Mestrado em Engenharia Informática
Estágio
Relatório de Estágio

Plataforma de download de Skins UI para Apps Mobile

Nuno André Fontes Vinhas
nvinhas@student.dei.uc.pt

Orientador do DEI:
Prof. Dr. Pedro Furtado

Orientador da Wit-Software, S.A.:
Eng. Rui Gil

Data: 29 de Junho de 2014

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

As operadoras de telecomunicações estão a perder um elevado número de receitas, já que os seus clientes começam a optar por soluções Over-The-Top (OTT), como o Skype, WhatsApp, Facebook ou Twitter, que oferecem melhores soluções a um custo reduzido. De forma a combater esta tendência, a Global System for Mobile Communications Association (GSMA) criou o Rich Comunication Suite (RCS) que consiste numa especificação de um conjunto de serviços - Voice Over IP (VoIP), Instant Message (IM), File Transfer (FT), entre outros - interoperáveis entre as várias redes das operadoras de telecomunicações. Várias soluções foram colocadas no mercado por diferentes operadoras, todas oferecendo um cliente de forma gratuita.

Após garantirem uma sólida base de utilizadores, as operadoras vêm nestas soluções uma oportunidade de negócio. O modelo de negócio incide essencialmente na disponibilização de conteúdos de valor acrescentado aos seus utilizadores. A venda de Skins e de Stickers, por exemplo, está a ter um enorme sucesso nas aplicações OTT, tendo estas reportado receitas significativas resultantes da venda destes conteúdos.

Neste relatório o autor apresenta as suas contribuições no desenvolvimento das funcionalidades de download e consumo de conteúdos de valor acrescentado nos clientes RCS da WIT Software e no cliente de *messaging* do grupo Vodafone intitulado de Message Plus.

Palavras-Chave

"Instant Messaging", "Rich Communication Suite", "Global System for Mobile Communications Association", "Skins", "In-App Purchases", "Stickers", "Short Message Service"

Índice

Capítulo 1 Introdução 12
1.1. Instituição 12
1.2. Contexto 12
1.3. Problema 13
1.4. Proposta e Objectivos 13
1.6. Estrutura do documento 14
Capítulo 2 Estado da Arte 15
2.1 Introdução 15
2.2 In-app purchases Android & iOS 15
2.2.1 História e definição de in-app purchases 15
2.2.2 iOS 15
2.2.3 Android 15
2.2.4 Conclusão 16
2.3 Descrição de Aplicações com Skins e/ou Stickers 16
2.3.1 Critérios de avaliação 16
2.3.2 Aplicações analisadas 17
2.3.3 Conclusão 18
Capítulo 3 Gestão do Projeto 21
3.1 Equipa WIT Mobile Communicator 21
3.2 Metodologia 21
3.1 Planeamento 22
3.1.1 Proposta inicial 22
3.1.2 Planeamento final 23
Capítulo 4 Arquitetura 27
4.1 Estrutura da aplicação 27
Capítulo 5 Trabalho Realizado 31
5.1 Desenvolvimento de uma Framework de Skins 31
5.1.1 Problema 31
5.1.2 Objectivos 31
5.1.3 Solução 31
5.1.3.1 *Components* 32

5.1.3.2 *Operations* 33
5.1.3.3 *Objects* 34
5.1.3.4 Conclusão acerca dos *Components, Operations* e *Objects* 34
5.1.4 Relação entre a instância de um componente e a respectiva configuração 34
5.1.5 Ficheiros de configuração em XML 35
5.1.6 *Assets* 37
5.1.7 Estrutura de ficheiros 37
5.1.8 Carregamento e aplicação de um determinado tema 38
5.1.9 Processo de personalização de um componente – Ponto de vista da Framework 38
5.1.10 iPhone e iPad 38
5.1.11 Mudar Skin sem desligar a aplicação 38
5.1.12 Herança 39
5.1.13 Processo de personalização de um componente – Ponto de vista do programador 39
5.1.14 Arquitetura 40
5.1.15 Potencialidades 41
5.1.16 Conclusão e Trabalho Futuro 41
5.2 Alterar a UI de Chat de forma a suportar Skins 43
5.2.1 Problema 43
5.2.2 Objectivos 43
5.2.3 Design e solução 43
5.2.4 Arquitetura do Chat 45
5.2.5 Desafios e Soluções 48
5.2.5.1 UITableViewCell e reutilização de *Views* 48
5.2.5.2 Calculo da altura de cada tipo de UITableViewCell 49
5.2.5.3 Auto Resising Masks 50
5.2.5.4 Reload Data versus Observer Pattern 52
5.2.5.5 Cache e o Core Data 53
5.2.6 Integração com a Framework de Skins 54
5.2.7 Conclusão 54
5.3 Desenvolvimento de uma loja que permite fazer download de conteúdos na solução RCS para iOS 57
5.3.1 Introdução 57
5.3.2 Objectivos 57
5.3.3 *User Interface* 57
5.3.4 Solução 58

5.3.5 *Models* 59
5.3.6 Download de Conteúdo 61
5.3.7 Framework de Skins 62
5.3.8 Stickers Manager 63
5.3.9 Arquitetura da loja 64
5.3.10 Conteúdo alojado da rede 65
5.3.11 Trabalho Futuro 66
5.3.12 Considerações finais 66
5.3.13 Conclusões 67
5.4 Solução de download de Stickers para o Message Plus 69
5.5 Optimização no carregamento dos Emojis e Emoticons no Message Plus 71
5.5.1 Introdução 71
5.5.2 Solução 71
5.5.3 Conclusões 72
5.6 Nova implementação da câmara do Message Plus e implementação da funcionalidade de Quick Share de vídeo 73
5.6.1 Introdução 73
5.6.2 Solução 73
5.6.3 Modos 76
5.6.3.1 Modo de Fotos 76
5.6.3.2 Modo de Vídeos 76
5.6.4 Modo Quick Share de Vídeos 77
5.6.5 Conclusões 80
Capítulo 6 Trabalho Futuro 81
Capítulo 7 Conclusões 83
Capítulo 8 Referências 85

Lista de Figuras

Figura 1 - Fases da metodologia implementada 22
Figura 2 - Diagrama de Gantt com os objetivos iniciais do estágio 23
Figura 3 - Diagrama de Gantt com os objetivos finais do estágio 26
Figura 4 - Arquitetura da UI da aplicação 28
Figura 5 - Relação entre a instância de um objeto e a especificação de UI 35
Figura 6 - Ficheiro de configuração em XML 36
Figura 7 - Configuração da UI de um UIButton 37
Figura 8 - Arquitetura da Framework de Skins 40
Figura 9 - Arquitetura do Chat 46
Figura 10 - Arquitetura das UITableViewCells do Chat 49
Figura 11 - Explicação de Auto Resizing Masks 52
Figura 12 – Temas Preto e Branco do WIT Mobile Communicator para iOS 54
Figura 13 - UI da loja de conteúdos 58
Figura 14 - Estrutura da loja 64
Figura 15 - Estrurua da solução da câmara 74
Figura 16 - Dinamismo da *input bar* de modo a suportar vários modos de Quick Share 77
Figura 17 - Estrutura da solução de Quick Share de Vídeo 79

Lista de Tabelas

Tabela 1 - Arquitetura do UI da aplicação 29
Tabela 2 - Componentes de UI disponibilizados pelo iOS 33
Tabela 3 - Propriedades de um objeto que herde de UIView possíveis de alterar 33
Tabela 4 - Tipos de *Object* 34
Tabela 5 - Descrição do ficheiro de configuração em XML 37
Tabela 6 - Descrição da configuração de um UIButton 37
Tabela 7 – Descrição da arquitetura do Chat 47
Tabela 8 - Calculo da altura de cada tipo de UITableViewCell 50
Tabela 9 - Propriedades do *ProductItem* 60
Tabela 10 - Estados de um produto 60
Tabela 11 - Critérios de escolha para cada estado dos produtos 61
Tabela 12 - Operações realizadas no download de um conteúdo 62
Tabela 13 - Estados de uma *task* de download de conteúdo 62
Tabela 14 - UI dos Stickers para a solução RCS para iOS 63
Tabela 15 - Descrição da estrutura da loja 65
Tabela 16 - Propriedades dos produtos presentes na lista de produtos disponíveis na loja 66
Tabela 17 - Propriedades do ficheiro de configuração JSON relativas à informação detalhada de cada produto 66
Tabela 18 - Estrutura do ficheiro binário de Emojis 72
Tabela 19 - Descrição da solução da câmara 75

Anexos

Anexo A: Vodafone Message Plus Stickers Test Cases Specification

Anexo B: Vodafone Message Plus Stickers Technical Specification

Anexo C: Especificações da UI e UX do Message Plus

Anexo D: Descrição da solução de download de Stickers no Message Plus

Acrónimos

API	Application Programming Interface
GSMA	Global System for Mobile Communications Association
IM	Instant Messaging
IMS	IP Multimedia Subsystem
MVC	Model View Controller
OTT	Over The Top
RCS	Rich Communication Suite
SMS	Short Message Service
UI	User Interface
VoIP	Voice over IP
XML	EXtensible Markup Language

Glossário

Chat	Comunicação através da internet que permite transmitir mensagens de texto.
Emoticon	Sequencia de caracteres – tipicamente uma combinação de letras, números e pontuação – que representam sentimentos, objetos ou ações.
Facade pattern	É uma design pattern que oferece uma interface simplificada para um conjunto assinalável de lógica[1].
Framework	Uma Framework pode ser definida brevemente como um conjunto de funções agrupados de forma a atingir um determinado objectivo.
iOS	Sistema operativo para dispositivos móveis desenvolvido pela Apple Inc.
MVC	É uma design pattern utilizada principalmente para construir UIs. Divide uma aplicação em três partes interligadas entre si, o Modelo, o Controller e a View[2].
Observer Pattern	Consiste numa design pattern em que um objecto mantem uma referencia para outros objectos, denominados observers, que são notificados quando um determinado estado altera[3].
Skin	As Skins permitem oferecer a um utilizador a possibilidade de personalizar a interface da aplicação a seu gosto, alterando o seu aspecto.
Sticker	Os Stickers são uma vertente dos tradicionais Emoticons cuja principal função é melhorar a experiência de utilização no contexto de um Chat. Consistem em ilustrações de personagens com personalidade, de dimensões maiores que um tradicional Emoticon. Enviar um Sticker é um ato cujo objectivo é expressar um sentimento e/ou emoção.
VCard	Formato de ficheiros relacionados com a partilha de um contacto.

Capítulo 1
Introdução

Este relatório descreve o trabalho elaborado pelo autor durante um estágio com a duração de um ano, inserido no curso de Mestrado em Engenharia Informática da Faculdade de Ciências e Tecnologias da Universidade de Coimbra. Este estágio está a decorrer na WIT Software, S.A. estando o autor a ser orientado pelo Professor Doutor Pedro Furtado, professor do Departamento de Engenharia Informática; e Rui Gil, Gestor de Projeto na WIT Software, S.A..

1.1. Instituição

A WIT Software, S.A. é uma empresa fundada em 2001 como uma spin-off da Universidade de Coimbra. Especializou-se em desenvolver software para operadoras de telecomunicações e, neste momento, tem como clientes muitas operadores líderes de mercado, como a Deutsche Telekom, TeliaSonera, Telefónica e grupo Vodafone.

1.2. Contexto

Uma das principais fontes de receitas das operadoras de Telecomunicações é o Short Message Service (SMS). No entanto, esta fonte de receitas tem vindo a diminuir consideravelmente, em tudo devido ao surgimento de soluções Over-The-Top (OTT) como o WhatsApp, o Facebook e o Skype, que, em regra, oferecem melhores serviços a um custo muito menor para os utilizadores. Devido ao elevado crescimento da utilização destes novos serviços nos últimos anos, existem previsões de que em 2016 o uso das Instant Messages (IM) oferecidas pelas OTT irá ultrapassar o uso das SMS.

De forma a combater esta tendência, a Global System for Mobile Communications Association (GSMA), que consiste numa organização que representa mais de 800 operadores de redes móveis, criou o Rich Comunication Suite (RCS). O RCS consiste numa especificação de um conjunto de serviços interoperáveis entre as redes das diferentes operadoras de telecomunicações. Exemplos desses serviços são o Voice Over IP (VoIP), que permite efetuar chamadas de voz; o File Transfer (FT), que permite transferir ficheiros; e o Instant Messaging (IM), que permite enviar e receber mensagens de texto. De forma a que todas as aplicações RCS sejam consistentes ao nível da experiência de utilização (UX), a GSMA especificou também a UX que uma aplicação deverá ter para poder ser considerada RCS.

Em 2011 a GSMA lançou um concurso para encontrar a organização que iria desenvolver de forma oficial as soluções RCS para Android e iOS. Depois de um longo e exaustivo processo de avaliação, a WIT Software foi a empresa selecionada para desenvolver essas soluções[4].

1.3. Problema

Quando as operadoras de redes móveis lançaram as suas soluções RCS fizeram-no de uma forma totalmente gratuita, oferecendo todos os serviços aos seus utilizadores. Através deste modelo de negócio, as operadoras de redes móveis não tinham como objectivo a obtenção de lucros monetários imediatos. Pretendiam, por outro lado, angariar uma sólida base de utilizadores para ficarem inseridos num mercado emergente, que cada vez mais se afigura como sendo o futuro das comunicações móveis.

Embora a obtenção de receitas a curto prazo não fosse o objectivo principal, qualquer organização gosta de ver o retorno do seu investimento. Por isso, uma solução pouco intrusiva de gerar receitas através das suas soluções RCS será muito bem vista pelas operadoras de telecomunicações.

Este estágio pretende desenvolver uma solução que gere receitas através das soluções RCS da WIT Software, SA.

1.4. Proposta e Objectivos

Através da análise de mercado, a WIT Software S.A. encontrou um modelo de negócio que está a ser praticado pela maioria dos serviços OTT e que tem gerado lucros bastante interessantes. Esse modelo de negócio consiste na disponibilização de conteúdos de valor acrescentado aos seus utilizadores.

A proposta para este estágio consiste no desenvolvimento de uma loja que permita o download de produtos de valor acrescentado a partir das aplicações RCS da WIT Software, SA (Android e iOS).

Esta aplicação deve estar preparada para se adaptar aos novos tipos de conteúdos, por isso será também necessário desenvolver o suporte a esses conteúdos nas aplicações. No âmbito deste estágio apenas está equacionado o suporte aos dois tipos de conteúdos que mais sucesso têm tido nas aplicações OTT, que são os Stickers e as Skins. Os Stickers são um caso de sucesso no que diz respeito a receitas nestas aplicações, chegando a atingir 10 milhões de euros de faturação por mês no caso da aplicação Line, o que representa 30% do total de receitas[5].

Este estágio está inserido nos produtos RCS da WIT Software, S.A., que se encontram em produção em vários países relacionados com várias operadoras de telecomunicações. Durante o período deste estágio, a equipa de *sales* da WIT Software, S.A. realizou várias demonstrações a potenciais novos clientes. Alguns deles mostraram-se bastante interessados no conceito de download de conteúdo de valor acrescentado, o que obrigou a uma adaptação dos objectivos do estágio para irem de encontro às necessidades especificas de cada novo cliente.

Os objetivos finais definidos para este estágio são:

- Desenvolver uma Framework de Skins que permita personalizar aplicações iOS;
- Implementar o suporte à Framework de Skins na solução RCS da WIT Software, S.A. para iOS;

- Desenvolver uma loja na solução RCS para iOS da WIT Software, S.A. que permita fazer o download de conteúdos;
- Desenvolver e especificar uma solução optimizada para envio, recepção e download de Stickers para a solução RCS para iOS do grupo Vodafone intitulada Message Plus (M+);

1.6. Estrutura do documento

Este documento está organizado em Capítulos, Secções e Subsecções. Sendo os capítulos:

- **Capitulo 1 Introdução:** este é o capitulo atual, introduz a instituição e o estágio, apresentando o problema que se propõe resolver, a solução proposta e os objectivos;
- **Capitulo 2 Estado de Arte:** apresenta os resultados da analise ao estado de arte;
- **Capitulo 3 Gestão do Projeto:** descreve a metodologia de gestão do projeto e o planeamento;
- **Capitulo 4 Arquitetura:** apresenta a arquitetura geral da solução RCS para iOS da WIT Software, S.A.;
- **Capitulo 5 Trabalho Realizado:** descreve com detalhe o trabalho realizado pelo autor. Apresentado os desafios de cada tarefa bem como a solução e respectiva arquitetura;
- **Capitulo 7 Trabalho Futuro:** este capitulo, descreve o potencial trabalho que, no futuro, ainda pode ser realizado;
- **Capitulo 6 Conclusão:** o ultimo capitulo, apresenta uma reflexão do autor em relação ao estágio e ao trabalho realizado;

Este documento é acompanhado por um conjunto de anexos de que contêm informação confidencial relacionada com o trabalho realizado pelo autor:

- **Anexo A: Vodafone Message Plus Stickers Test Cases Specification:** especificação, realizada pelo autor, dos testes relativos aos Stickers no Message Plus;
- **Anexo B: Vodafone Message Plus Stickers Technical Specification:** especificação da funcionalidade de Stickers;
- **Anexo C: Especificações da UI e UX do Message Plus:** conjunto de especificações UI e UX do Message Plus desenvolvidas pelo grupo Vodafone;
- **Anexo D: Descrição da solução de download de Stickers no Message Plus:** especificação da solução desenvolvida pelo autor para a tarefa relacionada com os Stickers no Message Plus;

Capítulo 2
Estado da Arte

2.1 Introdução

Este capítulo encontra-se organizado em dois subtópicos. O primeiro tópico consiste na análise aos mecanismos de in-app purchases de Android e iOS, destacando as diferenças entre ambos. O segundo tópico consiste na análise das aplicações concorrentes à solução RCS da WIT Software, S.A. que fazem uso de Skins e/ou Stickers.

2.2 In-app purchases Android & iOS

As in-app purchases têm peculiaridades próprias de cada plataforma, quer seja Android ou iOS. Este capítulo pretende expor as diferenças entre cada mecanismo, de forma a que, no final, o autor apresente uma solução que seja compatível com qualquer um dos mecanismo de cada plataforma.

2.2.1 História e definição de in-app purchases

In-app purchases definem-se como compras realizadas através de uma aplicação móvel, de forma a poder aceder a conteúdo ou a funcionalidades consideradas especiais. O processo de compra é realizado dentro da aplicação e pretende ser transparente para o utilizador, na medida em que este não precisa de sair da aplicação para efetuar a aquisição.

As in-app purchases surgiram com o iPhone OS 3.0, sistema operativo da Apple, em Outubro de 2009. Passado pouco tempo, as outras grandes potências do mercado de dispositivos móveis também acabaram por vir a oferecer este serviço aos seus utilizadores. A BlackBerry lançou eem Setembro de 2010 e, em Março de 2011, chegou a vez da Google. Por fim, a Microsoft aquando do lançamento de Windows Phone 8, em Outubro de 2012.

2.2.2 iOS

A Apple permite implementar o suporte de in-app purchases nas aplicações iOS através do uso da Framework Store Kit. Esta Framework é a que permite realizar o processo de pagamentos através da App Store. Em cada item que seja vendido através das in-app purchases o vendedor fica com 70% do preço do respectivo item[6].

A Apple permite alojar o conteúdo relacionado com os produtos. Se optar por esta solução o conteúdo estará alojado na mesma infraestrutura que suporta a App Store. Isto quer dizer que o conteúdo estará sempre disponível em qualquer parte do mundo e a qualquer hora[7].

2.2.3 Android

O suporte a este tipo de compras nesta plataforma denomina-se *in-app billing*. Existe uma API que possibilita a comunicação da aplicação com a aplicação do Google Play (App Store da Google) de forma a poder realizar as compras e os respectivos pagamentos[8].

No entanto, contrastando com a solução da Apple, esta solução não permite alojar o conteúdo relativo a cada produto.

2.2.4 Conclusão

Os resultados da análise às soluções das duas plataformas permitiram perceber que a principal diferença entre ambas está relacionada com o facto de a solução da Apple permitir alojar o conteúdo da in-app purchase na sua infraestrutura, ao contrário do Android. Concluiu-se, portanto, que existe a necessidade de uma entidade para alojar o conteúdo de cada in-app purchase.

2.3 Descrição de Aplicações com Skins e/ou Stickers

Nesta secção, o autor apresenta uma análise às diferentes soluções que o mercado dispõe no que diz respeito a aplicações que utilizem Skins e Stickers.

Visto que o estágio está centrado na solução RCS da WIT Software, S.A., onde o *instant messaging* é uma funcionalidade central, o autor focou-se principalmente em analisar aplicações que ofereçam o mesmo tipo de serviço, ou seja, aplicações que sejam concorrentes diretas. No entanto, por forma a estender o leque de funcionalidades oferecidas, foram também analisadas outras aplicações que aparentavam ter características que merecessem a sua análise.

Para a realização desta análise o autor instalou e testou as varias aplicações disponíveis de forma gratuita. Estas aplicações foram testadas num iPhone e num iPad, ambos com o sistema operativo iOS 7.0.1.

2.3.1 Critérios de avaliação

Existem dois pontos de principal destaque nesta análise. O primeiro consiste em perceber que aplicações fazem uso de Stickers, como o fazem, como os organizam, como é o processo de download e como funciona o método de pagamento.

O outro ponto de relevante importância são as *skins*. O suporte existente na plataforma iOS no que diz respeito a personalização de interface é limitado, não permitindo, por exemplo, alterar a posição de objetos. Por isso, será interessante analisar até que ponto as aplicações estão a personalizar as suas interfaces através de *skins* e se possível, perceber como é que o estão a fazer.

A análise das aplicações e da sua respectiva mais valia para este estágio teve como requisitos de seleção os seguintes aspectos:

- **A aplicação tem de oferecer conteúdo *downloadable*:** Este é o critério fundamental para considerar que uma aplicação merece uma análise mais detalhada.
- **A aplicação é concorrente da solução da WIT:** Isto permite fazer uma comparação direta com o mercado, sabendo quais são os pontos fortes e os pontos fracos no que diz respeito a conteúdo *downloadable*.

As principais características a reter na analise de cada aplicação são:

- Qual o tipo de conteúdo disponibilizado;
- Para o caso do conteúdo serem Skins, perceber até que ponto é que a *user interface* é customizada;
- Aquando da compra de um conteúdo, perceber se é usado o mecanismo de *in-app purchases* da Apple ou se existe alguma outra forma de efetuar o pagamento;

2.3.2 Aplicações analisadas

ChatON

O ChatON é um serviço de IM desenvolvido pelo fabricante de telemóveis Samsung Electronics. Além de se encontrar disponível para a maioria dos sistemas operativos móveis (incluindo o próprio sistema operativo da Samsung, o *bada*), encontra-se também disponível como Web App.

Esta aplicação não tem uma loja, mas existe uma secção onde é possível fazer download de Stickers (ao qual eles chamam Anicons). São totalmente gratuitos.

Facebook Messenger

O Facebook Messenger é um serviço oferecido pelo Facebook para as grandes plataformas móveis. Inicialmente consistia numa versão mobile do Chat do Facebook, no entanto tem evoluído para uma versão mais *standalone*, sendo possível trocar mensagens necessitando apenas de um nome e um número de telefone.

Este serviço oferece a possibilidade de adquirir *stickers*. A sua loja possibilita visualizar uma lista com todos os *stickers*, bem como uma lista dos mais comprados.

No momento desta análise todos os *stickers* eram gratuitos.

Kakao Talk

O Kakao Talk foi desenvolvido pela Coreana Kakao Corp e é utilizado por mais de 55 milhões de pessoas em todo o mundo[9]. Além do tradicional IM, esta aplicação permite também jogar jogos *multiplayer* entre os seus utilizadores.

Possui uma loja nativa onde se pode comprar Skins e Stickers. Permite ordenar ambos pelo mais recente e o mais popular. As compras são efectuadas com o recurso às in-app purchases da Apple.

Pelo que foi possível analisar, as Skins desta aplicação apenas alteram imagens, cores e fontes. Não foi possível verificar se o método que utilizam permite alterar posição e dimensões dos objetos.

Kik Messenger

O Kik Messenger é uma aplicação de IM disponível na maioria das plataformas móveis.

Esta permite o download de Stickers através de uma loja que consiste numa lista de todos os pacotes de Stickers disponíveis. As compras são efectuadas com o recurso às in-app purchases da Apple.

Line

O Line é uma aplicação de *instante messaging* e VoIP desenvolvida no Japão. Foi lançada em 2011 no Japão e atingiu 100 milhões de utilizadores em oito meses e 200 milhões de utilizadores passados 6 meses[10].

Esta aplicação oferece a possibilidade de adquirir Stickers. A sua loja possibilita ver o seu conteúdo ordenado pelo mais recente, pelo preço, pelos mais procurados e por categorias. As compras de Stickers são feitas com o recurso às in-app purchases da Apple.

O Line oferece também a possibilidade de aplicar Skins à aplicação. No entanto, essas Skins não se encontram numa loja como os Stickers. As Skins estão disponiveis nas definições da aplicação, onde é possível fazer download de uma pequena seleção delas.

As Skins do Line são obtidas através de download e é permitido apagar Skins já transferidas. De igual forma ao Kakao Talk, as Skins desta aplicação aparentam apenas que alteram imagens, cores e fontes. Mais uma vez, não foi possível verificar se o método que utilizam permite alterar posição e dimensões dos objetos.

Skype

O Skype consiste num serviço de VoIP e *instante messaging* disponível para os sistemas operativas de desktop e smartphones mais populares. Trabalha sobre um protocolo peer-to-peer intitulado de *Skype protocol* que requer uma diminuta infraestrutura centralizada. É usado por mais de 660 milhões de utilizadores[11].

Este serviço não oferece nenhum conteúdo *downloadable* mas oferece in-app purchases, que consistem em compra de crédito para poder fazer chamadas.

As compras são efectuadas com o recurso às in-app purchases da Apple.

Viber

O Viber é uma aplicação de *instante messagin* e de VoIP para smartphones, utilizada por mais de 200 milhões de pessoas em todo o mundo[12].

O Viber possibilita o download de Stickers através de uma loja que permite visualizar os Stickers mais recentes, os mais utilizados e os gratuitos.

Tal como o Skype, oferece a possibilidade de comprar crédito para poder fazer chamadas.

As compras são efectuadas com o recurso às in-app purchases da Apple.

2.3.3 Conclusão

Esta análise permitiu perceber que a maioria das aplicações oferece conteúdos para compra através das in-app purchases da Apple. Não foi, por isso, detectado outro mecanismo de compra diferente.

As aplicações disponibilizam conteúdos como Stickers e Skins. Algumas oferecem também a possibilidade de comprar crédito para poder fazer chamadas.

As Skins que foram utilizadas durante os testes não permitiram detectar mudanças na posição dos objetos ou em características mais sofisticadas do que cores, fontes e imagens.

Capítulo 3
Gestão do Projeto

Este capítulo destina-se a apresentar a metodologia de trabalho seguida pelo autor, o porquê da sua escolha bem como a descrição e apresentação o plano de estágio e sua respectiva evolução.

3.1 Equipa WIT Mobile Communicator

O autor está inserido na equipa do produto RCS da WIT Software, S.A. denominada WIT Mobile Communicator (WMC), esta equipa tem como objectivo o desenvolvimento das soluções RCS para várias plataformas, o que leva a que seja uma equipa numerosa composta por mais de quarenta colaboradores. Está dividida em vários grupos que correspondem às plataformas para que desenvolvem. Os grupos existentes à data são:

- Android;
- **iOS**;
- Desktop;
- Web;
- COMLib: A COMLib é uma *stack* desenvolvida pela WIT Software, S.A. cuja função é realizar toda a lógica de comunicação necessária a uma aplicação RCS, independentemente da sua plataforma;

A equipa do WMC é composta também por uma equipa de Quality Assurance dedicada aos seus produtos.

3.2 Metodologia

Na WIT Software, S.A. não existe uma metodologia de trabalho imposta de uma forma obrigatória. Cada equipa, projeto e cliente são diferentes, por isso cabe a cada uma das equipas escolher a metodologia de trabalho mais adequada ao cliente e às características do projeto em questão. Esta flexibilidade fez sobressair duas diferentes metodologias, o SCRUM[13] e o Modelo Waterfall[14]. No entanto, é possível acontecer que em projetos com certas e determinadas características se adopte uma metodologia que consiste numa variante do SCRUM e do Modelo Waterfall.

O autor foi integrado na equipa do WMC, por isso, não pôde participar na escolha da metodologia que iria aplicar no seu estágio. A equipa do WMC não segue nenhuma das metodologias principais (SCRUM ou Modelo Waterfall), trabalha sob metodologia AGILE[15] inspirada no SCRUM.

Na primeira fase desta metodologia são analisados os requisitos de cada funcionalidade a desenvolver. Desta fase resulta então uma lista de funcionalidades, divididas por categorias.

Após a primeira fase estar concluída é realizada uma reunião entre o autor e o orientador de estágio, onde são debatidas as tarefas que resultam de cada funcionalidade e, se necessário, divididas em tarefas mais pequenas. Nessa reunião também é especificado o tempo de execução de cada tarefa bem como a prioridade de execução de cada funcionalidade.

Sempre que é implementada uma nova funcionalidade as seguintes fases são executadas:

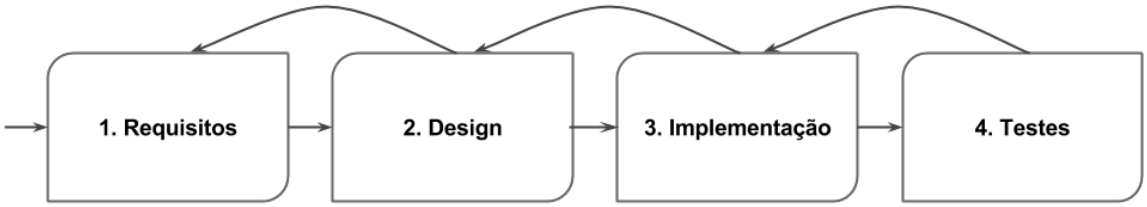

Figura 1 - Fases da metodologia implementada

1. **Requisitos:** Na primeira fase todos os requisitos da funcionalidade são revistos. Esta análise permite identificar eventuais necessidades de alterar, adicionar ou remover requisitos;
2. **Design:** Nesta fase, o colaborador estuda a melhor forma de implementar a funcionalidade. Para isso, analisa a arquitetura já existente com o intuito de perceber qual será a melhor forma de integrar a funcionalidade no projeto já existente. Esta fase pode originar um documento com a arquitetura e descrição da implementação funcionalidade;
3. **Implementação:** Esta fase consiste no desenvolvimento da funcionalidade;
4. **Testes:** Finalmente a equipa de qualidade da WIT Software, S.A. executa testes funcionais relativos à funcionalidade desenvolvida;

A qualquer momento é possível voltar a uma fase anterior, por exemplo, devido a uma mudança de requisitos ou quando um colaborador cria uma entrada na plataforma de bug-tracking Redmine relativa a um problema na funcionalidade implementada. O que leva a que, consequentemente, surja a necessidade de voltar à fase de implementação para corrigir esse problema.

3.1 Planeamento

Neste capítulo serão descritas as tarefas realizadas pelo autor, bem como a evolução do planeamento ao longo do estágio.

3.1.1 Proposta inicial

Este subcapítulo, pretende apresentar os objectivos traçados no início do estágio. O diagrama seguinte apresenta as tarefas que o autor teria de realizar para que no final os objectivos fossem cumpridos dentro dos prazos estipulados.

Em primeiro lugar, o autor iria ambientar-se ao contexto do estágio. Dispondo de quatro semanas para realizar o estado de arte sobre três importantes tópicos:

1. **Aplicações concorrentes OTT:** Esta análise tem como objectivo compreender o estado do mercado no que diz respeito a aplicações que disponibilizassem conteúdo de valor acrescentado. Sendo interessante registar os diferentes tipos de conteúdos e, se possível, perceber como é que a aplicação se consegue adaptar a cada um desses diferentes tipos.
2. **Mecanismos de in-app purchases iOS e Android:** As duas soluções para sistemas operativos de dispositivos móveis têm diferentes regras no que diz respeito à integração de in-app purchases nas aplicações. Por isso, existiu a necessidade de estudar as duas soluções de forma a que o autor desenvolva uma solução que se adapte aos dois sistemas operativos.

3. **Serviços de Cloud Storage:** Para que seja possível o desenvolvimento de uma plataforma que permita gerir os conteúdos disponíveis na loja, esses conteúdos terão obrigatoriamente de estar alojados num determinado local. O objectivo seria fazer uso de um serviço de Cloud Storage. Por isso, existiu a necessidade de fazer uma análise aos serviços existentes de Cloud Storage, para depois, a WIT Software, S.A. optar por uma delas.

Após a análise ao estado de arte o autor iria desenvolver uma Framework de Skins que permitisse aplicar diferentes Skins à solução RCS para iOS. O autor iria dispor de doze semanas para o desenvolvimento desta tarefa.

Em seguida, o autor iria desenvolver na solução iOS a loja que permitisse fazer o download de conteúdos. A solução teria de ser suficientemente genérica para se poder adaptar com facilidade a novos tipos de produtos. No âmbito do estágio, o autor teria de implementar o suporte ao download de Skins e Stickers, bem como à implementação do suporte para a utilização do novo conteúdo. Iria dispor de doze semanas para concluir esta tarefa.

Após a loja estar completa, o autor iria começar a sua última tarefa, para a qual dispunha de treze semanas. Esta ultima tarefa consiste na implementação de uma plataforma de gestão do conteúdo disponível na loja.

De salientar que, de acordo com este plano, o autor teria ainda dois períodos dedicados à escrita do relatório, coincidindo esses períodos com o final de cada semestre.

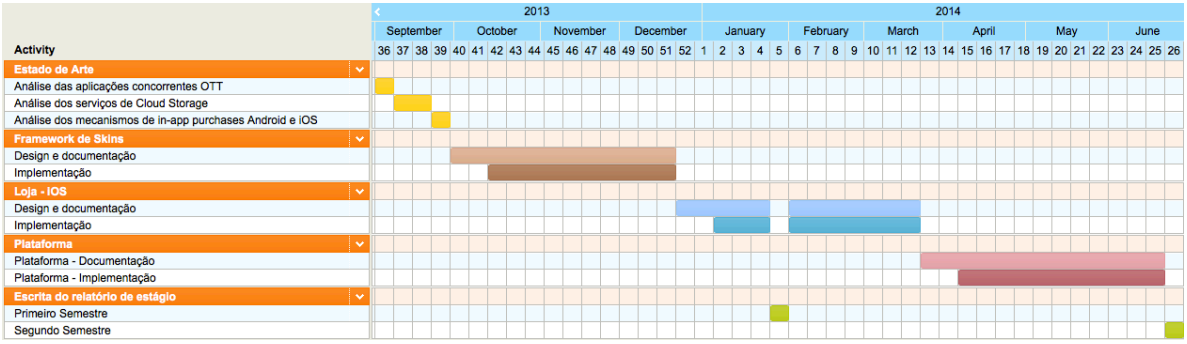

Figura 2 - Diagrama de Gantt com os objetivos iniciais do estágio

De uma forma sintetizada, os objectivos do estágio no seu início eram:

- Desenvolver uma Framework de Skins para aplicações iOS;
- Desenvolver uma loja que permita fazer download de conteúdos de valor acrescentado;
- Desenvolver uma plataforma de gestão dos conteúdos presentes na loja;

3.1.2 Planeamento final

O estágio encontra-se profundamente integrado no cliente RCS para iOS da WIT Software, S.A., o que faz com que os seus objetivos tenham uma relação bastante próxima com os do cliente RCS e, por isso, origina que os objetivos do estágio se tornem bastante dinâmicos. Este subcapítulo tem como objectivo apresentar os motivos que levaram à execução de cada tarefa, esperando que no final o leitor perceba toda a evolução do estágio no que ao planeamento e objetivos diz respeito.

Como planeado inicialmente, a primeira tarefa realizada pelo autor esta relacionada com o estado de arte, começando por fazer a análise das aplicações concorrentes OTT.

De seguida, surgiu a primeira alteração ao plano inicial. Inicialmente o esperado era que o autor continuasse a elaboração do estado de arte, concluindo o estudo dos restantes tópicos - mecanismos de in-app purchases iOS e Android e Serviços de Cloud Storage. O facto de a Framework de Skins ser de extrema importância para o cliente RCS iOS e também o facto de não existir nenhuma outra ferramenta semelhante na qual o autor se pudesse basear para reduzir o tempo de desenvolvimento, fez com que o começo da tarefa fosse antecipado três semanas para evitar o não cumprimento da meta estipulada. Passando assim a conclusão da execução do estado de arte para depois desta tarefa.

Em seguida, teve de ser adicionado o suporte à Framework de Skins à aplicação. Para isso, toda a aplicação teve de sofrer alterações para poder ser integrada com a Framework. A cada elemento da equipa foi incumbida a tarefa de adicionar suporte a uma determinada funcionalidade da aplicação, ficando o autor encarregue de toda a funcionalidade de Chat. A funcionalidade de Chat é a funcionalidade medular de toda a aplicação, tendo por isso requisitos de performance, *user interface* e *user experience* mais rigorosos. A solução relativa à funcionalidade de Chat existente na altura mostrou estar datada e sem possibilidade de ser reaproveitada, tendo por isso o autor de refazer toda a solução, tornando-se assim, numa tarefa extremamente interessante para ser desenvolvida no âmbito deste estágio. Esta tarefa foi denominada de "*refactoring* do chat" e o autor dispôs de sete semanas para a sua execução.

O *refactoring* do Chat terminou a meados do mês de Janeiro. Surgiu então a oportunidade para terminar o estado de arte, preparando a entrega de relatório intermedio de estágio. Sendo assim, em primeiro lugar o autor realizou a análise comparativa dos mecanismos de in-app purchases entre iOS e Android e, em seguida, começou a analisar os diferentes Serviços de Cloud Storage. No entanto, como a oferta de serviços era abundante e a execução de uma análise com a qualidade desejada tornou-se complicada, em tudo devido a falta de informação e falta de serviços com uma versão de demonstração gratuita, não foi possível terminar a análise a tempo da entrega do relatório intermédio de estágio. Por isso, o autor ficou de continuar a análise durante o segundo semestre.

No início do mês de Fevereiro o autor interrompeu a tarefa relacionada com a análise dos Serviços de Cloud Storage para preparar um *Proof of Concept* (POC) para o Mobile World Congress (maior feira mundial de telecomunicações) que se realizou entre os dias 24 e 27 de Fevereiro, em Barcelona, onde a WIT Software, S.A. esteve presente. Este POC tinha como objectivo demonstrar o conceito de disponibilização de conteúdo de valor acrescentado dentro de uma aplicação para dispositivos móveis. A demonstração consistia na introdução de uma loja, na solução RCS para iOS da WIT Software, S.A., onde fosse possível fazer download de Stickers e Skins. Para esta tarefa, o autor dispôs de aproximadamente três semanas, um curto espaço de tempo que impediu o autor de construir uma solução de maior complexidade. O principal objectivo era que no dia 24 de Fevereiro (dia de começo do MWC) se apresentasse uma loja onde fosse possível fazer download de Stickers e Skins e fazer uso desse conteúdo.

A demonstração da loja de Stickers e Skins no MWC mostrou-se bastante produtiva. Surgiram vários clientes e organizações interessadas no conceito de disponibilização de conteúdo de valor acrescentado nas suas aplicações/soluções. Este interesse fez com que surgisse a necessidade de ajustar os objectivos do estágio a esta nova realidade.

Devido ao sucesso que foi a demonstração no MWC a WIT Software, S.A. decidiu que queria introduzir a loja nas várias demonstrações realizadas pela equipa de *Sales*. Sendo que a versão que existia consistia num POC, tornou-se então prioritário o desenvolvimento de uma versão da loja mais próxima do resultado final pretendido. Para a execução dessa tarefa o autor dispôs de aproximadamente 4 semanas.

Em seguida, surgiu o interesse do grupo Vodafone. A WIT Software, S.A. está a desenvolver uma aplicação para o grupo Vodafone intitulada de Message Plus, que consiste numa aplicação de *messaging* baseada na solução RCS da WIT Software, S.A. mas com características específicas definidas pelo grupo Vodafone. O objectivo do grupo Vodafone era de preparar a aplicação para que num futuro próximo os utilizadores possam comprar Stickers dentro da aplicação. No entanto, os requisitos funcionais, de UX e de UI apresentados pelo grupo Vodafone apresentavam diferenças consideráveis em relação à solução desenvolvida pela WIT Software, S.A.. Por este motivo, o autor teve especificar e implementar uma solução que fosse de encontro ao pretendido pelo grupo Vodafone. Para a realização desta tarefa o autor dispôs de oito semanas.

A aplicação Message Plus encontra-se atualmente em produção em mais de quinze países. Apesar disso, os desenvolvimentos na aplicação não estão parados, estando em desenvolvimento novas funcionalidades que têm o objectivo de melhorar a experiência de utilização da aplicação e também conseguir rivalizar com as aplicações concorrentes. Devido a isso, a versão que se encontra disponível na App Store é regularmente atualizada para introduzir novas funcionalidades e melhoramentos. A tarefa anterior entrará em produção na nova atualização da aplicação, bem como outras funcionalidades desenvolvidas pela equipa.

Uma outra funcionalidade que estava prevista entrar nessa mesma atualização da aplicação consistia na implementação de uma nova versão da câmara de Fotos e Vídeos e a implementação de uma funcionalidade intitulada de Quick Share de Video. O Quick Share de Vídeo consiste na rápida partilha de vídeos de curta duração a partir da vista de Chat da aplicação. A funcionalidade consiste no simples *long press* de um botão, na vista de Chat, que desencadeará o começo de gravação de um vídeo que terá uma duração máxima de quinze segundos. Quando o utilizador deixa de fazer *long press* no botão, o vídeo para de ser gravado e é enviado de uma forma imediata para o outro utilizador. A principal vantagem desta funcionalidade reside na simplicidade de interações necessárias para fazer uma partilha de vídeo. A funcionalidade tem sofrido de uma popularidade crescente, exemplo disso é a solução de *messaging* do Facebook que implementa uma versão semelhante a esta, bem como, a Apple que anunciou que a próxima versão do iOS irá ter esta funcionalidade na sua aplicação nativa de *messaging*.

Devido aos factos desta funcionalidade ter de ser integrada na próxima versão do Message Plus, da equipa estar ocupada a terminar outras tarefas de igual importância e de não haver um recurso com experiência na câmara de vídeo e fotos dos dispositivos Apple como a do autor, foi tomada a decisão de integrar o desenvolvimento desta tarefa no estágio. O autor dispôs de quatro semanas.

No final, o autor dedicou, sensivelmente, duas semanas à escrita deste documento.

A tarefa relacionada com o desenvolvimento da plataforma de gestão dos conteúdos foi então retirada do âmbito deste estágio, pois foi dada prioridade aos novos clientes. Por esta razão, a analise aos Serviços de Cloud Storage ficou sem fazer sentido para o âmbito deste estágio e por isso não foi incluída neste documento.

No capítulo cinco é possível perceber, de uma forma detalhada, o contributo do autor em cada tarefa desenvolvida.

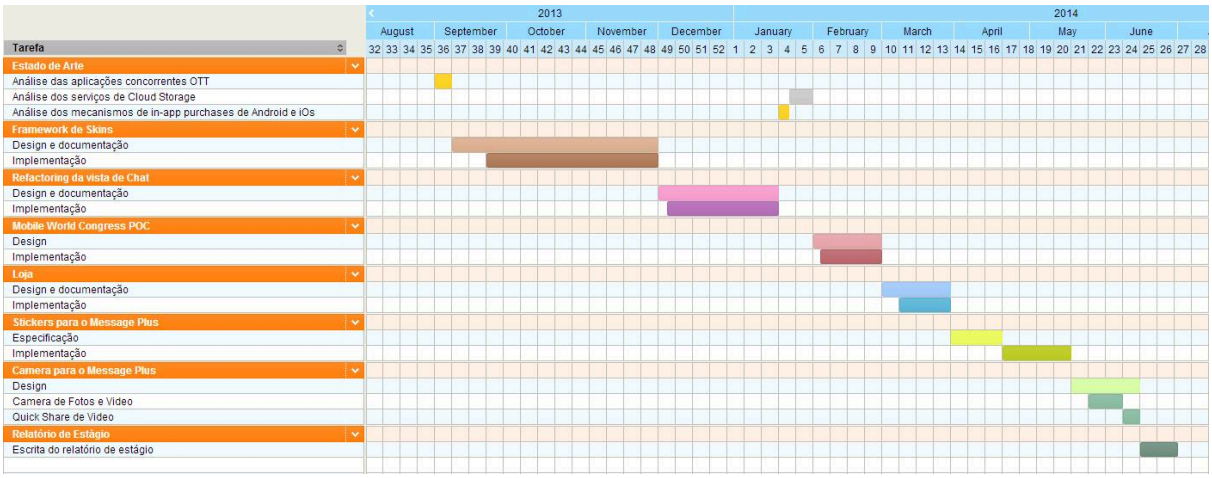

Figura 3 - Diagrama de Gantt com os objetivos finais do estágio

Capítulo 4
Arquitetura

Este capitulo detalha como está estruturada a solução RCS para iOS da WIT Software, S.A., sendo o principal objetivo que o leitor perceba de um modo geral todos os componentes da solução.

Neste capitulo não é dado foco aos componentes em que o autor esteve envolvido, isso será detalhado no capitulo seguinte aquando da descrição de cada tarefa realizada pelo autor.

4.1 Estrutura da aplicação

Devido ao facto da solução RCS da WIT Software, S.A. ser uma solução relativamente complexa, colocar toda a sua arquitetura numa única figura resultaria num diagrama igualmente complexo. A figura a baixo representa a estrutura da UI da aplicação, salientando todos os seus principais componentes.

O AppDelegate é o ponto de entrada único na aplicação, sempre que ela é inicializada ou resumida. O AppDelegate delega ao Containers Manager a responsabilidade de escolher a UI adequada de acordo com o dispositivo (iPad ou iPhone). De seguida, o Containers Manager, acede ao Features Availability para saber que módulos estão definidos como ativos nos ficheiros de configuração, de acordo com essa informação inicializa unicamente os módulos que estão configurados como ativos. Os módulos contêm a lógica e UI relacionada com cada serviço fornecido pela aplicação, como por exemplo o Chat e as chamadas VoIP. Cada modulo utiliza o Features Availability para saber se determinada funcionalidade esta disponível, por exemplo, o modulo de Chat utiliza o Features Availability para saber se é possível enviar um ficheiro através da funcionalidade File Transfer. A package Notifications é responsável pela gestão de todas as notificações da aplicação. Uma notificação pode assumir varias representações de acordo com o estado da aplicação. Nos Components estão disponíveis vários elementos de UI que podem ser reutilizados por toda a aplicação. O Utils contem objetos que possuem lógica passível de ser reaproveitada pela aplicação. Já o Core Data consiste uma Framework que funciona com uma *wrapper* para aceder ao SQLite.

A COMLib é uma *stack* proprietária desenvolvida pela WIT Software, S.A. que implementa a lógica de baixo nível relativa à comunicação com a rede de IMS de forma a oferecer as funcionalidades do RCS.

Sempre que algum componente tenha a possibilidade de ser customizado é necessário aceder à Framework de Skins.

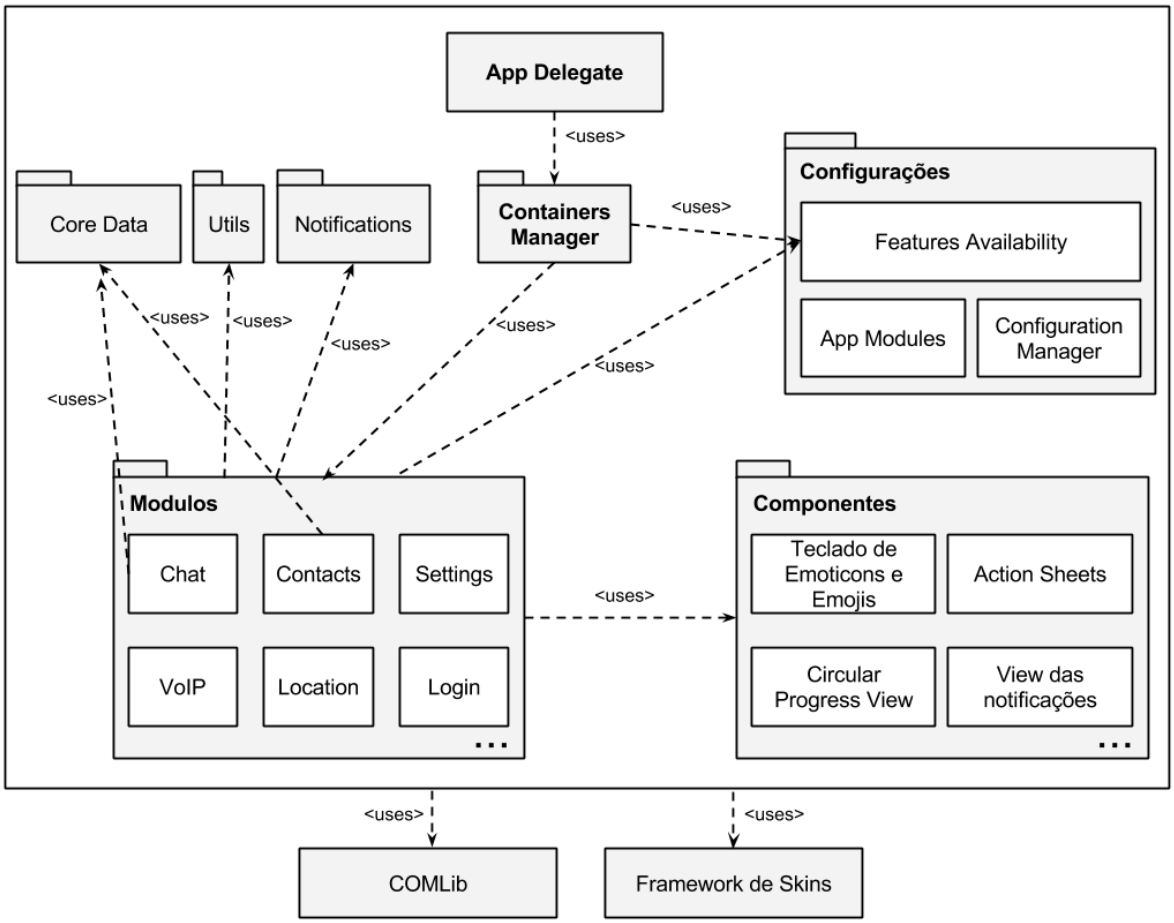

Figura 4 - Arquitetura da UI da aplicação

Package	**Descrição**
App Delegate	Consiste no ponto de entrada na aplicação.
Configurações	Existem dois ficheiros de configurações de extrema importância. Um deles é local e é usado para gerir os módulos e funcionalidades que determinada versão da aplicação deve disponibilizar. O segundo consiste num ficheiro resultante de um processo intitulado de *auto-configuração* que é enviado quando um utilizador se regista na rede. Este ficheiro possui, entre outras coisas, as funcionalidades que a respectiva rede suporta. O App Modules é o responsável pela leitura da primeira configuração e o Configuration Manager é o responsável pela segunda. O Features Availability consiste numa Facade [2] para os dois ficheiros de forma a saber se uma determinada funcionalidade se encontra disponível ou não.
Containers Manager	Este Manager é responsável por fazer a gestão dos módulos que são iniciados quando a aplicação arranca. Escolhe a UI consoante o dispositivo (iPad vs iPhone).
Utils	Consiste numa package que contem várias classes úteis e passiveis de serem reutilizadas no projeto.
Notifications	Esta package é responsável pela gestão de todas as notificações da aplicação. Uma notificação pode assumir varias representações de acordo com o estado da aplicação, esta

	package faz essa gestão.
Core Data	Framework que funciona como um wrapper para aceder ao SQLite [20].
Modulos	Esta package é a maior e a mais complexa de toda a UI e consiste no conjunto de sub-packages mais pequenas chamadas de Modulos. Cada Modulo contem a lógica associada a uma determinada funcionalidade ou serviço.
Componentes	Esta package consiste em vários componentes de UI que podem ser reutilizados por toda a aplicação.
COMLib	*Stack* proprietária desenvolvida pela WIT Software, S.A. que implementa a lógica de baixo nível relacionada com as funcionalidades do RCS.
Framework de Skins	Representa a Framework de Skins desenvolvida durante este estágio que permite aplicar skins em aplicações iOS.

Tabela 1 - Arquitetura do UI da aplicação

Capítulo 5
Trabalho Realizado

Este capítulo descreve em detalhe o trabalho realizado pelo autor durante este estágio. Está organizado em subcapítulos, cada um representando uma tarefa.

5.1 Desenvolvimento de uma Framework de Skins

No começo deste estágio a solução RCS para iOS da WIT Software, S.A. não suportava qualquer tipo de personalização no que diz respeito a Skins, por isso, uma Framework de Skins é vista como uma ferramenta de extrema importância para a solução RCS para iOS da WIT Software, S.A..

5.1.1 Problema

O SDK do iOS oferece a possibilidade de alterar propriedades de certos componentes através do protocolo UIAppearance[16]. O UIAppearance permite configurar a aparência de uma aplicação de forma a que toda a aplicação fique consistente. No entanto, as propriedades que implementam este protocolo não são suficientes para o grau de personalização desejado para a Framework. Surge assim a necessidade de desenvolver uma Framework.

5.1.2 Objectivos

De uma forma objectiva e sucinta, os objectivos para a Framework são os seguintes:

- A Framework deve possibilitar a customização de qualquer aplicação iOS;
- A Framework deve suportar iPhone e iPad;
- A Framework deve possibilitar a troca de Skin na aplicação sem a necessidade de a fechar e abrir de novo a aplicação, ou seja, deverá fazer a troca de Skin de uma forma *live*;
- Cada Skin deve estar o suficientemente isolada e independente, de forma a que no futuro possa ser possível fazer o download de skins;
- A utilização da Framework pelo *developer* deve ser fácil e objectiva;
- A Framework deverá ser estruturada de forma a que a implementação do seu suporte nas aplicações já existentes seja o menos intrusiva possível, de forma a evitar regressões;
- A Framework deverá ser flexível o suficiente de forma a que seja possível, de uma forma simples, estende-la com novas funcionalidades;

5.1.3 Solução

Em seguida, é apresentada uma descrição detalhada de toda a solução. De reter que todas as decisões de arquitetura foram tomadas de forma a que a solução seja o mais genérica possível, de forma a que seja possível estender a Framework com mais funcionalidades sempre que possível.

A solução desenvolvida permite, através de ficheiros de configuração XML, alterar propriedades de componentes da UI, bem como adicionar-lhes novas propriedades e

comportamentos. Além dos ficheiros de configuração, existem três outras estruturas de principal relevo, os *Components*, as *Operations* e os *Objects*.

Como introdução a uma explicação detalhada de cada uma das estruturas, é importante reter que os ficheiros de configuração XML referidos anteriormente permitem definir valores para determinadas propriedades de objetos da UI (i.e. cor de fundo, área, posição, etc). Mais à frente neste capítulo será apresentada uma explicação mais detalhada da estrutura dos ficheiros de configuração XML.

5.1.3.1 *Components*

Cada objecto que apresente uma componente de UI (i.e. área, posição, cor, etc.) encontra-se qualificado para poder ser customizado pela Framework de Skins. Por isso, todos os diferentes componentes de UI que o iOS dispõe estão analogamente qualificados para serem customizados. Os componentes de UI que o iOS disponibiliza são:

Componente	Herda	Descrição
UIView[17]	NSObject[18]	É uma *class* que define uma área rectangular no ecrã e que fornece uma API que permite gerir o conteúdo dentro dessa área. É a *class* mãe de todos os objetos que apresentam uma componente de UI.
UIButton[19]	UIView	É uma *subclass* de UIView que apresenta características típicas de um botão. Características como eventos de toque, diferentes estados (*disable* vs. *enable*), etc.
UISwitch[20]	UIView	É uma *subclass* de UIView que apresenta um botão com os estados *on* e *off*.
UISlider[21]	UIView	É uma *subclass* de UIView que apresenta um controlador visual que permite escolher um único valor de um conjunto de valores contínuos;
UILabel[22]	UIView	É uma *subclass* de UIView que permite desenhar uma ou várias linhas de texto estático.
UITextView[23]	UIView	É uma *subclass* de UIView que permite desenhar texto estático numa vista que tem a propriedade de ser *scrollable*. Uma vista tem a necessidade de ser *scrollable* quando a área ocupada pelo conteúdo a ser apresentado é maior que o espaço disponível e, por isso, surge a necessidade de fazer *scroll* para ver mais conteúdo.
UITextField[24]	UIView	É uma *subclass* de UIView que permite desenhar texto dando ainda a possibilidade de o editar.
UINavigationBar[25]	UIView	É uma *subclass* de UIView que representa uma barra horizontal que normalmente aprece no topo das aplicações cujo objectivo principal é navegar através da hierarquia de vistas da aplicação. Dispõe de três

		propriedades que permitem definir respectivamente uma UIView para o lado esquerdo da barra, uma UIView para o titulo da barra e uma outra UIView para o lado direito da barra.

Tabela 2 - Componentes de UI disponibilizados pelo iOS

Como é possível verificar pela análise à tabela anterior, todos os componentes herdam de UIView, por isso, para cada objecto será possível alterar, no mínimo, as propriedades da UIView, que se resumem à área, posição e cor de fundo. Mais ainda, em iOS todo o conteúdo de uma UIView é apresentado numa CALayer[26], cuja função é apresentar o conteúdo no ecrã em forma de imagem. A CALayer também tem diferentes propriedades qualificadas para serem alteradas pela Framework. A seguinte tabela agrega todas as propriedades que qualquer objecto que herde de UIView dispões para alterar.

Propriedade	Tipo	Descrição
frame	CGRect[27]	Permite definir a largura, altura e posição de uma UIView.
backgroundColor	UIColor[28]	Permite definir a cor de fundo para uma UIView.
cornerRadius	CGFloat[29]	Permite definir um ângulo para aplicar a cada canto da *frame* de uma UIView.
borderColor	UIColor	Permite definir a cor da linha de contorno de uma UIView.
borderWidth	CGFloat	Permite definir a grossura da linha de contorno de uma UIView.

Tabela 3 - Propriedades de um objeto que herde de UIView possíveis de alterar

Portanto, todos estes componentes possuem características que os tornam candidatos a ser personalizados pela Framework. Sendo assim, para cada um destes componentes foram criados objetos que fazem o relacionamento/a ponte entre os definidos nos ficheiros de configuração XML e o próprio objecto. Esses objetos são intitulados de *Components*. Por outras palavras, estes objetos permitem fazer o mapeamento entre uma UIView definida no XML de configuração e o componente UIView nativo.

É importante reter também que cada componente presente na tabela de *Components*, além das propriedades que herdam da UIView, também têm as suas próprias propriedades.

5.1.3.2 *Operations*

Por definição, uma *Operation* é uma *Class* cujo objectivo consiste na execução de uma determinada atividade lógica com um determinado fim.

Para cada propriedade de cada *Component* (tabela anterior) existe uma *Operation* associada. Por exemplo, existe uma responsável por alterar o *backgroundColor* de uma UIView.

Cada *Component* pode executar um determinado conjunto de operações, que normalmente coincidem com a execução da lógica necessária para alterar as suas propriedades.

Para cada *Component* é possível definir quais os tipos de operações que ele suporta.

5.1.3.3 *Objects*

Neste momento é possível perceber que uma *Operation* é capaz de alterar uma propriedade de um *Component*. Por exemplo, alterar a cor de fundo de uma UIView. No entanto, para poder alterar a cor a *Operation* necessita de receber no mínimo um *input* que lhe diga qual a cor final pretendida. Para isso foram criados os *Objects*.

Um *Object* é uma *Class* cujo objectivo é conter a informação de input de uma *Operation*. Essa informação pode tomar vários tipos. Esses tipos estão definidos na seguinte tabela:

Tipo de Objecto	Descrição
CGFloat[29]	Representa um *float* ou um *double* consoante o sistema seja de 32 ou 64 bits.
NSArray[30]	Consiste numa lista de objectos.
NSInteger[29]	Representa um *long* de 32 bits ou 64 bits consoante o sistema utilizado.
CGPoint[27]	Representa uma estrutura que contém um ponto em coordenadas bidimensionais.
CGRect[27]	Representa uma estrutura que contem a posição e dimensões de um retângulo.
CGSize[27]	Representa uma estrutura que contem os valor de largura e altura.
NSString[31]	Representa uma *string* imutável.

Tabela 4 - Tipos de *Object*

Através destes *Objects* é possível passar qualquer de *input* para uma *Operation*.

5.1.3.4 Conclusão acerca dos *Components, Operations* e *Objects*

Esta solução é genérica o suficiente de forma a que exista um qualquer tipo de *Component*, *Operation* e *Objects*. Assim, até uma vista *custom* implementada pela programador pode ser personalizada com a Framework, para isso, basta adicionar as *Operation, Components* e *Objects* necessários.

5.1.4 Relação entre a instância de um componente e a respectiva configuração

Como já foi mencionado anteriormente, a Framework de Skins trabalha sobre ficheiros de configuração em XML, que permitem definir o aspecto que cada *Component* irá ter na UI. A Framework começa por fazer *parse* do ficheiro em XML onde está definido o valor que a propriedade desse componente irá ter. A relação entre o componente da UI e a sua especificação é feita através de um identificador, ou seja, cada conjunto de *Operations* definidas para um determinado *Component* tem um identificador. De uma forma mais específica, imagine o leitor que num ficheiro de configuração está especificado um componente do tipo UIButton - que consiste num botão que é usado sempre que o

utilizador quer cancelar uma ação - e, por isso, foi especificado que esse tipo de botão terá como cor de fundo a cor vermelha. Este componente terá um identificador associado, por exemplo "button_cancel_action", definido no XML. Assim, todas as instâncias de UIButton a que lhe seja atribuído este identificador irão ter como cor de fundo a cor vermelha. A figura seguinte pretende demonstrar de uma forma mais visual e de alto nível o mapeamento entre a instância de um componente da UI da aplicação, com as definições das suas propriedades no ficheiro de configuração em XML.

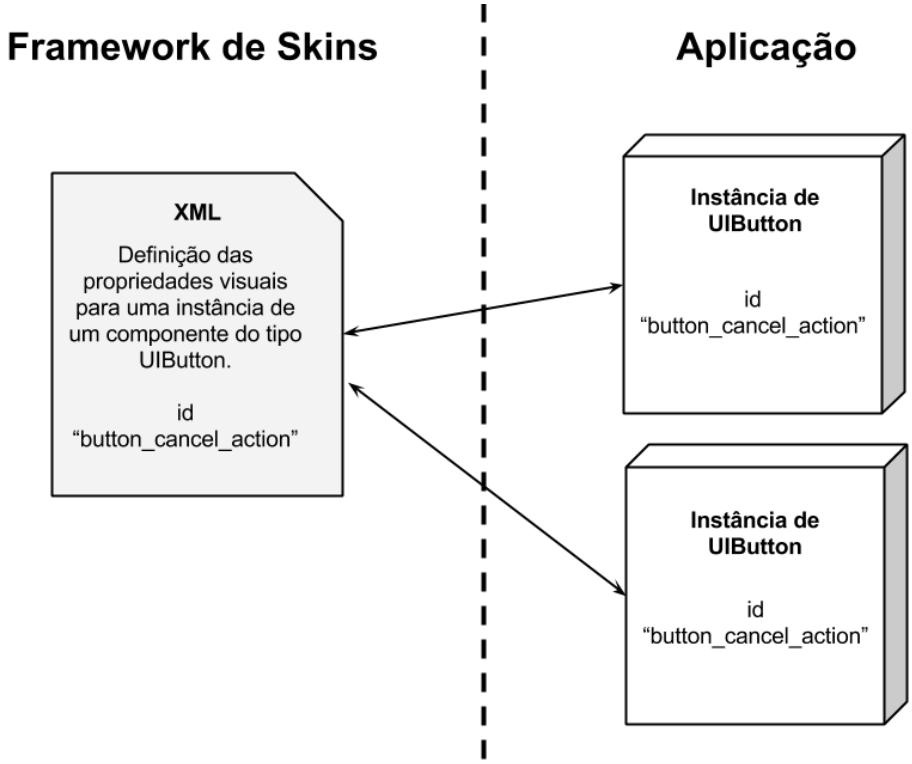

Figura 5 - Relação entre a instância de um objeto e a especificação de UI

Depois de ter a informação sobre o valor de uma propriedade de um determinado *Component* a Framework vai procurar por instâncias que tenham o respectivo identificador e que são do tipo do *Component* definido no XML. Depois de ter essas instâncias, a Framework aplica a *Operation* a cada instancia alterando assim o valor da propriedade em cada instancia.

Em seguida será explicada a estrutura do ficheiro XML.

5.1.5 Ficheiros de configuração em XML

Este subcapítulo pretende descrever a estrutura de um ficheiro de configuração em XML de uma Skin. No final, o leitor irá perceber como é possível personalizar um botão (UIButton) através de um ficheiro de configuração.

A imagem seguinte representa o modelo mais simples, do ficheiro de configuração em XML da estrutura de uma Skin. A *root* de cada ficheiro de configuração é definida com a *tag theme*. Em seguida, surge uma lista dentro da *tag instances* onde estão definidos todos os componentes e as suas respectivas propriedades e valores.

```
<theme>
    <instances>

        <!-- Lista com vários componentes -->

        <tipo_do_componente id="identificador">

            <!-- Lista com várias operações -->
            <operation name="identificador_da_operação">
                <param name="nome_do_parametro">
                    <tipo_do_objeto>valor_do_objeto</tipo_do_objeto>
                </param>
            </operation>

        </tipo_do_componente>

    </instances>
</theme>
```

Figura 6 - Ficheiro de configuração em XML

A tabela seguinte tem o objectivo de explicar de uma forma mais objectiva cada parte do ficheiro XML.

Tag ou parâmetro	Descrição
theme	Representa a *root* do ficheiro de configuração.
instances	Lista todos os *Componentes* que vão ser personalizados pela Framework.
tipo_do_componente	Representa o tipo de *Component*. Pode ser uma UIView, UILabel, etc.
identificador	Representa o identificador que permite mapear entre a instância do componente presente na Aplicação e o respectivo *Component* contendo a informação sobre as suas propriedades.
operation	É uma *tag* que define o começo da definição de uma *Operation* para um determinado *Component*.
identificador_da_operação	Cada *Operation* tem um determinado identificador de forma a que possa definido no XML. Por exemplo, a *Operation* que tem a lógica relativa à mudança da cor do fundo de uma UIView pode ter como identificador a *string* "backgroundColor".
param	É uma *tag* que defino o começo da definição do *input* da *Operation*.
nome_do_parametro	Cada input de uma determinada *Operation* tem um identificador associado. Por exemplo, a *Operation* que tem a lógica relativa à mudança da cor do fundo de uma UIView pode ter como nome para o parâmetro de entrada a string "color".
tipo_do_objecto	Esta é uma *tag* que representa o tipo do parâmetro de entrada. Por exemplo, a *Operation* que tem a lógica relativa à mudança da largura do contorno de uma UIView pode ter como identificador o tipo de parâmetro de entrada a string "float", visto que o que vai receber como *input* é um valor numérico.
valor_do_objecto	Aqui vai descriminado o valor da parâmetro de entrada. Por

	exemplo, a *Operation* que tem a lógica relativa à mudança da largura do contorno de uma UIView pode ter como valor do parâmetro o número "20".

Tabela 5 - Descrição do ficheiro de configuração em XML

Tendo agora conhecimento de como é a estrutura de um ficheiro de configurações de uma Skin, será explicado, através de um exemplo, como é que se consegue alterar a cor de fundo de um botão (UIButton).

```
<theme>
    <instances>
        <uibutton id="button_cancel_action">
            <operation name="backgroundColor">
                <param name="color">
                    <color format="rgb">1.0,0.0,0.0</color>
                </param>
            </operation>
        </uibutton>
    </instances>
</theme>
```

Figura 7 - Configuração da UI de um UIButton

Tag ou parâmetro	Descrição
uibutton	Representa o tipo de *Component*.
button_cancel_action	Representa o identificador que permite fazer a relação entre a instância do componente presente na Aplicação e o respectivo *Component* contendo a informação sobre as suas propriedades.
backgroundColor	Nome da operação que será feita no UIButton.
param name color	Nome que identifica o *input* da *Operation*.
param type color	Tipo do parâmetro de entrada.
format RGB	Define uma propriedade do *Object* que vai ser o *input* da *Operation*.
1.0, 0.0, 0.0	Cor vermelha. Representa o valor do *Object* que vai ser o *input* da *Operation*.

Tabela 6 - Descrição da configuração de um UIButton

5.1.6 *Assets*

Existem também certos tipos de propriedades que consistem em imagens, ou seja, pode haver uma imagem a definir o background de uma UIView. Por isso, faz sentido que possam existir imagens diferentes para cada Skin.

Quando, a partir do um ficheiro de configuração de uma determinada Skin, se aplica um determinado *asset* será obrigatoriamente usado o *asset* pertencente a esse tema.

5.1.7 Estrutura de ficheiros

As Skins estão estruturadas em pastas com a extensão ".theme". Dentro dessas pastas encontram-se os ficheiros de configuração em XML, os *assets* e um ficheiro com a informação básica da Skin, como por exemplo, o seu identificador e o nome.

5.1.8 Carregamento e aplicação de um determinado tema

Quando a Framework inicia começa automaticamente a procurar todas as pastas com a extensão ".theme". Para cada pasta que encontrar ela cria um objecto que irá guardar a informação básica relativa a cada Skin, como por exemplo, identificador, nome, *path*, etc. Num outro ficheiro de configuração está definida a Skin que deve ser considerada *default* para quando a aplicação é iniciada. Após saber qual a Skin default é feito o carregamento de toda a informação presente em cada XML de configuração da respectiva Skin. Tendo esta informação em memória, fica possível mapear instâncias de componentes com a sua respectiva configuração no à Skin diz respeito.

5.1.9 Processo de personalização de um componente – Ponto de vista da Framework

Em seguida, serão enumerados todos os passos que a Framework executa de forma a aplicar uma Skin a um determinado objeto (para este exemplo será utilizada uma UIImageView onde será colocada uma imagem que é diferente consoante a Skin):

1. A Framework é solicitada para personalizar uma UIImageView, para isso são passados para a Framework a instância de uma UIImageView e o identificador definido no ficheiro de configuração em XML;
2. A Framework guarda uma referência fraca para esse objecto numa lista;
3. Obtém o *Component* relativo ao objecto passado, ou seja, a uma UIImageView;
4. Obtém o objeto com o mesmo identificador que o que foi passado para a Framework, que guarda a informação relativa às operações que o utilizador introduziu no XML. Esse objecto é intitulado de *Skin Entity*.
5. Percorre todas as operações definidas pelo utilizador no XML e verifica se são compatíveis com o *Component* em questão;
6. Obtém o objeto *Operation* relativo à operação em questão, sendo neste caso a operação cuja lógica atribui uma imagem a uma UIImageView;
7. Obtém a imagem com o nome definido no XML. Essa imagem encontra-se guardada na pasta *assets* e pode ser diferente de Skin para Skin.
8. Tendo a imagem, basta aplicar a imagem à UIImageView;

5.1.10 iPhone e iPad

Cada ficheiro de configuração e respectivos *assets* tem uma terminação no seu nome orientada ao dispositivo. Isto é, os ficheiros pertencentes ao iPhone terminam em "~iPhone", enquanto que os relativos ao iPad terminam em "~iPad".

Assim, quando a Framework é iniciada, ela detecta em que tipo de dispositivo está e consoante o dispositivo carrega os ficheiros de configuração corretos.

5.1.11 Mudar Skin sem desligar a aplicação

Um dos requisitos da Framework consistia na possibilidade de mudar de Skin sem ter de desligar a aplicação. Para isso, sempre que é aplicado um tema a um objecto, a Framework guarda uma referência fraca para esse objecto e o respectivo identificador da respectiva

personalização. Assim, após carregar o novo tema, a Framework irá percorrer todas as referências e aplicar a respectiva personalização.

5.1.12 Herança

Durante os desenvolvimentos relativos à integração da Framework com a solução RCS para iOS da WIT Software, S.A. e devido ao aparecimento de novas Skins, surgiu um problema. O problema estava relacionado com o facto de alguns *assets* e configurações presentes em duas Skins distintas estarem a ser replicadas, trazendo problemas ao nível do espaço ocupado. Assim surgiu a necessidade de criar uma Skin base onde todos os recursos que fossem partilhados estariam e todas as Skins herdavam dessa Skin.

Este problema foi resolvido introduzindo uma propriedade no ficheiro de configuração do tema onde é especificado o identificador do tema que é herdado.

Quando o tema é carregado, ele carrega também o tema com o identificador do tema herdado, conseguindo assim, usar recursos e configurações do outro tema (*instances, classes* e constantes).

5.1.13 Processo de personalização de um componente – Ponto de vista do programador

A solução final tornou a personalização bastante simples, tornando possível personalizar um componente em dois passos.

De seguida é apresentado todo o processo necessário para personalizar um objecto, neste caso uma UIView.

1. Em primeiro lugar é necessário ir ao XML de personalização e adicionar a entrada respectiva à UIView com as *Operations* identificador respectivo;
2. No código, após a criação da UIView basta invocar a API que a Framework oferece de forma a personalizar o componente. Tendo para isso, de passar a isntância do componente que é para personalizar e o identificador único que foi definido no XML de personalização;

A seguinte imagem apresenta o código, em Objective-C, que exemplifica o passo número dois descrito anteriormente.

```
UIView *exampleView = [UIView new];
[[WMCThemes sharedThemes] themeView:exampleView withIdentifier:@"identificador_no_xml_de_configuração"];
```

5.1.14 Arquitetura

A figura abaixo representa a arquitetura da Framework de Skins.

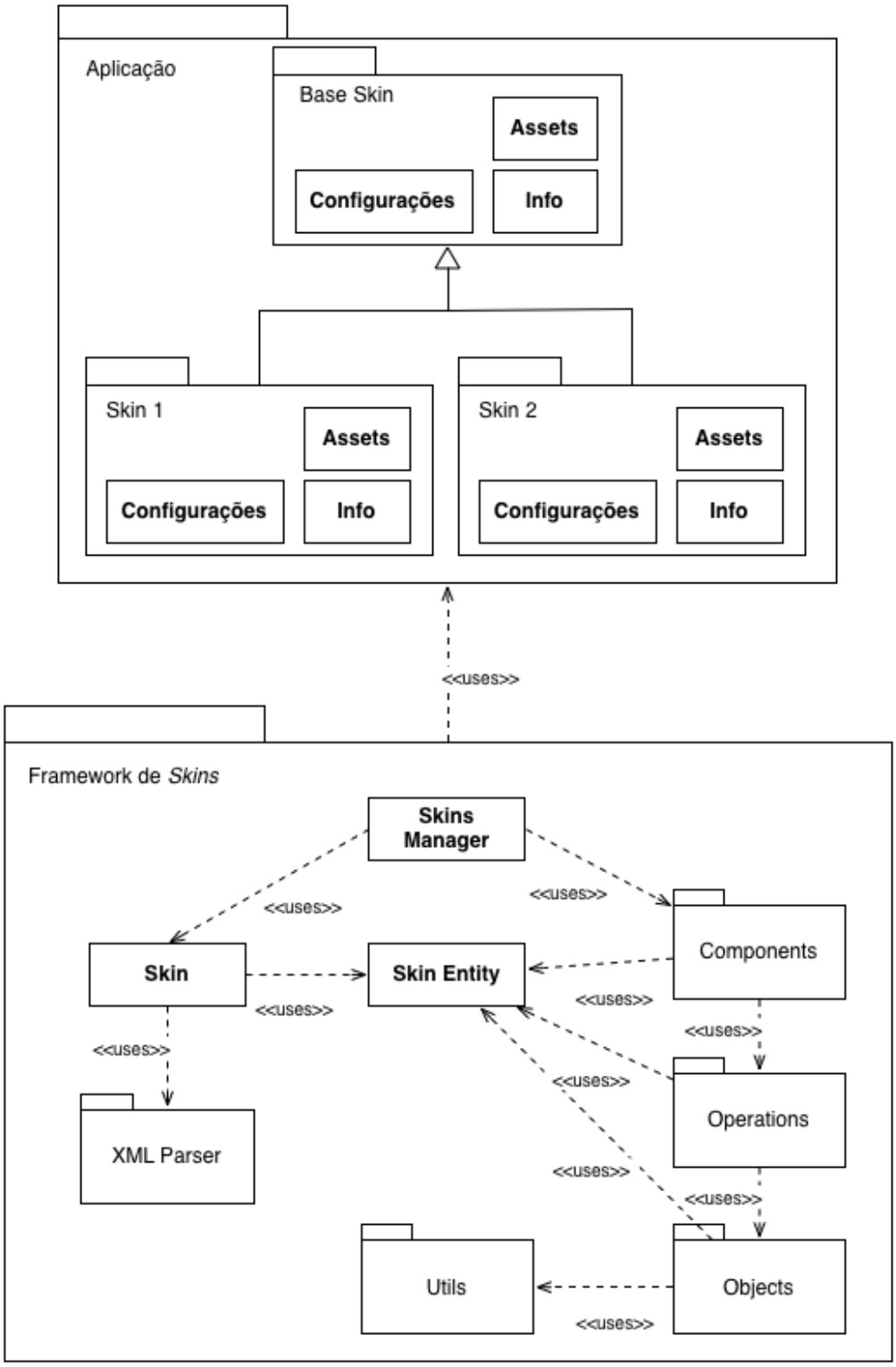

Figura 8 - Arquitetura da Framework de Skins

5.1.15 Potencialidades

A Framework, estruturada como está, permite definir bastante lógica através do ficheiro de configuração. Um dos principais desenvolvimentos consistiu em adicionar uma *flag* como atributo de uma imagem que permite informar a Framework se deve fazer cache daquela determinada imagem. Isto fez com que o acesso a imagens que estavam constantemente a ser utilizadas fosse mais rápido.

5.1.16 Conclusão e Trabalho Futuro

Todos os objectivos desta tarefa foram cumpridos, resultando então uma Framework que permite personalizar qualquer aplicação iOS com diferentes Skins.

Apesar da Framework se encontrar funcional e cumprir os requisitos, existe a possibilidade de a melhorar em diferentes aspectos. Um dos aspectos está relacionado com a performance de execução de todo o processo de *load* de uma Skin. Este processo ocorre quando a aplicação é iniciada ou quando é solicitada a alteração da Skin atual, qualquer melhoria na performance é vital pois tem impacto direto no tempo de arranque da aplicação. Neste processo, entre outras coisas, é feito o *parse* dos ficheiros de XML (este *parse* é bastante lento). Uma das soluções seria ter uma ferramenta que transforma esse XML num ficheiro binário. Assim, iria permitir que fazer o *load* de uma Skin de uma forma mais rápida pois seria possível ler estruturas de dados completas desse ficheiro binário.

A solução apresentou o nível versatilidade desejado estando por isso a ser usado em mais projetos desenvolvidos pela WIT Software, S.A..

5.2 Alterar a UI de Chat de forma a suportar Skins

Após o desenvolvimento da Framework de Skins toda a aplicação teve de ser integrada com a Framework. Cada elemento da equipa ficou responsável por integrar a UI de uma dada funcionalidade, tendo sido incumbida ao autor a UI relativa à funcionalidade de Chat.

O Chat é a principal funcionalidade da solução RCS para iOS da WIT Software, S.A., sendo assim a sua *user interface* e *user experience* apresentam requisitos rigorosos.

5.2.1 Problema

A primeira versão da solução RCS para iOS da WIT Software, S.A. foi implementada num curto espaço de tempo, devido ao facto da WIT Software, S.A. estar presente no concurso da GSMA que viria a ganhar. Devido a isso, nem sempre foram tomadas as melhores opções de *design* e implementação, fazendo com que a solução se tornasse incomportável com o decorrer do tempo.

Existiam vários problemas relacionados com a implementação existente à data. A vista de Chat foi construída em Javascript com o auxílio de uma UIWebView[32], que consiste num componente nativo para apresentar conteúdo Web. Assim, apenas pessoas que tivessem conhecimento de programação Web poderiam tentar desenvolver algo no Chat, fazendo disso uma enorme limitação ao desenvolvimento dessa componente. Um problema não menos importante passava pelo simples facto de que devido ao Chat estar construído em JavaScript inviabilizava a utilização da Framework de *skins*.

Por isso, o autor teve de construir uma solução de raiz de forma a que ela suportasse *skins* e que corresponde-se aos requisitos rigorosos de *user interface*, *user experience* e performance.

5.2.2 Objectivos

Este trabalho teve duas principais tarefas: a primeira consistiu em reconstruir a vista de Chat e a segunda passou pela integração da UI de Chat com a Framework de Skins.

Devido aos problemas encontrados na integração com a solução anterior foi decidido refazer essa solução. Foram definidos os seguintes objectivos para essa tarefa:

- **Performance:** a UI do Chat deve ser bastante fluida. Para isso, terá de ser possível fazer *scroll* da lista de Chat sem qualquer sinal da UI sofrer algum tipo de dificuldade em desenhar. O teste terá de ser realizado num iPhone 4 com o iOS 7;
- **Customização:** a UI do Chat deve poder ser customizada sem que com isso prejudique a performance;
- **Design e implementação:** o autor deverá ter especial cuidado com a solução encontrada de forma a que a UI possa ser alterada/expandida de uma forma rápida e relativamente fácil;

5.2.3 Design e solução

Na solução RCS para iOS sempre que possível é seguido o modelo de arquitetura *Model View Controller* (MVC)[2]. Sendo assim, o Chat é composto por *Models*, *Views* e *Controllers*.

No que aos *Controllers* diz respeito, existe um *Controller* para cada família de dispositivos, isto é, um *Controller* para o iPhone e outro para o iPad. Cada um destes *Controllers* tem como objectivo definir o *layout* das *Views* que dele fazem parte, bem como, a lógica especifica de cada família de dispositivos.

Cada um dos *Controllers* faz uso da mesma *View*. Nessa *View* são apresentadas as mensagens do Chat em forma de lista vertical.

Os *Models* representam os vários tipos de mensagens que podem ser apresentadas na lista de mensagens. As mensagens podem ser dos seguintes tipos:

- Texto;
- Localização;
- Contactos em forma de vCard;
- Stickers;
- Transferência de ficheiros de áudio, vídeo, imagem ou outros;

Estes são os tipos de mensagens existentes à data, no entanto existe sempre a possibilidade de introdução de novos tipos.

Neste momento é possível perceber que existe um *Controller* que possui uma *View* que por sua vez apresenta, em forma de lista, a informação contida em *Models*.

O componente do SDK de iOS escolhido para ser a *View* denomina-se de UITableView[33]. A UITableView é um componente nativo que permite apresentar conteúdo na forma de uma lista vertical.

A implementação da UITableView pressupõe que existe um objecto que lhe fornece a informação que ela necessita para apresentar o conteúdo de cada linha da lista. Por outras palavras, a UITableView necessita de alguém (objecto) que lhe forneça o *Model* de cada linha que ela tem de apresentar. Esse objecto é denominado de *Data Source*.

O *Data Source* da UITableView tem como objectivo fornecer à UITableView todas as mensagens que ela precisa de colocar na interface da aplicação. Sendo assim, o *Data Sourece* é o responsável de obter as mensagens de Chat que se encontram guardadas na COMLib e guardar essa informação no *Model* respectivo.

A UITableView sempre que necessita de apresentar uma mensagem na interface da aplicação irá pedir ao seu *Data Source* a respectiva mensagem.

Existem dois tipos de *Data Source*: um para o *Single Chat* – Chat em que apenas existem dois intervenientes, numa relação 1-para-1 – e outro para o *Group Chat* – Chat em que existem mais que dois intervenientes, numa relação de 1-para-N. A existência destes dois tipos deve-se ao facto de a API da COMLib ser diferente para cada uma do tipo de Chat.

Falta apenas mencionar que cada linha da UITableView consiste num objecto chamado UITableViewCell[34]. A UITableViewCell é uma UIView que representa a área de uma linha de uma UITableView. Cada UITableViewCell tem a responsabilidade de desenhar o tipo de UI especificado para cada tipo de mensagem. Por exemplo, para uma mensagem do tipo de texto existirá uma UITableViewCell que irá apresentar unicamente o texto da mensagem e a data da mensagem (de acordo com uma especificação de UI), ou ainda, se uma mensagem for do tipo File Transfer de imagem existirá uma UITableViewCell que irá mostrar a imagem recebida/enviada e data.

Depois desta breve explicação de como está estruturada a vista de Chat segue então uma explicação mais detalhada da solução, começando com a Arquitetura.

5.2.4 Arquitetura do Chat

O subcapítulo anterior funcionou como um introdução à estrutura da vista de Chat. Este subcapítulo tem o objectivo de tornar mais visual a explicação anteriormente descrita.

Sendo assim, a figura seguinte ilustra a arquitetura do Chat da solução RCS para iOS da WIT Software, S.A..

A azul estão identificados os componentes que foram desenvolvidos na integra pelo autor. De salientar que além dos componentes de cor azul o autor também esteve envolvido no desenvolvimento dos *Data Sources* e *Models* e de alguma lógica no Core Data.

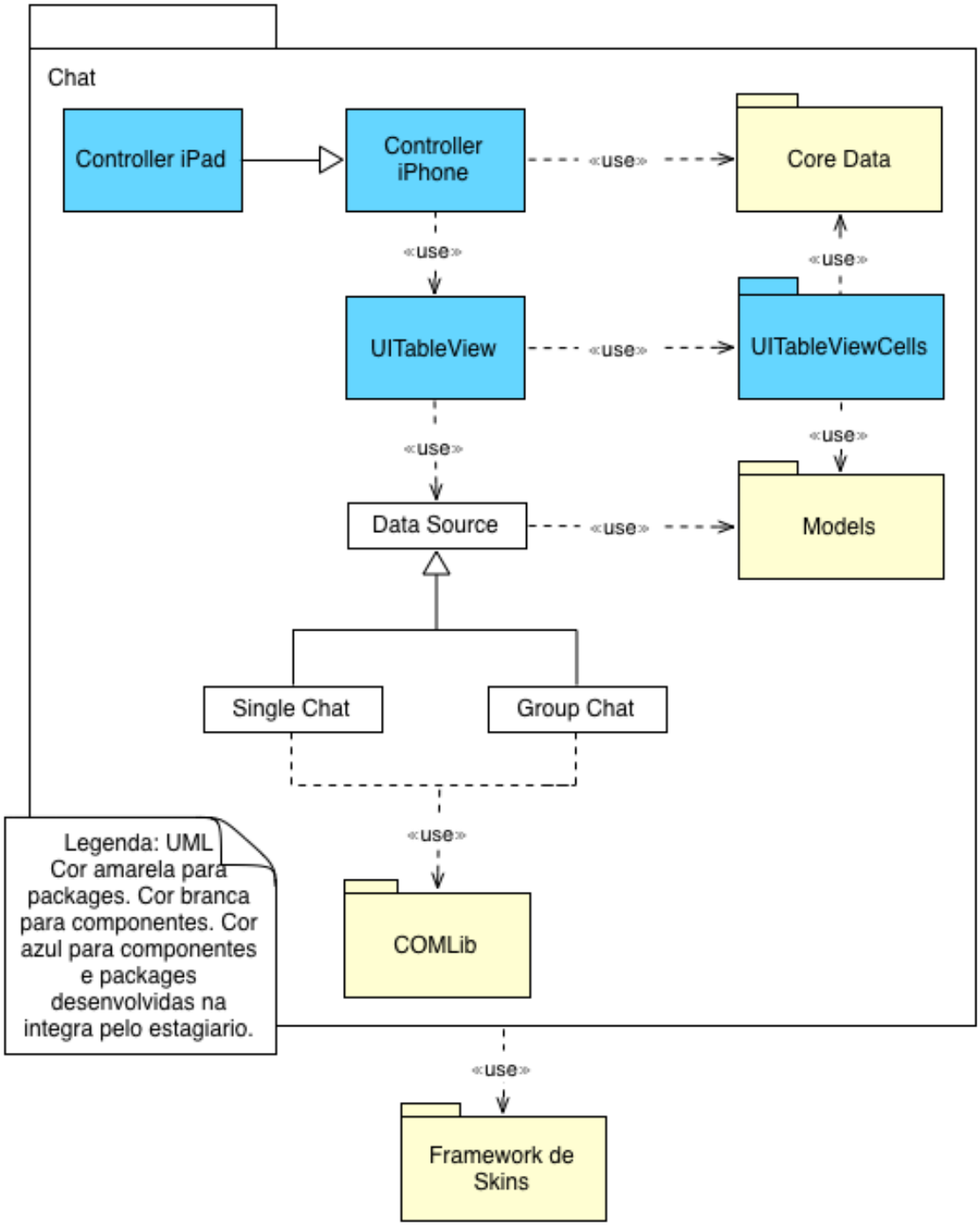

Figura 9 - Arquitetura do Chat

Componente	**Descrição**
Core Data	Consiste num *wrapper* nativo de acesso ao SQLite.
Controller iPhone	Este é o *Controller* que faz por apresentar a vista de Chat e serve de intermediário entre os *Models* e as *Views*, de forma a notificar alterações nos *Models*. Determinados

	componentes de UI do iPhone são distintos dos de iPad, por isso, existe um *Controller* para iPhone e outro para iPad. O *Controller* comunica com o Core Data para persistir e obter as imagens de background do Chat definidas pelo utilizador.
Controller iPad	Este *Controller* herda do Controller iPhone e apenas altera o *layout* relativos às *Views* que aparecem para o utilizador.
UITableView	A UITableView é um componente nativo que permite apresentar conteúdo na forma de uma lista vertical.
UITableViewCells	As UITableViewCells correspondem às *Views* que são apresentadas em cada linha de uma UITableView. Ou seja, estas são as *Views* que vão apresentar a informação relativa a um determinado *Model*. Estas *Views* comunicam com o Core Data de forma a persistir dados relacionados com os File Transfers (este tema encontra-se mais detalhado na próxima secção).
Single Chat Data Source and Group Chat Data Source	Estes componentes são os que interagem com a COMLib com o objectivo de obter a informação relativa a uma determinada mensagem. Como a API relativa ao Group Chat e ao Single Chat são distintas houve a necessidade de criar dois tipos de Data Sources.
COMLib	*Stack* proprietária desenvolvida pela WIT Software, S.A. que implementa a lógica de baixo nível relacionada com a lógica de comunicação de aplicações RCS.
Models	Os *Models* são os objetos que mantêm a informação relativa a cada tipo de mensagem.
Framework de Skins	Toda a UI necessita de ser personalizada, por isso, sempre que um componente é adicionado à UI existe a necessidade de comunicar com a Framework de Skins para o tornar personalizável.

Tabela 7 – Descrição da arquitetura do Chat

5.2.5 Desafios e Soluções

Nesta secção são descritos os vários desafios e soluções que foram encontrados durante a execução desta tarefa. Os tópicos consistem no cálculo da altura de cada UITableViewCell, na *cache*, no *reload data*, na *observer pattern* e na integração com a Framework de Skins.

5.2.5.1 UITableViewCell e reutilização de *Views*

Como foi já descrito anteriormente, a UITableViewCell é a UIView responsável por apresentar o conteúdo das mensagens do Chat. Existe uma UITableViewCell para cada tipo de mensagem de Chat, isto porque cada tipo de mensagem tem as suas características e especificidades ao nível da interface.

É comum que ao fim de um tempo o histórico de uma conversa de Chat entre duas pessoas atinja milhares de mensagens. Visto que cada UITableViewCell representa uma mensagem faz sentido dizer que, para este caso, o número de UITableViewCells existentes em memória é igual ao número de mensagens de Chat, ou seja, milhares. No entanto isso nunca poderá acontecer, em tudo devido aos recursos limitados que um dispositivo móvel dispõe. Por isso, são aplicadas técnicas de forma a que num determinado instante não estejam alocadas mais que um conjunto aceitável de UITableViewCells. Em seguida será explicada com mais detalhe a técnica utilizada para resolver este problema.

A técnica mencionada anteriormente é denominada de reutilização de *Views* e baseia-se nestas duas ideias:

1. Só devem estar instanciadas as *Views* que estão visíveis na interface da aplicação;
2. O processo de instanciação de uma *View* é bastante pesado e por isso não deve consistir numa ação repetitiva;

Por isso, a solução passa por manter uma *pool* limitada de *Views*. Sempre que for necessário utilizar uma *View*, utiliza-se uma *View* disponível na *pool*. Quando a *View* deixa de ser necessária é adicionada novamente à *pool*. Assim, é possível evitar ter um número elevado de *Views* e cada *View* utilizada só é instanciada uma única vez, evitando a instanciação repetitiva. Segue a explicação de como esta solução foi implementada no Chat.

Quando a UITableView é instanciada são definidos os tipos de UITableViewCells que irão ser apresentados na tabela. Ao fazer isto, o iOS vai instanciar um conjunto de UITableViewCells para cada tipo e guarda-las numa lista interna de reutilização. Assim, sempre que for necessário usar um determinado tipo de UITableViewCell a UITableView vai utilizar uma instancia do tipo desejado que já esteja na lista interna de reutilização. Sempre que uma UITableViewCell desaparece do ecrã será adicionada novamente á lista interna de reutilização.

A aplicação desta técnica permitiu obter resultados bastante satisfatórios, no que à performance e uso de recursos diz respeito.

A figura seguinte pretende demonstrar de uma forma visual a explicação da estrutura de UITableViewCells descrita anteriormente neste capitulo.

Como o *layout* de cada mensagem varia consoante a mensagem seja *incoming* ou *outgoing* houve a necessidade de criar diferentes *Views* para esses tipos de mensagens.

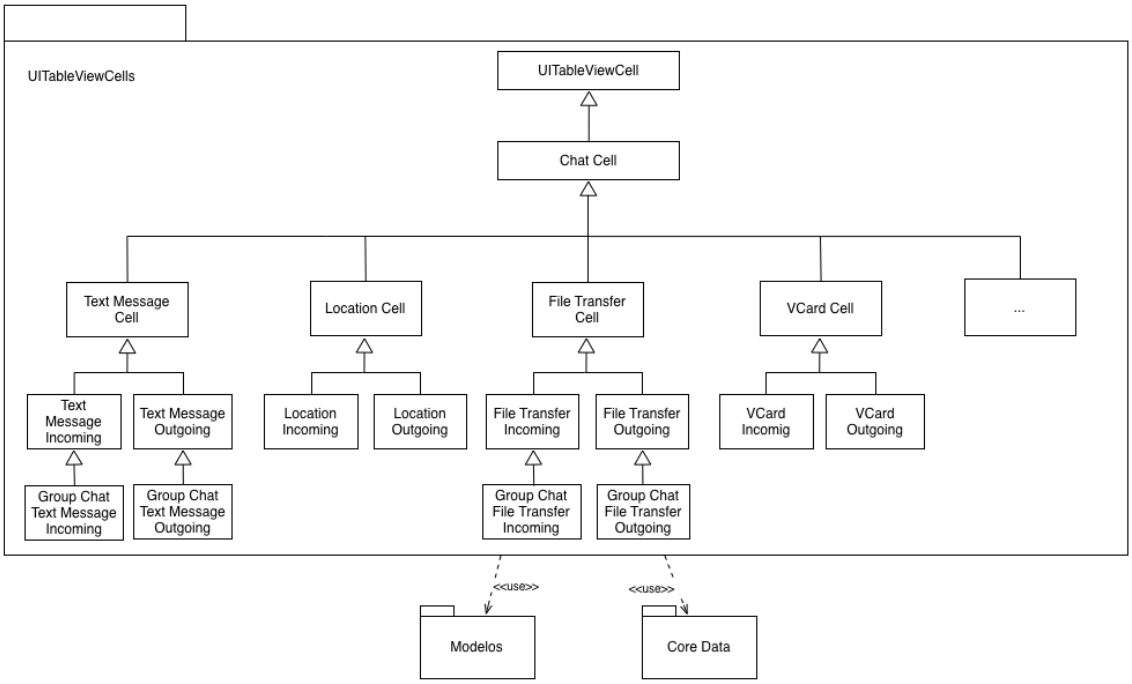

Figura 10 - Arquitetura das UITableViewCells do Chat

5.2.5.2 Calculo da altura de cada tipo de UITableViewCell

A UITableView antes de ser desenhada necessita sempre de saber qual vai ser a sua altura final. Pensa-se que uma das razões para a Apple ter desenhado a UITableView desta forma terá que ver com o facto de poder colocar o indicador de *scroll* na posição correta, visto que a posição do mesmo depende da altura total da UITableView.

Para saber a sua altura final, a UITableView invoca o método *tableView:heightForRowAtIndexPath:*, que lhe retorna a altura que cada linha irá ter. Assim, basta efetuar a soma das alturas de todas linhas para obter altura final da tabela.

Numa UITableView comum, cada tipo de UITableViewCell tem uma altura predefinida e estática. No entanto, no Chat, isso não acontece. Cada tipo de UITableViewCell tem uma altura especifica de acordo com o tipo e estado de uma determinada mensagem. Por exemplo, o tamanho de uma UITableViewCell para uma mensagem de texto depende do texto que irá conter. Por isso, houve a necessidade de perceber qual seria a melhor abordagem para devolver a altura correta de cada UITableViewCell antes sequer de ela aparecer na própria UITableView. Por questões de performance, os cálculos necessários para obter a altura de uma UITableViewCell para uma determinada mensagem têm de ser executados num tempo muito reduzido, porque terá impacto no tempo de carregamento da UITableView.

A tabela seguinte ilustra os pormenores de como é calculada a altura de cada tipo de UITableViewCell.

Tipo de UITableViewCell	**Descrição**
UITableViewCells cujo conteúdo	*CTFramesetterSuggestFrameSizeWithConstraints[35].*

dinâmico se caracteriza por ser na forma de texto	Este método devolve o CGSize (estrutura que contém dois valores, uma altura e uma largura) necessário para apresentar um determinado texto com determinadas propriedades (fonte, tamanho, emojis, emoticons, ...). É também possível definir limitações relacionadas com a altura ou largura máximas que é possível comportar. Através do valor da altura devolvido conseguimos saber o espaço que o texto irá ocupar e assim devolver a altura que a UITableViewCell irá ter.
As UITableViewCells que variam consoante o estado e conteúdo.	Os File Transfers são um exemplo deste tipo de UITableViewCells. A altura de um File Transfer é influenciada pela orientação da imagem que é para ser apresentada, isto é, se a imagem for em *Landscape* a UITableViewCell tem uma altura diferente do que se for em *Portrait*. O próprio estado do File Transfer tem impacto, imaginemos que o File Transfer está no estado *Transferring* (que representa que um determinado ficheiro está a ser transferido ou recebido) deverá existir uma barra de progresso por baixo da imagem que esta a ser transferida. Por isso, existe a necessidade de alocar espaço para essa barra de progresso. A solução passa por saber quais os elementos que têm de ser mostrados para um determinado estado e somar à altura da foto.
UITableViewCells que têm tamanho fixo.	É devolvido o tamanho que está definido no NIB[36].

Tabela 8 - Calculo da altura de cada tipo de UITableViewCell

Com esta informação, a tabela já se pode reajustar a este tamanho final e assim já se pode construir cada UITableViewCell de acordo com o tipo de mensagem que se pretende.

5.2.5.3 Auto Resising Masks

Existem duas formas de construir uma UITableViewCell, a primeira consiste na construção integral através de código, a segunda forma consiste numa forma híbrida entre código e o *interface builder* disponibilizado pelo SDK do iOS. A primeira abordagem tem o potencial de tornar mais complexa a tarefa de perceber como está estruturada cada UITableViewCell, isto porque, a segunda abordagem apresenta a estrutura da UITableViewCell de uma forma visual e, por isso, de mais fácil compreensão. Os ficheiros resultantes do *interface builder* tem o nome de XIB.

A fácil atualização dos componentes de UI de uma UITableViewCell merece uma grande atenção, visto que um dos objectivos é que a UI possa ser alterada de uma forma rápida e fácil. A utilização de XIBs já proporcionou um avanço no que diz respeito à concretização desse ponto dos requisitos, no entanto as Auto Resizing Masks (ARMs)[17] vêm elevar a solução a para outro nível.

As ARMs são constantes que se atribuem a uma UIView que têm impacto na sua posição e dimensões em relação à UIView que as contem – denominada de *superview*. Com as ARMs é possível indicar a uma UIView que quando a sua *superview* aumenta em altura esta deve aumentar de altura na mesma proporção e manter-se sempre centrada verticalmente.

Esta técnica permitiu diminuir de uma forma bastante considerável os cálculos que tinham de ser feitos para reajustar o tamanho e posições de elementos.

A figura seguinte permite ilustrar de uma forma simples uma UITableViewCell, para uma mensagem do tipo *Text Message*, com ARMs definidas.

O primeiro retângulo representa uma UITableViewCell sem qualquer ARM definida. Dentro da UITableViewCell é possível observar dois retângulos que representam duas UILabels. A UILabel superior é responsável por apresentar o texto da mensagem, a UILabel inferior representa a hora a que a mensagem foi recebida.

O Segundo retângulo permite ver a definição das ARMs para a UILabel onde é apresentado o texto da mensagem. As regras definidas são as seguintes:

- Manter a distância ao lado esquerdo da UITableViewCell;
- Manter a distância ao topo da UITableViewCell;
- Manter a distância ao lado direito da UITableViewCell;
- Manter a distância ao fundo da UITableViewCell;
- Aumentar ou reduzir a largura da UILabel de forma a respeitar as regras anteriores;
- Aumentar ou reduzir a altura da UILabel de forma a respeitar as regras anteriores;

Por fim são aplicadas ARMs à UILabel onde é apresentada a hora a que a mensagem foi recebida. Para esse caso as regras são as seguintes:

- Manter a distância ao lado esquerdo da UITableViewCell;
- Manter a distância ao lado direito da UITableViewCell;
- Manter a distância ao fundo da UITableViewCell;
- Manter a altura da UILabel;
- Aumentar ou reduzir a distância ao topo da UITableViewCell de forma a respeitar as regras anteriores;
- Aumentar ou reduzir a largura da UILabel de forma a respeitar as regras anteriores;

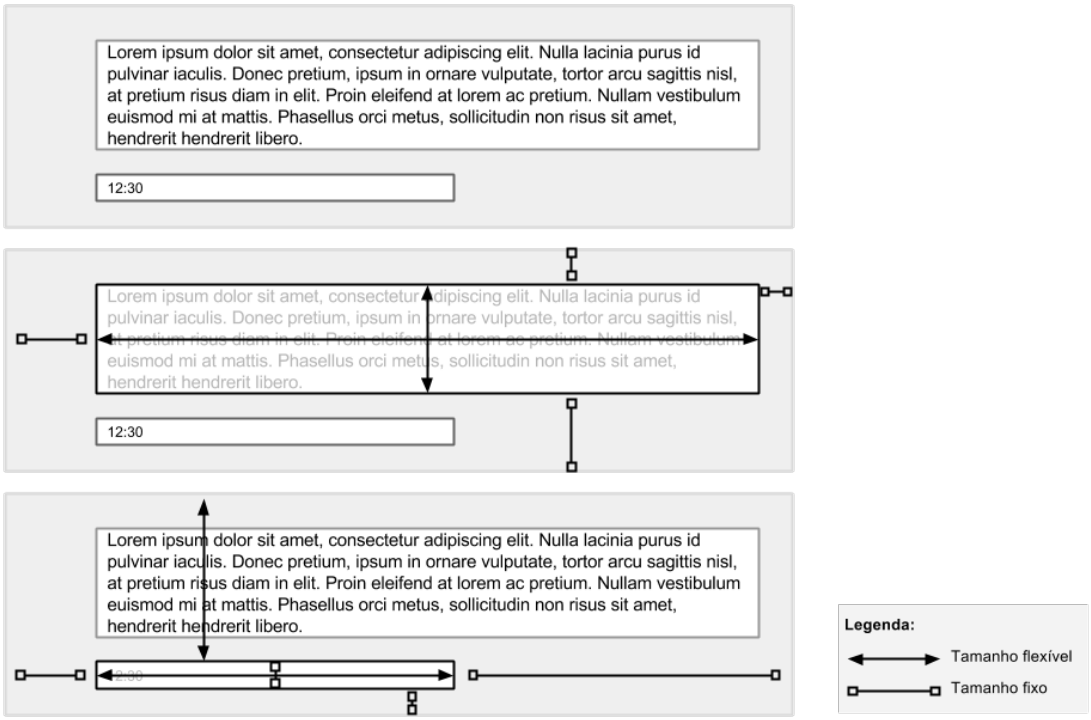

Figura 11 - Explicação de Auto Resizing Masks

A aplicação de ARMs reduziu a complexidade que seria necessária introduzir para calcular as posições e tamanhos que cada uma dos componentes da UITableViewCell deveriam ter. Neste caso de uma mensagem de texto a lógica não seria de complexidade elevada, mas quando surgissem mais componentes de UI dentro da UITableViewCell, toda a lógica teria de ser revista para ter em conta o novo componente. Utilizando as ARMs isso não é necessário facilitando assim o desenvolvimento.

5.2.5.4 Reload Data versus Observer Pattern

Para a UITableView a desenhar o conteúdo de cada mensagem existe uma única forma, que consiste em invocar o método *–(void)reloadData* [33]. Sempre que este método é invocado a UITableView, além de desenhar cada mensagem, vai chamar todos os métodos que lhe fornecem informação sobre si mesmo, como por exemplo, o método referido anteriormente que obtém a altura para cada UITableViewCell que permite à UITableView saber a sua altura final. No entanto, existem situações onde é garantido que o tamanho da UITableView não teve alterações e que a única alteração que existiu apenas teve impacto numa mensagem especifica. Exemplo disso, é quando o utilizador está a enviar uma imagem para um outro utilizador e recebe atualizações do progresso da transferência. Neste caso, o único valor que alterou consiste numa percentagem do ficheiro transferido que se reflete na UI pela atualização do valor presente na barra de transferências daquela mensagem especifica. Chegamos então à conclusão que, para certas e determinadas situações, o *reladData* executa uma quantidade de lógica desnecessária.

Esta lógica em excesso veio fazer com que, em situações em que a UI está constantemente a ser atualizada, a performance do Chat diminuísse para valores inaceitáveis, chegando a bloquear por momentos a UI da aplicação.

A solução encontrada passou por utilizar a design pattern "Observer Pattern"[3]. Esta design pattern permite que uma instancia de uma UITableViewCell fosse notificada quando uma determinada propriedade do *Model* (mensagem) correspondente sofresse alterações no seu valor. Um exemplo consiste em observar as alterações da propriedade relativa ao progresso de um File Transfer e alterar unicamente a barra de progresso.

A "simples" implementação desta *pattern* fez com que a UI atingisse os níveis de performance desejados, visto que reduziu drasticamente o número de invocações do método *reloadData*.

De salientar também que o facto de não se estar constantemente a fazer *reloadData* diminui o processamento da aplicação o que faz com que se poupe a bateria do dispositivo.

5.2.5.5 Cache e o Core Data

Todas as imagens e vídeos recebidas/os ou enviadas/os durante a execução de um File Transfer são guardados/as numa pasta dentro do Bundle da aplicação[37]. Sempre que um utilizador acedesse a um Chat para ver o histórico de uma determinada conversa seria necessário ler a imagem ou vídeo para poder apresentar o *thumbnail* respectivo. A leitura das imagens a partir do File System é na verdade um processo bastante lento, o que iria causar um grande impacto na fluidez do Chat. Devido a isso, foi necessário encontrar outras formas de tornar o acesso a essas imagens mais rápido.

A solução passou por usar duas caches em que uma delas é volátil e a outra é persistente. O tipo de informação guardada em cada uma é idêntica e consiste numa imagem com a resolução mínima necessária para aparecer no balão de Chat (quando se trata de um File Transder de um vídeo é guardada uma imagem de uma determinada *frame* do vídeo). Para a cache persistente é usado o Core Data[38] que consiste num wrapper nativo para aceder ao SQLite.

Os acessos à cache persistente não são rápidos o suficiente para satisfazerem os requisitos da Chat View e, devido a isso, houve a necessidade de uma cache temporária de rápido acesso.

A segunda cache consiste numa cache em memória que permite aceder às imagens de uma forma bastante rápida, possibilitando assim a fluidez necessária para a tabela de Chat. A informação presente em cada cache é limpa com um intervalo de tempo diferente para cada uma. A cache persistente é limpa de acordo com um valor definido num ficheiro de configuração. A cache de memoria é preenchida quando se entra na vista de Chat e é limpa quando se sai, tendo um limite definido para um máximo número de imagens em simultâneo na cache.

Em conclusão, existem duas caches. A primeira cache é persistente e é onde são guardadas imagens com a resolução mínima necessária para aparecerem nos balões do Chat, evitando assim estar sempre a gerar imagens com essa resolução. A segunda cache é uma cache volátil para efeitos de performance.

5.2.6 Integração com a Framework de Skins

Durante o processo relacionado com a construção de cada UITableViewCell já foram tidas em conta as necessidades relativas ao suporte de Skins. O resultado final pode ser observado pelas seguintes imagens, que demonstram o Chat com diferentes Skins.

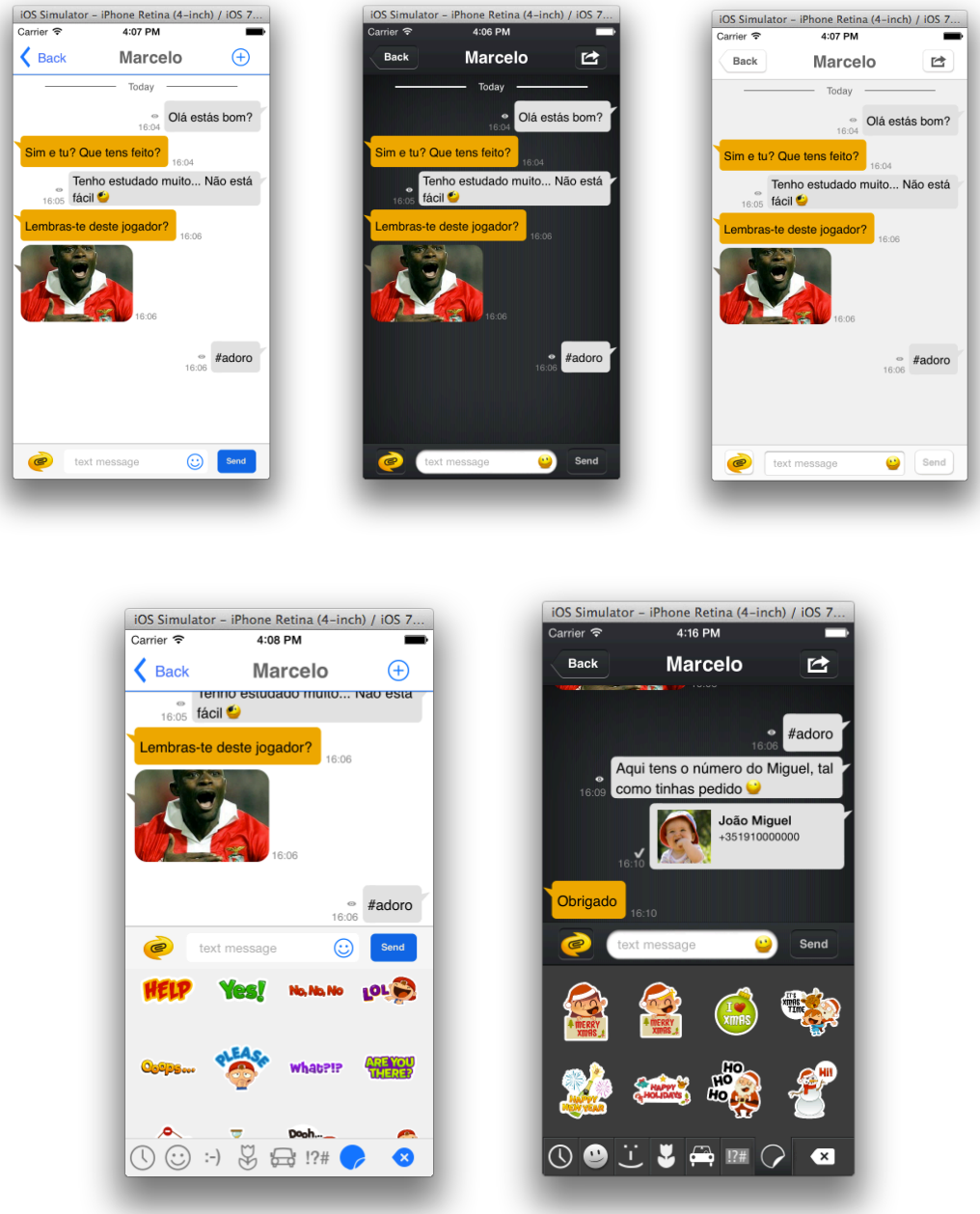

Figura 12 – Temas Preto e Branco do WIT Mobile Communicator para iOS

5.2.7 Conclusão

Esta tarefa apresentava um grau de responsabilidade máximo, isto porque se iria alterar toda a estrutura da *View* de Chat da aplicação que consiste na principal funcionalidade da aplicação.

Ao longo dos desenvolvimentos, foram surgindo desafios de *design* e *performance* que tiveram de ser ultrapassados.
A nova solução desenvolvida cumpriu todos os objetivos propostos, possibilitando assim aplicar o suporte à Framework de Skins com o nível de performance desejado.

5.3 Desenvolvimento de uma loja que permite fazer download de conteúdos na solução RCS para iOS

5.3.1 Introdução

Esta tarefa consiste em desenvolver uma loja, para a solução RCS para iOS da WIT Software, S.A., que permita fazer o download de conteúdos.

5.3.2 Objectivos

O primeiro objectivo consiste em disponibilizar um local na aplicação onde o utilizador possa visualizar o conteúdo e respectiva informação, disponível para download. O utilizador deverá poder ver informação detalhada sobre cada conteúdo.

Após fazer o download de um determinado conteúdo o utilizador deverá poder utilizá-lo e, caso pretenda, desinstalá-lo.

A solução terá de ser suficientemente dinâmica de forma a poder introduzir um novo tipo de conteúdos quando for necessário.

Em forma de síntese, os objectivos para a loja são:

- A solução deverá apresentar o conteúdo disponível para download;
- Deverão ser apresentadas informações detalhadas sobre cada conteúdo;
- Deverá ser possível Fazer download do conteúdo disponível;
- Deverá ser possível utilizar o conteúdo ao qual foi feito o download;
- Deverá ser possível desinstalar conteúdos;
- A solução deverá estar preparada para a introdução de novos tipos de conteúdos;

5.3.3 *User Interface*

A equipa de design da WIT Software, S.A. especificou UI que a loja deverá ter. Em seguida serão apresentadas três imagens com o objectivo de ilustrar o aspecto especificado para a loja.

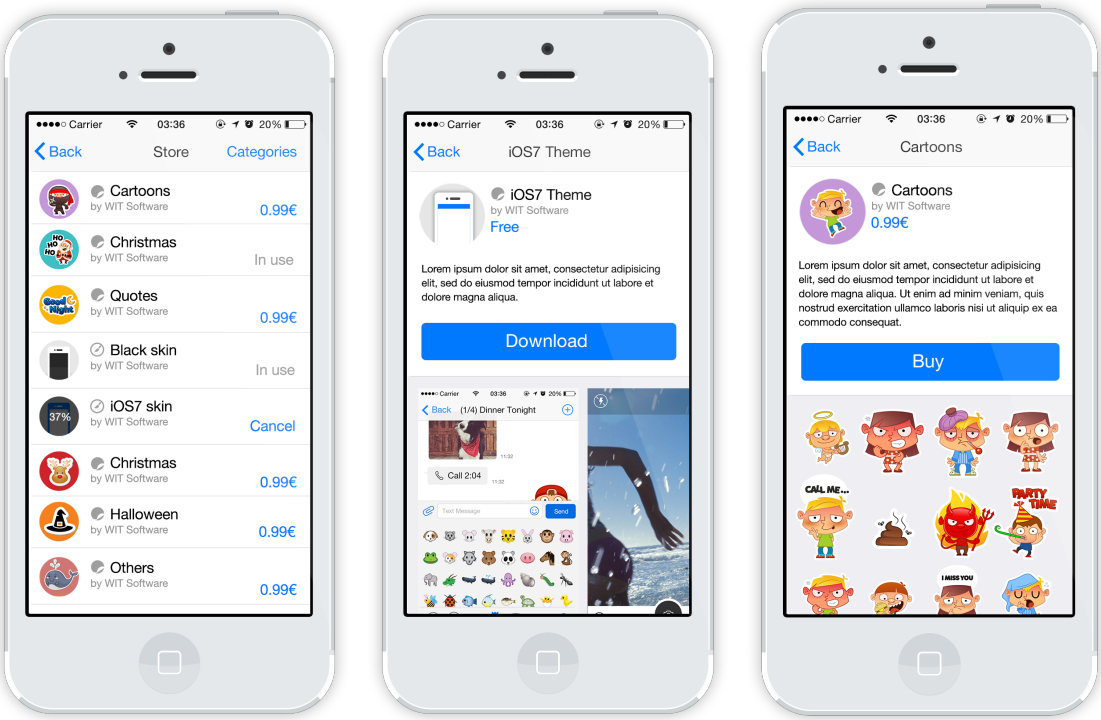

Figura 13 - UI da loja de conteúdos

5.3.4 Solução

Este capitulo tem como objectivo explicar com detalhe a solução desenvolvida.

Em primeiro lugar, de forma a fazer com que a explicação da solução seja de simples compreensão para o leitor, vamos partir do princípio de que os conteúdos que são disponibilizados na loja encontram-se alojados na rede e que existe forma de lhes aceder. No final da explicação da solução será detalhada a estrutura dos conteúdos na rede.

O primeiro objectivo para a loja consistia em apresentar os conteúdos disponíveis para download ao utilizador. Para isso, é necessário obter a informação sobre cada um dos conteúdos.

Com o intuito de fazer a gestão dos conteúdos disponíveis para download foi criado o *Store Manager*. Este *manager* é o responsável pela obtenção da informação de cada conteúdo.

Um conteúdo tem um *Model* associado, onde são guardadas todas as informações relativas. Existe um *Model* para cada tipo de conteúdo.

Para apresentar o conteúdo disponível para *download* foi seguido mais uma vez o modelo MVC. Neste caso foi criado um *Controller* que contém uma *View* cujo objectivo é apresentar uma lista vertical com a informação contida nos *Models*. Mais uma vez a UITableView é o candidato ideal para fazer o papel de lista vertical, por isso, foi a escolhida para ser a *View* do modelo. Ao contrário do Chat, esta UITableView só terá um tipo de UITableViewCells, visto que a interface de cada um dos tipos de conteúdos é idêntica, não fazendo assim sentido haver um tipo de UITableViewCell para cada tipo de conteúdo.

De igual forma foi tirado partido do mecanismo de reutilização de *Views* de forma a apresentar a melhor performance possível para a lista de conteúdos.
Por fim, a implementação de uma solução para apresentar a informação detalhada de um determinado conteúdo. A informação apresentada nesse ecrã pode ser vista no subcapítulo *user interface*.

A solução consistiu numa UIView com *scroll* – denominada de UIScrollView – de forma a que seja possível ver todo o conteúdo no ecrã. Consoante o estado do produto é apresentado um botão diferente. Nessa vista, de acordo com a informação de cada produto, é possível visualizar ou uma coleção de várias imagens, ilustrando o produto em ação na aplicação, ou uma imagem com o conteúdo de cada produto.

5.3.5 *Models*

Como seria de esperar, existem vários tipos de conteúdos. Sendo assim, existe a necessidade de criar um *Model* para cada um dos tipos. Os tipos de conteúdos existentes à data são os Stickers e as Skins.

No entanto, existem propriedades que são partilhadas entre todos os tipos dos conteúdos. Sendo assim existe uma *Class* denominada *ProductItem* que contem toda essa informação. A informação que é partilhada entre produtos está presente na seguinte tabela:

Propriedade	**Tipo**	**Descrição**
productIdentifier	String	O valor desta propriedade permite identificar o produto de uma forma única.
productName	String	Nome do produto. Este nome é o que é apresentado ao utilizador na loja.
productPrice	String	Representa o valor monetário do produto.
productResourceImageURL	URL	URL para a imagem de apresentação do produto.
productResourceURL	URL	URL que encaminha para um ZIP que contém os dados relativos ao produto.
productResourceSize	Int	Tamanho dos dados disponíveis em productResourceURL.
productDetailsURL	URL	URL que encaminha para um *json* que contem informação sobre o produto.
productDetailsInfo	String	Representa um texto informativo sobre o produto.
productDetailsSpriteImageURL	URL	URL para uma imagem onde será possível visualizar conteúdo do produto.

productDetailsPreviewImagesURLsArray	Array	Array que contem URLs para várias imagens que demonstram o produto integrado na aplicação.

Tabela 9 - Propriedades do *ProductItem*

Até à data não se verificou a existência de alguma propriedade que não seja partilhada entre produtos, o que não invalida o facto de haver necessidade de haver um *Model* para cada tipo. Isso deve-se a vários comportamentos que são distintos entre os vários tipos de produtos.

Quando visualizamos um produto na loja pode acontecer que já tenhamos comprado esse produto, não fazendo assim sentido ter de o comprar novamente. Por isso, foram criados vários estados para os produtos, de acordo com a seguinte tabela:

Estado	**Descrição**
Purchased	Produto que foi comprado com sucesso pelo utilizador
Downloaded	Produto que já se encontra presente na aplicação, ou seja, cujo *download* já foi realizado e não foi apagado.
Downloading	Produto que está a ser *downloaded* no momento. Neste instante existe uma tarefa a executar o download do conteúdo deste produto.
In Use	Alguns produtos têm a característica de poderem estar a ser utilizados no instante em que se está a navegar pela loja. Exemplo disso são as Skins. Este estado permite, entre outras coisas, informar o utilizador dessa situação.
Free	Os produtos que são classificados como gratuitos têm uma atenção especial visto que torna-se desnecessário a realização do processo de pagamento. Esses artigos são classificados como Free.

Tabela 10 - Estados de um produto

A determinação do estado do produto pode variar consoante o próprio produto em sim, por isso existe então a necessidade de ter um *Model* para cada tipo de produto para poder separar esta lógica especifica a cada um. De seguida, é apresentada uma tabela que explica como é obtido cada um dos estados para cada tipo de conteúdo.

Tipo de conteúdo	**Descrição**
Estado	

Purchased	
Stickers e Skins	É guardada uma lista onde constam todos os produtos comprados por um determinado utilizador.

Downloaded	
Stickers	Para saber se uma determinada *package* de Stickers está instalada na aplicação o *Model* pergunta ao Stickers Manager se existe uma *package* com um determinado identificador. A resposta do Manager indicará o estado deste conteúdo.
Skins	Para saber se uma determinada Skin se encontra instalada na aplicação o *Model* pergunta à Framework de Skins se existe uma Skin com um determinado identificador. A resposta da Framework indicará o estado deste conteúdo.
Downloading	
Stickers e Skins	Existe um Manager responsável por gerir todas as tarefas relacionadas com o *download* dos conteúdos. Para saber se está a ser feito o *download* de um conteúdo basta perguntar ao Manager se existe algum *download* para o conteúdo com um determinado identificador. A resposta do Manager indicará o estado deste conteúdo.
In Use	
Skins	A Framework de Skins retorna o identificador da Skin que está a ser utilizada num determinado instante. Se a resposta coincidir com o identificador da Skin deste conteúdo, significa que está a ser utilizada.
Stickers	Para este tipo de conteúdo não faz sentido saber se está a ser utilizado ou não. Por isso, este conteúdo nunca estará neste estado.
Free	
Stickers e Skins	A informação relativa a se um determinado produto é gratuito é originada do mesmo local

Tabela 11 - Critérios de escolha para cada estado dos produtos

5.3.6 Download de Conteúdo

O Download Manager foi criado com o intuito de gerir todos os *downloads* que estão a ser executados.

Sempre que é solicitado ao Manager para começar um *download* do conteúdo de um determinado produto ele lança uma *task* que fica encarregue de fazer esse mesmo *download* de uma forma assíncrona. Este Manager mantém um dicionário com o par *productIdentifier* – *task* de forma a poder controlar quais os *downloads* que estão a ser feitos. Quando uma *task* termina de fazer o *download* notifica o Manager e este retira a o par *productIdentifier* – *task* do seu dicionário.

A *task* consiste num conjunto de operações assíncronas, dependentes entre si, que garantem o *download* do conteúdo. Essas operações são explicadas na seguinte tabela:

Ordem	Operação	Descrição
1	*download*	Consiste na operação que faz o *donwload* dos conteúdo de um determinado produto. Nesta fase, o conteúdo está a ser guardado numa pasta temporária.
2	*unzip*	O conteúdo de um determinado produto vem comprimido no formato zip. Por isso, é necessário fazer *unzip*.
3	*save*	Esta tarefa move o ficheiro da pasta temporária para a pasta correcta.

Tabela 12 - Operações realizadas no download de um conteúdo

Durante o processo de *download* a UI da aplicação tem de ser notificada com a informação relativa ao *download*. Informação essa que vai desde a percentagem atual de *download* até a uma notificação de que o *download* terminou com sucesso ou com erro.

Para poder controlar todas estas variantes, cada *task* possui um conjunto de estados. As alterações nesses estados são observadas pelo Downloads Manager e sempre que ele detecta uma alteração comunica com as entidades que se registaram para receber notificações de um determinado *download*. De seguida, é apresentada uma tabela com os possíveis estados de uma *task*.

Estado	Descrição
Initializing	Neste estado a *task* está a estabelecer a conexão necessária para fazer o *download*.
Downloading	A conexão está estabelecida e a aplicação já recebeu informação sobre o *download*. Por isso já se encontra a transferir o ficheiro.
Downloaded	A transferência está concluída. No entanto a *task* ainda tem de fazer *unzip* e *save* do ficheiro.
Finished	A *task* já executou todas as operações com sucesso.
Cancelled	A *task* foi cancelada por ordem exterior.
Failed	A *task* terminou por falha numa das operações.

Tabela 13 - Estados de uma *task* de download de conteúdo

5.3.7 Framework de Skins

A Framework de Skins teve de ser alterada para de forma a ser possível realizar uma distinção das Skins que vêm predefinidas na aplicação e as que foram feitas download.

Para isso, foi feito um pequeno desenvolvimento que consistiu na criação de um ficheiro de configuração para a Framework. Este ficheiro é em XML e permite definir várias configurações para a Framework. Nesse ficheiro, torna-se possível definir as extensões das pastas que são referentes as Skins, podendo assim, ter diferentes extensões para Skins que vêm pré-instaladas e para Skins que são adquiridas na loja.

Sendo assim, é possível definir que as pastas que têm a extensão ".theme" são representativas de Skins que vêm pré instaladas na aplicação. E que as Skins originárias da loja têm a extensão ".purchasedTheme".

5.3.8 Stickers Manager

A solução existente à data, no que diz respeito à gestão dos Stickers, consistia na simples leitura de uma lista de imagens guardadas numa pasta especifica de uma Skin. Esta solução teve de ser refeita, já que todo o processo de gestão tomou diferentes contornos com a introdução de conteúdo proveniente da loja.

A nova solução vai procurar todas as pastas com a extensão ".stickers" presentes no bundle da aplicação. A cada pasta corresponde uma *package* de Stickers, que contem informação sobre a *package* e as imagens relativas aos Stickers.

A equipa de *design* também especificou um novo UI para os Stickers. As próximas imagens consistem nos *mockups* relacionados com o novo UI.

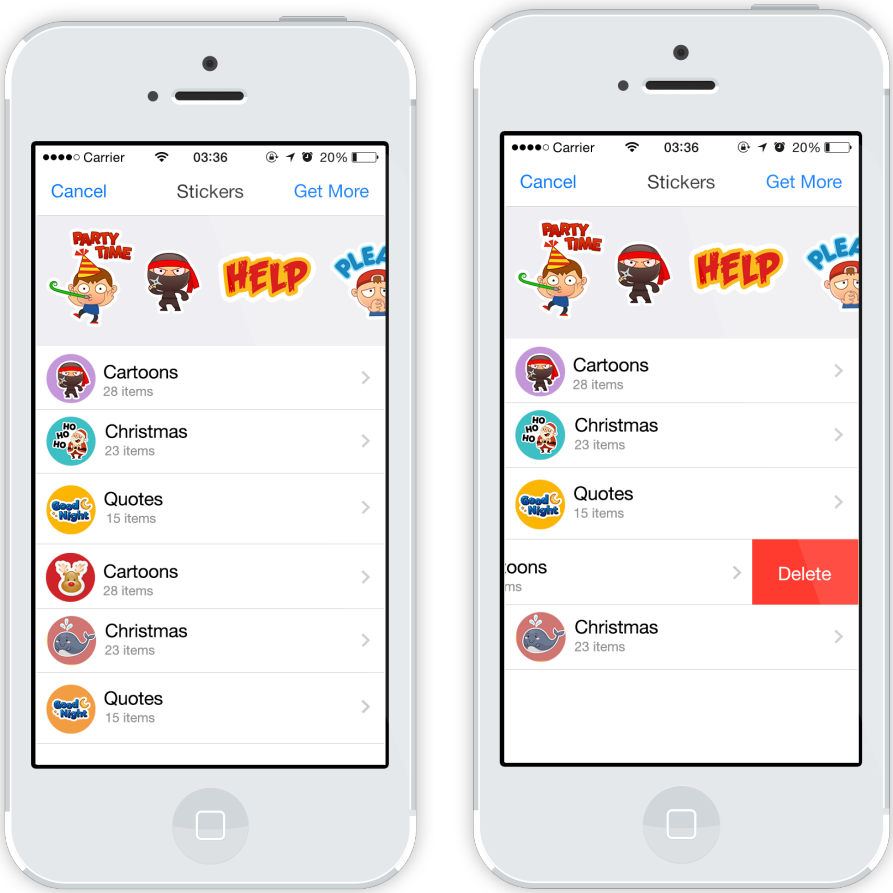

Tabela 14 - UI dos Stickers para a solução RCS para iOS

Consiste numa lista vertical onde surgem todas as *packages* de Stickers instaladas na aplicação. O utilizador consegue ver o número de Stickers que cada *package* tem, como pode também desinstalar uma *package* se assim o desejar.

No topo da lista é apresentada uma lista horizontal onde estão presentes os últimos Stickers enviados, ou seja, os mais recentes.

A lista vertical relativa às *packages* de Stickers corresponde a uma UITableView em que o seu *Data Source* obtém a informação das *packages* através do Stickers Manager.

O ato de desinstalar uma *package* de Stickers consiste na remoção da respectiva pasta do *bundle* da aplicação.

Os *Stickers* mais recentes utilizados pelo utilizador são persistidos no Core Data guardando a informação respectiva à *package* a que pertencem. Assim, sempre que uma *package* é removida, as respectivas entradas de Stickers no Core Data também são removidos.

5.3.9 Arquitetura da loja

A seguinte imagem pretende ilustrar a estrutura dos vários intervenientes na loja.

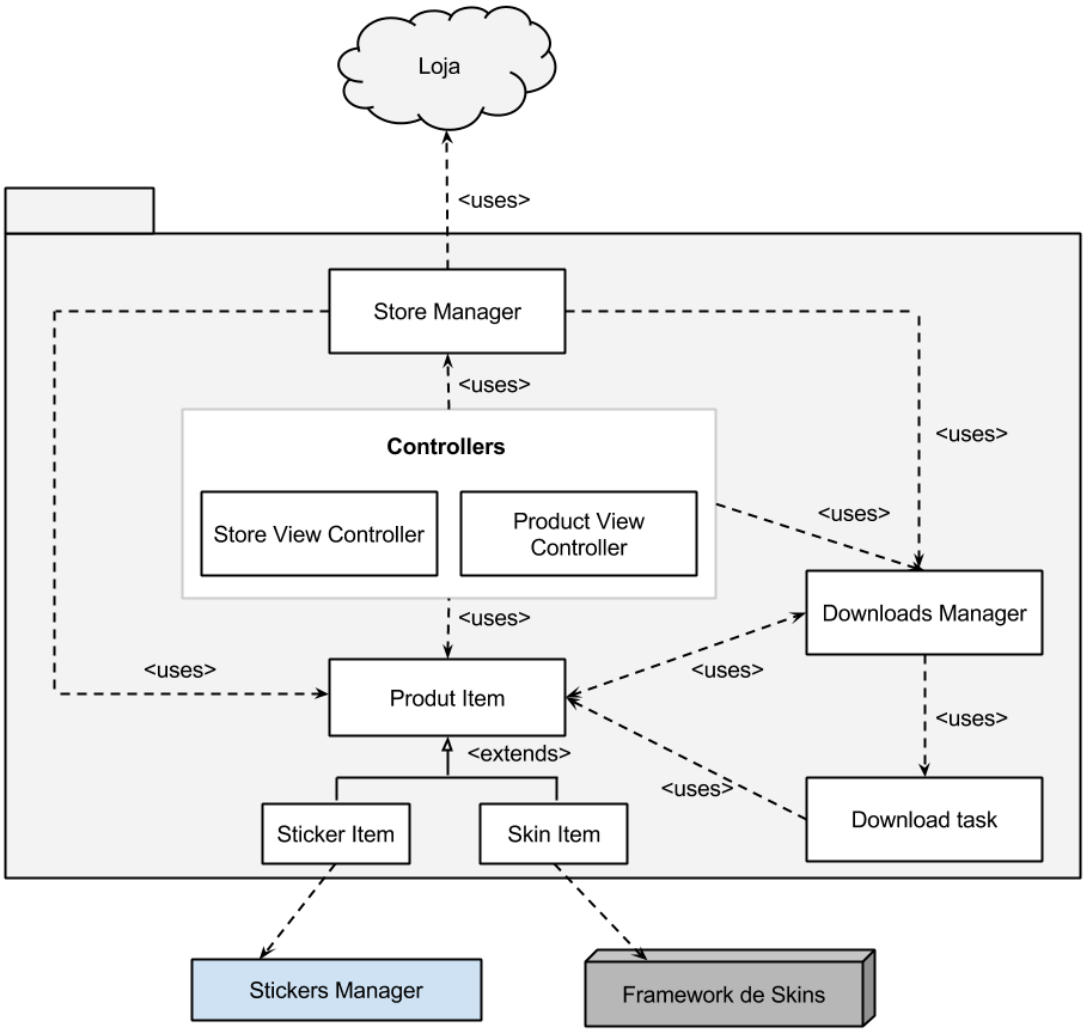

Figura 14 - Estrutura da loja

Componente	Descrição
Loja	Local remoto onde se encontra a informação relativa à loja.
Store Manager	Objecto que faz a gestão do conteúdo disponível para ser apresentado na loja.
Store View Controller	Permite apresentar os conteúdos da loja em forma de lista vertical.
Product View Controller	Apresenta informação detalhada sobre um determinado conteúdo.
Product Item, Sticker Item e Skin Item	Representam os vários tipos de conteúdos disponíveis na loja bem como lógica especifica de cada um deles.
Downloads Manager	Objecto que faz a gestão de todos os *downloads* de conteúdos que estão a ser feitos num determinado instante.
Download Task	Objecto que está encarregue de fazer o *download* de um determinado conteúdo.
Stickers Manager	Objecto que faz a gestão de todos os Stickers presentes na aplicação num determinado instante.
Framework de Skins	É a Framework encarregue de gerir e aplicar todas as Skins disponíveis na aplicação num determinado instante.

Tabela 15 - Descrição da estrutura da loja

5.3.10 Conteúdo alojado da rede

O conteúdo alojado na rede consiste, essencialmente, em ficheiros JSON. Existe um ficheiro que lista todos os produtos existentes na loja. Cada produto é listado com a seguinte informação:

Propriedade	Descrição
Nome do produto	*String* que representa o nome do produto que vai aparecer na loja.
Tipo do produto	*String* que permite distinguir se um produto é uma Skin, Sticker ou outro.
URL para o conteúdo	*String* que contem o caminho para o ficheiro *zip* que contém o conteúdo do produto.
URL para um *thumbnail*	*String* com o caminho para a imagem que vai surgir na loja.

URL para informação detalhada	*String* com o caminho para o ficheiro JSON que contém a informação detalhada de um determinado produto.
Tamanho do conteúdo	*String* onde é guardada a informação relativa ao tamanho do ficheiro *zip*. Informação que pode ser útil ao utilizador em certas situações.

Tabela 16 - Propriedades dos produtos presentes na lista de produtos disponíveis na loja

Outro tipo ficheiro corresponde a um ficheiro que contém a informação detalhada de um determinado produto. Esse ficheiro tem a seguinte estrutura:

Propriedade	**Descrição**
Texto descritivo	*String* que contém um texto que apresenta o produto ao consumidor, fazendo um breve resumo.
URL para uma imagem	*String* com o caminho para a imagem que deve aparecer ao lado do nome da aplicação na vista de detalhes.
Preview images	*Array* contendo o caminho para várias imagens que serão apresentadas na *View* de detalhes do produto.
Sprite image	*String* contendo o caminho para uma imagem que pretende demonstrar o conteúdo do produto. Utilizada no contexto dos Stickers.

Tabela 17 - Propriedades do ficheiro de configuração JSON relativas à informação detalhada de cada produto

5.3.11 Trabalho Futuro

A estrutura principal da loja já se encontra desenvolvida, no entanto existem ainda algumas funcionalidade que irão ser necessárias para a loja funcionar na sua plenitude. Em seguida serão enumeradas as varias funcionalidades que são esperadas implementar no futuro.

- Desenvolvimento de um mecanismo de controlo de versões: Quando uma nova versão está disponível da loja, o utilizador deverá ser notificado e poder fazer *update* da mesma;
- Integração com a App Store para poder realizar o processo de pagamento;

5.3.12 Considerações finais

A WIT Software, S.A. tem uma equipa a desenvolver um motor de *plugins* que irá permitir estender a solução RCS para iOS com novas funcionalidades.

A equipa de *sales* tem incorporado nas suas apresentações vários *plugins* que funcionam como *proof of concept*. Esses *plugins* são descarregados desta loja, por isso, houve também a necessidade do autor adicionar o suporte a mais um tipo de conteúdo, neste caso os *plugins*.

5.3.13 Conclusões

A tarefa foi concluída com sucesso, cumprindo assim todos os objectivos. O principal desafio desta tarefa consistiu em desenvolver um solução dinâmica o suficiente de forma a que possa ser estendida e melhorada no futuro.

5.4 Solução de download de Stickers para o Message Plus

Por motivos de confidencialidade toda a descrição da solução relacionada com a funcionalidade de Stickers encontra-se descrita no Anexo D auxiliada pelos anexos A, B e C.

5.5 Optimização no carregamento dos Emojis e Emoticons no Message Plus

5.5.1 Introdução

Aquando da implementação no Message Plus da tarefa relacionada com os Stickers, foi possível observar que a instanciação da *View* onde estão incluídos os Emoticons, Emojis e Stickers estava aquém de esperado.

Através da ferramenta disponibilizada pelo IDE da Apple – o XCode -, denominada de Time Profiler [39], é possível perceber quanto tempo demora cada invocação no código permitindo assim perceber quais as tarefas mais longas efectuadas pela aplicação.

A utilização desta ferramenta permitiu perceber que o que estava a fazer com que a *View* estivesse com tempos de instanciação elevados consistia nos contínuo acesso ao File System para serem carregadas mais de 800 imagens relativas aos Emojis.

5.5.2 Solução

A solução consistiu em diminuir os acessos ao File System, passando de um número de acessos igual ao número de Emojis para um único acesso.

Para isso, é importante perceber como funcionam os Emojis. Os Emojis consistem em pequenas imagens que surgem entre uma mensagem de texto. Em regra, a cada imagem corresponde um único caractere Unicode, no entanto existem dez exceções que são representadas por dois caracteres Unicode.

A solução passou por criar um ficheiro binário contendo todos as imagens e respectivo(s) caractere(s) Unicode de forma a que com um único acesso ao File System seja possível ler essa informação sempre que necessário.

Para isso foi criado um ficheiro de configuração em XML com o os seguintes objetivos:

1. Definir a ordem como os Emojis são apresentados ao utilizador;
2. Definir a que categoria pertence cada Emoji;
3. Definir o(s) Unicode(s) de cada Emoji;
4. Definir a imagem de cada Emoji;
5. Definir o número total de Emojis;

O próximo passo da solução consistiu na implementação de um programa em JAVA que interpretasse o ficheiro de configuração em XML e criasse um ficheiro binário com a estrutura e ordem da seguinte tabela.

***Header* do Ficheiro**	
Propriedade	Tamanho
Versão	1 byte
Numero total de Emojis	4 bytes
Tamanho do *Header* do Ficheiro igual a 5 bytes	
Header do Emoji	

Propriedade	Tamanho
Código Unicode	8 bytes
Offset	4 bytes
Categoria do Emoji	1 byte
Tamanho de Emoji	2 bytes
Tamanho do *Header* do Emoji igual a 15 bytes	

Tabela 18 - Estrutura do ficheiro binário de Emojis

O ficheiro binário começa com 5 bytes que contêm a versão e o número total de Emojis. Segue a informação relativa a cada Emoji, ocupando cada Emoji 15 bytes. No final desta informação constam as imagens dos Emojis.

Assim sendo, quando a aplicação é iniciada, todos os *Headers* são lidos para memória e é mantida uma ligação aberta ao ficheiro binário. Sempre que for necessário apresentar a imagem de um Emoji através de um Código Unicode é possível carregar a imagem através do *offset* e tamanho do Emoji.

De salientar que a construção do ficheiro binário teve de ter em atenção o facto de os processadores dos dispositivos Apple serem little-endian e os de Android serem big-endian. Para isso, foi adicionada uma *flag* que permite gerar o ficheiro binário num tipo ou noutro.

5.5.3 Conclusões

Esta solução, além de permitir melhorar o tempo de instanciação da *View* de Emojis fez com que a memória utilizada pela aplicação fosse reduzida. Isto porque não existe agora necessidade de manter as imagens dos Emojis em memória.

Esta solução preparou também as soluções da WIT Software, S.A. para que conteúdo do tipo dos Emojis sejam disponibilizados na loja.

A razão que levou a que o programa que gera o ficheiro binário tenha sido feito em JAVA e não em Objective-C foi o facto de que este também será utilizado pelos colegas de equipa que estão a desenvolver as soluções para Android e que não têm experiência em Objective-C.

5.6 Nova implementação da câmara do Message Plus e implementação da funcionalidade de Quick Share de vídeo

5.6.1 Introdução

Este subcapítulo tem como objectivo detalhar a implementação de duas tarefas que têm muitos pontes em comum. Uma das tarefas consiste na implementação de uma nova versão da câmara do Message Plus de forma a ir de encontro à especificação de UX e UI definidas pelo grupo Vodafone. A outra tarefa consiste na implementação de uma nova funcionalidade intitulada de Quick Share de Vídeo.

O principal desafio passa por tornar a solução de acesso à câmara do dispositivo genérica de forma a poder ser reaproveitada independentemente do tipo de câmara – fotos, vídeos e Quick Share de Video.

O próximo subcapítulo explica detalhadamente a implementação da solução para cada tarefa.

5.6.2 Solução

Na primeira versão do Message Plus existia uma vista diferente para cada câmara, isto deve-se ao facto de o Message Plus ter derivado da solução RCS para iOS da WIT Software, S.A.. No entanto o grupo Vodafone pretendia que existisse uma única vista onde se pudesse trocar entre modos, ou seja, uma vista que através de apenas uma interação fosse possível alterar o modo da câmara, por exemplo, de modo de fotos para o modo de vídeos.

Uma outra funcionalidade pretendida pelo grupo Vodafone era o Quick Share de vídeo, que, resumidamente, consiste na possibilidade de partilha rápida de vídeo com apenas um gesto.

Estas duas tarefas têm em comum o facto de usarem a câmara do dispositivo. Devido a isso, a solução desenvolvida foi idealizada de forma a que se reutilize ao máximo este ponto em comum.

Primeiramente é necessário que o leitor perceba como funciona o acesso á câmara de um dispositivo iOS. Depois do perceber, será mais fácil entender como foi estruturada a solução e o porquê dessas escolhas.

O iOS disponibiliza uma Framework intitulada de AV Foundation[40] cujo objectivo é oferecer uma interface em Objective-C para gerir conteúdos áudio visuais. Essa Framework foi utilizada para realizar o acesso á câmara nativa.

Nessa Framework, para apresentar ao utilizador a imagem que está a ser capturada pela câmara do dispositivo são necessários, no mínimo, três intervenientes:

1. **Input**: Câmara do dispositivo que captura imagens;
2. **Layer**: Objecto onde será renderizada a imagem que a câmara está a capturar;
3. **Objecto de sessão**: Responsável por gerir as *frames* enviadas pelo **input**;

Com estes três intervenientes é possível apresentar na interface da aplicação o que a câmara está a captar.

Estes intervenientes são, por isso, partilhados por todos os tipos de câmaras pretendidos pelo grupo Vodafone e devido a isso a lógica associada deve estar preparada para ser reutilizada nos três tipos de câmara – fotos, vídeos e Quick Share de Vídeo.

A imagem seguinte pretende ilustrar a versão final de como ficou estruturada a solução.

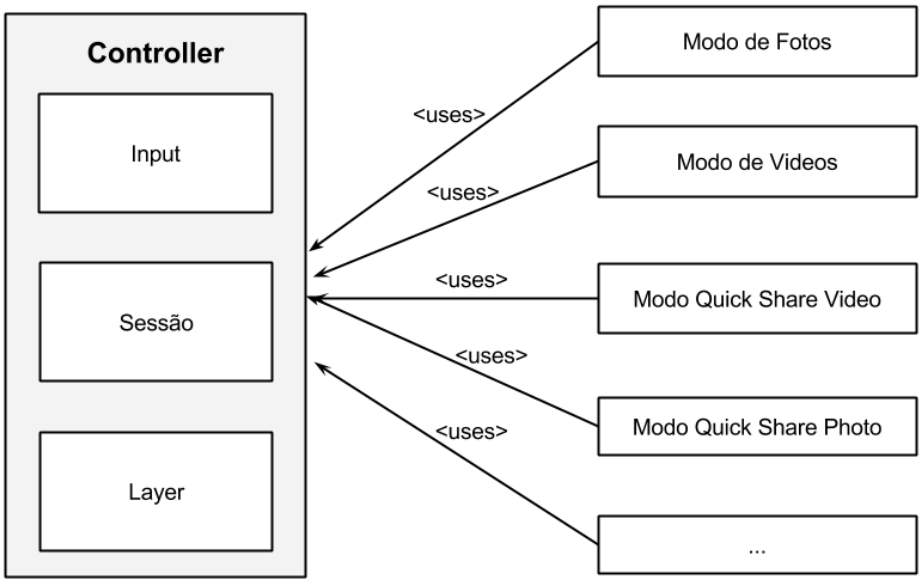

Figura 15 - Estrurua da solução da câmara

Elemento	Descrição
Input	Consiste na câmara do dispositivo. Caso o dispositivo em questão tenha duas câmaras, é possível escolher uma das duas para funcionar como input.
Sessão	Este objecto é responsável por gerir toda uma sessão de captura da imagem da câmara, gravação de vídeo, captura de uma foto, etc. Aqui é possível definir propriedades específicas da captura atual, como por exemplo, a qualidade da imagem processada.
Layer	Consiste no objecto onde irá ser apresentada a imagem que está a ser capturada. O AV Foundation disponibiliza um objecto intitulado AVCaptureVideoPreviewLayer que consiste numa *layer* que quando associada a uma sessão apresenta o que a câmara está a capturar[41].
Controller	Este objecto consiste no *Controller* que apresenta uma UIView onde a AVCaptureVideoPreviewLayer irá estar contida. É responsável por fornecer uma API que possibilita que os elementos, input, sessão e layer possam ser utilizados por diferentes modos.
Fotos, Videos, Quick Share de Video, Quick Share de Fotos,	Representam os vários modos que a câmara pode tomar. Estes modos têm acesso ao input, sessão e layer através do Controller, podendo assim alterar as características da câmara consoante as necessidades do seu tipo de modo. Cada modo encapsula a lógica relacionada com o modo em

Outros	questão, por exemplo, o modo de vídeo tem a lógica associada à gravação de um vídeo, bem como a lógica de UI associada, que permite apresentar os botões e informação relacionada com modo.

Tabela 19 - Descrição da solução da câmara

Quando o *Controller* é iniciado, é possível indicar quais os modos suportados e qual o modo com que a câmara deve arrancar.

O grupo Vodafone definiu o gesto horizontal de *swipe* a interação que faz mudar de tipo de câmara. Um gesto de *swipe* consiste no toque contínuo no ecrã do dispositivo com um movimento translacional. Esse gesto está implementado na UIView do *Controller* e a resposta a esse gesto consiste no carregamento de um novo módulo.

Todos os módulos respondem a um protocolo que consiste nos seguintes métodos:

- *Load* – este método é utilizado para inicializar, caso necessário, a sessão e o input disponibilizadas pelo *Controller*;
- *ConfigureMode* – este método é o local onde são definidas configurações especificas de cada modo. Por exemplo, é definida a resolução das imagens capturadas pela câmara;
- *ConfigureUI* – este método permite que sejam adicionados componentes de UI (como botões, labels, etc) relacionados com o modo especifico. Aqui também é inicializada a layer, caso necessário.
- *Run* – garante que a sessão está ativa e a fazer com que a imagem proveniente da câmara apareça na interface da aplicação;
- *Unload* – quando se pretende invalidar o modo que está ativo. Por exemplo, devido ao facto de o utilizador ter escolhido mudar de modo. Este método é utilizado para remover todos os componentes de UI adicionados pelo próprio modo;

Era requisito também a existência de *flash*, *focus* e *zoom* na solução das câmaras. A implementação de cada uma destas funcionalidades foi realizada através de APIs disponibilizadas pelo AV Foundation. Segue uma breve descrição de como foi implementada cada funcionalidade.

- ***Flash***: o AV Foundation permite definir três tipos de *flash*. O primeiro tipo é denominado de *AVCaptureTorchModeOn*. Neste tipo o flash está constantemente ligado. O segundo tipo é o contrario do primeiro, é denominado de *AVCaptureTorchModeOff*, que se traduz num *flash* sempre desligado. Por último, o tipo *AVCaptureTorchModeAuto* em que consoante certas condições (por exemplo a claridade) o flash é ligado ou desligado. Para implementar esta funcionalidade foi adicionado à UI um botão com estes três estados;
- ***Focus***: Na objecto que representa a câmara do dispositivo, ou seja, o input, existe um método intitulado de *setFocusPointOfInterest*[42] que recebe como argumento um ponto onde a câmara deve focar. Assim, para implementar esta funcionalidade, foi adicionado um gesto à UIView do *Controller* que detecta o ponto onde o utilizador toca, sendo esse ponto passado para o método do input que realiza o *focus*. Quando o utilizador toca no ecrã é adicionada uma imagem, por breves instantes, indicativa do ponto de *focus*;
- ***Zoom***: Também no objecto de input é possível definir o *zoom*. Para isso é utilizado o método *setVideoZoomFactor* [42] que consiste num factor que define a escala das imagens que são capturadas pela câmara. Quanto maior for a escala, maior será a imagem e vice versa. O utilizador ao fazer um gesto intitulado de *Pinch*[43], que consiste no toque continuo de dois dedos no ecrã em que ambos se afastam ou

diminuem, aumenta ou diminui o valor do factor, traduzindo-se assim num maior ou menor *zoom*;

Por último, era pretendido que fosse possível alterar entre a câmara traseira e frontal do dispositivo. Para isso, foi adicionado uma botão que ao ser pressionado altera o input da sessão para o desejado, ou seja, altera entre a câmara traseira e frontal do dispositivo.

As últimas quatro funcionalidades descritas podem ser ativadas ou desativadas pelos módulos, escondendo assim a os controlos de UI respectivos.

5.6.3 Modos

Este subcapítulo pretende enumerar e detalhar características especificas de cada modo, de forma a perceber o que os distingue e como foi implementada a lógica de cada um.

Um dos factores que distingue cada um dos módulos é a respectiva UI. A UI destas funcionalidades foi especificada pelo grupo Vodafone. Por razões de confidencialidade, essa especificação está disponível no **Anexo C: Especificações de UI e UX do Message Plus.**

5.6.3.1 Modo de Fotos

O modo de fotos permite que o utilizador tire fotos de elevada qualidade fazendo uso da câmara do seu dispositivo.

Por isso, o primeiro factor de distinção reside no facto de que a sessão ter de ser configurada com a qualidade de imagem máxima suportada pela câmara do dispositivo.

A Framework AV Foundation permite retirar output da sessão de três formas, ou na forma de vídeo com áudio, na forma de vídeo sem áudio, ou na forma de uma imagem. Nesta caso é pretendido obter uma imagem por isso é adicionada uma instancia do tipo AVCaptureStillImageOutput[44].

A classe AVCapturaStillImageOutput permite tirar uma foto, com as características definidas na sessão, através do seguinte método:

captureStillImagesAsynchronouslyFromConnection:completionHandler

5.6.3.2 Modo de Vídeos

Este modo possibilita que o utilizador faça a gravação de um vídeo fazendo uso da câmara do seu dispositivo.

Por requisitos impostos pelo grupo Vodafone, o limite do vídeo não será temporal mas sim espacial, ou seja, o limite máximo do vídeo estará relacionado com o tamanho que ele ocupa em disco e não com a duração do mesmo. O valor do tamanho máximo para o vídeo é configurável.

Além do input de vídeo, este modo pretende também ter um input de áudio de forma a poder gravar vídeo com áudio. Por isso, quando é feito Load deste modo é adicionado um input que irá adicionar áudio à sessão capturado pelo microfone do dispositivo.

De forma a suportar vídeos com uma duração aceitável o grupo Vodafone decidiu que a resolução da imagem do vídeo seria de 352x288. Este valor é definido na sessão disponibilizada pelo *Controller*.

O tipo de output pretendido neste modo é na forma de vídeo, por isso, é adicionada à sessão uma instancia de output do tipo AVCaptureMovieFileOutput[45]. Este output permite ser configurado com o limite máximo que o ficheiro resultante da gravação pode tomar.

O modo adiciona um botão à UI que permite começar/parar a gravação do vídeo. O seguinte método da classe AVCaptureMovieFileOutput é o que possibilita o começo da a gravação:

startRecordingToOutputFileURL:recordingDelegate:

Este método recebe um URL para um ficheiro onde será guardada a gravação do vídeo e um *delegate* que irá receber informação sobre quando e porquê terminou uma gravação.

5.6.4 Modo Quick Share de Vídeos

Através da analise das especificações de UI e UX (**Anexo C: Especificações da UI e UX do Message Plus**) é possível perceber que o botão para gravar um vídeo está contido na *input bar* da vista de Chat. Por isso, a UIView respectiva a essa barra foi alterada de modo a poder conter o botão respectivo ao modo de Quick Share.

A solução desenvolvida está preparada para apresentar vários tipos de modos de Quick Share na *input bar*. Preparando assim para a eventualidade do grupo Vodafone desejar colocar mais tipos de Quick Share nessa mesma *input bar*.

A imagem seguinte tem o objectivo de elucidar o leitor do dinamismo que a *input bar* necessitou de suportar. Segundo a especificação, quando não existe texto no input deverá ser visualizado o modo de Quick Share disponível (neste caso de exemplo estão presentes vários tipos de Quick Share que consistem em vídeo, foto e áudio), caso contrário deverá aparecer o botão para enviar o texto que se encontra no *input*. Toda esta transição é animada, o input de texto anima a sua largura e os botões fazem uma animação de *fade in* ou *fade out*. Estas animações oferecem um aspecto mais polido à aplicação.

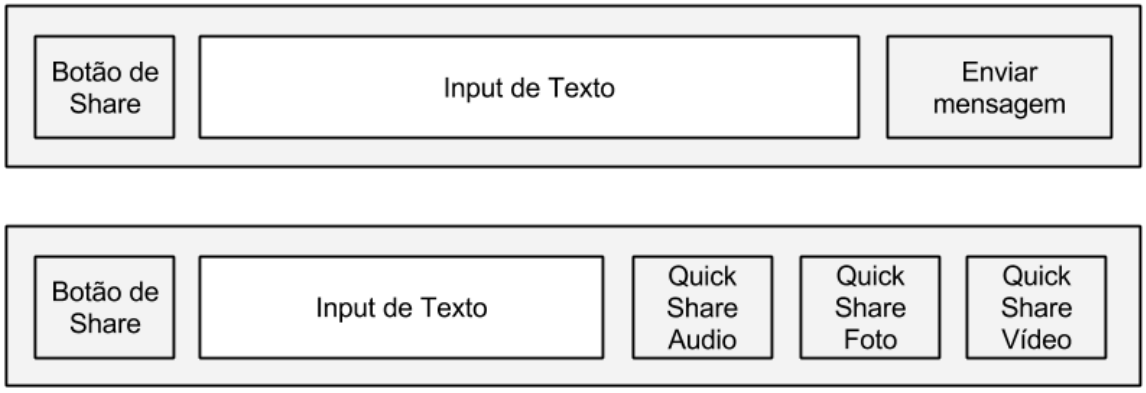

Figura 16 - Dinamismo da *input bar* de modo a suportar vários modos de Quick Share

Em seguida, será explicada a estrutura de toda a solução do Quick Share de Vídeo. Existe, para cada modo de Quick Share, um objecto que herda de uma classe intitulada QuickShareMode. Este objecto é responsável por gerir todo o modo de Quick Share.

Quando a *input bar* é instanciada recebe como argumento um *array* com os tipos de Quick Share que que suporta. Para cada tipo de Quick Share é criada a respectiva instancia que herda da classe QuickShareMode.

Cada QuickShareMode devolve uma *view* que é adicionada à *input bar* do chat. Essa *view* contém os controlos necessários que permitem ao utilizador interagir com a funcionalidade. De forma a obedecer à UX e UI especificadas, quando o utilizador carrega no controlo do Quick Share de vídeo, este cria o Quick Share *Controller* e adiciona-o ao *Chat Controller*. Por sua vez, o Quick Share *Controller* instancia o Camera *Controller* com o modo Quick Share de Video.

Após uma contagem decrescente (de acordo com a especificação de UX), é iniciada a gravação do vídeo.

Quando o utilizador deixa de pressionar no botão de Quick Share na *input bar* o Câmara *Controller* é notificado, que por sua vez notifica o modo de Quick Share da câmara que deve cancelar ou parar a gravação.

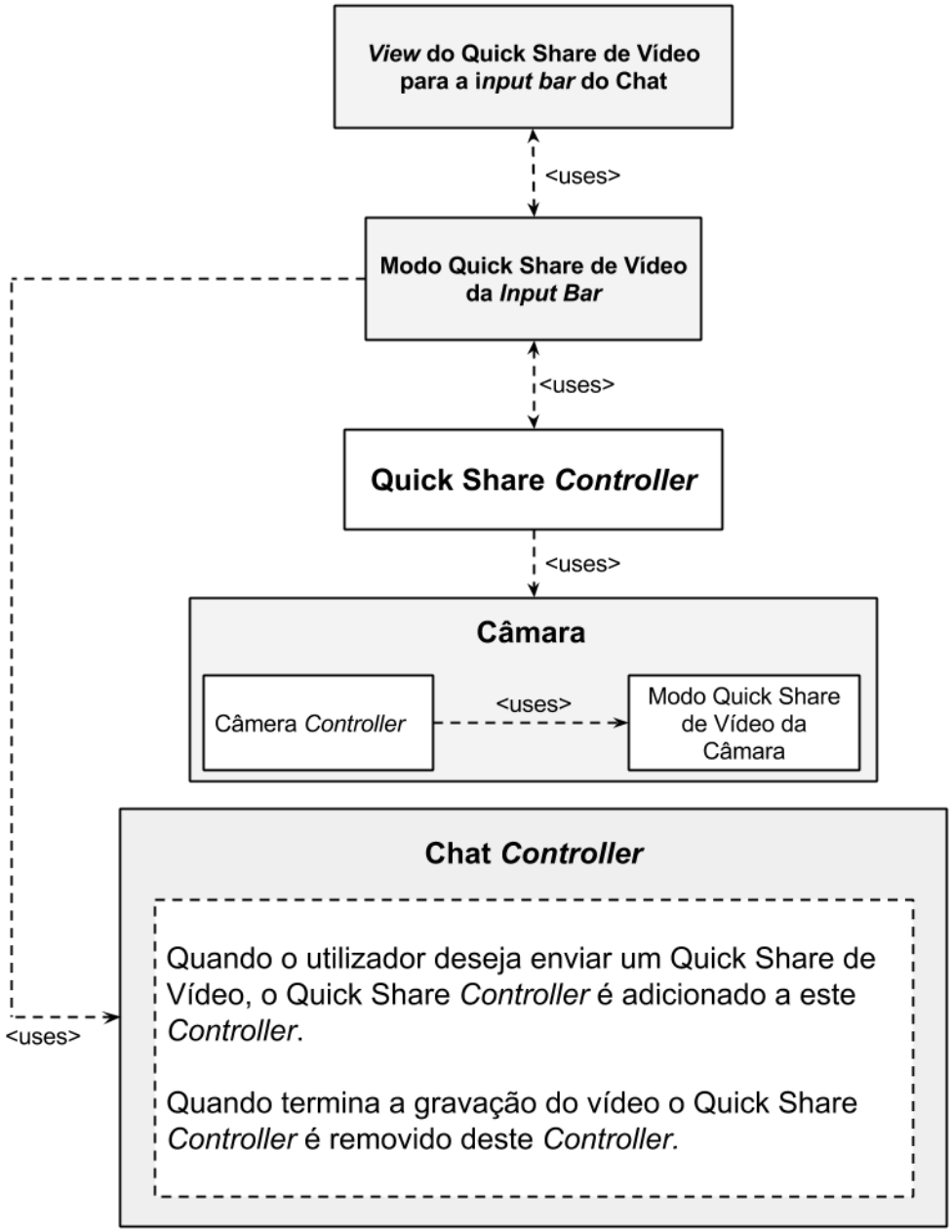

Figura 17 - Estrutura da solução de Quick Share de Vídeo

Em comparação ao modo de gravação de vídeo da câmara, este modo apresenta uma lógica semelhante. As diferenças residuais consistem em:

- O começo e cancelamento da gravação é delegado a outro componente. No modo de gravação de vídeo da câmara era o próprio modo que adicionava botões para controlar o começo e cancelamento de gravação de vídeo. Neste modo, os botões irão pertencer ao outro componente presenta na *input bar*;
- A gravação está limitada temporalmente com um valor de quinze segundos e não ao nível do espaço que é ocupado pela gravação;

5.6.5 Conclusões

Todas as tarefas foram implementadas com sucesso. A implementação da câmara mostrou ser genérica o suficiente para facilitar o desenvolvimento do modo de Quick Share de vídeo, ficando ainda preparada para adicionar, de uma forma simples, novos modos de câmara caso seja necessário.

Neste momento toda esta tarefa encontra-se em fase de aceitação de forma a poder ser integrada na próxima versão do Message Plus, disponibilizada na App Store.

Capítulo 6
Trabalho Futuro

A Framework de Skins apresenta um certo grau de maturidade. estando, por isso, a ser utilizada noutros projetos da empresa. No entanto existe a possibilidade de a melhorar em diferentes aspectos. Um dos aspectos está relacionado com a performance de execução de todo o processo de *load* de uma Skin. Neste processo, entre outras coisas, é feito o *parse* dos ficheiros de XML (este *parse* é lento). Uma das soluções seria ter uma ferramenta que transforma esse XML num ficheiro binário. Assim, iria permitir que fazer o *load* de uma Skin de uma forma mais rápida pois seria possível ler estruturas de dados completas desse ficheiro binário.

No que diz respeito à loja para a solução RCS para iOS da WIT Software, S.A., o autor desenvolveu uma versão completa o suficiente de forma a ser possível à equipa de *sales* fazer demonstrações a possíveis clientes interessados na funcionalidade. No entanto, para a loja ir para produção ainda faltam desenvolver algumas funcionalidades. Exemplo disso, é a integração com as in-app purchases da Apple e um sistema de controlo de versões de forma a permitir que um utilizador faça *update* do conteúdo que já fez download.

De forma a poder gerir o conteúdo existente na loja existe a necessidade de desenvolver um *back office* que permitirá adicionar, remover e fazer *update* dos conteúdos, entre outras funcionalidades.

O trabalho desenvolvido no âmbito dos Stickers no Message Plus permitiu adquirir experiência que poderá agora ser aproveitada para evoluir a versão da loja da WIT Software, S.A.

A WIT Software, S.A. irá continuar a investir nesta solução de forma a que ela evolua para um produto cada vez mais final.

Capítulo 7
Conclusões

O crescimento da popularidade das aplicações OTT durante os últimos anos teve um impacto negativo e substancial nas receitas da indústria das telecomunicações. Face a esta ameaça a indústria procurou reinventar e modernizar os seus serviços de comunicação, mas ainda pouco explorou a monetização dos mesmos. Identificando uma oportunidade de negócio nesta nova realidade, a WIT Software, S.A. lançou este estágio para o desenvolvimento de uma plataforma de download de conteúdos digitais de valor acrescentado, nomeadamente Skins e Stickers. Com esta génese, o estágio ficou muito bem posicionado no contexto atual da indústria e a sua originalidade e inovação tornaram o seu desenvolvimento uma experiência profundamente enriquecedora para o autor.

O balanço do estágio foi muito positivo. O autor ficou encarregue de várias tarefas de desenvolvimento de elevada complexidade técnica e também originalidade. Devido a esta originalidade, foi necessário o desenvolvimento de soluções novas de raiz sem qualquer tipo de software pré-existente que pudesse servir de base ou inspiração. Muito do trabalho realizado pelo autor também teve grande visibilidade na indústria, tendo sido apresentado a várias operadoras de telecomunicações em diversas ocasiões, inclusive na maior feira mundial da indústria, por representantes de vendas da WIT Software.

Durante este estágio o autor também adquiriu experiência inestimável de desenvolvimento de software em mundo real. Esteve integrado na equipa de desenvolvimento de produtos RCS, uma das maiores da empresa e que conta com mais de 40 colaboradores, e seguiu uma metodologia ágil que tinha sido previamente adoptada pela equipa. Trabalhou em projetos reais para várias operadoras internacionais, inclusive uma das maiores da indústria, o Grupo Vodafone, num projeto de elevada importância para a WIT Software. Todo o código desenvolvido pelo autor já se encontra em produção e é utilizado diariamente por milhares de utilizadores.

Todo os objectivos propostos inicialmente foram atingidos. A Framework de Skins foi desenvolvida e graças à sua arquitetura modular e qualidade já está a ser utilizada internamente na WIT Software por equipas de desenvolvimento de outros produtos. Foi desenvolvida a loja na aplicação RCS de iOS que permite a transferência de Skins e Stickers, e também foi desenvolvida a solução que permite o envio, recepção e download de Stickers. Assim, pode-se considerar que o estágio foi concluído com sucesso.

Por fim, a título conclusivo, o autor gostaria de referir o privilégio que foi trabalhar neste projeto com uma equipa tão talentosa e experiente quanto a equipa de RCS e agradecer a oportunidade e voto de confiança que lhe foi depositado pela WIT Sofware, S.A..

Capítulo 8
Referências

[1] S. Making. *Facade Design Pattern*. Available: http://sourcemaking.com/design_patterns/facade

[2] I. Apple. (2014). *Concepts in Objective-C Programming*. Available: https://developer.apple.com/library/ios/documentation/general/conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html

[3] Codedesign. *Observer Pattern*. Available: http://www.oodesign.com/observer-pattern.html

[4] GSMA. (2012). *GSMA Selects WIT Software to Provide RCS-e Application*. Available: http://www.gsma.com/membership/gsma-selects-wit-software-to-provide-rcs-e-application

[5] J. Russel. (2013). *First earnings from messaging service Line show revenue of $58m in Q1 2013; $17m from stickers alone*. Available: http://thenextweb.com/asia/2013/05/10/first-earnings-from-messaging-service-line-show-revenue-of-58m-in-q1-2013-17m-from-stickers-alone/ - !q9VAr

[6] I. Apple. (2014). *Getting Started with In-App Purchase on iOS and OS X*. Available: https://developer.apple.com/in-app-purchase/In-App-Purchase-Guidelines.pdf

[7] I. Apple. (2014). *In-App Purchase Programming Guide*. Available: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Introduction.html

[8] Google. *In-app Billing Overview*. Available: https://developer.android.com/google/play/billing/billing_overview.html

[9] E. News. (2012). *KakaoTalk Daily Traffic Hits 3 Billion*. Available: http://english.chosun.com/site/data/html_dir/2012/07/27/2012072701392.html

[10] E. Lukman. (2013). *Line Hits 200 Million Users, Adding 100 Million in Just 6 Months*. Available: http://www.techinasia.com/line-hits-200-million-users-adding-100-million-users-6-months/

[11] Telecompaper. (2011). *Skype grows FY revenues 20%, reaches 663 mln users*. Available: http://www.telecompaper.com/news/skype-grows-fy-revenues-20-reaches-663-mln-users--790254

[12] WSJ. (2013). *Viber Unveils Desktop App*. Available: http://blogs.wsj.com/digits/2013/05/07/viber-unveils-desktop-app/

[13] J. S. Ken Schwaber. *O Guia do Scrum (TM)*. Available: https://http://www.scrum.org/Portals/0/Documents/Scrum Guides/Scrum Guide - Portuguese_European.pdf - zoom=100

[14] C. Melonfire. (2006). *Understanding the pros and cons of the Waterfall Model of software development*. Available: http://archive.today/20130102112959/http://articles.techrepublic.com.com/5100-10878_11-6118423.html?part=rss&tag=feed&subj=tr

[15] K. B. Beck, Mike; van Bennekum, Arie; Cockburn, Alistair; Cunningham, Ward; Fowler, Martin; Grenning, James; Highsmith, Jim; Hunt, Andrew; Jeffries, Ron; Kern, Jon; Marick, Brian; Martin, Robert; Mellor, Steve; Schwaber, Ken; Sutherland, Jeff; Thomas, Dave. (2001). *Manifesto para o Desenvolvimento Ágil de Software*. Available: http://agilemanifesto.org/iso/ptpt/

[16] I. Apple. (2014). *UIAppearance Protocol Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UIAppearance_Protocol/Reference/Reference.html

[17] I. Apple. (2014). *UIView Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/uiview_class/uiview/uiview.html

[18] I. Apple. (2014). *NSObject Class Reference*. Available: https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSObject_Class/Reference/Reference.html

[19] I. Apple. (2014). *UIButton Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UIButton_Class/UIButton/UIButton.html

[20] I. Apple. (3014). *UISwitch Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/uiswitch_class/Reference/Reference.html

[21] I. Apple. (2014). *UISlider Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UISlider_Class/Reference/Reference.html

[22] I. Apple. (2014). *UILabel Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UILabel_Class/Reference/UILabel.html

[23] I. Apple. (2014). *UITextView Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/uitextview_class/Reference/UITextView.html

[24] I. Apple. (2014). *UITextField Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UITextField_Class/Reference/UITextField.html

[25] I. Apple. (2014). *UINavigationBar Class Reference*. Available: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UINavigationBar_Class/Reference/UINavigationBar.html

[26] I. Apple. (2014). *CALayer Class Reference*. Available: https://developer.apple.com/library/mac/documentation/graphicsimaging/reference/CALayer_class/Introduction/Introduction.html

[27] I. Apple. (2014). *CGGeometry Reference*. Available: https://developer.apple.com/library/ios/documentation/GraphicsImaging/Reference/CGGeometry/Reference/reference.html#//apple_ref/doc/c_ref/CGRect

[28] I. Apple. (2014). *UIColor Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UIColor_Class/Reference/Reference.html

[29] I. Apple. (2014). *NSNumber Class Reference*. Available: https://developer.apple.com/library/mac/documentation/cocoa/reference/foundation/classes/nsnumber_class/Reference/Reference.html

[30] I. Apple. (2014). *NSArray Class Reference*. Available: https://developer.apple.com/library/ios/Documentation/Cocoa/Reference/Foundation/Classes/NSArray_Class/NSArray.html

[31] I. Apple. (2014). *NSString Class Reference*. Available: https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/Reference/NSString.html

[32] I. Apple. (2014). *UIWebView Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UIWebView_Class/Reference/Reference.html

[33] I. Apple. (2014). *UITableView Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UITableView_Class/Reference/Reference.html

[34] I. Apple. (2014). *UITableViewCell Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UITableViewCell_Class/Reference/Reference.html

[35] I. Apple. (2014). *CTFramesetter Reference*. Available: https://developer.apple.com/library/mac/documentation/Carbon/reference/CTFramesetterRef/Reference/reference.html

[36] I. Apple. (2014). *UINib Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UINib_Ref/Reference/Reference.html

[37] I. Apple. (2014). *Bundle Programming Guide*. Available: https://developer.apple.com/library/mac/documentation/corefoundation/conceptual/cfbundles/BundleTypes/BundleTypes.html

[38] I. Apple, "Core Data Framework Reference," 2014.

[39] I. Apple. (2014). *Instruments User Reference*. Available: https://developer.apple.com/library/ios/documentation/AnalysisTools/Reference/Instruments_User_Reference/TimeProfilerInstrument/TimeProfilerInstrument.html

[40] I. Apple. (2014). *AV Foundation Framework Reference*. Available: https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVFoundationFramework/_index.html

[41] I. Apple. (2014). *AVCaptureVideoPreviewLayer*. Available: https://developer.apple.com/library/mac/documentation/AVFoundation/Reference/AVCaptureVideoPreviewLayer_Class/Reference/Reference.html

[42] I. Apple. (2014). *AVCaptureDevice Class Reference*. Available: https://developer.apple.com/library/mac/documentation/AVFoundation/Reference/AVCaptureDevice_Class/Reference/Reference.html

[43] I. Apple. (2014). *UIPinchGestureRecognizer Class Reference*. Available: https://developer.apple.com/library/ios/documentation/uikit/reference/UIPinchGestureRecognizer_Class/Reference/Reference.html

[44] I. Apple. (2014). *AVCaptureSillImageOutput Class Reference*. Available: https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVCaptureStillImageOutput_Class/Reference/Reference.html

[45] I. Apple. (2014). *AVCaptureMovieFileOutput Class Reference*. Available: https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVCaptureMovieFileOutput_Class/Reference/Reference.html