figura 12 *Printscreens* da aplicação SyncSpace Shared Whiteboard (Tablet e iPhone)

Esta aplicação tem algumas das funcionalidades consideradas importantes. No entanto, não permite a inserção de formas nem imagens.

Em termos de características:

- Pontos positivos:
  - Não requer uma conta para poder criar ou participar num desenho colaborativo;
  - Está disponível para dispositivos Android, iPhone e iPad, e é ainda possível de visualizar num *browser*.
- Pontos negativos:
  - A versão *browser* apenas permite a visualização do desenho.

3.3.10. Shared Board

Shared Board é a unica aplicação analisada que está disponível para PC e MAC. No entanto, também está disponível para dispositivos iPad e Android.

Figura 13 *Printscreen* da aplicação Shared Board

Esta aplicação tem grande parte das funcionalidades consideradas importantes. No entanto, não suporta a inserção de formas, nem permite salvar o estado do desenho para se continuar mais tarde.

Em termos de características:

- Pontos positivos:
  - Não requer uma conta para poder criar ou participar num desenho colaborativo;
  - A ferramenta está disponível para múltiplas plataformas, Android, iPad, PC e MAC.
- Pontos negativos:
  - A versão de PC e MAC necessita da aplicação Adobe Air instalada para funcionar.

3.3.11. Lucidchart

Lucidchart é uma ferramenta de colaboração em tempo real dedicada à criação de diagramas, permitindo a criação de vários tipos dos mesmos tais como, diagramas UML, ER, *flowchart*, *mind-maping*, e até *mockups*. Esta ferramenta apenas está disponível para *browsers* de *internet*.

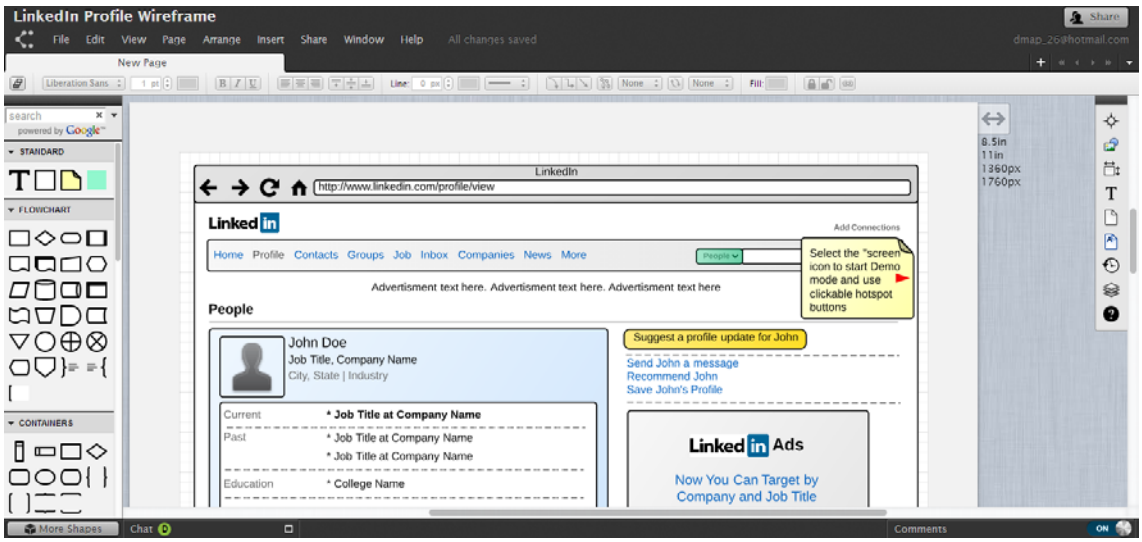


Figura 14 *Printscreen* da aplicação Lucidchart

Trata-se de uma aplicação muito completa em termos de funcionalidades. No entanto, não é especificamente uma ferramenta de desenho colaborativo e, portanto, não disponibiliza uma das funcionalidades mais importantes que é a possibilidade de fazer desenho livre.

Em termos de características:

- Pontos positivos:
  - Não é necessário qualquer extra para executar a aplicação já que esta funciona apenas com JavaScript;
  - É uma ferramenta totalmente gratuita enquanto estiver em versão Beta.
- Pontos negativos:
  - Apenas está disponível para *browsers* de *internet*;
  - É necessário uma conta para participar na elaboração de qualquer diagrama.

3.3.12. Concept Board

Concept Board é uma aplicação de desenho colaborativo apenas para *browsers* de *internet* cujo objetivo é ser utilizada num ambiente empresarial, suportando um conjunto de funcionalidades que dão ênfase à interação entre pessoas e à troca de ideias.

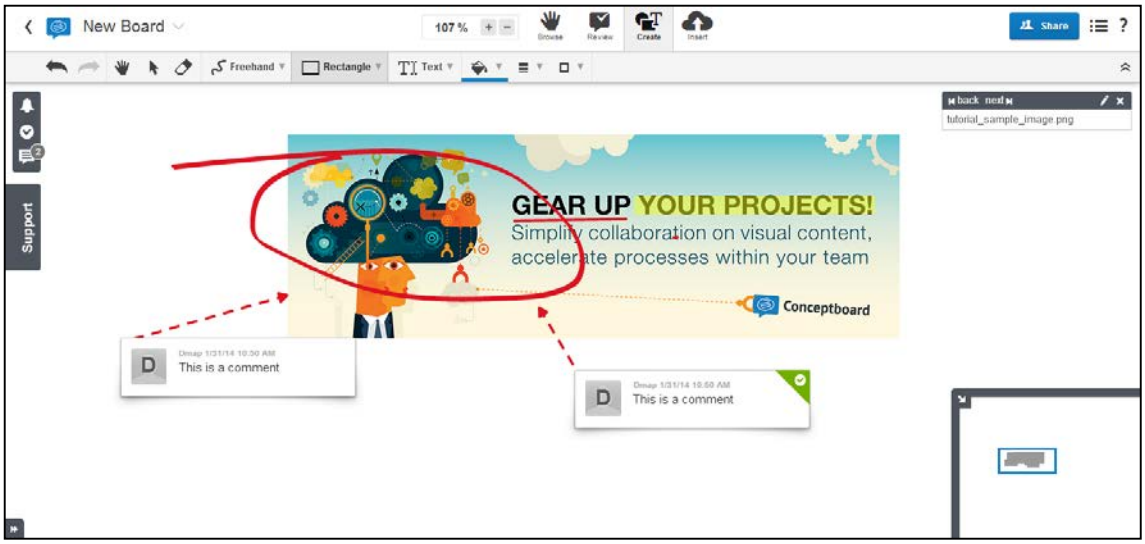


Figura 15 *Printscreen* da aplicação Concept Board

Esta aplicação tem todas as funcionalidades consideradas importantes, bem como algumas extra, tais como, possibilitar o *undo*/*redo* das ações dos utilizadores e permitir anotações e comentários que podem ser colocadas em qualquer parte do desenho.

Em termos de características:

- Pontos positivos:
  - Não é necessário qualquer extra para executar a aplicação já que esta funciona apenas com JavaScript;
  - Grátis até 5 utilizadores por sessão.
- Pontos negativos:
  - É necessário uma conta para participar na elaboração de qualquer desenho;
  - Apenas está disponível para *browsers* de *internet*.

3.4. Conclusões

Depois desta análise detalhada às ferramentas existentes, concluiu-se que, embora existe uma grande variedade de opções com um fator comum que é a partilha de um desenho em tempo real entre vários participantes, é possível identificar múltiplas diferenças no que toca ao público-alvo e ao propósito da ferramenta.

De facto, foram identificados os seguintes tipos:

- Ferramentas com múltiplo propósito, de forma a alcançar um público-alvo vasto (Google Drawings, Scribblar, CoSketch.com, GroupBoard, SyncSpace Shared Whiteboard, Concept Board);

- Ferramentas direcionadas apenas para desenho artístico (FlockDraw, Chalkboard, Whiteboard: Collaborative Draw, Shared Board);
- Ferramentas mais focadas na elaboração de diagramas (Lucidchart);
- Ferramentas para partilha e elaboração de documentos PowerPoint (Mighty Meeting).

Através da observação do gráfico seguinte, é possível ter uma noção das funcionalidades mais comuns de entre aquelas que são disponibilizadas pelas ferramentas em análise.

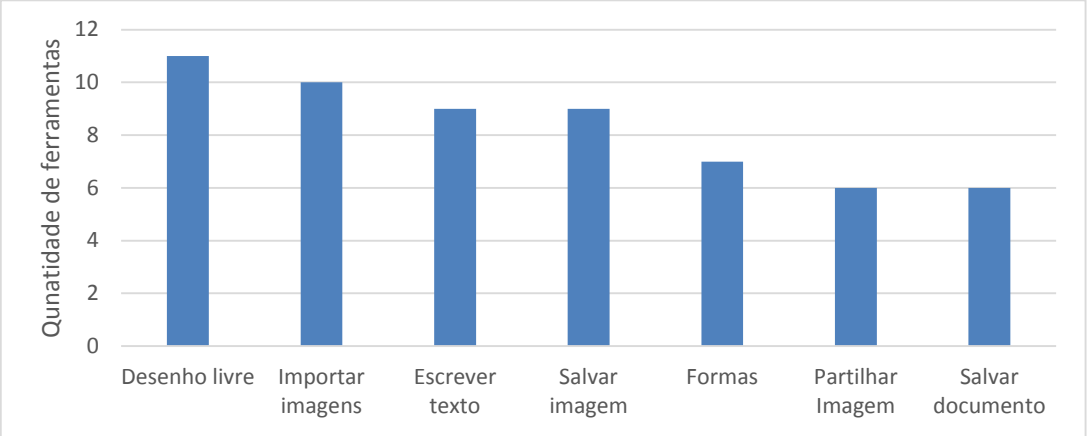


Gráfico 1 Funcionalidades mais comuns

Na verdade, é possível perceber que algumas das funcionalidades mais comuns estão em discordância com as anteriormente selecionadas como sendo importantes pois, no âmbito para o qual a nova aplicação será desenvolvida, ou seja, para o contexto empresarial, existem funcionalidades que não são tão relevantes, tais como salvar a imagem ou partilhá-la.

O contrário também acontece com o salvar o estado do desenho colaborativo para se continuar mais tarde. Isto porque, no contexto empresarial, o objetivo não é partilhar a informação numa rede social mas sim discutir ideias. Também é importante poder salvar-se o estado do desenho para o caso de surgirem imprevistos e de ter que se continuar a discussão mais tarde, o que até é benéfico em termos de inovação pois a pessoa pode ter novos *insights* e ficar inspirada enquanto estiver a realizar outras tarefas.

Uma das características mais importantes neste tipo de aplicação é a facilidade que os utilizadores têm em partilhar informação. Nesse sentido, foram analisados vários fatores tais como as plataformas que têm mais ou menos acesso a este tipo de ferramentas. Outro fator também importante é a facilidade que os utilizadores têm em partilhar uma sessão de desenho colaborativo.

No que toca à quantidade de plataformas para as quais estas aplicações estão disponíveis, esta informação está bastante clara no Gráfico 2.

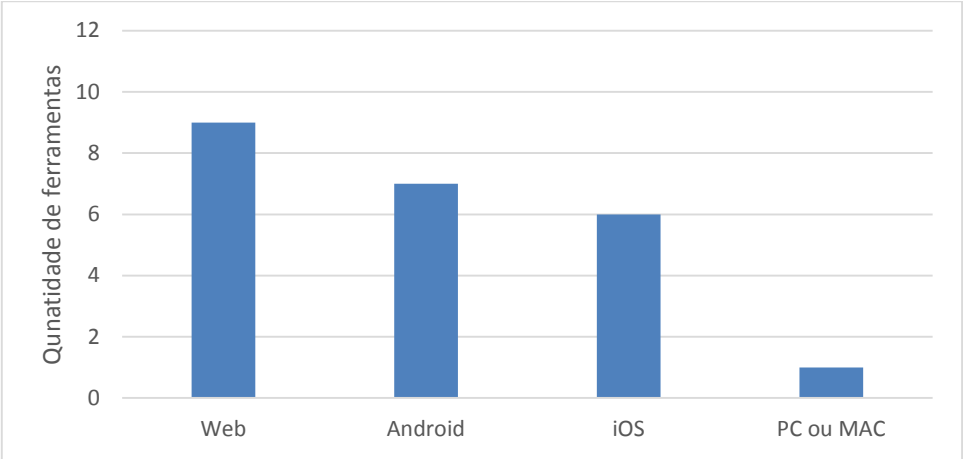


Gráfico 2 Número de aplicações disponíveis por plataforma

De facto, é possível identificar que a grande maioria das aplicações está primeiramente mais vocacionada para *browsers* de internet e só depois para dispositivos móveis. No entanto, existem muitas aplicações que apenas estão disponíveis para uma destas plataformas o que aumenta o risco de não ser possível partilhar a informação. No caso das ferramentas analisadas, 50% apenas estão disponíveis para uma plataforma, e 17% apenas estão disponíveis para duas plataformas, o que resulta em 33% das ferramentas suportarem mais de duas destas plataformas.

No que toca à facilidade que um utilizador tem em partilhar ou participar numa sessão de desenho colaborativo, foi considerada importante a necessidade de criar ou não uma conta para se poder utilizar a ferramenta, bem como a quantidade de passos que o utilizador tem de dar até poder utilizar a mesma. Estes passos podem ser a criação de uma conta, efetuar autenticação, e instalação e criação de documento de desenho. Isto foi feito com o objetivo de ter uma métrica de forma a poder ter-se uma melhor ideia da dificuldade que é começar a partilhar um desenho.

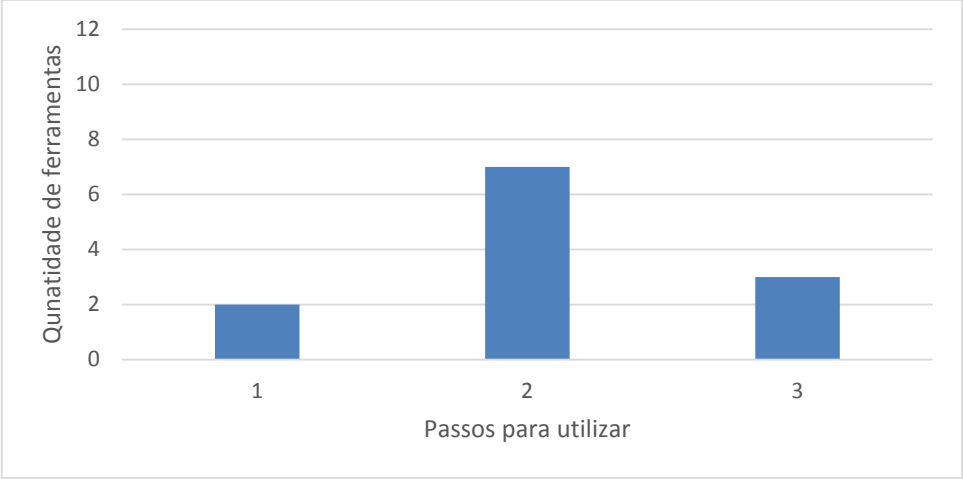


Gráfico 3 Passos necessários para começar a utilizar as aplicações

Pelo gráfico anterior, é possível verificar que grande parte das aplicações tem um número de passos a realizar relativamente baixo. No entanto, quanto maior for o número de passos, mais difícil é a utilização, e mais prejudicial se torna pois as pessoas facilmente podem desistir ou até se esquecer de parte do que queriam partilhar e discutir.

O objetivo da aplicação a desenvolver neste estágio é que esta seja integrada com o RCS Suite, e, portanto, esteja lá pronta a ser utilizada. A *quote* “It’s just there, it just works.” é a *quote* do RCS.

4. Desenvolvimento da aplicação

Nesta secção é feita uma descrição pormenorizada de todos os passos tomados e escolhas efetuadas durante o decorrer do estágio. Esta secção contém também outra informação, como a descrição das ferramentas e tecnologias que foram utilizadas.

4.1. Ferramentas e tecnologias utilizadas durante o desenvolvimento

4.1.1. Ferramentas

Redmine[18] – *Software open-source* para gerir projetos que disponibiliza ferramentas para gerir os *bugs* e as tarefas a serem desenvolvidas durante um projeto. Também possui um calendário e gráficos de Gantt para ajudar no decorrer do projeto. Possibilita também a integração com vários repositórios de controlo de versões de ficheiros, tais como, Svn, Git, Mercurial, entre outros.

Subversion[19] – Sistema de controlo de versões de ficheiros que permite guardar todas as versões submetidas, comparar as versões de ficheiros, fazer *merge*, reversão de ficheiros, e muitas outras funcionalidades que facilitam muito o trabalho em equipa no desenvolvimento de projetos.

Jenkins[20] – *Software open-source* que disponibiliza serviços de integração contínua para o desenvolvimento de software. Este permite automatizar, gerir e catalogar *builds* do software. Esta ferramenta pode obter o código fonte de uma aplicação através de um sistema de controlo de versões e juntamente com as configurações definidas constrói uma *build* da aplicação pronta a ser testada e entregue. As *builds* do Jenkins podem ser iniciadas de várias formas: podem ser iniciadas manualmente, programadas para serem executadas a uma determinada altura, ou até quando é feito um *commit* para o sistema de controlo de versões. Esta ferramenta foi utilizada quando a aplicação foi integrada com o projeto RCS Suite, para criar as *releases*.

Ant[21] – Ferramenta de automação de compilação e construção de *software*. A sua utilização foi necessária para fazer o *setup* do projeto base (RCS Suite) no qual foi integrada a ferramenta de desenho colaborativo.

Java[22] – Linguagem de programação utilizada para o desenvolvimento da aplicação Android.

Eclipse[23] – IDE escolhido para o desenvolvimento da aplicação, o qual permite a integração com o SDK do Android necessário para o desenvolvimento de aplicações para o sistema operativo Android.

Android SDK[24] – Plataforma e API base que permite o desenvolvimento de aplicações para o sistema operativo Android.

Modelio[25] –Ferramenta de modelação UML que foi utilizada para fazer os primeiros diagramas, tendo sido posteriormente substituída pelo Visio devido à sua complexidade desnecessária e falta de funcionalidades importantes.

Visio[26] – Ferramenta para criação de vários tipos de diagramas, foi utilizada para criar muitos dos diagramas UML desenvolvidos ao longo do estágio.

JUnit[27] – Ferramenta de testes unitários para a linguagem de programação Java. Permite a criação de testes para módulos de funcionalidades de uma aplicação, e assim testar o comportamento dos vários módulos em separado.

Mockito[28] – F*ramework open-source* para Java que permite a criação de objetos falsos (*mocked*) para facilitar os testes unitários. Com esta *framework* é possível fazer uma separação de módulos ainda maior, o que aumenta muito a eficácia dos testes unitários, pois permite serem testadas muitas outras funcionalidades e módulos que seriam extremamente complicados de testar de outra maneira. Esta ferramenta é especialmente útil para testar código que foi desenhado tendo em conta as técnicas de BDD (Behaviour Driven Development) ou TDD (Test Driven Development).

Android JUnit Framework[29]– *Framework* que vem em conjunto com o SDK do Android e que permite a criação dos mais variados testes: testes unitários, testes de integração, testes de *user-interface* e também testes *end-to-end*. Esta permite simular a interação do utilizador com a aplicação, e assim automatizar tarefas comuns para validar se a aplicação está a funcionar devidamente. Com esta ferramenta pode-se evitar ter alguém constantemente a testar as funcionalidades já implementadas sempre que é feita uma modificação ou desenvolvida uma nova funcionalidade.

Robotium[30] –É uma *framework* de automatização de testes para Android que permite a criação de testes de caixa-preta para o interface do utilizador de aplicações Android. A *framework* funciona em junção com a *framework* JUnit do SDK do Android para simplificar a criação destes testes. Esta ferramenta foi utilizada devido a facilitar bastante a criação de testes

de UI em relação a utilizar apenas a *framework* disponibilizada pelo SDK do Android, e assim acelerar o desenvolvimento.

Roboletric[31] – Ferramenta que permite a execução de testes unitários para aplicações Android de forma significativamente mais rápida pelo facto de serem executados diretamente na JVM (Java Virtual Machine), ou seja, no computador/MAC. Esta Aplicação foi utilizada porque, fazer o *building*, *deploying* e execução de uma aplicação Android, quer seja num emulador, quer num dispositivo real, é lento, e os testes unitários desenvolvidos para Android têm de ser executados na plataforma Android. Consequentemente é bastante demorada a execução dos testes, o que se torna muitas das vezes num entrave para a criação de testes para a aplicação. A ferramenta permite assim a criação de testes mais rápidos graças a simular a API do Android diretamente na JVM.

Kryonet[32] – Biblioteca *open-source* para aplicações Java que disponibiliza uma API simples de utilizar para comunicações TCP e UDP eficientes, utilizando as bibliotecas NIO (Non-blocking I/O) do Java para comunicações via internet entre dispositivos.

TimeTracker – Ferramenta de registo de tempo e tarefas, sendo a utilizada para controlo de tempo dos funcionários na WIT.

Red Notebook[33] – Aplicação para Windows que tem um calendário e permite fazer anotações e apontamentos associados a cada dia. Esta ferramenta foi escolhida para ser utilizada como diário, de forma a apontar as tarefas que foram executadas ao longo do estágio.

ToDoList[34] – Esta ferramenta permite fazer a gestão de tarefas. Permite a inserção de uma data prevista de finalização, colocar o tempo previsto de execução, e fazer *tracking* do progresso das várias tarefas. Embora esta ferramenta permita fazer toda a gestão de tarefas, esta apenas foi utilizada para lembretes e anotação de tarefas de forma mais informal. Para gerir as tarefas relacionadas com a aplicação foi utilizado o Redmine.

TeamGantt[35] – Ferramenta de criação de diagramas de Gantt, utilizado para criar diagramas de Gantt dos vários *sprints* no decorrer do estágio.

4.1.2. Protocolos

SIP (Session Initiation Protocol)[36] - Protocolo de iniciação de sessão que utiliza o modelo "requisição-resposta" para iniciar sessões de comunicação interativa entre dispositivos. Este

protocolo é um padrão da IETF (Internet Engineering Task Force), sendo este o padrão RFC2543[36].

MSRP (Message Session Relay Protocol)[37] – Protocolo para transmissão de uma sequência de mensagens instantâneas no contexto de uma sessão de comunicação. Este protocolo é utilizado diretamente por cima do protocolo SIP e é usado no contexto RCS especialmente para mensagens instantâneas e transferência de ficheiros. O protocolo MSRP está definido no padrão RFC4975.

JSON (JavaScript Object Notation**)**[38] – Um formato *standard* similar ao XML que utiliza texto para armazenar ou transmitir informação. Foi utilizado como formato para transferir informação entre os dispositivos dentro da mesma sessão Whiteboard.

4.2. Requisitos do Software

Como neste projeto é seguida uma metodologia ágil, os requisitos foram inferidos através das *user-stories* especificadas pelos clientes, neste caso, os *product owners*. Após a apresentação do estado da arte elaborada durante a fase inicial do estágio, foi feita uma reunião que marcou o início real do desenvolvimento da aplicação, na qual foram definidas as *user stories* iniciais. Posteriormente em outras reuniões foram criadas novas *user stories* e outras foram removidas.

Para representação de cada *user story* foi utilizado o *template* "**No papel de** "<papel>", **quero** "<ação com o sistema>" **para** "<beneficio externo>"". Este *template* facilita a interpretação das *user stories*, uma vez que estas ficam padronizadas.

4.2.1. *User stories* iniciais

Apresenta-se, em seguida, a lista das *user stories* inicialmente definidas.

Nº	**No papel de…**	**quero..**	**para..**
1	utilizador	poder inserir imagens dos meus ficheiros	ter a possibilidade de mostrar algo mais elaborado que um desenho.
2	utilizador	poder tirar uma fotografia e inserir a mesma	ter a possibilidade de mostrar algo mais elaborado que um desenho.
3	utilizador	poder desenhar livremente linhas com o dedo	ser mais fácil alterar o desenho.
4	utilizador	poder apagar o desenho conforme movo o dedo	ser mais fácil alterar o desenho.
5	utilizador	poder convidar um ou mais contactos RCS para participar comigo no desenho	poder partilhar e discutir as ideias com estas pessoas.
6	utilizador	poder mover escalar e rodar as imagens inseridas	poder salientar e ajustar a informação que eu quero.

7	utilizador	poder inserir as seguintes formas retângulos, círculos, triângulos, setas e estrelas no desenho	poder criar uma representação mais fiel das minhas ideias e para salientar e explicar informação.
8	utilizador	poder mover escalar e rodar as formas inseridas, e escolher a sua cor	poder salientar e ajustar a informação que eu quero.
9	utilizador	poder inserir texto no desenho	mais facilmente poder partilhar/discutir alguns tipos de informação.
10	utilizador	poder mover escalar e rodar o texto, e escolher a sua cor	poder salientar e ajustar a informação que eu quero.
11	utilizador	escolher uma cor, e os componentes inseridos posteriormente ficarem com essa cor	ser mais fácil trabalhar com a ferramenta e sem ter de estar sempre a escolher uma cor.
12	utilizador	poder ver tudo o que os outros utilizadores inserirem no mesmo desenho que eu	ter uma representação fiel do desenho que os outros utilizadores também estão a ver.
13	utilizador	que enquanto uma imagem está a ser transferida para mim seja visível um *placeholder* da imagem, este *placeholder* deve mover-se, escalar-se e rodar como se se tratasse da imagem real, e quando a transferência acabar tem que ser trocado pela imagem real	eu ter a noção que a imagem está a ser transferida, mesmo não a conseguindo ver ainda.
14	utilizador	que as ações dos outros utilizadores sejam vistas em	criar uma experiência mais agradável.

		tempo real e que sejam fluidas no meu dispositivo	
15	utilizador	Ter a possibilidade de mudar a cor do fundo de ecrã	criar uma experiência mais agradável e poder fazer sobressair alguma informação.
16	utilizador	poder ter várias sessões de desenho com a mesma pessoa e com pessoas e grupos diferentes, cada uma com o seu estado e desenho independente de todas as outras	poder ter vários tópicos e poder criar novos desenhos sem ter de apagar informação que pode ser importante.
17	utilizador	ser notificado quando me convidam para uma nova sessão de desenho	não perder informação atualizada que pode ser importante.
18	utilizador	que a informação do desenho seja recebida mesmo quando não estou no desenho	poder fazer outras tarefas em paralelo.
19	utilizador	ser notificado quando existe uma modificação num dos desenhos e eu não estou com essa sessão aberta	estar a par das modificações nos desenhos.
20	utilizador	que a sessão de desenho seja representado como um objeto no chat da aplicação RCS que pode ser aberta clicando neste	ser mais fácil aceder ao desenho pretendido, que pode ser identificado facilmente através do contexto das mensagens de chat.
21	utilizador	que o objeto com a sessão de chat mude de cor quando o desenho for modificado, e ficar nesta cor até	no meio de várias sessões saber quais foram as modificadas.

		esta sessão ser aberta e o desenho modificado ser visto	
22	utilizador	poder escolher um nome para a sessão de chat, e que esta fique visível no objeto correspondente a este, tanto no meu dispositivo como nos dispositivos dos outros participantes	facilitar a identificação de cada desenho.
23	utilizador	poder selecionar e editar qualquer componente que já exista no desenho	poder modificar o desenho mais facilmente.
24	utilizador	poder modificar texto que já tenha sido inserido	não ter de estar a apagar e voltar a escrever outra vez no caso de me enganar.
25	utilizador	poder colocar no topo de todos os componentes qualquer componente que já exista no desenho	ser mais fácil sobressair o que é mais importante em cada momento e poder reorganizar a informação.
26	utilizador	poder escolher uma imagem dos meus ficheiros e defini-la como fundo de ecrã no desenho	poder desenhar por cima de uma imagem e assim facilitar na criação de esquemas.
27	utilizador	poder tirar uma fotografia e defini-la como fundo de ecrã no desenho	poder desenhar por cima de uma imagem e assim facilitar na criação de esquemas.
28	utilizador	que quando uma imagem de *background* foi inserida por outro utilizador, e está a ser transferida, como não está visível quero ser notificado deste evento	eu ter a noção que a imagem está a ser transferida, mesmo não a conseguindo ver ainda.

4.2.2. Modificação das *user stories* ao longo do estágio

No decorrer do estágio foram feitas várias reuniões no sentido de avaliar e ajustar as funcionalidades mais importantes, resultando na criação de novas *user stories* e na remoção de algumas já implementadas.

Foram removidas as *user stories* relacionadas com a funcionalidade de inserir imagens e poder mover, escalar e rodar as imagens, ou seja, as *user stories* 1, 2, 6 e 13.

Para substituir esta funcionalidade das imagens, foram adicionadas as *user stories* 26, 27 e 28.

4.2.3. Requisitos não-funcionais

Durante a primeira reunião foram igualmente especificados e inferidos os requisitos não funcionais da aplicação, sendo estes os seguintes:

1. É importante reduzir o melhor possível a quantidade de informação transferida entre os dispositivos desde que isto não prejudique a experiência do utilizador.
2. É importante a aplicação ser modular, para que seja fácil adicionar e remover novas funcionalidades sem ter de se estar a modificar outras funcionalidades.
3. A aplicação deve ser estável e funcionar previsivelmente.
4. A aplicação deve suportar dispositivos com a versão do sistema operativo Android 2.3 Gingerbread e todas as versões mais recentes que esta.
5. A aplicação deve ter um interface com o utilizador bastante simples e que facilite, em primeiro lugar, a utilização das ferramentas mais importantes.

Posteriormente não houve qualquer modificação aos requisitos não-funcionais.

4.2.4. Atributos de Qualidade

Tendo em conta toda a informação recolhida, principalmente no desenvolvimento do estado da arte e na reunião de definição de requisitos, foram escolhidos alguns atributos de qualidade para a aplicação. Nomeadamente.

- ***Maintainability*** – Uma vez que a aplicação vai ser integrada mais tarde com o produto RCS Suite e um dos requisitos não-funcionais é em relação à modularidade, torna-se de

grande importância a capacidade do sistema sofrer mudanças sem ter de se estar a mudar uma grande parte da aplicação, e de forma relativamente fácil. É pois essencial a criação de uma arquitetura robusta para o código da aplicação.

- **Usabilidade** – Embora esta versão da aplicação não seja um produto final, pode um dia vir a ser. Consequentemente, ter uma aplicação agradável, intuitiva e fácil de utilizar é muito importante.
- **Performance** – A capacidade de resposta é algo muito relevante numa aplicação interativa pois, do ponto de vista do utilizador, algo que não é fluído é desagradável, mesmo que seja eficaz. Deste modo, foi também escolhido este atributo de qualidade que influenciou em vários aspetos o desenvolvimento da aplicação.

4.2.5. Priorização das tarefas

Na reunião inicial foi também definida com o Scrum Master e os *product owners*, a planificação inicial das tarefas tendo em conta a prioridade. Mais tarde foi feita uma estimativa do planeamento e implementação do esqueleto da arquitetura da aplicação, e de cada uma das *user stories*. Estas estimativas tiveram também em conta a planificação e execução dos testes das várias funcionalidades.

Ficou estimado que o planeamento da arquitetura da aplicação iria demorar aproximadamente 7 dias, e sua a implementação iria demorar 6 dias. Relativamente ao tempo estimado para cada user *user story* estimou-se o seguinte:

User Stories	**Prioridade (0-10)**	**Tempo estimado (em dias)**
US-1 Inserir imagens	6	3
US-2 Inserir fotografias	6	2
US-3 Desenhar com o dedo	9	3
US-4 Apagar desenho com o dedo	8	2
US-5 Convidar contactos RCS	3	8
US-6 Mover, escalar e rodar imagens	6	5

US-7 Inserir formas	8	3
US-8 Mover, escalar, rodar e escolher cor das formas	8	5
US-9 Inserir texto	7	2
US-10 Mover, escalar, rodar e escolher cor do texto	7	2
US-11 Definir cor para componentes	8	2
US-12 Representação fiel do desenho entre dispositivos	9	3
US-13 Placeholder durante transferência de imagem	6	3
US-14 Ações fluidas e em tempo real	5	3
US-15 Mudar o fundo de ecrã para uma cor	4	1
US-16 Múltiplas sessões de desenho	3	4
US-17 Notificação de convite para desenho	3	2
US-18 Sincronização do desenho mesmo não estando neste	3	4
US-19 Notificação de alteração de desenho	3	1
US-20 Sessão de desenho representada como objeto no *chat* RCS	3	3
US-21 Sobressair desenhos modificados desde última abertura, no *chat*	3	1
US-22 Nome para sessão de *chat*	3	1
US-23 Selecionar e editar componentes	6	4
US-24 Alterar texto já inserido no desenho	5	1
US-25 Colocar componente no topo de todos os outros	5	2

No total o estágio correspondeu a 102 dias úteis de trabalho, dos quais se estimou:

- 13 dias para desenvolvimento da arquitetura;
- 70 dias para implementação das *user stories*;
- 5 dias para a elaboração do relatório;

No total a estimativa resultou em 83 dias de trabalho e 19 dias de *buffer*, incluindo os 5 dias para elaboração do relatório de estágio, que serão utilizados nos dias de buffer que não forem necessários para elaboração de outras tarefas. Estes dias foram distribuídos pelos vários *sprints*. O *buffer* foi também utilizado para reuniões, planeamento e correção de erros na aplicação reportados pela equipa de testes da WIT Software.

Mais tarde foram introduzidas outras tarefas, sendo estas:

User Stories	**Prioridade (0-10)**	**Tempo estimado (em dias)**
US-26 Escolher imagem dos ficheiros e definir como fundo de ecrã	5	3
US-27 Tirar fotografia e definir como fundo de ecrã	5	2
US-28 Notificação de nova imagem de background a ser recebida	5	1

Isto concluiu em 89 dias para implementação da arquitetura e das *user stories*, e 13 dias de *buffer*.

4.3. Atividades no decorrer do estágio

4.3.1. *Sprint* 0

O *sprint* 0 consta de tudo o que foi feito durante o estágio antes de se começar a desenvolver a aplicação. O *sprint* 0 foi de apenas 1 mês, tendo inicio no dia 13 de Janeiro e acabou no dia 12 de Fevereiro.

No inicio do estágio começou-se por dar a conhecer ao estagiário o funcionamento dos protocolos SIP e MSRP, e a especificação Joyn Blackbird, utilizada no produto RCS Suite da WIT Software. De seguida foi feita a análise do Estado da Arte apresentada no capítulo 3 deste documento. E por fim foi feito um primeiro protótipo.

Durante este *sprint* inicial a equipa fez várias reuniões de modo a definir o plano e objetivos do estágio.

4.3.2. *Sprint* 1

Planificação das tarefas para o primeiro sprint

Para o primeiro *sprint* ficou definido fazer-se primeiramente a planificação e implementação da arquitetura base, e foram escolhidas *user stories* tendo em conta a prioridade para serem implementadas após a arquitetura base estar desenvolvida. Estas *user stories* foram analisadas para obter os requisitos e a sua estimativa. Sendo estas as seguintes:

US-12 Representação fiel do desenho entre dispositivos

- Implementação da fachada de comunicação – 8h;
- Implementação do sistema de comunicação entre dispositivos – 13h;
- Definição inicial do protocolo de comunicação – 3h.

US-3 Desenhar com o dedo

- Implementação do sistema de *input* do utilizador – 5h;
- Implementação da ação de desenhar – 20h.

Na figura seguinte, apresenta-se o diagrama de Gantt com as tarefas detalhadas distribuídas pelo *sprint*:

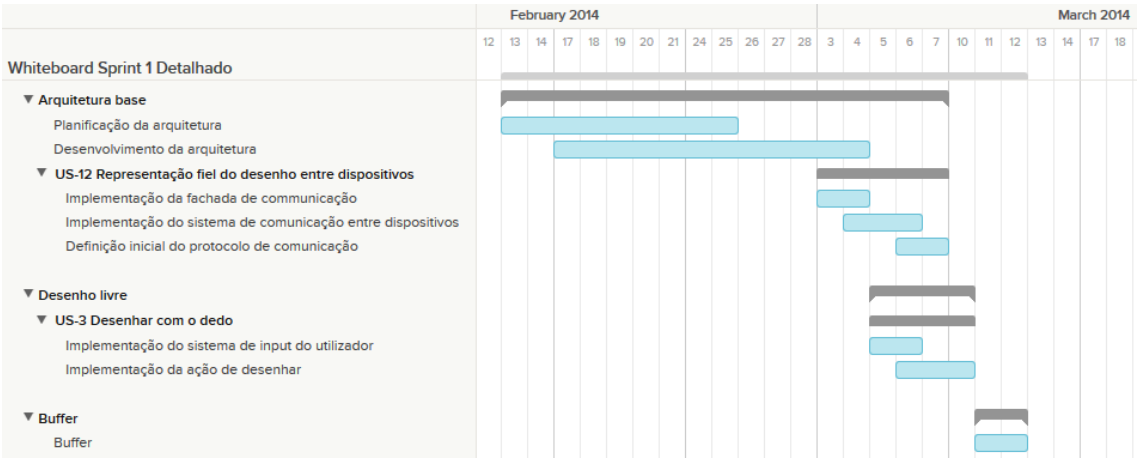


Figura 16 Diagrama de Gantt detalhado das tarefas para o primeiro *sprint*

Algumas das tarefas no diagrama de Gantt estão sobrepostas devido a ter sido considerada a necessidade de estas serem implementadas e testadas juntamente.

Finalização do primeiro sprint

O seguinte gráfico Burndown mostra o progresso que foi feito durante o primeiro *sprint* em relação ao progresso esperado tendo em conta as estimativas. A informação foi atualizada a cada 5 dias para mostrar o progresso. A penúltima amostra dá a conhecer o estado no fim estimado da sprint, e a última dá a conhecer o estado no final da sprint, ou seja, depois de acabar os dias de buffer.

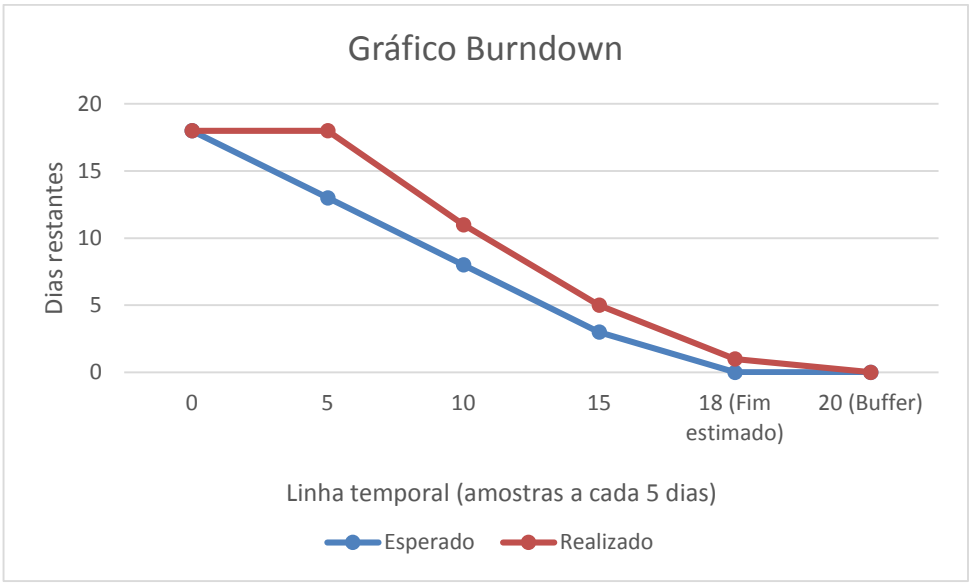


Gráfico 4 Gráfico Burndown do progresso das tarefas ao longo do primeiro *sprint*

Através do gráfico Burndown é possível ter a noção dos atrasos do projeto, conforme as tarefas são acabadas os dias correspondentes às tarefas são descontados e assim é possível ter a noção do progresso, assim quando o realizado está acima do esperado significa que o projeto está atrasado, e se o realizado estiver abaixo do esperado significa que está adiantado. No entanto uma situação particular que é necessária ter em atenção ao analisar estes gráficos é que os dias só são descontados quando cada *user story* é finalizada por completo, neste caso parece que não foi feito nada nos primeiros 5 dias apenas porque nenhuma tarefa foi finalizada por completo.

Neste *sprint* a tarefa "Implementação do sistema de *input* do utilizador" acabou por demorar consideravelmente mais tempo, devido a já ter sido preparada para as funcionalidades que iriam necessitar de *input multitouch* e à criação de testes automáticos para testar este módulo. Esta funcionalidade fez com que fossem utilizados grande parte dos dois dias de *buffer*.

A tarefa "Implementação da ação de desenhar" demorou significativamente menos tempo, ou seja, aproximadamente 15 horas.

O restante tempo do *buffer* foi utilizados para a elaboração deste documento e para a planificação das tarefas para o *sprint* seguinte.

4.3.3. *Sprint* 2

Planificação das tarefas para o segundo sprint

Para o segundo *sprint* foram novamente selecionadas as *user stories* com maior prioridade e foram analisadas para obter os requisitos e a sua estimativa. Sendo estas as seguintes:

US-4 Apagar desenho com o dedo

- Implementação da ação de apagar – 13h.

US-7 Inserir formas

- Definição do modelo representativo das formas (coordenadas da forma em relação a um ponto central) – 5h;
- Criação das formas – 2h;
- Implementação do sistema de inserção de formas no desenho – 13h.

US-8 Mover, escalar, rodar e escolher cor das formas

- Implementação do sistema de mover as formas – 2h;
- Implementação do sistema de escalar as formas – 13h;
- Implementação do sistema de rodar as formas – 13h;

- Implementação do sistema da forma ter uma cor – 2h.
- Implementação das ações de mover, escalar e rodar a nível do *input* – 8h.

US-11 Definir cor para componentes

- Implementação do interface para seleção de cor – 8h;
- Integração do sistema de mudança de cor na arquitetura base – 8h.

US-9 Inserir texto

- Implementação do interface de inserção de texto na aplicação – 3h.
- Implementação do sistema de inserção de texto no desenho – 13h.

US-10 Mover, escalar, rodar e escolher cor do texto

- Implementação do sistema de mover o texto (adaptado do sistema das formas) – 1h;
- Implementação do sistema de escalar o texto (adaptado do sistema das formas) – 5h;
- Implementação do sistema de rodar o texto (adaptado do sistema das formas) – 5h;
- Implementação do sistema do texto ter uma cor (adaptado do sistema das formas) – 1h.
- Implementação das ações de mover, escalar e rodar a nível do *input* (adaptado do sistema das formas) – 3h.

US-1 Inserir imagens

- Implementação do sistema de escolha e obtenção da imagem – 8h;
- Implementação do sistema de visualização da imagem – 8h;
- Implementação do sistema de transferência da imagem – 8h.

Na figura seguinte, apresenta-se o diagrama de Gantt com as tarefas detalhadas distribuídas pelo *sprint*:

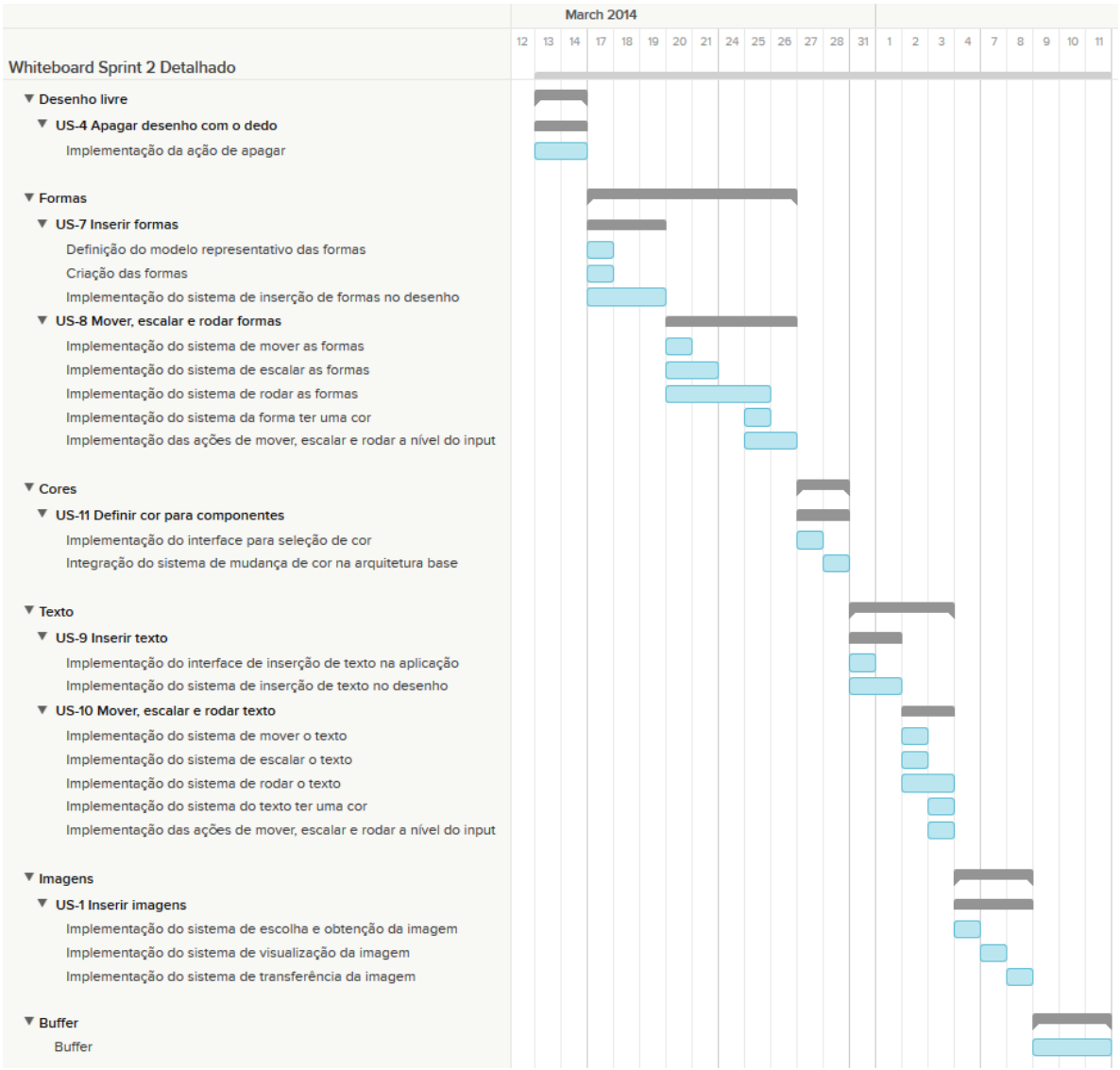


Figura 17 Diagrama de Gantt detalhado das tarefas para o segundo *sprint*

Finalização do segundo sprint

O seguinte gráfico Burndown mostra o progresso que foi feito durante o segundo *sprint* em relação ao progresso esperado tendo em conta as estimativas.

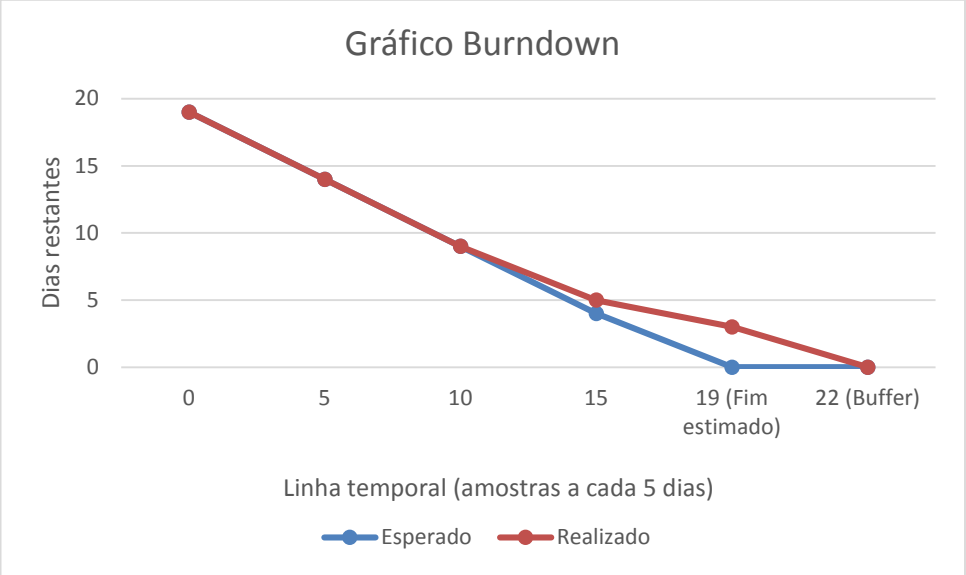


Gráfico 5 Gráfico Burndown do progresso das tarefas ao longo do segundo *sprint*

Este *sprint* começou a correr como estava planeado, no entanto mais tarde a tarefa de "Implementação do sistema de transferência da imagem" demorou muito mais tempo do que estava planeado, ocupando os 3 dias de *buffer* para além das 8h que estavam planeadas.

4.3.4. *Sprint* 3

Planificação das tarefas para o terceiro sprint

As *user stories* que foram selecionadas e detalhadas para este *sprint* são as seguintes:

US-2 Inserir fotografias

- Implementação do sistema de obtenção de fotografias – 13h;
- Implementação do interface do utilizador para poder tirar fotografias – 3h.

US-6 Mover, escalar e rodar imagens

- Implementação do sistema de mover as imagens – 5h;
- Implementação do sistema de escalar as imagens – 13h;
- Implementação do sistema de rodar as imagens – 13h;
- Implementação das ações de mover, escalar e rodar a nível do *input* – 5h.

US-13 Placeholder durante transferência de imagem

- Desenvolvimento do *placeholder* – 8h;
- Adaptação do sistema e arquitetura para acomodar a funcionalidade – 13h.

US-23 Selecionar e editar componentes

- Adaptação do sistema de camadas dos componentes – 8h;
- Implementação do sistema de seleção de componentes – 20h.

US-25 Selecionar e editar componentes

- Implementação da funcionalidade da alteração das camadas dos componentes – 13h;
- Implementação do interface do utilizador para permitir esta funcionalidade – 3h.

Na figura seguinte apresenta-se o diagrama de Gantt com as tarefas detalhadas:

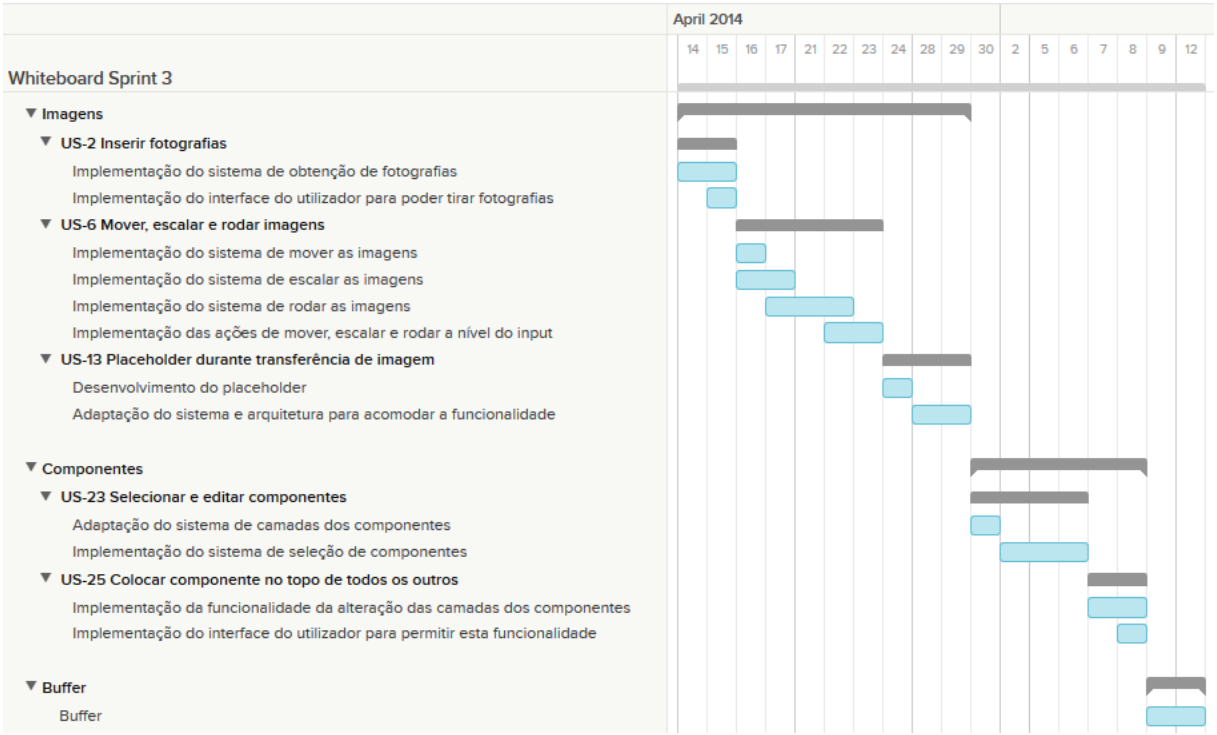


Figura 18 Diagrama de Gantt detalhado das tarefas para o terceiro *sprint*

Devido a terem existido três feriados durante o período de tempo destinado ao terceiro *sprint*, este foi consideravelmente mais curto.

Finalização do terceiro sprint

O seguinte gráfico Burndown mostra o progresso que foi feito durante o terceiro *sprint* em relação ao progresso esperado tendo em conta as estimativas.

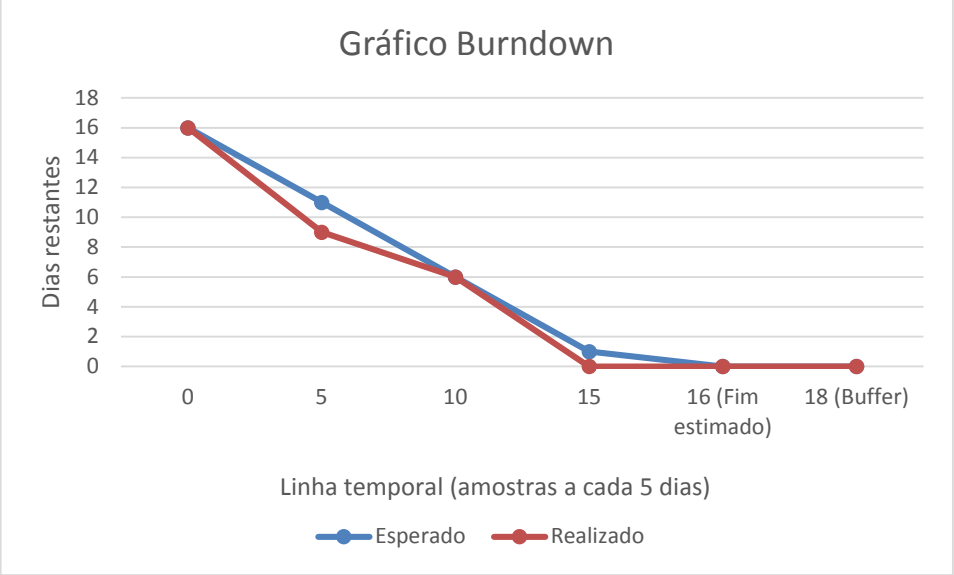


Gráfico 6 Gráfico Burndown do progresso das tarefas ao longo do terceiro *sprint*

Este *sprint* começou a correr melhor do que estava planeado devido ao facto da implementação da *user story* 6 ter sido tão similar à implementação da *user story* 8, o que facilitou a sua implementação. Este avanço nas tarefas permitiu que o *sprint* fosse terminado 1 dia antes do tempo inicialmente previsto.

4.3.5. *Sprint* 4

Planificação das tarefas para o quarto sprint

Na reunião de início de *sprint* foram removidas algumas *user stories* que resultaram na eliminação de algumas funcionalidades já implementadas, e foram adicionadas outras. Também ficou decidido que a integração com o produto RCS iria começar neste *sprint* e acabar no próximo.

Efetivamente as tarefas estipuladas para este *sprint* foram as seguintes:

US-24 Alterar texto já inserido no desenho

- Implementação da funcionalidade – 5h;
- Implementação do interface do utilizador para permitir esta funcionalidade – 2h.

US-14 Ações fluidas e em tempo real

- Otimizar sistema de transferência de informação – 13h;

- Ajustes de interface para permitir visualizar ações fluidas – 8h.

US-26 Escolher imagem dos ficheiros e definir como fundo de ecrã

- Ligação com sistema de obtenção de imagens – 8h;
- Ajustes do ecrã para a imagem escalar da mesma forma nos vários utilizadores – 13h.

US-27 Tirar fotografia e definir como fundo de ecrã

- Ligação com sistema de obtenção de fotografia – 8h;
- Ajustes da fotografia conforme a orientação desta – 5h.

US-28 Notificação de nova imagem de background a ser recebida

- Adaptação da arquitetura para permitir esta funcionalidade – 8h;
- Criação da mensagem de notificação – 1h.

US-15 Mudar o fundo de ecrã para uma cor

- Interface com o utilizador para seleção de cor – 3h;
- Atualização do ecrã e transferência da informação – 5h.

US-5 Convidar contactos RCS

- Leitura de documentação e aprendizagem da API de comunicação – 13h;
- Adaptação do sistema de comunicação para suportar o funcionamento da API do RCS – 8h;
- Implementação do módulo de comunicação utilizando a API do RCS – 13h;
- Integração do sistema de transferência de imagens – 13h;
- Integração do interface do utilizador com a aplicação RCS – 8h;
- Convite de contactos e permitir entrarem na sessão de desenho – 8h.

Na figura seguinte, apresenta-se o diagrama de Gantt com as tarefas detalhadas distribuídas pelo *sprint*:

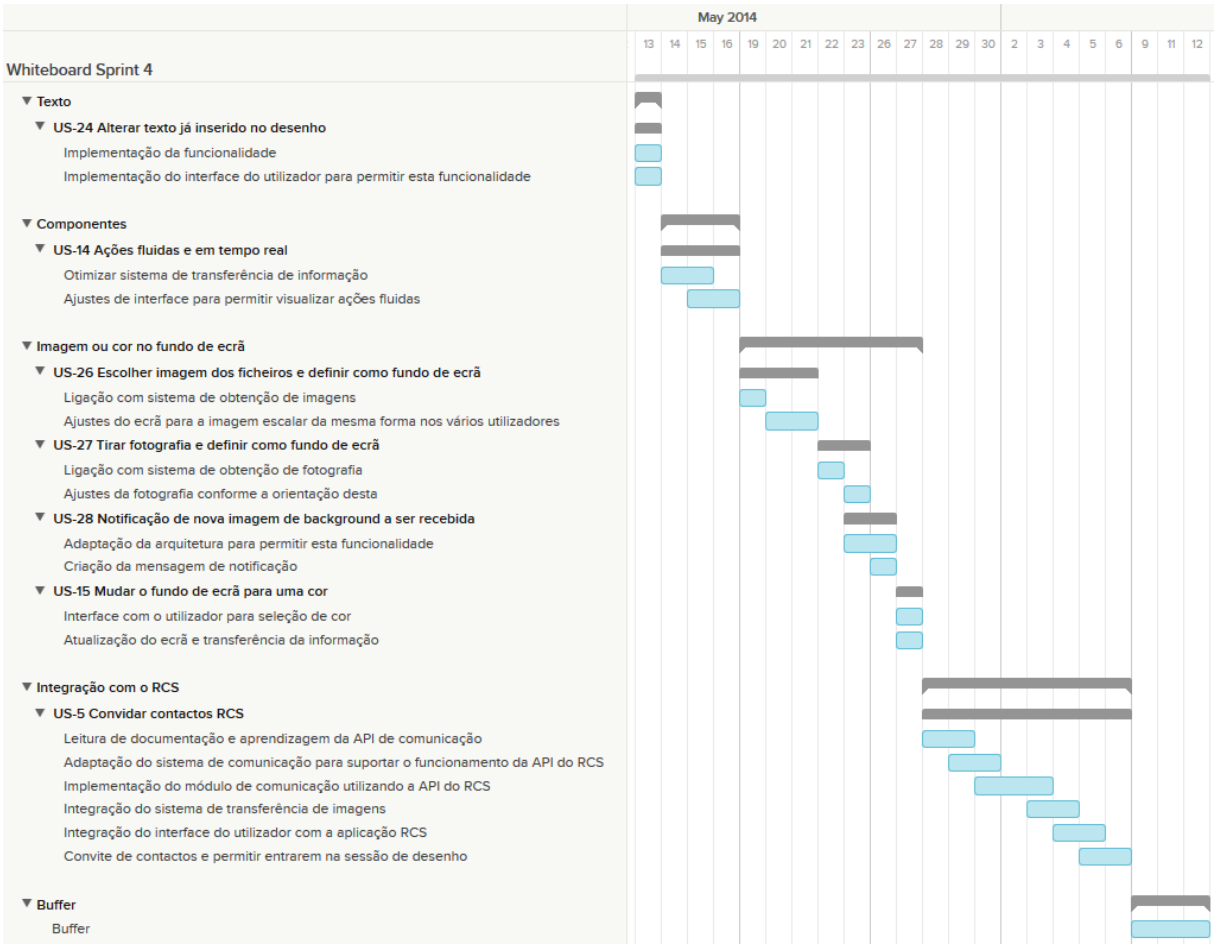


Figura 19 Diagrama de Gantt detalhado das tarefas para o quarto *sprint*

Finalização do quarto sprint

O seguinte gráfico Burndown mostra o progresso que foi feito durante o terceiro *sprint* em relação ao progresso esperado tendo em conta as estimativas.

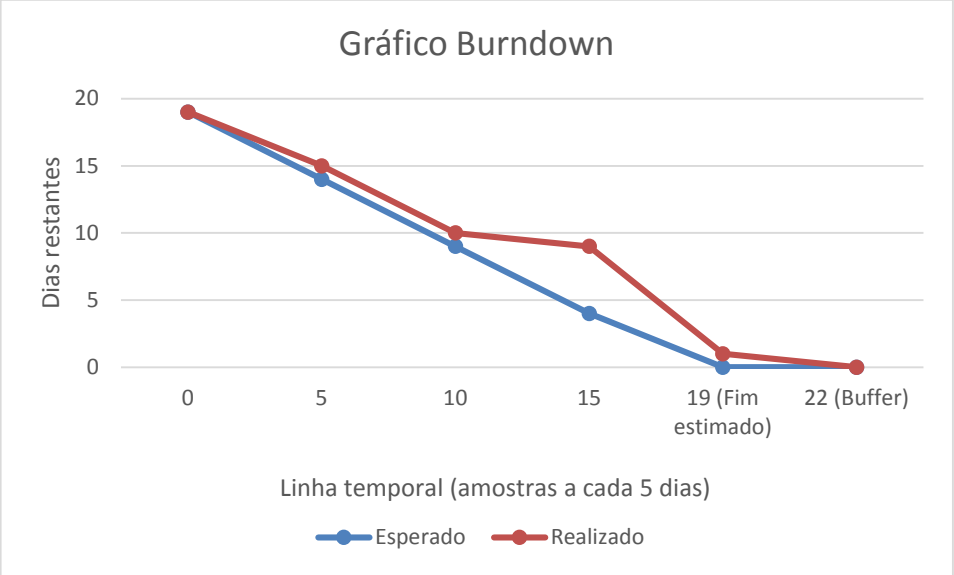


Gráfico 7 Gráfico Burndown do progresso das tarefas ao longo do quarto *sprint*

Durante o quarto *sprint* houve um atraso de um dia na tarefa "Ajustes do ecrã para a imagem escalar da mesma forma nos vários utilizadores" que se propagou ao longo do *sprint*, e acabou por se utilizar 1 dia do buffer para compensar este atraso.

4.3.6. *Sprint* 5

Planificação das tarefas para o quinto sprint

Este subcapítulo descreve o quinto e ultimo *sprint* no desenvolvimento da aplicação Whiteboard. Neste foram implementadas as últimas funcionalidades e as ultimas modificações necessárias para se poder considerar esta uma aplicação de desenho colaborativo acabada e integrada na aplicação RCS Suite da WIT Software.

Para o quinto *sprint* foram novamente selecionadas as *user stories* com maior prioridade e foram analisadas para obter os requisitos e a sua estimativa. Sendo estas as seguintes:

US-16 Múltiplas sessões de desenho

- Construir um sistema de identificação única de cada sessão – 8h;
- Sistema de filtragem de informação por sessão – 8h;
- Gravação da informação de cada sessão separadamente – 13h.

US-20 Sessão de desenho representada como objeto no *chat* RCS

- Criação do objeto para *chat* 1 para 1 – 13h;

- Criação do objeto para *chat* de grupo – 8h.

US-22 Nome para sessão de *chat*

- Interface para escrita do nome da sessão – 3h;
- Partilha do nome da sessão para os utilizadores da sessão – 5h.

US-17 Notificação de convite para desenho

- Implementação da notificação de convite – 13h.

US-18 Sincronização do desenho mesmo não estando neste

- Criação de sistema externo à aplicação a ser integrado na aplicação RCS para receção de informação – 13h;
- Sistema de filtragem de informação por sessão – 8h;
- Gravação da informação por sessão – 8h.

US-19 Notificação de alteração de desenho

- Implementação da notificação de aviso – 5h.

US-21 Sobressair desenhos modificados desde última abertura, no *chat*

- Modificação dos objetos de *chat* para o efeito – 8h.

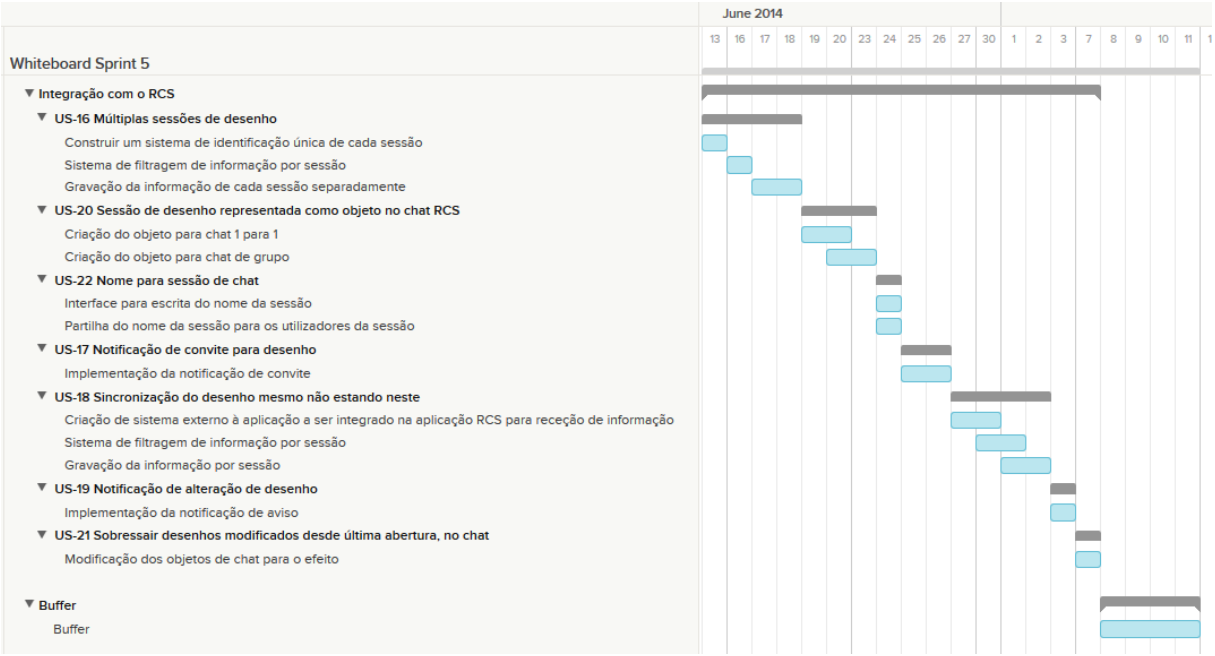


Figura 20 Diagrama de Gantt detalhado das tarefas para o quinto *sprint*

Finalização do quinto sprint

O seguinte gráfico Burndown mostra o progresso que foi feito durante o terceiro *sprint* em relação ao progresso esperado tendo em conta as estimativas.

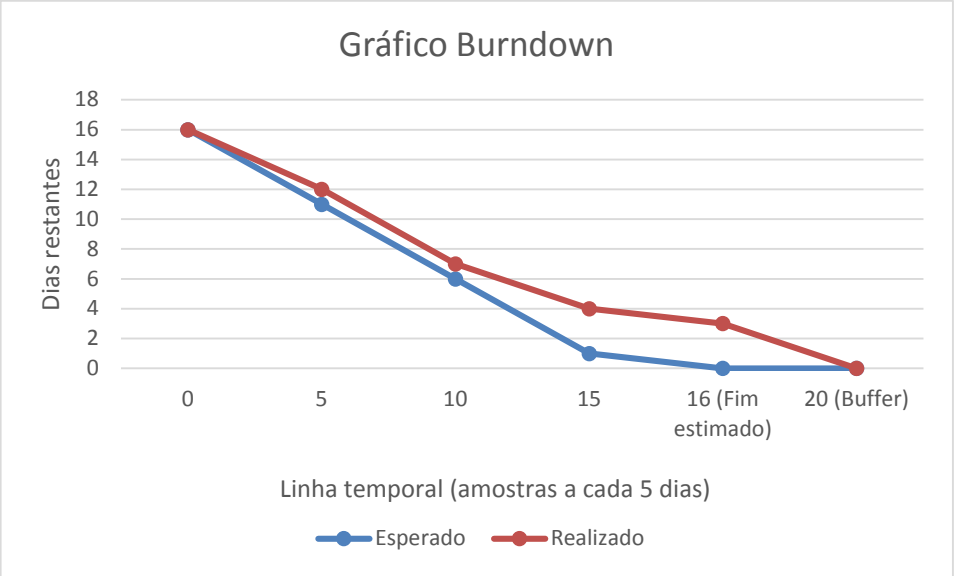


Gráfico 8 Gráfico Burndown do progresso das tarefas ao longo do quinto *sprint*

Durante o quinto e último *sprint* existiram vários atrasos, a tarefa "Criação do objeto para *chat* 1 para 1" demorou aproximadamente mais um dia, e a *user story* 18 demorou aproximadamente mais dois dias. E assim foram utilizados 3 dias de *buffer*.

4.4. Arquitetura geral da aplicação

A arquitetura da aplicação foi um dos aspetos que mereceu grande atenção durante todo o período de desenvolvimento. De facto, grande parte do primeiro *sprint* foi focado no planeamento e desenvolvimento da mesma, de forma a conseguir-se uma base suficientemente flexível para suportar as necessidades da ferramenta. Como foi descrito anteriormente, *maintainability* é um dos atributos de qualidade devido à natureza da aplicação, modificar a arquitetura de um sistema deste género seria consideravelmente trabalhoso durante o desenvolvimento. Outra mais-valia do investimento inicial foi a facilidade com que se integrou com a aplicação RCS Suite da WIT Software, feita apenas na fase final do desenvolvimento do projeto.

A aplicação tem 5 módulos distintos, sendo estes: 1) máquina de estados - contêm vários estados e cada um representa uma ação que o utilizador pode fazer; 2) sistema para gerir a comunicação entre dispositivos; 3) módulo de interface - gere a apresentação da informação mostrada ao utilizador; 4) módulo de *input* do utilizador - está muito interligado com o módulo da máquina de estados uma vez que faz o pré-processamento do *input* do utilizador para a máquina de estados usar.

Para apresentar a arquitetura geral da aplicação foram utilizados diagramas C4[39] que dão a conhecer a arquitetura de uma aplicação a 4 níveis de detalhe que têm como nome: contexto, módulos, componentes e classes. O contexto é composto por um ou mais módulos, cada módulo contem um ou mais componentes, e cada componente por sua vez é implementado por uma ou mais classes. No entanto neste relatório, preferiu utilizar-se apenas os primeiros 3 níveis de diagramas para apresentar a arquitetura geral, e no subcapítulo seguinte, “Módulos da aplicação”, recorrer-se a outros diagramas para facilitar a compreensão do funcionamento de cada um destes módulos.

4.4.1. Diagrama C4 contextual

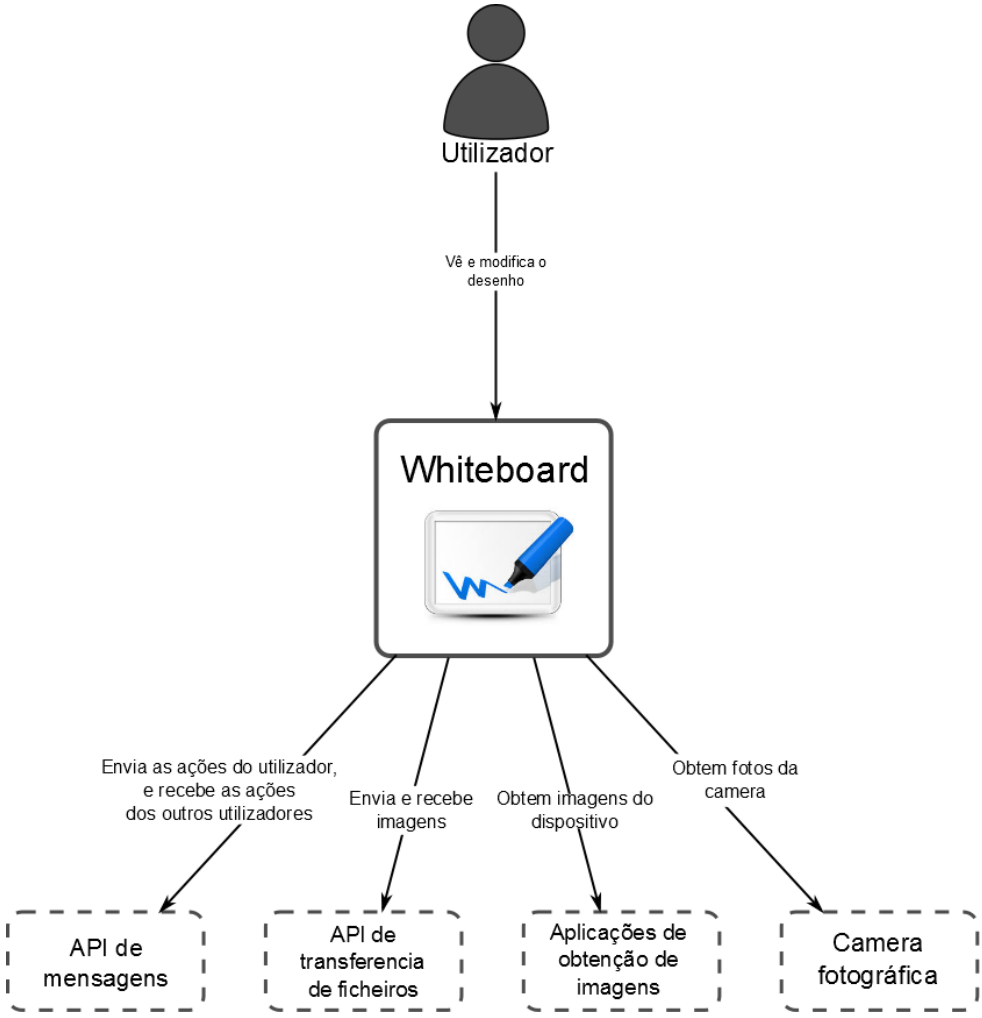


Figura 21 Diagrama C4 contextual da aplicação

No diagrama C4 contextual é possível ver as interações da aplicação com entidades externas à mesma. A nível de utilização só existe um único tipo de utilizador, sendo este que interage com o desenho, modificando-o e visualizando-o. Em relação às entidades externas que a aplicação utiliza, são entidades para troca de informação entre dispositivos ligados à mesma sessão, tais como a API de mensagens e a API de transferência de ficheiros, as quais são disponibilizadas pela *stack* de comunicações pertencente à aplicação RCS. A aplicação utiliza também, através de *intents* do Android, outras aplicações do dispositivo que disponibilizam funcionalidades como, por exemplo, utilizar aplicações de obtenção de imagens do dispositivo e fotografias da câmara de filmar, imagens essas posteriormente utilizadas para colocar no fundo de ecrã do desenho.

4.4.2. Diagrama C4 de módulos

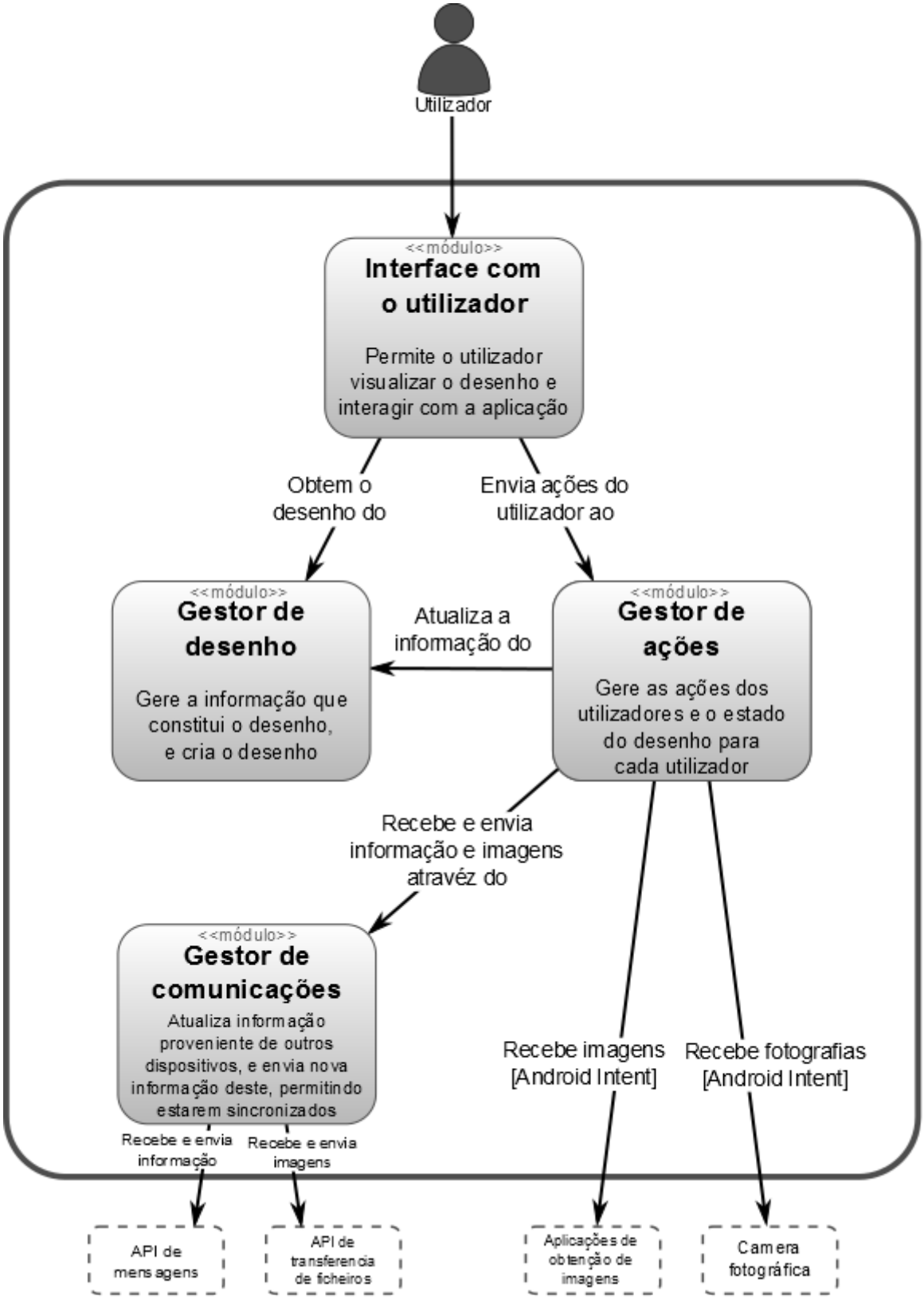


Figura 22 Diagrama C4 de módulos da aplicação

No diagrama C4 de módulos é possível ver o mesmo que no diagrama C4 de contexto mas de forma mais detalhada em relação à organização da arquitetura da aplicação.

Existe um módulo com o nome de “Interface com o utilizador” que faz a separação entre a lógica da aplicação, e o que o utilizador vê e faz. Com esta separação é mais fácil gerir os *inputs* de forma abstraída da maneira como o Android os processa e disponibiliza, e poder utilizá-los numa forma mais simples para a plataforma. Esta separação permite também ignorar o tamanho do ecrã e assim facilitar todo o sistema de cálculos, translações e rotações dos componentes.

O módulo “Gestor de ações” processa a informação recebida do “interface de utilizador” e do “Gestor de comunicações”, para as transformar em ações de utilizadores que são geradas pela máquina de estados interna a este módulo. Se as ações consistirem em obter imagens ou fotos do dispositivo, este gestor utiliza *intents* do Android para obter essa informação. No fim, estas ações são utilizadas para atualizar a informação no “Gestor de desenho” e para enviar as ações do utilizador local para os outros dispositivos ligados à mesma sessão através do “Gestor de comunicações”.

O módulo “Gestor de comunicações” faz a separação entre o sistema de lógica e o mundo exterior, ou seja, os outros dispositivos ligados à mesma sessão. É utilizado para processar a informação recebida que é proveniente de outros dispositivos, e transmiti-la para o “Gestor de ações”. Este módulo é também utilizado para enviar informação proveniente deste dispositivo para os outros ligados à mesma sessão.

O módulo “Gestor de desenho” grava e faz a gestão da informação do desenho guardando todos os componentes visuais prontos a serem impressos para uma imagem, a pedido do módulo “Interface com o utilizador”, que posteriormente utiliza a imagem para a apresentar no ecrã.

4.4.3. Diagrama C4 dos componentes do módulo “Interface com o utilizador”

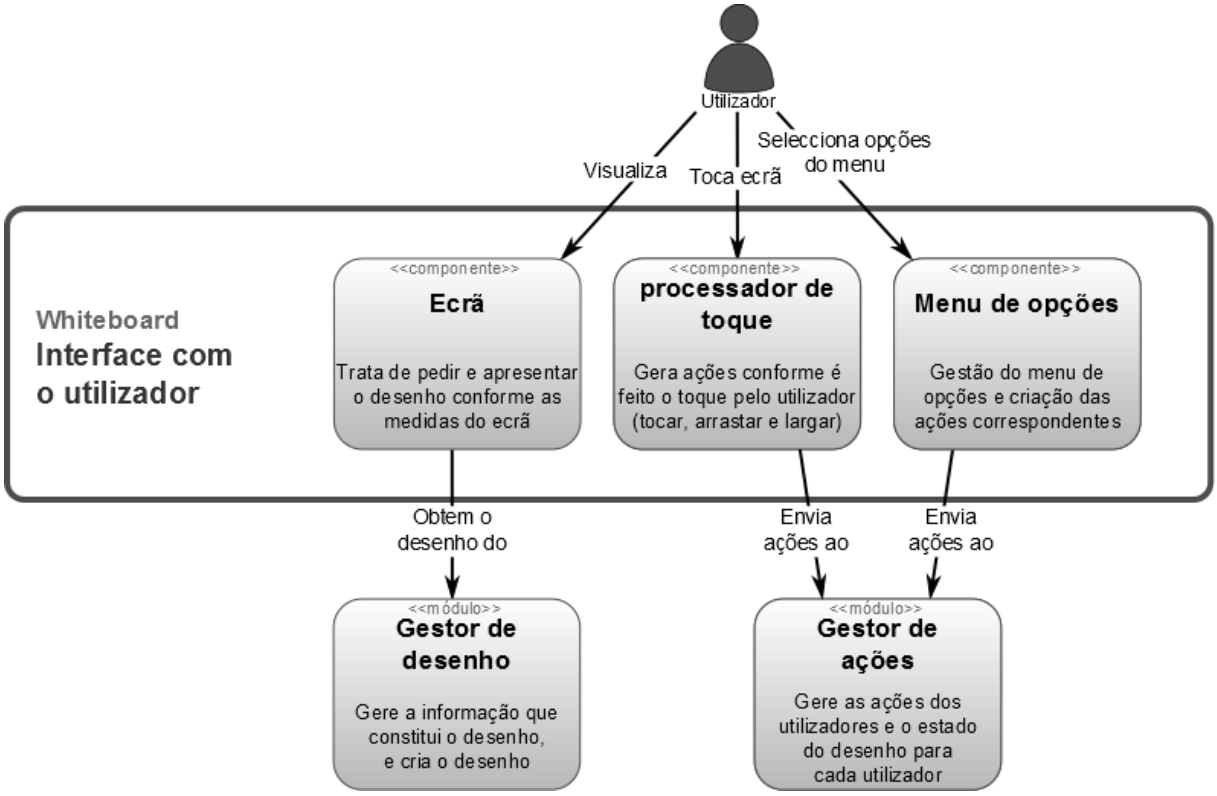


Figura 23 Diagrama C4 de componentes do módulo “Interface com o utilizador”

O módulo “Interface com o utilizador” é composto por três componentes, “Ecrã”, “Processador de toque”, e “Menu de opções”.

O componente “Ecrã” faz a gestão da apresentação do desenho no ecrã. Este contem um subcomponente câmara que permite facilmente pedir a imagem ao “Gestor de desenho” com o tamanho pretendido, e assim poder ajustá-la ao ecrã como for pretendido.

O componente “Processador de toque” é utilizado para processar todos os toques, e uniformizá-los para os encaminhar para o “Gestor de ações”. Este componente também usa a câmara, de forma a transformar coordenadas do ecrã real para o ecrã fictício no qual toda a lógica do sistema é baseada.

O componente “Menu de opções” faz a gestão do menu de opções e cria ações conforme a opção que seja selecionada pelo utilizador. Algumas opções deste menu são para inserir formas ou background no desenho, as quais criam ações que são depois delegadas ao “Gestor de ações”.

4.4.4. Diagrama C4 dos componentes do módulo “Gestor de ações”

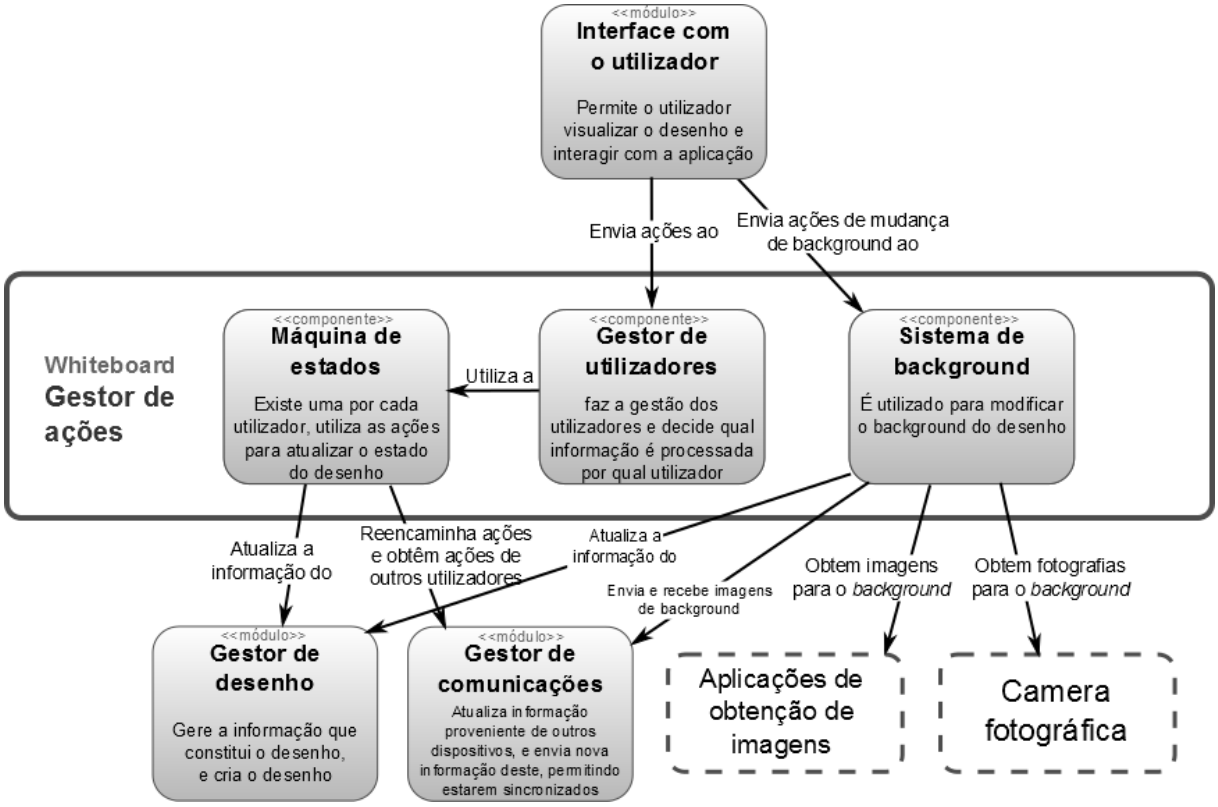


Figura 24 Diagrama C4 de componentes do módulo “Gestor de ações”

O módulo “Gestor de ações” é composto pelo componente “Máquina de estados”, pelo “Gestor de utilizadores” e pelo Sistema de background.

O componente “Máquina de estados” recebe as ações provenientes do componente “Gestor de utilizadores” e do módulo “Gestor de comunicações”. Este utiliza uma instância da máquina de estados para cada utilizador e aplica as ações conforme o estado anterior, ou seja, para que quando é uma ação de arrastar o dedo no ecrã, quando está a desenhar uma linha continua a desenhar a linha, ou quando está a arrastar um objeto, continua a arrastar o objeto.

Esta gestão da máquina de estados acontece de forma quase independente do estado dos outros utilizadores, exceto em situação de concorrência, em que dois ou mais utilizadores estão a fazer uma ação sobre o mesmo objeto, por exemplo, mover o mesmo objeto. E, nesse caso, apenas a ação do primeiro utilizador é considerada. Utiliza também entidades externas para alguns tipos de ações e no fim atualiza a informação do “Gestor de desenho”.

O componente “Gestor de utilizadores” recebe as ações do módulo “Interface com o utilizador” e reencaminha-as para a máquina de estados associada ao utilizador local.

O componente “Sistema de background” recebe ações associadas à mudança de *background* do módulo “Interface com o utilizador” e utiliza entidades externas para obter imagens para inserir como *background* do desenho. Também utiliza o módulo “Gestor de comunicações” para enviar as imagens colocadas como *background* e para receber imagens colocadas como *background* do desenho por outros utilizadores na sessão. No fim, atualiza a informação do “Gestor de desenho”.

4.4.5. Diagrama C4 dos componentes do módulo "Gestor de comunicações"

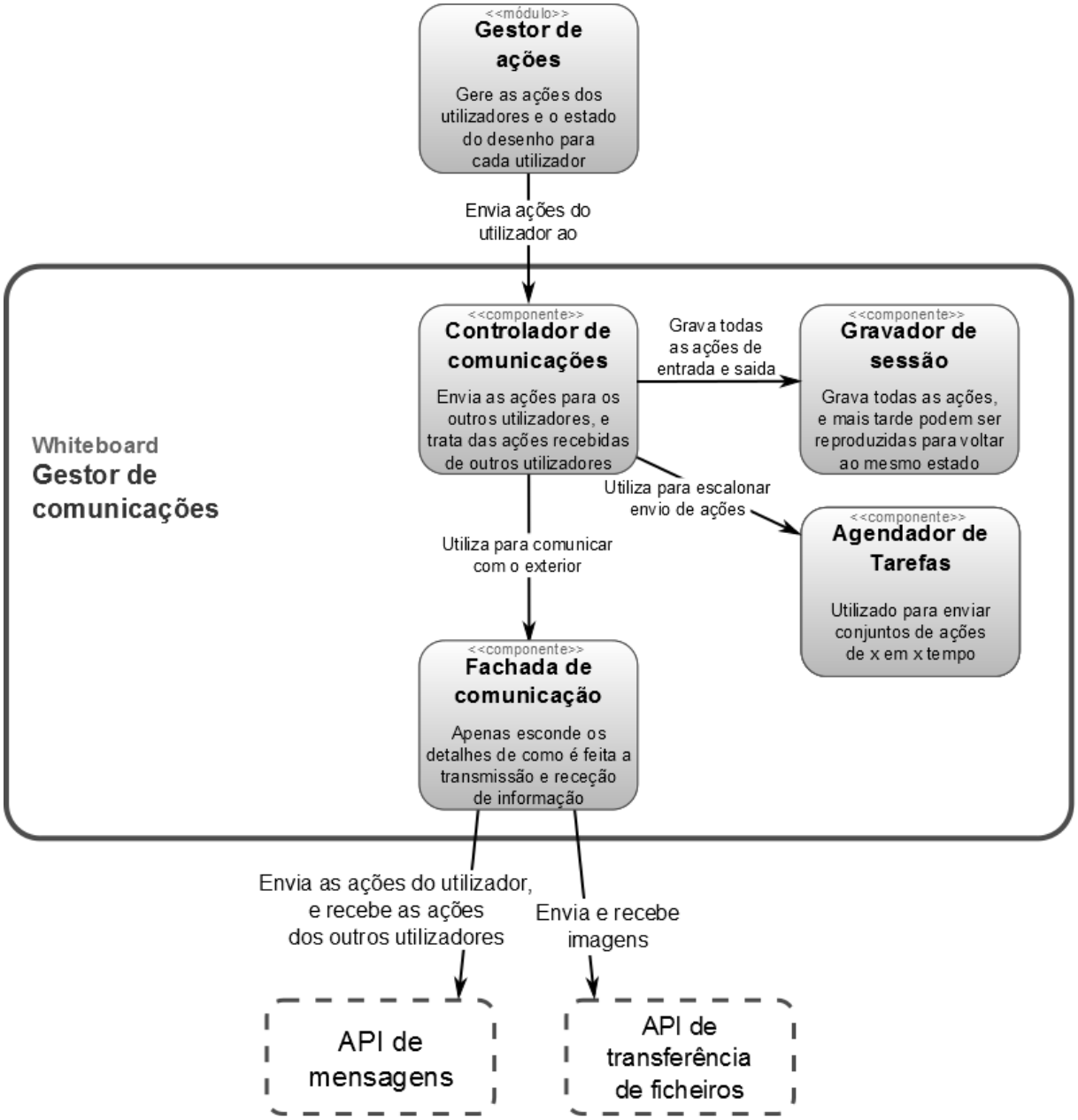


Figura 25 Diagrama C4 de componentes do módulo "Gestor de comunicações"

O módulo "Gestor de comunicações" é composto por quatro componentes: "Controlador de comunicações", "Gravador de sessão", "Agendador de tarefas" e "Fachada de comunicação".

O componente "Controlador de comunicações" faz toda a gestão de comunicações de entrada e de saída. Para as comunicações de saída, este agrupa várias ações provenientes do "Gestor de ações" e utiliza o "Agendador de tarefas" para ser notificado de quando deve enviar a informação, ou, no caso de chegar ao limite máximo de informação por mensagem, envia logo.

Quanto às comunicações de entrada, este redireciona-as para o “Gestor de ações”. Este componente tem também a tarefa de enviar todas estas ações de entrada e saída para o componente “Gravador de sessão”.

O componente “Gravador de sessão” grava toda a informação das ações para um ficheiro e também permite ler esse ficheiro para a aplicação voltar ao mesmo estado. Este componente é de grande utilidade pois permite que a aplicação volte ao mesmo estado, mesmo que esta seja fechada.

O componente “Agendador de tarefas” serve apenas para o “Gestor de comunicações” enviar as ações em intervalos de tempo para a “Fachada de comunicações”, de forma a enviar informação o mínimo de vezes possível, e de forma equilibrada.

O componente “Fachada de comunicações” é utilizado para fazer a separação entre o sistema de comunicações e o mundo exterior, permitindo assim existirem vários tipos de fachadas sem que isso influencie o resto da aplicação, tal como aconteceu durante o desenvolvimento. Com efeito, no início foi utilizada uma fachada que utilizava *sockets* TCP do Java para interagir com os outros dispositivos e, na altura da integração com a aplicação RCS, substituiu-se por uma fachada que utilizava a API de mensagens e a API de transferência de ficheiros da aplicação RCS.

4.4.6. Diagrama C4 do componente do módulo “Gestor de desenho”

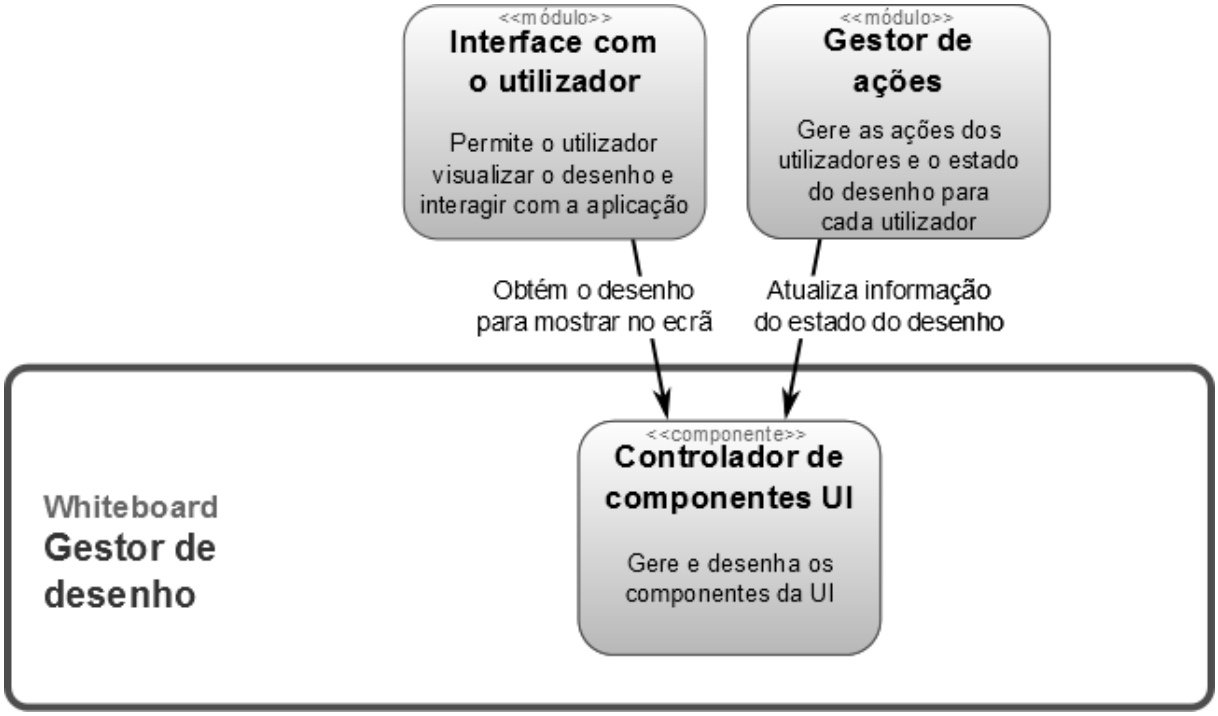


Figura 26 Diagrama C4 de componentes do módulo “Gestor de desenho”

O módulo “Gestor de desenho” é composto apenas pelo componente “Controlador de componentes UI”.

O componente “Controlador de componentes UI” recebe as atualizações do “Gestor de ações” e mantém a informação do desenho em memória, de forma a estar sempre pronto para imprimir o desenho quando o módulo “Interface com o utilizador” precisar. Guarda os vários componentes que constituem o desenho, e a sua ordem.

No Anexo B podem ver-se alguns dos diagramas desenvolvidos durante o estágio, mas que no entanto não refletem totalmente a arquitetura atual da aplicação.

4.5. Implementação dos módulos da aplicação

4.5.1. Gestor de ações

O módulo da máquina de estados permite manter um estado atual das ações dos utilizadores, e juntamente com esses estado atual (ou seja o contexto), e os novos *inputs* dos utilizadores tem o objetivo de identificar e executar a ação pretendida. Esta funcionalidade é necessária porque uma ação de um utilizador, como arrastar o dedo no ecrã, pode ter vários significados, por exemplo desenhar uma linha ou mover um objeto. Com este contexto é possível saber exatamente qual é a ação pretendida pelo utilizador.

Funcionamento detalhado da máquina de estados

O diagrama seguinte faz a representação da máquina de estados, a um nível abstraído da implementação dos estados, de forma a facilitar a compreensão das possíveis ações do utilizador.

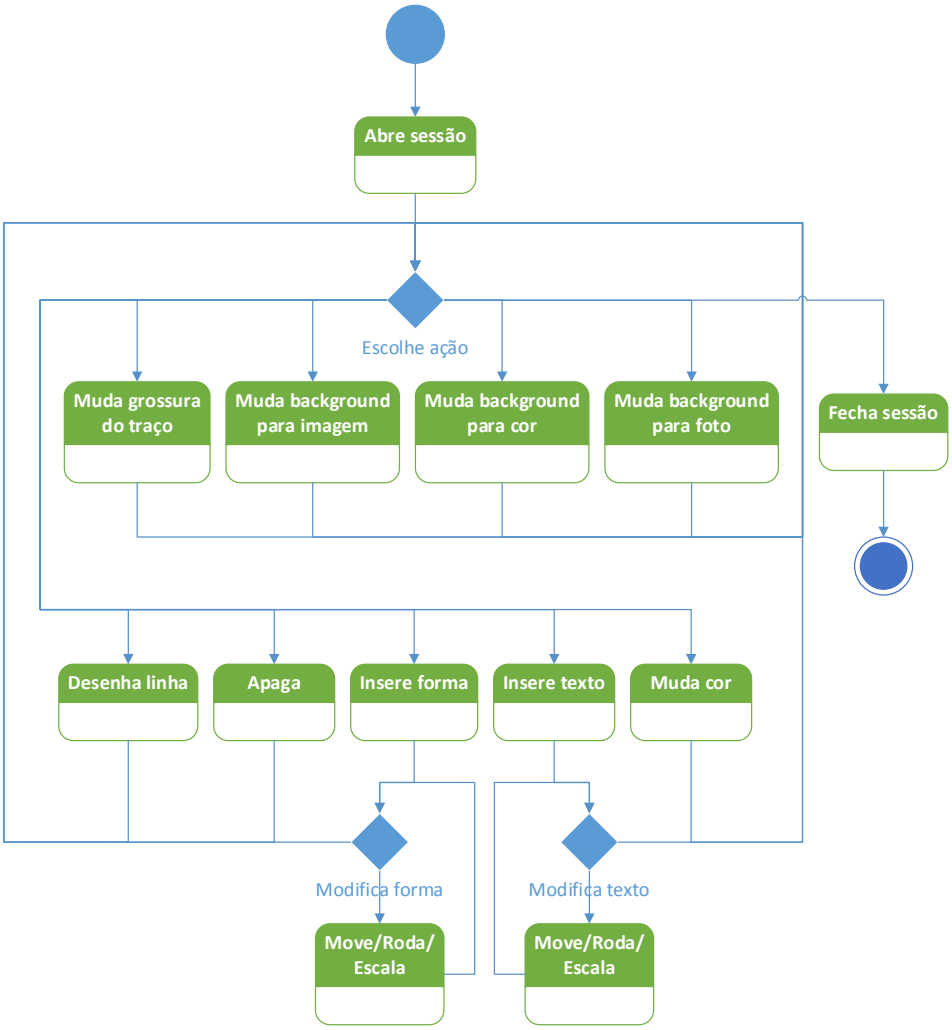


Figura 27 Diagrama de estados da aplicação

Ao nível de implementação, a máquina de estados é constituída por vários estados, em que cada um representa um contexto que permite que as ações dos utilizadores tenham significados diferentes, facilitando assim a separação de responsabilidades. Os estados da máquina de estados estão diretamente relacionados com as funcionalidades que a aplicação disponibiliza, sendo, portanto, os seguintes:

- Estado linha;
- Estado forma;
- Estado texto;
- Estado Imagem;
- Estado seleção de componente;
- Estado raiz.

Dos estados anteriormente descritos existe um que se destaca, sendo este o estado raiz. Este é utilizado como estado inicial e decide para qual estado a ação deve ser reencaminhada quando o estado está ativo. Este estado estipula ainda qual é o estado definido por defeito quando se

toca no ecrã – sendo este, de momento, o estado linha, que serve para desenhar linhas tocando no ecrã. Outros estados utilizam este estado raiz para mudarem o estado atual da máquina de estados quando consideram que o seu estado acabou, ou para quando este recebe erradamente informação que não lhe pertence (no eventual caso de se perder uma ou mais ações de utilizadores remotos ou existir um *bug* no sistema, a aplicação não *crashar* e ainda tentar recuperar, implicando apenas que o desenho possa ficar diferente dos desenhos dos outros utilizadores remotos).

Os outros estados utilizam as ações dos utilizadores para modificar o desenho de acordo com as suas responsabilidades e estado atual.

Implementação da máquina de estados

Para implementar o módulo da máquina de estados optou-se por utilizar a arquitetura mais usual de implementação de máquinas de estado que é usando herança: foi implementada uma classe abstrata “State” para ser a classe base da qual todos os outros estados são estendidos. Esta classe tem várias funções que podem ser *overloaded* pelos estados para poderem receber a informação e fazer as ações que lhes competem.

Seguidamente apresenta-se a classe “State” e as funções importantes para esta funcionalidade.

```
public abstract class State {

        public abstract void decodeMessage(JSONObject message);

        public void onPress(Vector2 point) {}

        public void onMove(Vector2 point) {}

        public void onRelease() {}

        public void onRelease(Vector2 point) {}

        public void onResizeRotateStart() {}

        public void onResizeRotate(float scaleFactor, float rotation) {}

        public void onChangeText(String text) {}
}
```

No Anexo C é possível ver o código e documentação desta classe.

Devido aos estados receberem informação proveniente de outros utilizadores remotos, existe uma função “decodeMessage(JSONObject message)” que é utilizada pelo módulo “Gestor de comunicações” que recebe um objeto JSON que contém a informação da mensagem, a qual

deve ser descodificada por cada entidade de forma a obter a informação necessária para efetuar a ação dos utilizadores remotos.

Em seguida é possível ver um excerto de código do construtor e da função de descodificação de mensagens do estado linha, a funcionar em junção para receber informação e criar a ação associada.

```
public LineState(Model model, UIContainer uiContainer,  NetworkController networkController,
      User user, JSONObject jsonObject, FixedCamera camera) throws JSONException {
   super(model, uiContainer, networkController, user, camera);

   onPress(new Vector2((float)jsonObject.getDouble(NetStr.POS_X),
         (float)jsonObject.getDouble(NetStr.POS_Y)));
}

@Override
public void decodeMessage(JSONObject message) {

   String action = message.getString(NetStr.ACTION);

   if(action.equals(NetStr.Action.MOVE)) {

      Vector2 previousPoint = line.getPoints().
                                    get(line.getPoints().size() - 1);

      Vector2 point = new Vector2(message.getInt(NetStr.POS_X) + previousPoint.x,
            message.getInt(NetStr.POS_Y) + previousPoint.y);
      onMove(point);

   } else if(action.equals(NetStr.Action.RELEASE)) {
      onRelease();

   } else {
   State state = new RootState(mModel, mUIContainer, mNetworkController, mOwner, camera);
      mOwner.setState(state);
      state.decodeMessage(message);
   }

}
```

As outras funções da classe “State” servem para o descodificador de mensagens as usar, e para serem utilizadas diretamente pelo módulo “Interface com o utilizador” com informação proveniente do utilizador local.

Implementação do gestor de utilizadores

Para manter uma organização coerente e uma maior separação de responsabilidades, foi implementado um componente que simula todos os utilizadores em cada dispositivo de forma transparente, dos quais apenas um dos utilizadores é um utilizador local, e o resto são utilizadores remotos. Este comportamento é ilustrado pela imagem seguinte.

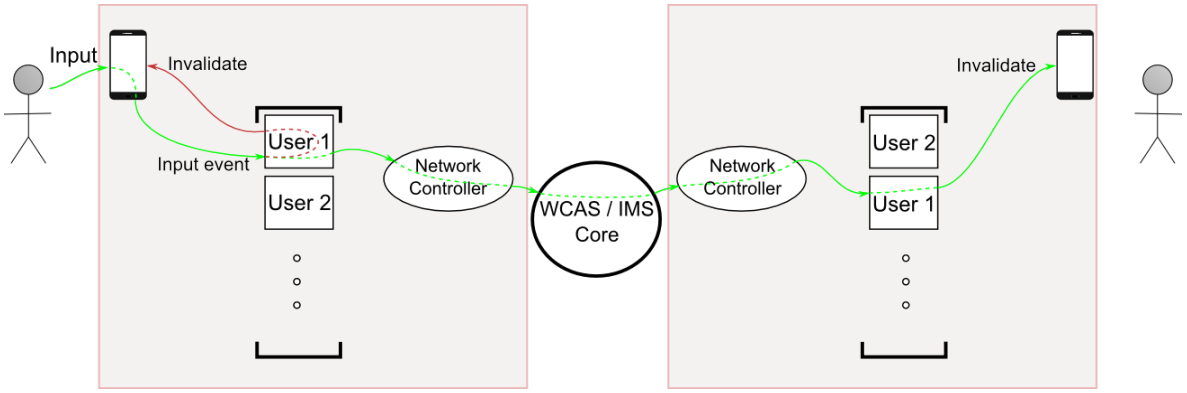


Figura 28 Diagrama de interação entre dispositivos

Esta arquitetura facilita a distribuição de responsabilidades o que simplifica o código e outras funcionalidades que tenham de ser implementadas.

Implementação do sistema de background

O sistema de *background* existe separadamente da máquina de estados devido a ser uma ação direta e não ser necessário guardar um estado anterior. No entanto, tem um funcionamento similar a esta, em que existem funções de um gestor que altera o componente UI do *background*, que são chamadas pelo componente que trata de receber eventos da API de ficheiros. Apresentam-se, em seguida, fragmentos de código que executam esta funcionalidade.

Fragmento de código da classe que recebe a informação da API de transferência de ficheiros:

```
public class FileReceiver implements EventFileTransferStateChangedCallback {

   @Override
   public void onEventFileTransferStateChanged(FileTransferInfo ftInfo) {
      switch (ftInfo.getState()) {
      case FT_STATE_TRANSFERRED:
            final Bitmap backgroundImage = BitmapFactory.decodeFile(ftInfo.getFilePath());
            FileTransferAPI.setFileTransferDisplayed(ftInfo.getId());

            FileOutputStream out = new FileOutputStream(backgroundPath, false);
            backgroundImage.compress(Bitmap.CompressFormat.PNG, 100, out);

            backgroundChangeHandler.changeBackground(
                  backgroundImage, "");

         break;
      case FT_STATE_TRANSFERRING:
         backgroundChangeHandler.changeBackgroundToLoading();
         break;
      }
   }
}
```

Fragmento de código da classe que faz a gestão da mudança de *background*:

```
public class BackgroundChangeHandler {

    public void setBackgroundBitmap(Bitmap bitmap) {
        backgroundUIComponent.changeBackground(bitmap);
    }

    public void setBackgroundLoading() {
        backgroundUIComponent.changeBackgroundToLoading();
    }
}
```

Esta última classe serve apenas como uma simples fachada para manter a divisão de responsabilidades.

4.5.2. Interface com o utilizador

Implementação do sistema de ecrã

Devido à necessidade de suportar muitas resoluções de ecrãs teve que se implementar um sistema que tratasse da gestão do ecrã e que possibilitasse uma abstração das particularidades do dispositivo a nível da lógica da aplicação. Com efeito, foi feita a codificação da aplicação para que a nível da lógica da aplicação exista um ecrã fictício que tenha sempre o mesmo tamanho em todos os dispositivos, e foi criada uma classe "FixedCamera" que trata de todas as transformações de informação do ecrã real para o ecrã fictício e vice-versa. Este sistema ajuda imenso no que toca à visualização de informação, o sistema de *input*, e também na troca de informação entre os vários dispositivos, visto que esta está sempre padronizada dentro dos mesmos limites.

Um problema encontrado durante o desenvolvimento desta aplicação foi não só a existência de dispositivos com resoluções diferentes, mas também com ou rácio de resolução diferente. Tal implica que em alguns dispositivos o desenho tenha de ser mais esticado num dos eixos para preencher todo o ecrã, ou, alternativamente, fazer com que o desenho seja centrado e esticado apenas até ao ponto máximo que mantém o rácio e, portanto, fiquem barras na borda do ecrã dos dispositivos nas quais não se pode desenhar, como se pode ver na Figura 30. Visto tratar-se de um problema em que clientes diferentes podem optar por soluções diferentes, estes componentes foram desenvolvidos tendo isso em conta, sendo suficientemente flexíveis para qualquer destas situações e podendo ser modificados apenas com a alteração de alguns valores na aplicação.

Figura 29 Ilustração do problema de rácio de tamanhos de ecrã em dispositivos

De seguida, apresenta-se um fragmento do código da classe “FixedCamera”:

```
public class FixedCamera {

    /**
     * Changes coordinates from screen to camera
     * @param point original screen coordinates
     * @return the same vector with the changed coordinates
     */
    public Vector2 unproject(Vector2 point) {
        point.x = unprojectX(point.x);
        point.y = unprojectY(point.y);
        return point;
    }

    /**
     * Changes coordinates from camera to screen
     * @param point original camera coordinates
     * @return the same vector with the changed coordinates
     */
    public Vector2 project(Vector2 point) {
        point.x = projectX((int) point.x);
        point.y = projectY((int) point.y);
        return point;
    }

    public int unprojectX(float x) {
        return (int) (x/ratioWidth);
    }

    public int unprojectY(float y) {
        return (int) (y/ratioHeight);
    }

    public float projectX(int x) {
        return x*ratioWidth;
    }

    public float projectY(int y) {
        return y*ratioHeight;
    }
}
```

Como é possível constatar, as funções tratam de transformar coordenadas do ecrã real para o ecrã fictício e vice-versa. Esta classe contém também outras variáveis e funções para ajudar neste processo, as quais permitem também o ajuste de valores que configuram a câmara uma

vez que é esta classe que contém também o tamanho da câmara fictícia e do rácio a usar. No Anexo C é possível ver todo o código e documentação desta classe.

Implementação do processador de toque

O processador de toque é utilizado para processar o toque no ecrã por parte do utilizador, e simplificar e padronizar esta informação conforme a aplicação precisa dela. Para este processamento foi criada uma classe chamada "GestureHandler" que identifica os vários gestos que a aplicação utiliza, como toque com um dedo e toque com dois dedos. Este processador também é utilizado para abrir o menu de opções quando o utilizador faz um *input* do tipo "long press" no desenho. Depois de analisar a informação converte-a para coordenadas do ecrã fictício utilizando a câmara e envia a informação para o "Gestor de ações".

Para fazer esta análise do *input* o processador de toque obtêm todos os MotionEvent da View do desenho e, internamente, utiliza uma pequena máquina de estados para saber o que o *input* significa.

Apresenta-se em seguida o código do enumerador Java que é utilizado para o processador de toque saber em que estado está:

```
public enum GestureState {
   NONE,
   ONE_FINGER,
   TWO_FINGERS,
   WAITING_FOR_FINGER_RELEASE
}
```

O seguinte diagrama ilustra o funcionamento desta máquina de estados.

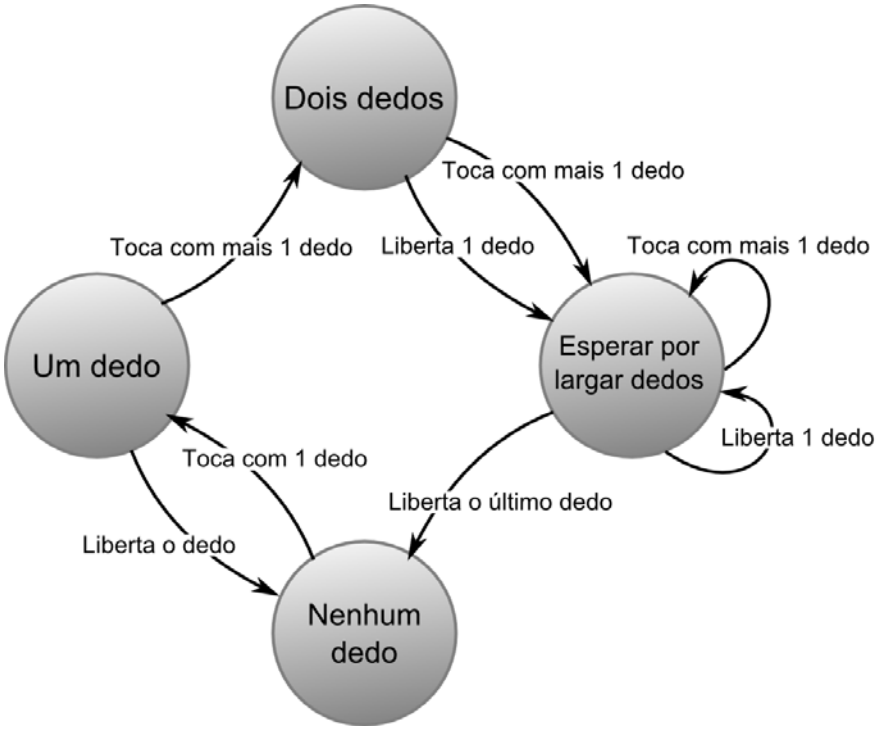


Figura 30 Diagrama de estados da máquina de estados do processador de toque

Funcionamento do menu de opções

Para implementar o menu de opções foi utilizada uma biblioteca *open-source* [40] para fazer menus, a qual foi selecionada por proporcionar menus bastante flexíveis de utilizar e, do ponto de vista estético, os mais bonitos de entre os pesquisados.

O menu tem várias opções em árvore que permitem facilmente selecionar a opção pretendida. Na Figura 32 é possível ver o menu em diferentes estados, quando se pressionam opções que têm outras opções filho.

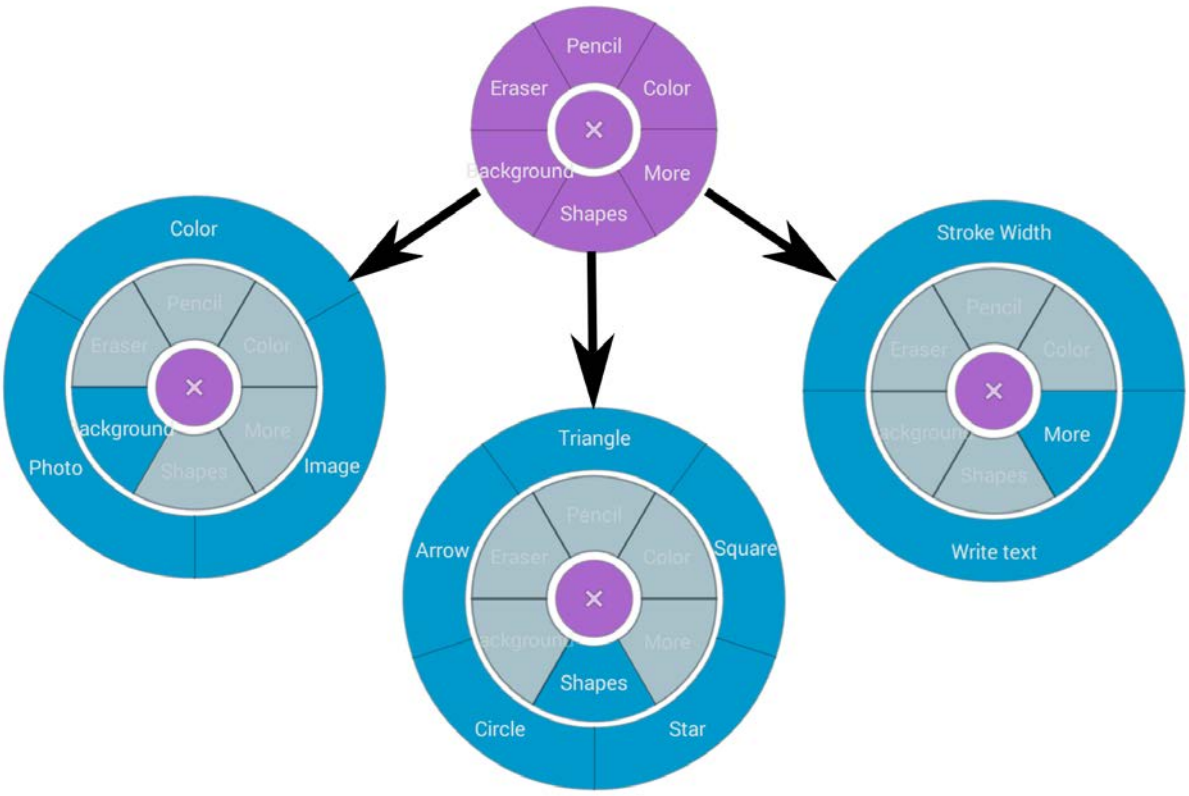


Figura 31 Menu de opções da aplicação

A árvore de opções é a seguinte:

- Forma:
  - Circulo;
  - Seta;
  - Triângulo;
  - Quadrado;
  - Estrela.
- Background:
  - Fotografia (tirada da câmara do dispositivo) como *background*;
  - Cor como *background*;
  - Imagem (armazenada no dispositivo) como *background*.
- Borracha;
- Desenho com toque;
- Selecionar cor;
- Mais opções:
  - Escrever texto;
  - Mudar a espessura das linhas que são desenhadas.

Quando uma destas opções é selecionada, a ação é redirecionada para o Gestor de ações, que por sua vez a redireciona para a máquina de estados ou para o sistema de *background* se for uma ação desse tipo.

Existem também janelas que são abertas em algumas das opções do menu, sendo estas para seleção de cor, escrita do texto para inserção no desenho, e seleção da grossura das linhas. As janelas referidas podem ser visualizadas na Figura 33.

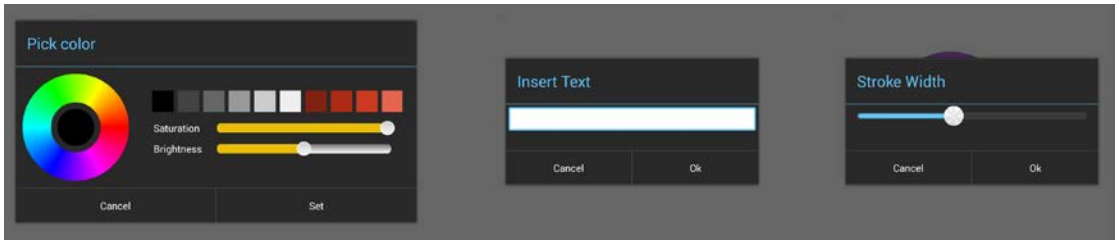


Figura 32 Janelas de inserção de informação adicional

4.5.3. Gestor de comunicações

Implementação do controlador de comunicações

Para desenvolver o controlador de comunicações criou-se um interface que permite uma abstração da fachada de comunicação ao contactar o controlador de comunicações, sendo o código deste interface o seguinte:

```
public interface CommunicatorCallback {

   public void receivedMessage(String userId, JSONObject message);
}
```

O seguinte fragmento de código mostra as funções mais importantes do controlador:

```
public class NetworkController implements CommunicatorCallback {

        @Override
        public synchronized void receivedMessage(final String userId, final JSONObject message) {}

        public synchronized void sendAction(JSONObject jsonObject) {}

        private synchronized void sendPackage(JSONObject message) {}

        public synchronized void sendMessageAndFlush(JSONObject jsonObject) {}

        public synchronized void flush() {}

        public synchronized void disconnect() {}

        public synchronized void connect() {}
}
```

A função “receivedMessage()” serve para a fachada de comunicação avisar o controlador que foi recebida uma nova mensagem de um utilizador especifico.

A função “sendActions()” serve para o “Gestor de ações” enviar ações para outros utilizadores. No entanto, estas podem não ser enviadas imediatamente, visto o envio das mensagens depende da quantidade de informação à espera de ser enviada e do agendador de tarefas.

A função sendPackage() é a função que efetivamente envia informação à fachada de comunicações. Esta função é chamada pela “sendActions()” e pelo agendador de tarefas.

A função “sendMessageAndFlush()” é uma maneira que o “Gestor de ações” tem de enviar imediatamente uma mensagem sem ter que estar à espera do agendador de tarefas.

A função “flush()” permite também ao “Gestor de ações” forçar o envio de todas as mensagens pendentes. Esta função é útil na situação em que o “Gestor de ações” sabe que nos próximos momentos não vão ser enviadas mais ações, por exemplo, quando o utilizador acaba de desenhar uma linha com o dedo.

As funções disconnect() e connect() apenas servem para avisar a fachada de comunicações que esta deve terminar/iniciar as suas funções, por exemplo, registar-se na API de comunicação para receber os eventos.

Implementação do gravador de sessão

O gravador de sessão é um simples mecanismo que faz uso de todas as mensagens já estarem no formato JSON e guarda-as todas num *array* também em formato JSON pela mesma ordem em que são recebidas/enviadas. Este mecanismo permite que com facilidade as mensagens possam ser novamente reproduzidas para a aplicação voltar a um estado específico. O seguinte fragmento de código é o que lê o ficheiro e reproduz o estado do desenho:

```
BufferedReader br = new BufferedReader(new FileReader(filePath));
String fileData = br.readLine();
if(fileData != null && !fileData.equals("")) {
   JSONArray array = new JSONArray(fileData);
   br.close();
   thisUser.setRemote(true);
   for (int i = 0, size = array.length(); i < size; i++) {
      JSONObject action = array.getJSONObject(i);
      networkController.receivedMessage(action.getString(NetStr.USER_ID), action);
   }
   thisUser.setRemote(false);
}
```

Como é possível ver no código, para este sistema funcionar basta reencaminhar cada uma das mensagens para o controlador de comunicações. Torna-se necessário colocar temporariamente o utilizador local como um utilizador remoto, de maneira a que o funcionamento interno seja o

pretendido. Caso contrário, o módulo “Gestor de ações” iria reenviar novamente a informação para os outros utilizadores.

Implementação do agendador de tarefas

O agendador de tarefas é uma utilização das classes Timer e TimerTask do Java. Esta foi considerada como um componente separado do componente controlador de comunicações para dar ênfase a este sistema assíncrono de envio de mensagens.

A implementação deste sistema é a seguinte:

```
timer = new Timer();
timerTask = new TimerTask() {

   @Override
   public void run() {
      synchronized (NetworkController.this) {
         if(messagesHolder != null)
            sendPackage(messagesHolder);
         }
      }
   };
timer.schedule(timerTask, MAX_DELAY);
```

O sistema é iniciado sempre que o controlador de comunicações recebe um pedido de envio de informação, e pode ser cancelado no caso de mais mensagens serem recebidas e o tamanho máximo de uma mensagem ser atingido. Nesta situação outra tarefa é agendada para enviar a restante informação mais tarde.

Implementação da fachada de comunicação

A separação da fachada de comunicação foi uma escolha muito importante que facilitou imenso o trabalho de integração com a aplicação RCS. No início foi utilizada uma fachada que utilizava *sockets* TCP para a comunicação e, começar a utilizar a API de comunicação da aplicação RCS Suite, foi tão simples como implementar uma nova fachada e substituir pela outra.

Todas as fachadas de comunicação tinham de implementar uma classe abstrata que se pode ser vista no seguinte código:

```
public abstract class Communicator {

   protected CommunicatorCallback callbackListener;

   public void setListener(CommunicatorCallback callbackListener) {
      this.callbackListener = callbackListener;
   }

   public CommunicatorCallback getListener() {
      return callbackListener;
   }

   public abstract void sendMessage(JSONObject message);

   public abstract void connect();

   public abstract void disconnect();

   public abstract int getBufferSize();

}
```

Com a exceção da função “getBufferSize()”, todas as funções já foram explicadas anteriormente.

Assim, a função “getBufferSize()” permite ao controlador de comunicações saber se cada mensagem recebida pelo “Gestor de ações”, juntamente com as mensagens à espera de serem enviadas, ainda cabem no tamanho máximo permitido para cada mensagem.

4.5.4. Gestor de desenho

Implementação do controlador de componentes UI

Este controlador é composto maioritariamente por uma classe UIContainer e um conjunto de instâncias de componentes que estendem um interface UIComponent. O UIContainer mantém uma lista de todos os componentes pela ordem correta para que, quando o desenho é pedido pelo módulo “Interface com o utilizador”, este possa solicitar a cada componente que ele se desenhe na tela. Quando o módulo “Interface com o utilizador” precisa de atualizar o desenho, envia uma instância da classe Canvas do Android ao UIContainer, que por sua vez o envia a cada um dos componentes no *array* para que se desenhem neste.

Todos os componentes são implementados pelo seguinte interface:

```
public interface UIComponent {

   public void drawOnCanvas(Canvas canvas, FixedCamera camera);

   public State getHandlerState();

   public int isTouching(Bitmap bitmap, Canvas canvas, FixedCamera camera, int posX, int posY);

   public void drawForTouchDetection(Canvas canvas, FixedCamera camera);

   public String getId();

   public void destroy();
}
```

A função “drawOnCanvas()” serve para se pedir ao componente para ele se desenhar no Canvas.

A função “getHandlerState()” serve para obter o estado do objeto a nível de estar a ser editado por outro utilizador, ou não, e serve para identificar se é possível fazer-se certas operações.

As funções “isTouching()” e “drawForTouchDetection()” são utilizadas para uma funcionalidade que, embora esteja implementada, não está a ser usada na aplicação, a qual permite que se possam editar componentes que já existem no desenho.

A função “getId()” serve apenas para obter o identificador único do componente.

Por fim, a função “destroy()” serve para o componente ter a possibilidade de finalizar alguns componentes internos.

4.6. Outras soluções técnicas

4.6.1. Reduzir a transmissão mensagens, e ações fluidas

Num sistema deste género onde é necessária a transmissão de grandes quantidades de informação, torna-se muito importante otimizar o sistema e a informação enviada.

Uma das soluções aplicadas ao sistema de envio de mensagens foi a agregação de várias mensagens que são enviadas todas juntas de maneira a reduzir a frequência de envio das mesmas. Devido a esta situação, os desenhos deixavam de ser tão fluidos e, portanto, foi necessário introduzir *timestamps* nas mensagens e nos clientes que recebiam esta informação. Desta forma, o sistema, ao receber este grupo de mensagens, apenas as enviava ao Gestor de ações quando fosse a altura correta, tendo sempre assim um *delay* entre as várias mensagens.

4.6.2. Sistema de seleção de componentes

Embora não faça parte da versão final, o sistema de seleção de componentes não deixou de ser um desafio. O sistema percorre todos os componentes pela ordem inversa em que é impressa no ecrã, ou seja, primeiro analisa os componentes que estão por cima. A análise passa por imprimir os componentes num Canvas e, cada vez que um dos componentes for impresso, detetar se o *pixel* correspondente ao ponto onde o utilizador tocou contínua transparente. Se assim for, o sistema continua a procura e passa para o próximo componente. Caso contrário, significa que é este o componente selecionado.

Outro problema deste sistema era a seleção de objetos tais como texto com imensos espaços em branco entre as letras, que tornava complicado selecionar o componente. Para resolver este problema, a impressão do componente para verificar o toque passou a ser feita por outra função que não imprimiria necessariamente o seu componente, mas sim o espaço que seria considerado pertencer-lhe. No caso do texto, o que realmente é impresso por esta função é o retângulo que rodeia o texto. Este comportamento é ilustrado pela Figura 34, na qual o objeto selecionado para edição seria o retângulo azul.

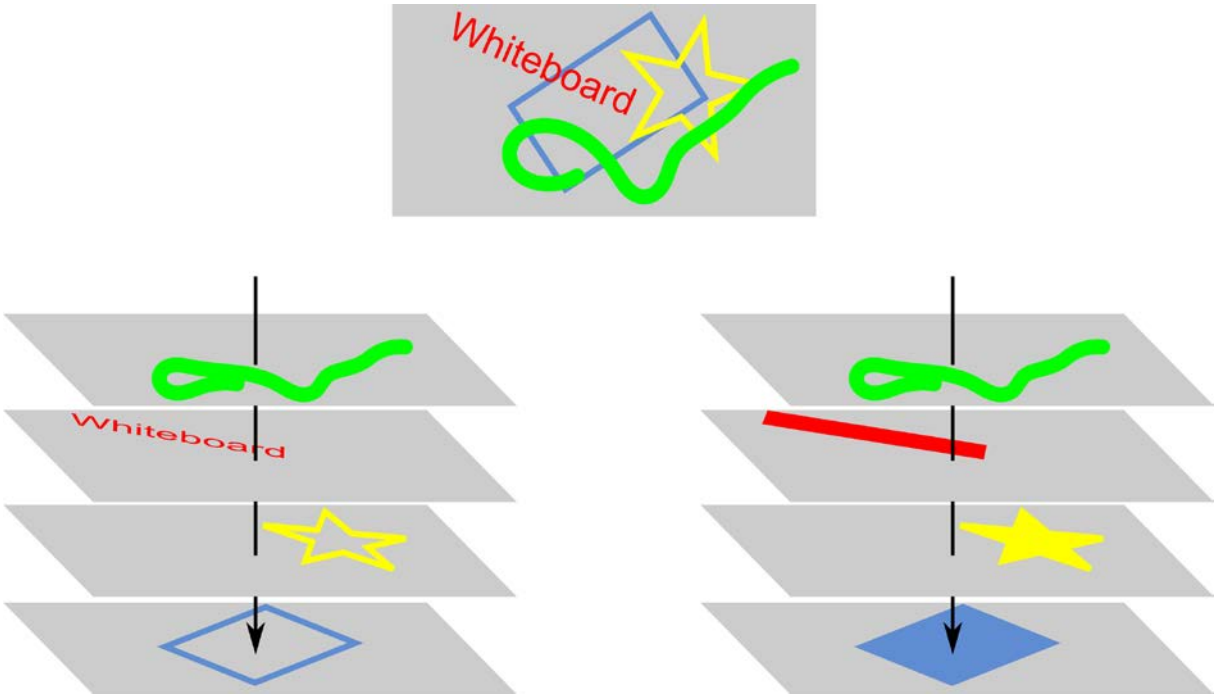


Figura 33 Funcionamento da seleção de componentes, o que é visível à esquerda, e o que realmente conta para fazer a seleção, à direita.

4.7. Dificuldades encontradas

De todas as dificuldades encontradas durante o estágio, a maior foi criar uma arquitetura suficientemente flexível que permitisse uma integração fácil, tanto das funcionalidades planeadas para desenvolvimento, como das funcionalidades que não estavam planeadas e não foram implementadas, mas podem vir a sê-lo num futuro próximo, tais como a funcionalidade de *undo/redo*.

Outra das grandes dificuldades foi a integração da aplicação com o RCS Suite que ficou a dever-se à complexidade do projeto, à sua arquitetura ser bastante complexa e ao funcionamento das API's disponibilizadas não ser muito claro.

4.8. Testes

Embora não tenha existido um plano de testes ficou definido que seriam feitos vários tipos de testes durante o desenvolvimento para as partes mais cruciais da aplicação.

4.8.1. Testes unitários

Durante o desenvolvimento da aplicação foram criados vários testes unitários, estes testes tinham como objetivo manter a qualidade dos módulos já implementados, e de manter a confiança na aplicação.

De seguida está um excerto do código de duas classes de testes unitários da aplicação.

```
public class Vector2Test extends AndroidTestCase {

   public void testTruncate2() {
      Vector2 vector2 = new Vector2(3.432355f, 7.321f, 2);
      assertTrue(vector2.x == 3.43f);
      assertTrue(vector2.y == 7.32f);
   }

   public void testTruncate3() {
      Vector2 vector2 = new Vector2(-3.432355f, -7.321f, 2);
      assertTrue(vector2.x == -3.43f);
      assertTrue(vector2.y == -7.32f);
   }

   public void testCpy1() {
      Vector2 vector2 = new Vector2(-3.432355f, -7.321f, 2);
      Vector2 temp = vector2.cpy();
      vector2.x = 0;
      vector2.y = 0;
      assertTrue(temp.x == -3.43f);
      assertTrue(temp.y == -7.32f);
   }

}
```

```
public class BitmapRepositoryTest extends AndroidTestCase {
   public void testChangeMaxBitmapSize1() {
      BitmapManager bitmapManager = new BitmapManager();
      int width = bitmapManager.getMaxBitmapWidth();
      int height = bitmapManager.getMaxBitmapHeight();

      bitmapManager.setMaxBitmapWidth(width + 100);
      bitmapManager.setMaxBitmapHeight(height + 100);

      assertEquals(width + 100, bitmapManager.getMaxBitmapWidth());
      assertEquals(height + 100, bitmapManager.getMaxBitmapWidth());
   }
   public void testChangeMaxBitmapSize2() {
      BitmapManager bitmapManager = new BitmapManager();
      try {
         bitmapManager.setMaxBitmapWidth(-1);
         assertFalse(true);
      } catch(IllegalArgumentException e) {
         assertFalse(false);
      }
   }
}
```

4.8.1. Testes de interface do utilizador

Foram feitos alguns testes ao interface com o utilizador, de maneira aumentar a confiança do sistema de introdução de *input*. Estes testes foram desenvolvidos com o objetivo de testar o módulo descrito na secção **Implementação do processador de toque**, numa situação realista. Estes testes são possíveis graças à framework de testes do Android, que possibilita a simulação de toques reais na aplicação, dai estes testes não serem também considerados testes unitários.

Alguns destes testes podem ser visualizados no seguinte excerto de código:

```
public void testFinger1() {
   Instrumentation inst = getInstrumentation();
   try {
      //Needs to connect to the server!
      Thread.sleep(1000);
   } catch (InterruptedException e) {}
   MotionEvent motionEvent = MotionEvent.obtain(SystemClock.uptimeMillis(),
         SystemClock.uptimeMillis(), MotionEvent.ACTION_DOWN, 100, 100, 0);
   inst.sendPointerSync(motionEvent);

   assertEquals(GestureState.ONE_FINGER, gestureHandler.getGestureState());

   motionEvent.recycle();
}

public void testTwoFinger1() {
   Instrumentation inst = getInstrumentation();
   try {
      //Needs to connect to the server!
      Thread.sleep(1000);
   } catch (InterruptedException e) {}
   MotionEvent motionEvent = MotionEvent.obtain(SystemClock.uptimeMillis(),
         SystemClock.uptimeMillis(), MotionEvent.ACTION_DOWN, 100, 100, 0);
   inst.sendPointerSync(motionEvent);

   motionEvent.recycle();
   motionEvent = MotionEvent.obtain(SystemClock.uptimeMillis(), SystemClock.uptimeMillis(),
         MotionEvent.ACTION_DOWN, 200, 100, 0);
   inst.sendPointerSync(motionEvent);

   assertEquals(GestureState.TWO_FINGERS, gestureHandler.getGestureState());

   motionEvent.recycle();
}
```

4.8.2. Testes de sistema (end-to-end)

Devido à aplicação ter a necessidade de fazer comunicações entre vários dispositivos e à complexidade de criação de testes que permitam a interação entre dispositivos, os testes de sistema foram feitos manualmente pelos *testers* da WIT Software, e pelo estagiário.

Os testes efetuados foram executados no fim de cada *sprint* de forma a poder validar o funcionamento das novas funcionalidades, e para garantir que as funcionalidades anteriormente implementadas ainda estavam a funcionar devidamente.

4.9. O produto final

Esta secção dá a conhecer a aplicação no seu estado final. A secção foca-se em apresentar visualmente a integração da aplicação com o RCS Suite, e dar a conhecer o aspeto visual da aplicação desenvolvida. No Anexo D encontra-se o manual do utilizador onde se pode conhecer a aplicação a um nível do utilizador.

4.9.1. Integração com o RCS Suite

Em termos de interface com o utilizador a integração da aplicação com o RCS Suite passou pela criação de uma nova opção para desenho colaborativo. Esta opção foi adicionada ao menu de partilha do RCS Suite, como evidencia a imagem abaixo.

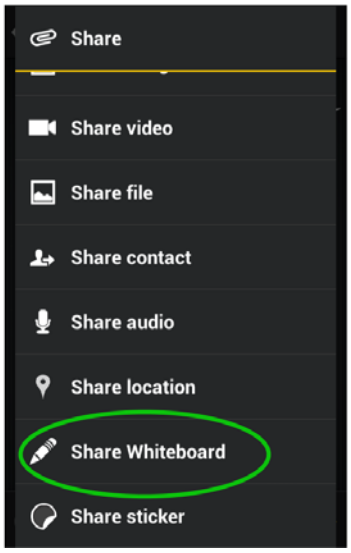


Figura 34 Menu de partilha da aplicação RCS Suite

Após a seleção da opção de partilha de Whiteboard é feito um pedido ao utilizador de um nome para a sessão de modo a esta ser facilmente identificada no *chat*. Esta identificação é importante porque podem existir várias sessões distintas entre dois utilizadores.

Figura 35 Inserção de nome para sessão (à esquerda) e objetos de sessão de desenho colaborativo no chat do RCS Suit (à direita)

Quando uma sessão é atualizada por outro utilizador o objeto do *chat* fica da cor azul até as mudanças serem visualizadas, nesse momento o objeto do *chat* volta para branco.

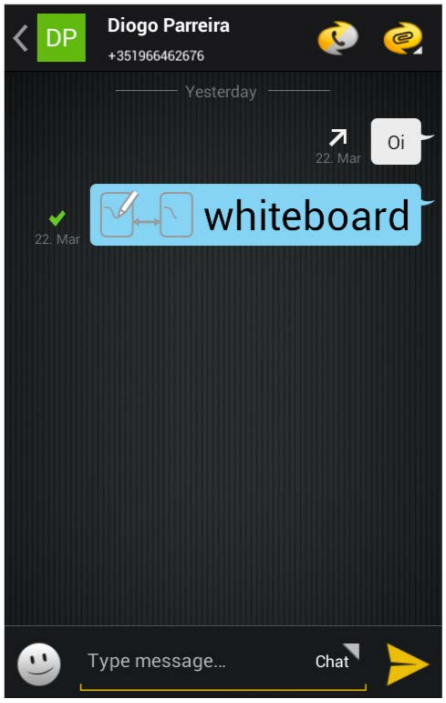


Figura 36 Objeto de *chat* com cor diferente devido a atualização do seu conteúdo

4.9.1. Visual da aplicação

Visto a aplicação ser muito visual, de seguida apresentam-se algumas imagens da aplicação em funcionamento para a dar a conhecer e os seus componentes.

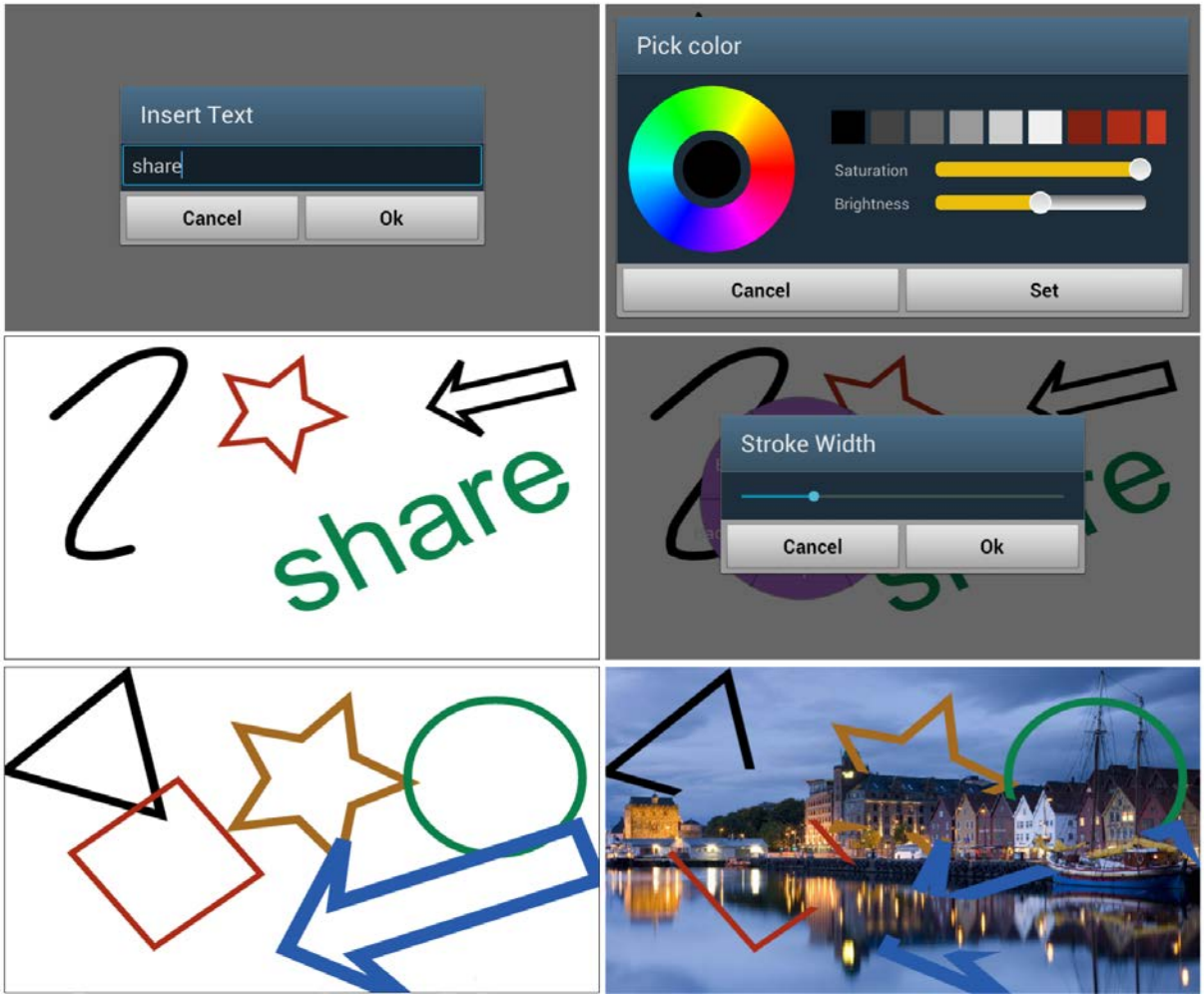


Figura 37 Várias imagens da aplicação

5. Conclusões

Tal como anteriormente referido, o principal objetivo deste estágio era a elaboração de uma ferramenta de desenho colaborativo a integrar no produto RCS Suite da WIT Software.

Para o efeito, analisaram-se muitas das ferramentas de desenho colaborativo disponíveis no mercado o que permitiu concluir que estas não estavam a ser utilizadas no seu potencial máximo. De facto, uma das maiores fraquezas detetadas nestas aplicações é que a grande maioria é feita para serem utilizadas num computador. Para o mundo empresarial estas condicionantes são muito críticas, pelo que se concluiu que a aplicação a desenvolver seria um bom investimento.

Na verdade, a nova aplicação veio acrescentar uma nova funcionalidade que não é oferecida por nenhum dos concorrentes da WIT Software, permitindo manter e até impulsionar a posição de destaque em que a empresa já se encontra e, consequentemente, chamar ainda mais a atenção de possíveis novos clientes.

Como balanço final, pode considerar-se que as metas foram atingidas e que neste momento a WIT Software dispõe da primeira iteração da ferramenta completa, pronta para ser apresentada aos clientes. Pode dizer-se que o trabalho decorreu com normalidade, sendo que o único entrave foi a falta de experiência do estagiário na área de trabalho da empresa e a complexidade do projeto RCS Suite. No entanto, com dedicação e perseverança, e com o apoio da própria empresa, as dificuldades foram ultrapassadas e o projeto chegou a bom porto.

Espera-se que a curto prazo comecem a haver clientes interessados neste módulo, sendo previsível que tenham que ser feitas adaptações às necessidades destes.

Evidentemente que há aspetos da aplicação a melhorar. Por exemplo, apesar da arquitetura da aplicação ter sido a área desenhada mais cuidadosamente, com o desenrolar do estágio e a necessidade de mudança de requisitos, esta sofreu vários processos de *refactoring* que levaram a que no final se concluísse que a aplicação poderia ter ficado ainda mais eficiente. Por outro lado, como o objetivo principal era criar uma aplicação completa e funcional, existem módulos que poderiam ser mais eficientes. Um desses módulos é, por exemplo, o protocolo de comunicação. Com efeito, o JSON não é a ferramenta para *encoding* de informação que a torna o mais compacta possível. Concluindo, estes aspetos de eficiência são algo que deve ser considerado para futuras melhorias.

Durante o desenvolvimento teve-se sempre em conta os requisitos não-funcionais e os atributos de qualidade da aplicação. Em relação aos requisitos não-funcionais todos foram atingidos e validados, no entanto alguns dos requisitos como o de ter um bom interface com utilizador poderia ser melhorados ainda mais. Os atributos de qualidade vão de encontro aos requisitos não-funcionais, e poderiam também ser melhorados nos mesmos aspetos.

Infelizmente acabou por não ser possível colocar a ferramenta de desenho colaborativo a funcionar com mais de 2 utilizadores em simultâneo devido a problemas com a API de Comunicação da aplicação RCS Suite, no entanto a aplicação está completamente preparada para esta funcionalidade, pelo que apenas seria necessário corrigir a situação para a colocar a funcionar.

Para terminar, e como reflexão pessoal, o estagiário gostaria de referir que a experiência de estagiar numa empresa de topo como a WIT Software foi única e extremamente enriquecedora, quer a nível profissional, quer a nível pessoal. De facto, esta oportunidade de contacto direto com a realidade do mundo do trabalho e de poder aprender com profissionais experientes, possibilitou ao aluno o ganho de conhecimentos através da vivência de situações práticas, reais, mas também o ganho de autonomia.

6. Referências Bibliográficas

[1] “WIT Software Products”, WIT Software, [Online]. Disponível em: https://www.wit-software.com/products/ [Acedido em Julho 2014].

[2] “joyn Blackbird Product Definition Document”, GSMA, [Online]. Disponível em: http://www.gsma.com/network2020/wp-content/uploads/2014/01/joyn-Blackbird-PDD-V3-0.pdf [Acedido em Fevereiro 2014].

[3] “Google Drawings”, Google Inc., [Online]. Disponível em: https://drive.google.com/. [Acedido em Fevereiro 2014].

[4] “Scribblar”, scribblar.com, [Online]. Disponível em: https://www.scribblar.com/. [Acedido em Fevereiro 2014].

[5] “CoSketch.com”, CoSketch.com, [Online]. Disponível em: http://www.cosketch.com/. [Acedido em Fevereiro 2014].

[6] “FlockDraw”, Flockdraw, [Online]. Disponível em: http://flockdraw.com/. [Acedido em Fevereiro 2014].

[7] “GroupBoard”, Group Technologies Inc., [Online]. Disponível em: http://www.groupboard.com/products/. [Acedido em Fevereiro 2014].

[8] “Mighty Meeting”, MightyMeeting Inc., [Online]. Disponível em: http://www.mightymeeting.com/. [Acedido em Fevereiro 2014].

[9] “Chalkboard”, Signature Creative Inc, [Online]. Disponível em: http://www.thechalkapp.com/. [Acedido em Fevereiro 2014].

[10] Whiteboard: Collaborative Draw”, Green Gar, [Online]. Disponível em: http://www.greengar.com/apps/whiteboard/. [Acedido em Fevereiro 2014].

[11] “SyncSpace Shared Whiteboard”, The Infinite Kind, [Online]. Disponível em: http://infinitekind.com/syncspace. [Acedido em Fevereiro 2014].

[12] “Shared Board”, Gábor Csomák, [Online]. Disponível em: http://www.sharedboard.net/. [Acedido em Fevereiro 2014].

[13] “Lucidchart”, Lucid Software Inc., [Online]. Disponível em: https://www.lucidchart.com/. [Acedido em Fevereiro 2014].

[14] “Concept Board”, Conceptboard, [Online]. Disponível em: http://conceptboard.com/. [Acedido em Fevereiro 2014].

[15] “JIRA”, Atlassian, [Online]. Disponível em: https://www.assembla.com/ [Acedido em Fevereiro 2014].

[16] “Assembla”, Assembla Inc., [Online]. Disponível em: https://www.atlassian.com/software/jira [Acedido em Fevereiro 2014].

[17] “Whiteboard: Collaborative Draw in App Store”, Apple Inc., [Online]. Disponível em: https://itunes.apple.com/br/app/whiteboard-collaborative-drawing/id321986541 [Acedido em Fevereiro 2014].

[18] “Redmine”, Jean-Philippe Lang, [Online]. Disponível em: http://www.redmine.org/ [Acedido em Novembro 2014].

[19] “Subversion”, Apache, [Online]. Disponível em: http://subversion.apache.org/ [Acedido em Fevereiro 2014].

[20] “Jenkins”, Jenkins, [Online]. Disponível em: http://jenkins-ci.org/ [Acedido em Fevereiro 2014].

[21] “Apache Ant”, Apache, [Online]. Disponível em: http://ant.apache.org/ [Acedido em Fevereiro 2014].

[22] “Java”, Oracle, [Online]. Disponível em: http://www.oracle.com/technetwork/java/index.html [Acedido em Novembro 2014].

[23] “Eclipse IDE”, Eclipse Foundation, [Online]. Disponível em: https://eclipse.org/ [Acedido em Novembro 2014].

[24] “Android SDK”, Google Inc., [Online]. Disponível em: http://developer.android.com/sdk/index.html [Acedido em Fevereiro 2014].

[25] “Modelio”, Modeliosoft, [Online]. Disponível em: https://www.modelio.org/ [Acedido em Fevereiro 2014].

[26] “Visio”, Microsoft, [Online]. Disponível em: https://products.office.com/pt-pt/visio/flowchart-software [Acedido em Abril 2014].

[27] “JUnit”, JUnit, [Online]. Disponível em: http://junit.org/ [Acedido em Março 2014].

[28] “Mockito”, Szczepan Faber, [Online]. Disponível em: http://mockito.org/ [Acedido em Março 2014].

[29] “Android JUnit Framework”, Google Inc., [Online]. Disponível em: http://developer.android.com/tools/testing/testing_android.html [Acedido em Março 2014].

[30] “Robotium”, Renas Reda, [Online]. Disponível em: http://www.robotium.org/ [Acedido em Março 2014].

[31] “Roboletric”, Roboletric, [Online]. Disponível em: http://robolectric.org/ [Acedido em Março 2014].

[32] “Kryonet”, Esoteric Software, [Online]. Disponível em: https://github.com/EsotericSoftware/kryonet [Acedido em Fevereiro 2014].

[33] “Red Notebook”, Jendrik Seipp, [Online]. Disponível em: http://rednotebook.sourceforge.net/ [Acedido em Março 2014].

[34] “ToDoList”, ToDoList, [Online]. Disponível em: http://www.codeproject.com/Articles/5371/ToDoList-An-effective-and-flexible-way-to-keep-on [Acedido em Fevereiro 2014].

[35] “TeamGantt”, TeamGantt, [Online]. Disponível em: https://teamgantt.com/ [Acedido em Fevereiro 2014].

[36] “SIP”, [Online]. Disponível em: https://www.ietf.org/rfc/rfc3261.txt [Acedido em Janeiro 2014].

[37] “MSRP”, [Online]. Disponível em: https://tools.ietf.org/html/rfc4975 [Acedido em Janeiro 2014].

[38] “JSON”, [Online]. Disponível em: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf [Acedido em Fevereiro 2014].

[39] Simon Brown (2014). Software Architecture for Developers, Paris (em Inglês).

[40] “Radial Menu Widget”, [Online]. Disponível em: https://code.google.com/p/radial-menu-widget/ [Acedido em Março 2014].

Anexo A

Tabela com informação de todas as ferramentas de desenho colaborativo.

Features \ Tools	Google Drawings	Scribblar	CoSketch.com	FlockDraw	GroupBoard	Mighty Meeting
Link	ogle.com/drive/apps.ht	ttp://www.scribblar.com	http://cosketch.com/	http://flockdraw.com/	ttp://www.groupboard.com	gle.com/store/apps/detai
Free Draw	YES	YES	YES	YES	YES	YES
Lines	YES	YES	YES	YES	YES	NO
Simple shapes	YES	YES	YES	NO	YES	NO
Complex shapes	YES	NO	NO	NO	NO	NO
Change color	YES	YES	YES	YES	YES	YES
Erase (Draws / Objects)	YES	YES	YES	YES	YES	YES
Change pen size	YES	YES	YES (3 options)	YES	YES	YES
Write text	YES	YES	YES	YES	YES	NO
Import Images	YES	YES	YES	NO	BKG ONLY	PowerPoint & PDF
Continue later	YES,cloud	YES,cloud	NO	NO	YES,cloud	YES,cloud
Resize canvas	YES	2 PAGES	NO	NO	NO	Multiple screens
Save image	YES	YES, not intuitive	YES	YES, not intuitive	YES, not intuitive	NO
Users drawing at the same time	50	2 FREE, 50 PAID	2+	10	5 FREE, 25 PAID	2 FREE
Users using the tool/downloads	N/A	N/A	N/A	N/A	100,000+	10k+ A
Free/Paid	FREE	FREE for 2 users	FREE	EE, PAID for iOS 2.	FREE for 5 users	FREE&PAID
Available for Android	NO	NO	NO	NO	YES	YES
Available for iOS	NO	NO	NO	YES	YES	YES
Available for Web	YES	YES	YES	YES	YES	YES, mighty.ws
Available for PC or MAC	NO	NO	NO	NO	NO	NO
Clear canvas (reset)	CTRL+A; DEL	YES	YES	NO	YES	YES
Highlighting tools	YES, text only	YES	NO	NO	NO	NO
Zoom	YES	NO	NO	NO	YES, MOBILE	NO
Grid	YES	YES	IMAGE AS BKG	NO	NO	YES
Share image	YES,Link	NO	NO	YES.face,twit,link	NO	NO
Share invite to join	ount; Link; face;tw	Email;Link	LINK	LINK	LINK	EMAIL, LINK
Plugins required (Java, Flash...)	NO	Flash	NO	Flash	NO	NO
Required Account	YES, to edit	YES to open room	NO	NO	NO, only passw	YES
Size of canvas (0-5)	5	4	3	3	3	5
Responsiveness (0-5)	5	3 (mouse release)	3 (mouse release)	3 (slow mouse)	3 (mouse release)	4
Easy interface (0-5)	4	4	3	3	4	4
Steps to start using	3-ACC,Login,doc	3-ACC,Login,doc	1-Create room	1-Create room	2-create;form fill	2-meeting, sketchbook
Other Utilities	Undo/Redo; Tables; G.Drive integration; chat	Wolfram Alpha paid; Undo/Redo; Chat&Voz; API available	Google MAPS; Chat; API for embed version; Stamps&backgrounds	Chat	Chat;Embed for paid version;password protected	Google hangouts integration; PPT presentations;Undo

Features \ Tools	Chalkboard	Whiteboard: Colla	SyncSpace Share	Shared Board	Lucid Chart	Concept Board
Link	/app/detail/?id=com.zian	http://app/detail/?id=co	http://infinitekind.com/	http://www.sharedboard.net	http://www.lucidchart.com	http://conceptboard.com
Free Draw	YES	YES	YES	YES	NO	YES
Lines	NO	NO	NO	NO	YES	YES
Simple shapes	NO	NO	NO	NO	YES	YES
Complex shapes	NO	NO	NO	NO	YES	NO
Change color	YES	YES	YES	YES	YES	YES
Erase (Draws / Objects)	YES	YES	YES	YES	YES	YES
Change pen size	YES	YES	YES	YES	YES	YES
Write text	NO	NO	YES	YES	YES	YES
Import Images	YES	YES	NO	YES	YES	YES
Continue later	NO	NO	YES	NO	YES	YES
Resize canvas	YES	NO	YES	NO	YES	YES
Save image	NO	YES	YES	YES	YES	NO
Users drawing at the same time	2	2	2+	2+	FREEwhile in beta	5 FREE, 10+
Users using the tool/downloads	1k+A	7m+ ios, 100k+ A	10k+ A	1k+ A	N/A	N/A
Free/Paid	FREE	FREE&PAID, 1$	FREE	FREE	FREE&PAID	FREE&PAID
Available for Android	YES	YES	YES	YES	NO	NO
Available for iOS	NO	YES	YES	YES	NO	NO
Available for Web	NO	NO	VIEW ONLY	NO	YES	YES
Available for PC or MAC	NO	NO	NO	YES	NO	NO
Clear canvas (reset)	YES	YES	DELETE ONLY	YES	CTRL+A; DEL	NO
Highlighting tools	YES	YES,transp.	NO	NO	YES	YES
Zoom	YES	NO	YES	YES	YES	YES
Grid	NO	NO	NO	NO	YES	NO
Share image	Face,picassa,twit	FACE	Face,picassa,twit	NO	YES	NO
Share invite to join	Automatic	Local WI-FI	LINK	Automatic (groups)	EMAIL,LINK	Em,Li,face,tw,G+
Plugins required (Java, Flash...)	NO	NO	NO	Adobe Air/Flash	NO	NO
Required Account	Name and picture	NO	NO	NO	YES	YES
Size of canvas (0-5)	5	2	5	4	5	5
Responsiveness (0-5)	2	5	5	4	5	5
Easy interface (0-5)	5	5	4	3	5	4
Steps to start using	2-install,use	2,install,use	2,install,use	3-down,install,exec	2-register,use	2
Other Utilities	Undo-Redo	Layers			Embed; diagrams; publish in web; color themes	Comments; sticky notes; insert pdf and other files; Undo-Redo; chat&video

Anexo B

Digrama de classes:

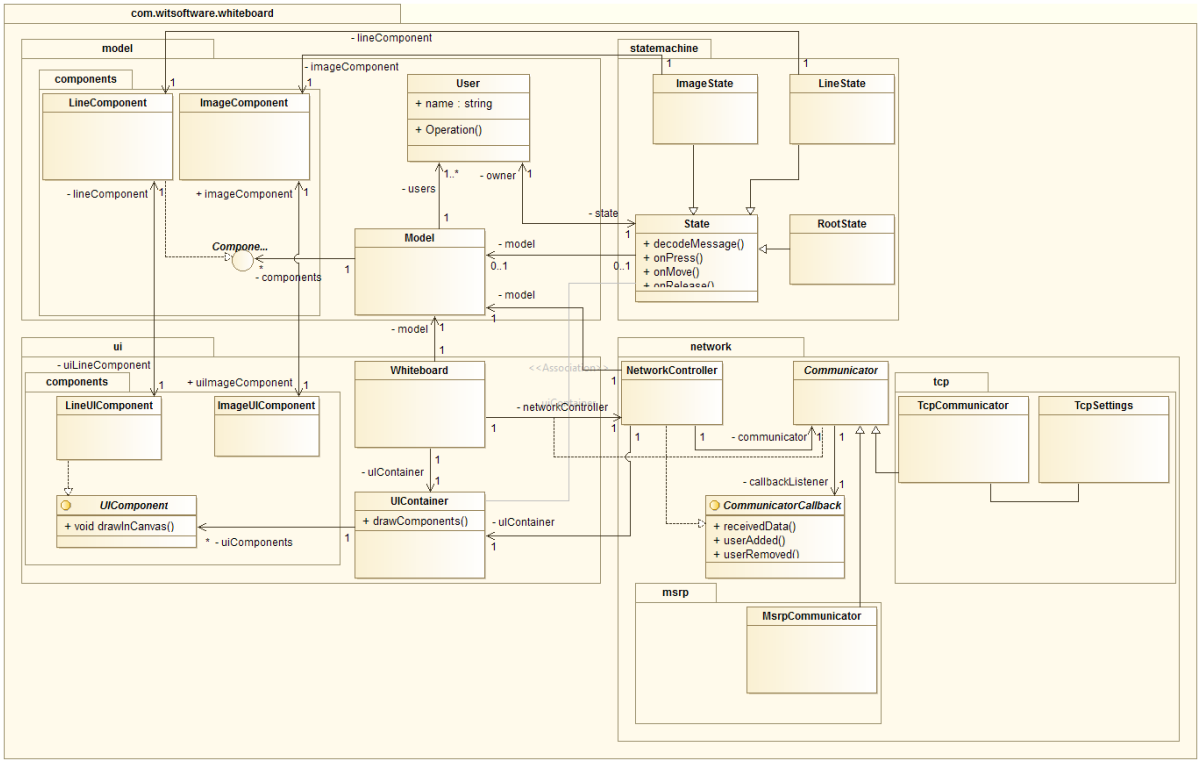


Figura 38 Diagrama de classes da primeira iteração da aplicação

Diagrama de sequência do processo de inicialização da sessão de desenho colaborativo:

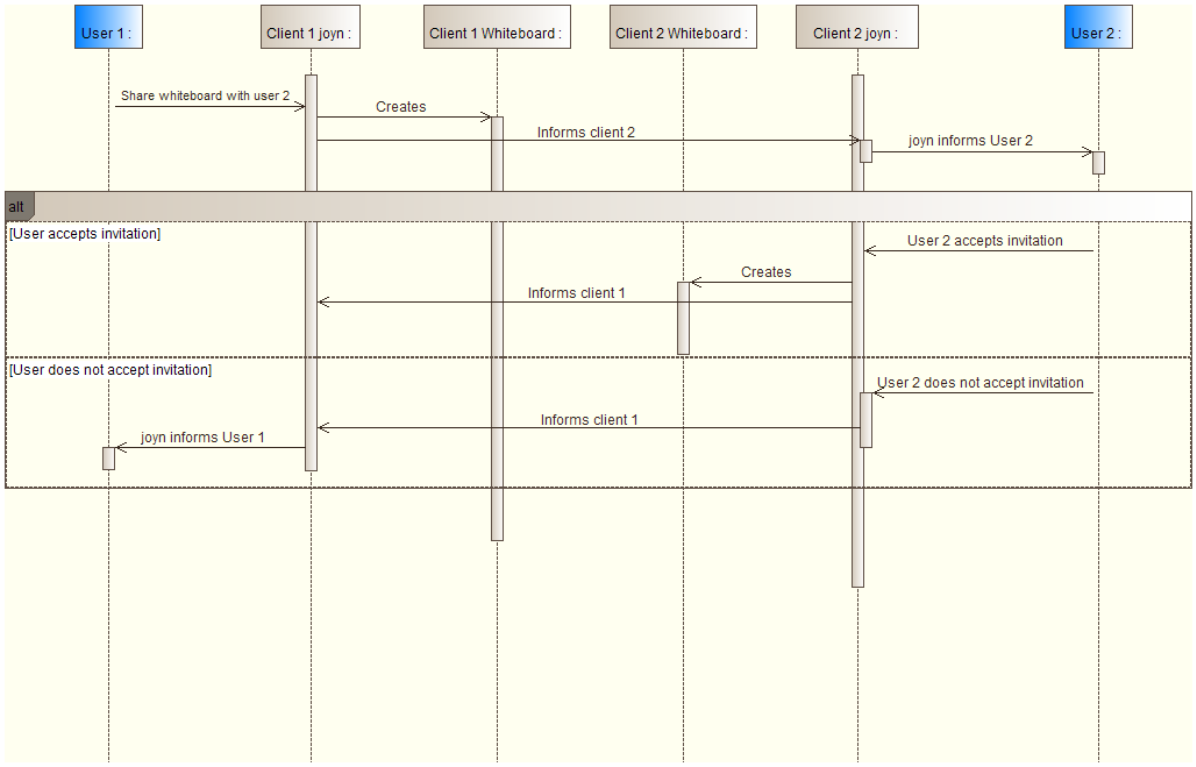


Figura 39 Diagrama de sequência do processo de inicialização da sessão

Anexo C

Código da classe Java “State”:

```
/**<p>Abstract base state, should be implemented by the multiple controllers of each kind of visual component,
 * this controllers can receive messages from any user and should be able to know the message purpose. </p>
 * <p> There are multiple functions to receive data that can be override by the component controllers. This
 * functions are separated in:
 * <li>Decoder function, should decode JSON objects received from other users (expect multiple messages,
 * should forward messages for specific functions dedicated to handle the task, like create an object
 * or resizing it, depends on the kind of object);</li>
 * <li>Raw input message handlers (to be used only for the interface listeners).</li></p>
 */
public abstract class State {

   protected Model mModel;
   protected UIContainer mUIContainer;
   protected NetworkController mNetworkController;
   protected User mOwner;
   protected FixedCamera camera;
   protected boolean locked = false;
   protected long lockedTime;

   public State(Model model, UIContainer uiContainer, NetworkController networkController, User user, FixedCamera camera) {
       this.mModel = model;
       this.mUIContainer = uiContainer;
       this.mNetworkController = networkController;
       this.mOwner = user;
       this.camera = camera;
   }

   /**
    * Decodes messages originating from other users, messages like creating a component or moving it.
    * @param message JSONObject with the received message
    */
   public abstract void decodeMessage(JSONObject message);

   //Raw input handlers functions

   /**
    * Used for the state to know a finger started touching the screen
    * @param point position of the finger in camera coordinates
    */
   public void onPress(Vector2 point) {}

   /**
    * Used for the state to know the finger moved
    * @param point position of the finger in camera coordinates
    */
   public void onMove(Vector2 point) {}

   /**
    * Used to inform that one pointer was release
    */
   public void onRelease() {}

   /**
    * Used for the state to know the finger was released
    * @param point position of the finger in camera coordinates
    */
   public void onRelease(Vector2 point) {}

   public void onResizeRotateStart() {}
```

```
    /**
     * Used to inform that two pointers were moved for resize/rotate
     */
    public void onResizeRotate(float scaleFactor, float rotation) {}

    public void onChangeText(String text) {}

    /**
     * Used to notify that this state is now active
     * @throws LockedStateException
     */
    public void enterState(User user) throws LockedStateException {
        if(user == null) {
            throw new NullPointerException("user is null");
        }
        if(locked && mOwner != user) {
            throw new LockedStateException();
        }
        mOwner = user;
        locked = true;
        lockedTime = System.currentTimeMillis();
    }

    /**
     * Used to notify that this state is now deactivated. Might be used to end an operation
     * @throws LockedStateException
     */
    public final void exitState(User user) throws LockedStateException {
        if(user == null) {
            throw new NullPointerException("user is null");
        }
        if(locked && mOwner != user) {
            throw new LockedStateException();
        }
        onExitState();
        locked = false;
    }

    public void onExitState() {}

    public boolean isLockedForUser(User user) {
        if(locked) {
            if(mOwner == user) {
                return false;
            }
            return true;
        }
        return false;
    }

    public boolean isLocked() {
        return locked;
    }

    public long getLockedTime() {
        return lockedTime;
    }

    public boolean lockState(User user, long time) {
        if(!locked || time < lockedTime) {
            mOwner.setState(new RootState(mModel, mUIContainer, mNetworkController, mOwner,
camera));
            locked = true;
            lockedTime = time;
            mOwner = user;
            return true;
        }
        return false;
    }
}
```

Código da classe Java “FixedCamera”:

```
public class FixedCamera {

   private static final float DEFAULT_WIDTH = 300;
   private static final float DEFAULT_HEIGHT = 480;
   private static float RATIO = DEFAULT_WIDTH / DEFAULT_HEIGHT;

   /** Size of the fictitious camera */
   final public int CAMERA_SIZE = 1000;

   public float ratioWidth, ratioHeight, originalWidth, originalHeight, scaleFactor;
   private int realWidth;
   private int realHeight;

   public FixedCamera(int width, int height) {
      recalculate(width, height, width, height);
   }

   public void recalculate(int width, int height, int realWidth, int realHeight) {
      Vector2 idealSize = getMaxSizeOfCameraInsideScreen(width, height);
      originalWidth = idealSize.x;
      originalHeight = idealSize.y;
      ratioWidth = idealSize.x/CAMERA_SIZE;
      ratioHeight = idealSize.y/CAMERA_SIZE;
      scaleFactor = idealSize.x / DEFAULT_WIDTH;
      this.realWidth = realWidth;
      this.realHeight = realHeight;
   }

   public static Vector2 getMaxSizeOfCameraInsideScreen(float maxWidth, float maxHeight) {

      if(maxWidth*RATIO > maxHeight) {
         return new Vector2(maxHeight/RATIO, maxHeight);
      } else {
         return new Vector2(maxWidth, maxWidth*RATIO);
      }
   }

   public int unprojectX(float x) {
      return (int) (x/ratioWidth);
   }

   public int unprojectY(float y) {
      return (int) (y/ratioHeight);
   }

   public float projectX(int x) {
      return x*ratioWidth;
   }

   public float projectY(int y) {
      return y*ratioHeight;
   }

   /**
    * Changes coordinates from screen to camera
    * @param point original screen coordinates
    * @return the same vector with the changed coordinates
    */
   public Vector2 unproject(Vector2 point) {
      point.x = unprojectX(point.x);
      point.y = unprojectY(point.y);
      return point;
   }
```

```
    /**
     * Changes coordinates from camera to screen
     * @param point original camera coordinates
     * @return the same vector with the changed coordinates
     */
    public Vector2 project(Vector2 point) {
        point.x = projectX((int) point.x);
        point.y = projectY((int) point.y);
        return point;
    }

    /**
     * Changes coordinates from real screen to screen then to camera
     * @param point original screen coordinates
     * @return the same vector with the changed coordinates
     */
    public Vector2 unprojectFromRealScreen(Vector2 point) {

        point.x = point.x*originalWidth/realWidth;
        point.y = point.y*originalHeight/realHeight;

        point.x = unprojectX(point.x);
        point.y = unprojectY(point.y);
        return point;
    }

    public int scaledToRealX(float x) {
        return (int) (x*realWidth/originalWidth);
    }
    public int scaledToRealY(float y) {
        return (int) (y*realHeight/originalHeight);
    }

}
```

Anexo D

Manual do utilizador

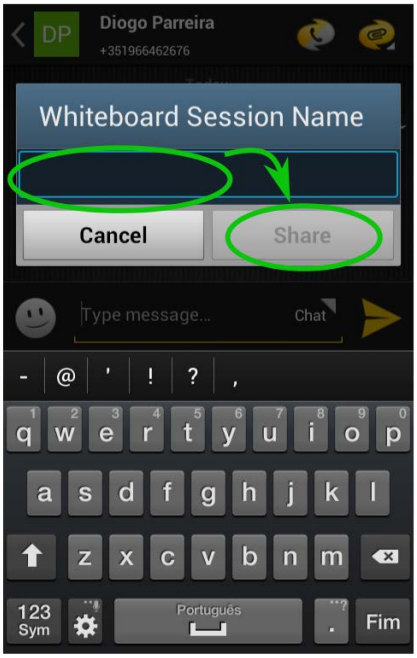
Para utilizar a aplicação Whiteboard basta ter a versão da aplicação RCS Suite da WIT Software com a aplicação Whiteboard integrada instalada num dispositivo com o sistema operativo Android 2.3 ou superior. Também é necessário ter um ou mais contactos com a mesma aplicação instalada.

Partilha de desenho colaborativo

Abrir a aplicação e selecionar o contacto com o qual se pretende partilhar uma sessão de desenho colaborativo, selecionar a opção de “Share”, e selecionar a opção “Share Whiteboard” como está apresentado nas imagens abaixo.

De seguida abre-se uma janela onde se deve inserir o nome para a sessão – o qual será posteriormente visível na janela de *chat* do utilizador e do seu contacto que serve para identificar cada sessão.

Insira o nome que pretendido, e pressione “Share”. Estas operações podem ser visualizada na imagem abaixo.



Após esta operação ser concluída a sessão será aberta, e um ecrã em branco ficará visível onde será possível desenha, inserir objetos e imagens como fundo de ecrã.

O contacto com o qual se partilhou a sessão de desenho colaborativo será notificado imediatamente se este se encontrar *online* na aplicação RCS Suite, caso contrário este será notificado após se ligar novamente.

Abrir sessão já existente

Para abrir uma sessão já existente, apenas é necessário abrir a aplicação e selecionar o contacto com o qual a sessão de desenho colaborativo que se pretende abrir está criada e selecionar a opção de “Chat”. Quando o chat do contacto estiver aberto basta selecionar o objeto de chat com o nome da sessão que se pretende abrir, e a sessão será aberta. É possível ver esta sequência nas imagens abaixo.

Menu de opções

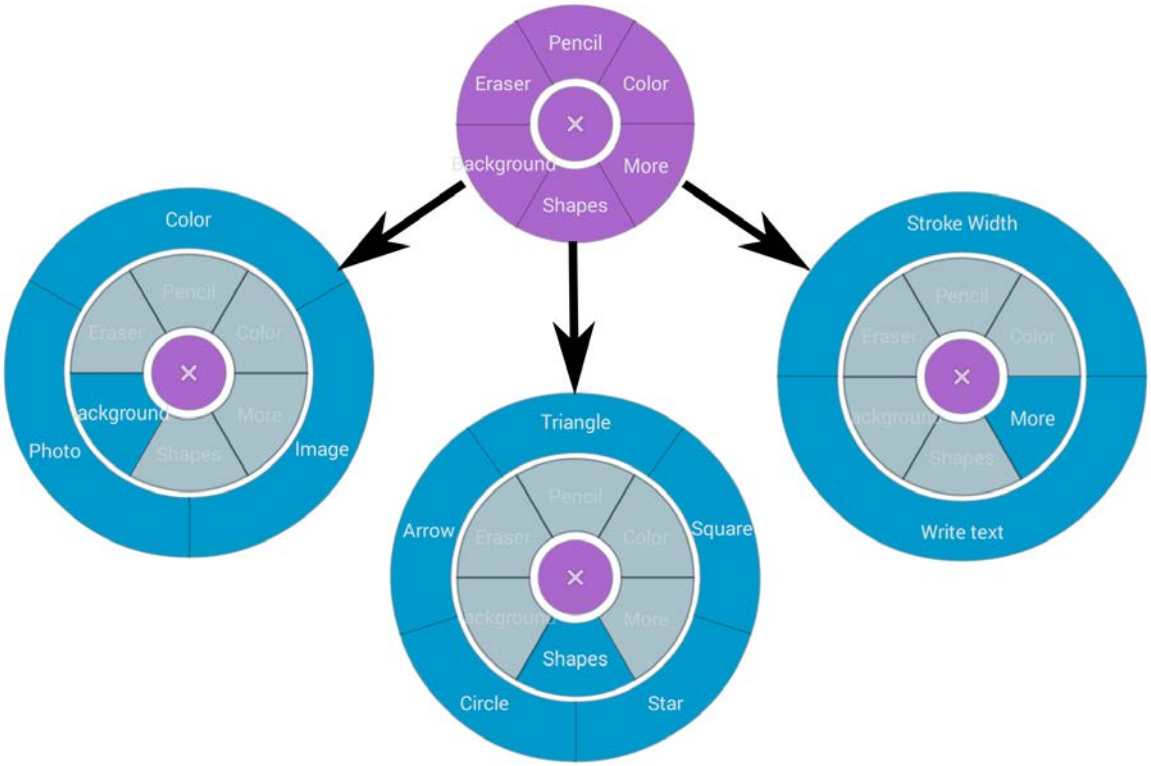
O menu de opções da aplicação disponibiliza uma série de opções para modificar o desenho. Para abrir o menu basta ter a aplicação de desenho colaborativo aberta e pressionar no ecrã por alguns segundos sem mover o dedo. Após este ser aberto é possível ver algumas das opções disponíveis ao utilizador, sendo que alguns dos botões deste menu abrem submenus com mais opções.

A árvore de opções é a seguinte:

- Forma, permite inserir objetos no desenho:
  - Circulo;
  - Seta;
  - Triângulo;
  - Quadrado;
  - Estrela.
- Background, permite modificar o *background* do desenho:
  - Fotografia (tirada da câmara do dispositivo) como *background*;
  - Cor como *background*;
  - Imagem (armazenada no dispositivo) como *background*.
- Borracha, permite apagar qualquer parte do desenho apenas arrastando o dedo;
- Desenho com toque, permite desenhar ao arrastar o dedo;
- Selecionar cor, permite definir a cor que vai ser utilizada ao desenhar com o dedo, pelas formas e pelo texto inserido;
- Mais opções:
  - Escrever texto, permite inserir texto no desenho;

- Mudar a espessura das linhas que são desenhadas.

É possível visualizar a árvore de opções do menu na seguinte imagem.



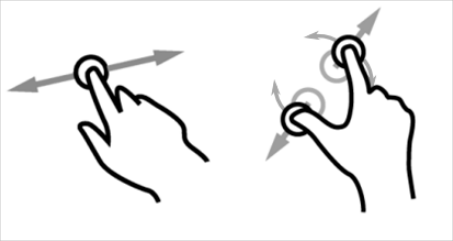
Informações adicionais

Mover, rodar e escalar

É possível mover rodar e escalar formas e texto. Para o fazer insira um objeto e a partir desse momento até inserir outro objeto ou selecionar outra opção (como apagar ou desenhar), este objeto pode ser modificado.

- Para mover apenas se arrasta um dedo no ecrã.
- Para rodar e escalar toque o ecrã com dois dedos e afaste ou aproxime os dedos para escalar o objeto, ou rode ambos em relação ao ecrã para rodar o objeto.

Estas opções estão ilustradas na imagem seguinte (mover à esquerda, rodar e escalar à direita).



Notificação de desenho

Quando se recebe uma atualização do desenho de uma sessão de desenho colaborativo e não se tem essa sessão aberta o objeto que representa a sessão de desenho que está no *chat* fica azul em vez de branco de forma a notificar o utilizador que existem modificações ao desenho que ainda não foram visualizadas. Este comportamento é visível na imagem abaixo.