Ricardo Jorge Matos Amaro

CONCEPÇÃO E DESENVOLVIMENTO DE UMA PLATAFORMA DE GESTÃO DE SERVIÇOS SaaS PARA O SECTOR DO ALOJAMENTO: INTEGRAÇÃO E MIGRAÇÃO DE SERVIÇOS CLOUD

Relatório de Estágio
Mestrado em Engenharia Informática
orientado por Prof. Paulo Rupino da Cunha, Eng. Bruno Almeida e Eng. Carlos Lopes
e apresentado ao Departamento de Engenharia Informática
da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Setembro de 2018

· U C ·

Universidade de Coimbra

Resumo

Este estágio enquadra-se num projecto Portugal 2020, de investigação e desenvolvimento, designado Hotelcracy Apps, cujo objectivo é desenvolver uma plataforma de integração *cloud* de serviços *Software-as-a-Service* (SaaS) utilizados no sector do alojamento, disponibilizando o acesso às suas funcionalidades através de uma interface centralizada e homogénea. O objectivo deste trabalho em particular é a concepção e desenvolvimento de uma prova de conceito dos mecanismos de migração entre serviços SaaS, sempre que o utilizador os decida substituir.

Para a concepção da prova de conceito foi realizada uma análise do estado da arte da integração de serviços *cloud*, onde se analisaram abordagens tradicionais e se estudaram plataformas de integração actualmente disponíveis no mercado. Foi ainda realizado um estudo dos desafios da migração de serviços SaaS, onde se identificaram dificuldades e problemas a esta associados, e para os quais se propuseram soluções.

Dos estudos realizados teceram-se considerações com impacto significativo no desenho da prova de conceito e da própria plataforma como um todo. Estas considerações traduziram-se em decisões que influenciaram a arquitectura da plataforma, levando à identificação e especificação de componentes que não tinham sido considerados em fase de candidatura do projecto. Foi definido um protocolo interno para estruturação dos dados a comunicar entre os componentes da plataforma, e um processo estruturado para a execução da migração de serviços. Este protocolo e este processo foram implementados na prova de conceito, tendo sido desenvolvidos componentes específicos na plataforma, respeitando a arquitectura definida.

Os contributos ao nível da arquitectura, do protocolo interno e do processo de migração são de grande valor para o projecto Hotelcracy Apps. Estes ajudaram a estabelecer uma base sólida para a construção da plataforma, de modo a que esta possa dar resposta a diferentes desafios no futuro, principalmente em relação à integração de novos serviços SaaS e ao aumento da quantidade de utilizadores.

Palavras-Chave

Integração de serviços, *Integration Platform-as-a-Service*, iPaaS, Migração de serviços, SaaS, Sector do alojamento, Sector hoteleiro, Serviços *cloud*, *Software-as-a-Service*

Índice

Lista de tabelas

Lista de figuras

Tabela de acrónimos

Acrónimo	Descrição
API	*Application Programming Interface*
ATAM	*Architecture Trade-off Analysis Method*
BoB	*Best-of-Breed*
BPM	*Business Process Management*
BPMN	*Business Process Model and Notation*
CRM	*Customer Relationship Management*
DSL	*Domain-Specific Language*
ESB	*Enterprise Service Bus*
HTTPS	*Hyper Text Transfer Protocol Secure*
I&D	Investigação e Desenvolvimento
iPaaS	*Integration Platform-as-a-Service*
IPNlis	Laboratório de Informática e Sistemas do Instituto Pedro Nunes - Associação para a Inovação e Desenvolvimento em Ciência e tecnologia
JSON	*JavaScript Object Notation*
MVC	*Model-View-Controller*
OTA	*Online Travel Agency*
PMS	*Property Management System*
POS	*Point-of-Sale*
PT2020	Portugal 2020 - Sistema de Incentivos para Investigação e Desenvolvimento Tecnológico
RPC	*Remote Procedure Call*
REST	*Representational State Transfer*
SaaS	*Software-as-a-Service*

Tabela de acrónimos

Acrónimo	Descrição
SDCP	*Service Delivery Cloud Platform*
SI	Sistema de Informação
SIP	*SaaS Integration Platform*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
TDD	*Test-Driven Development*
TI	Tecnologias da Informação
TPA	Terminal de Pagamento Automático
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*

Capítulo 1
Introdução

O presente relatório tem como propósito documentar o trabalho realizado no estágio "Concepção e desenvolvimento de uma plataforma de gestão de serviços SaaS para o sector do alojamento - integração e migração de serviços *cloud*". Iniciou-se em Fevereiro de 2017, e terminou em Julho de 2018.

O estágio insere-se no projecto Hotelcracy Apps, co-financiado pelo programa Portugal 2020 - Sistema de Incentivos para Investigação e Desenvolvimento Tecnológico (PT2020). O trabalho aqui apresentado foi desenvolvido no Laboratório de Informática e Sistemas do Instituto Pedro Nunes - Associação para a Inovação e Desenvolvimento em Ciência e Tecnologia (IPNlis), co-promotor em associação com a empresa Hotelcracy Software Lda., líder do consórcio. O estagiário foi inserido na equipa do IPNlis, composta por dois professores universitários, que possuíam o papel de consultores do projecto, um gestor de projecto, e um elemento dedicado ao desenvolvimento. Este é um projecto de investigação e desenvolvimento com a duração de 33 meses, iniciado no dia 1 de Outubro de 2016 e com término a 30 de Junho de 2019, tendo o estágio coincidido com a sua fase inicial.

Neste primeiro capítulo será feito o enquadramento do estágio, sendo dado a conhecer o projecto Hotelcracy Apps e o problema endereçado pelo mesmo. Seguidamente, apresentam-se os objectivos do estágio e enuncia-se a contribuição e importância deste para o projecto, listam-se os artefactos produzidos e descreve-se a estrutura do relatório.

1.1. Enquadramento e motivação

O projecto Hotelcracy Apps é orientado para o sector do alojamento, e aborda um problema concreto identificado neste sector: a falta de integração entre serviços *cloud* usados na gestão hoteleira. Tem como objectivo a definição de estratégias que permitam dar resposta a este problema, e a concepção e desenvolvimento de uma plataforma web dedicada para aplicação dessas estratégias, permitindo o acesso às funcionalidades desses serviços de um modo integrado.

Nesta sub-secção descreve-se o problema identificado, e apresenta-se o âmbito do projecto Hotelcracy Apps, como resposta a esse problema. Esta informação é baseada no anexo técnico do projecto [1].

1.1.1. A necessidade de integração de serviços SaaS do sector do alojamento

O sector do alojamento é constituído, em grande número, por hotéis e alojamentos independentes – empresas não afiliadas a cadeias ou marcas que imponham standards de serviços e/ou processos – além das grandes cadeias hoteleiras habitualmente conhecidas. Estas unidades independentes representam 67% das ofertas na Europa e 30% das ofertas nos Estados Unidos da América [2], e são caracterizadas pela sua diversidade, possuindo especificidades amplas em áreas como o preço, posicionamento, dimensão, e estrutura organizacional. Devido a este estatuto de independência e à sua reduzida dimensão, a maioria das unidades possuem limitações de recursos e conhecimentos técnicos na área das tecnologias

da informação (TI), dificultando a adopção de ferramentas adequadas à sua realidade. Nos últimos 20 anos, as tecnologias da informação e comunicação têm ganho enorme importância na gestão de unidades hoteleiras e similares, e a sua inexistência ou incorrecta aplicação pode ser um entrave ao sucesso destas unidades. Um exemplo desta evolução é o surgimento de plataformas online de distribuição de viagens (OTA – *Online Travel Agency*), como a Expedia ou a Booking.com, através das quais é possível efectuar a reserva de quartos sem o contacto directo com a unidade hoteleira, e que em 2012 eram já consideradas dos maiores agentes de viagens do mundo (Expedia o maior, Booking.com o 3º maior) [3].

Inicialmente, as ferramentas de TI utilizadas no sector do alojamento baseavam-se, sobretudo, em sistemas de gestão de propriedade, os *Property Management Systems* (PMS) – usados na gestão das operações centrais dos hotéis, tais como *check-in* e *check-out* de hóspedes e gestão de quartos – ou em suites que expandem estes sistemas. Estas são ferramentas monolíticas e que possuem alguns constrangimentos associados. Um destes está relacionado com a expansão de um sistema deste tipo estar restringida a soluções comercializadas pela organização por ele responsável, o chamado *vendor lock-in*, devido à inexistência de integrações possíveis com outros sistemas. Outro constrangimento são as limitações no suporte a todos os processos de negócio do sector, que diferem consoante as características da unidade hoteleira, e para os quais estas ferramentas podem não ser capazes de oferecer as funcionalidades adequadas e/ou necessárias. As limitações no acompanhamento das novas necessidades do mercado de forma rápida e ágil são outro problema na adopção de uma destas ferramentas.

Estas novas necessidades, que passam, por exemplo, pela capacidade de as empresas possuírem uma presença digital, oferecerem serviços a preços competitivos diariamente e responderem rapidamente às necessidades dos seus clientes, levaram ao aparecimento e adopção de ferramentas que rapidamente se tornaram factores críticos de sucesso no sector. Alguns exemplos de categorias destas ferramentas são: o *website* do hotel, o gestor de canais de distribuição (*channel manager*), o gestor de inventário e preços (*revenue management system*), a ferramenta de análise de preços (*rate shopper*), o gestor de reputação (*reputation management system*), a ferramenta de facturação, entre outros. Em cada uma destas categorias normalmente seleccionam-se as ferramentas *best-of-breed* (BoB) – as melhores na sua área de actuação [4] – muitas vezes comercializadas como soluções *Software-as-a-Service* (SaaS), cuja utilização pode ser realizada mediante a sua subscrição. Para endereçar as diferentes necessidades do negócio, em alguns casos, os hotéis têm a necessidade de subscrever vários tipos destes serviços, dificultando a gestão desse portfólio de ferramentas.

A adopção de tais serviços pelos hotéis e alojamentos independentes pode tornar-se numa operação morosa e dispendiosa, devido a factores como: a análise e selecção de serviços; a aprendizagem/habituação à utilização dos mesmos (curva de aprendizagem); e a integração dos novos serviços com o portfólio actual da organização. Na maioria dos casos, a adopção dos serviços é feita pelo próprio pessoal do hotel, o qual pode não possuir conhecimentos técnicos suficientes, resultando em portfólios de serviços desorganizados, sobrepostos, e pouco alinhados com os processos de negócio da organização, levando à duplicação de tarefas e dados. A duplicação de dados é propensa a erros que podem resultar em perdas consideráveis para o hotel, com impacto na sua competitividade, e implica a alocação de recursos humanos que deveriam estar focados nas tarefas mais relevantes do negócio – garantir uma boa estadia/experiência aos seus clientes.

1.1.2. O projecto Hotelcracy Apps como resposta

Consciente dos desafios do sector, foi criado o consórcio do projecto Hotelcracy Apps, a partir de uma parceria entre o IPNlis e a empresa Hotelcracy Software Lda.

O projecto Hotelcracy Apps propõe o desenvolvimento de uma plataforma que permitirá ao hoteleiro subscrever, migrar, cancelar e utilizar serviços SaaS BoB de terceiros através de uma interface centralizada e homogénea. A plataforma funcionará, ela própria, como um serviço SaaS, e será orientada para as unidades hoteleiras independentes.

A escolha e subscrição das soluções SaaS será feita com recurso a um *Marketplace* da plataforma a desenvolver, no qual o hoteleiro poderá escolher quais os serviços SaaS BoB que pretende utilizar para cada área do negócio (por exemplo, a facturação e a gestão de quartos e reservas). A integração dos vários serviços ficará a cargo da plataforma, e deverá permitir que o utilizador tire partido dela de um modo suave e contínuo, sem interrupções aparentes. Como exemplo desta integração e posterior utilização nas operações do hotel, a plataforma permitirá a um hoteleiro emitir uma factura relativa à estadia de um cliente, e realizar a cobrança da mesma. Para tal, os dados do cliente e dos consumos efectuados são obtidos a partir do serviço de PMS, e inseridos no serviço de facturação, para emissão da factura. Os dados para pagamento podem depois ser introduzidos no serviço de pagamento, e a cobrança ser efectuada.

O hoteleiro poderá ainda, a qualquer momento, reconfigurar o conjunto de serviços em uso, através da subscrição de um novo serviço, ou através da migração para outro da mesma categoria, encarregando-se a plataforma de realizar todas as integrações e transferências de dados necessárias. Como exemplo de migração de serviços, a plataforma permitirá a um hoteleiro alterar o serviço de *channel manager* que é utilizado no hotel. Este tipo de serviço é responsável pela distribuição dos quartos pelos vários canais de distribuição, e pela gestão das reservas, disponibilidade e preços. A alteração do serviço A para o serviço B deverá, quando possível, migrar os dados relativos a quartos, canais de distribuição, planos de tarifa e outros, presentes no serviço A inicial, para o novo serviço B.

A interacção com o utilizador será feita através de uma interface homogénea, que o abstrairá da complexidade e heterogeneidade das soluções subjacentes. Para além de permitir gerir os serviços subscritos, permitirá o uso desses mesmos serviços, sem necessidade de aceder à interface nativa de cada um deles, tirando partido das integrações entre todos. A alteração dos serviços subscritos para outros da mesma categoria manterá a interface com o utilizador, reduzindo, ou até removendo, a curva de aprendizagem dos colaboradores do hotel.

A plataforma permitirá, assim, mitigar dificuldades dos hoteleiros na integração dos serviços utilizados no hotel, e fomentar a adopção dos que melhor sirvam as suas necessidades de negócio, reduzindo a actual dependência de suites monolíticas e o *vendor lock-in*.

1.2. Objectivos do estágio

O presente estágio contribui para o projecto descrito com o desenvolvimento de uma prova de conceito que implemente a migração entre serviços SaaS. Esta operação permitirá aos hoteleiros alterarem de uma solução subscrita para outra mais favorável, mantendo o máximo de informação possível.

A prova de conceito tem como propósito avaliar a viabilidade desta operação através da plataforma Hotelcracy proposta. Deve possuir funcionalidades que permitam a migração de um serviço SaaS para outro serviço da mesma categoria, sendo realizada, quando possível, a transição de configurações e dados do serviço inicial para o novo serviço, utilizando as funcionalidades disponibilizadas nas APIs (*Application Programming Interface*) destes. Será considerado apenas o caso em que o utilizador subscreveu o serviço SaaS inicial através da plataforma Hotelcracy Apps, utilizou o serviço apenas através da plataforma, e pretende realizar a migração para um serviço SaaS novo da mesma categoria.

Ao nível da integração de serviços SaaS, deve ser realizado um estudo para identificação e análise das abordagens de integração existentes, avaliando a sua aplicabilidade e adequação ao projecto.

Os processos e artefactos relacionados com a subscrição original e o cancelamento de serviços SaaS ficam fora do âmbito deste estágio, sendo tratados noutras tarefas do projecto.

1.3. Documentação produzida

Um dos principais contributos do estágio foi a produção de documentos técnicos do projecto, incluindo entregáveis que serão avaliados pela entidade de gestão do programa PT2020.

Na Tabela 1 listam-se e descrevem-se sucintamente os documentos que contemplam o conhecimento criado durante o estágio.

Documento	**Análise de soluções de integração de serviços *cloud***
Descrição	Este documento contém informação recolhida relativa à análise de soluções de integração existentes no mercado. Contém tabelas com informação detalhada de cada solução de integração, e uma avaliação de cada, em relação à sua aplicação ou não no projecto.
Anexo	HC_A1-Analise_Solucoes_Integracao
Documento	**Descrição do modelo conceptual do sistema**
Descrição	Este documento contém informação acerca do modelo conceptual definido para a plataforma Hotelcracy Apps, produzido na Actividade 1 do projecto. O trabalho do estagiário neste documento incidiu na revisão do mesmo.
Anexo	HC_A2-Modelo_Conceptual
Documento	**Documento de arquitectura da plataforma Hotelcracy Apps**
Descrição	Este documento possui a especificação detalhada da arquitectura da plataforma Hotelcracy Apps. É um documento colaborativo, que contempla informação dos *drivers* arquitecturais identificados, as decisões tomadas, e os diagramas que ilustram a arquitectura. O trabalho do estagiário no documento divide-se em: ajuda na produção das secções de restrições (secção 4) e atributos de qualidade (secção 5); produção das secções de decisões (secção 6), de diagramas (secção 7), e de conclusões (secção 8); e revisão de todo o documento.
Anexo	HC_A3-Arquitectura_Plataforma

Documento	**Documento de especificação detalhada do componente Services Catalogue**
Descrição	Este documento possui a especificação detalhada do componente Services Catalogue. É um documento colaborativo, que contempla informação relativa a entidades de negócio do componente, funcionalidades que este disponibiliza, tecnologias aplicadas, contexto de execução, arquitectura interna, e especificação das suas interfaces. O trabalho do estagiário no documento dividiu-se na produção inicial das secções de entidades de negócio (secção 2), funcionalidades (secção 3), tecnologias aplicadas (secção 4), aspectos arquitecturais (secção 5) e especificação da interface do componente (secção 6), bem como na revisão de todo o documento e das várias evoluções que outros elementos da equipa realizaram.
Anexo	HC_A4-Services_Catalogue
Documento	**Documento de especificação detalhada do componente Orchestrator**
Descrição	Este documento possui a especificação detalhada do componente Orchestrator. É um documento colaborativo, que contempla informação relativa a entidades de negócio do componente, funcionalidades que este disponibiliza, tecnologias aplicadas, contexto de execução, arquitectura interna, e especificação das suas interfaces. O trabalho do estagiário no documento dividiu-se na sua produção inicial, bem como na revisão das várias evoluções que outros elementos da equipa realizaram.
Anexo	HC_A5-Orchestrator
Documento	**Documento de especificação detalhada do componente Migrator**
Descrição	Este documento possui a especificação detalhada do componente Migrator. É um documento colaborativo, que contempla informação relativa a entidades de negócio do componente, funcionalidades que este disponibiliza, tecnologias aplicadas, contexto de execução, arquitectura interna, e especificação das suas interfaces. O trabalho do estagiário no documento incidiu na sua produção e na aplicação de alterações e sugestões de outros membros da equipa.
Anexo	HC_A6-Migrator

Tabela 1 - Documentação produzida

1.4. Estrutura do relatório

Realizada a presente introdução, o resto do relatório encontra-se organizado da seguinte forma: no capítulo 2 são descritos alguns conceitos base aplicados ao longo do relatório; o capítulo 3 apresenta o planeamento inicial e o final do estágio, bem como as justificações para

os desvios verificados; no capítulo 4 é apresentado o estudo do estado da arte levado a cabo, o qual se foca na integração de serviços *cloud*; o capítulo 5 descreve a arquitectura da plataforma, desde a sua concepção inicial até à especificação final, com passagem pelo estudo realizado em relação à migração de serviços e pelo impacto desta na estruturação da plataforma; no capítulo 6 é apresentado o Modelo Canónico de dados definido para a plataforma; o capítulo 7 aborda o desenvolvimento da prova de conceito; por último, no capítulo 8 são apresentadas as conclusões do estágio, sendo feito um sumário do trabalho realizado, as limitações deste, e o trabalho a realizar no futuro.

Capítulo 2
Conceitos e contextualização

O presente capítulo tem como propósito descrever e clarificar conceitos que serão utilizados ao longo deste relatório, tornando claro o seu significado e âmbito no contexto do projecto.

2.1. Serviços *Software-as-a-Service* (SaaS)

Segundo o *National Institute of Standards and Technology* dos Estados Unidos da América, um serviço SaaS é uma aplicação em execução numa infraestrutura de *cloud computing* que pode ser acedida e utilizada por vários clientes através de um *browser* ou de um outro programa que sirva de interface com o utilizador [5]. O consumidor do serviço não gere nem controla as capacidades da aplicação ou a infraestrutura em que esta executa, sendo-lhe apenas possível controlar determinadas configurações específicas ao utilizador.

Como referido anteriormente no relatório, existe um conjunto de ferramentas especializadas utilizadas no sector hoteleiro que são comercializadas como serviços SaaS. É no aproveitamento deste tipo de ferramentas especializadas que o projecto se foca.

2.2. Categorias de serviços SaaS do sector do alojamento

As categorias de serviços referem-se aos grupos de ferramentas/serviços comumente utilizados no sector do alojamento. Numa das primeiras tarefas do projecto identificaram-se 15 categorias de serviços [6]. Seguidamente, listam-se e descrevem-se sumariamente as categorias de serviços identificadas:

- ***Property Management System*** **(PMS)** – Serviços usados na gestão das operações centrais do hotel, tais como: *check-in* e *check-out* de hóspedes; gestão de perfis de hóspedes; funções de *front* e *back office*; auditoria; gestão de informação relativa a quartos e serviços de limpeza.
- ***Channel Manager*** – Serviços usados na gestão de quartos, para: disponibilização de quartos pelos diversos canais de distribuição, como *websites* de reservas (p.e., Booking.com) ou operadores turísticos; gestão de reservas efectuadas em diferentes locais *online*; gestão de preços e disponibilidade dos quartos.
- ***Revenue Management System*** – Serviços usados no controlo do inventário do hotel (quantidades de quartos e respectivos preços), com o objectivo de obter o máximo de retorno ou lucro, através de, por exemplo, aplicação de tarifas dinâmicas em função da ocupação dos quartos.
- ***Reputation Manager*** – Serviços usados na automatização da análise de críticas ao hotel, partilhadas pelos hóspedes nas redes sociais, de modo a que este possa actuar mais prontamente sobre possíveis problemas identificados.
- ***Rate Shopper*** – Serviços usados na recolha e análise de preços da concorrência, sendo a informação obtida utilizada pelo *Revenue Management System* a fim de obter o máximo de lucro.

- ***Customer Relationship Management* (CRM)** – Conjunto de princípios, práticas e directrizes que uma organização segue na interacção com os seus clientes, sendo o sector hoteleiro um caso em que estas estratégias se tornam essenciais, pois todos os seus serviços giram em torno dos hóspedes. Para facilitar, ou até tornar possíveis estas estratégias, existem ferramentas que automatizam determinadas tarefas de CRM, tais como: recolha e análise de informação de clientes; gestão de reclamações; desenvolvimento e execução de campanhas publicitárias orientadas para o cliente.
- ***Booking Engine*** – Serviços que permitem ao hotel disponibilizar aos seus clientes um modo de efectuarem reservas directamente no seu *website*, evitando, ao hotel, as comissões cobradas pelos *websites* especializados na venda de produtos de viagens.
- **Facturação e pagamentos** – Serviços que permitem gerir o rastreamento de produtos e serviços prestados, e facturá-los aos clientes. Este tipo de ferramentas facilita a administração de um negócio, automatizando tarefas demoradas como a preparação de facturas.
- **Sistema *Point-of-Sale* (POS)** – Um sistema POS é a substituição informatizada de uma caixa registadora, e é utilizado na gestão de transacções realizadas no interior do hotel, mais propriamente nos serviços de bar e restauração.
- **Terminal de Pagamento Automático (TPA) Virtual** – Sistemas de pagamentos que visam facilitar as transacções comerciais, permitindo a realização de pagamentos electrónicos com recurso a cartões bancários, sem a necessidade de um TPA físico.
- **Operações de manutenção e relacionados** – Serviços usados na gestão da manutenção de infraestruturas e equipamentos, como gestão de avarias e de intervenções de reparação em edifícios e equipamentos.
- **Aplicações móveis** – Sendo disponibilizadas a clientes e não-clientes, as aplicações móveis podem funcionar como guia turístico digital e recepcionista, fornecendo informações relevantes, como direcções para o alojamento ou pontos de interesse nas imediações deste. São uma mais valia na retenção e relação com os clientes, aumentando a sua fidelidade.
- **Gestão de recursos humanos** – Serviços que permitem gerir e processar toda a informação referente aos recursos humanos de uma organização, permitindo, por exemplo, agendar horas de trabalho e comunicar com os trabalhadores.
- **Gestão financeira/tesouraria** – Serviços usados no planeamento, análise e controlo de contas, permitindo aos gestores visualizarem a situação financeira da organização.
- **Sistemas de gestão de acessos** – Serviços usados na gestão de acessos a propriedades, permitindo a abertura de portas de quartos e casas através da Internet, gerar acessos temporários, e saber quem acedeu, evitando a necessidade de uma presença física do proprietário.

2.3. *Application Programming Interface* (API)

Uma *Application Programming Interface* (API) é um conjunto de comandos, funções, protocolos, e objectos, que os programadores podem utilizar para desenvolver *software* que interage com um sistema externo [7]. Uma característica habitual dos serviços SaaS é a disponibilização de uma API, permitindo que uma determinada solução (*software)* comunique directamente com outra.

2.4. Integração de serviços

No contexto do projecto, a integração de serviços refere-se à execução de operações que envolvam a comunicação com duas ou mais soluções SaaS subscritas/fornecidas através da plataforma Hotelcracy Apps. Esta integração envolve o estabelecimento de comunicações entre as várias soluções SaaS consideradas, para acesso às funcionalidades por estas disponibilizadas, com recurso à API de cada uma. A plataforma proposta funcionará como iniciador e intermediário dessas comunicações. Fora deste âmbito fica a integração com serviços que não estejam contemplados pela plataforma, ou com sistemas *on-premises* e/ou proprietários das unidades hoteleiras.

2.5. Migração de serviços

No contexto do projecto, a migração de serviços refere-se à execução de operações que permitam que um hoteleiro cliente da plataforma Hotelcracy Apps transite de um serviço SaaS previamente subscrito através desta, para outro da mesma categoria, também ele subscrito através da plataforma. Esta operação implica, quando possível, a migração das configurações e dados presentes no serviço SaaS inicial para o novo serviço subscrito. A motivação para a transição para outro serviço pode dever-se à existência de uma solução com um menor custo, e/ou o acesso a funcionalidades disponíveis. Fora deste âmbito fica a migração de serviços SaaS que não estejam contemplados pela plataforma, ou de sistemas *on-premises* e/ou proprietários das unidades hoteleiras.

Capítulo 3
Planeamento e execução do estágio

Nesta secção é apresentado o planeamento do estágio e a metodologia de trabalho utilizada. O planeamento pretende mostrar a evolução deste, desde a sua especificação inicial até ao final do estágio, sendo enunciados e justificados os desvios que se verificaram. É também apresentada a participação do estagiário na gestão do projecto Hotelcracy Apps.

3.1. Planeamento do estágio

Como enunciado anteriormente, o projecto Hotelcracy Apps possui uma duração de 33 meses, e possui um planeamento próprio, especificado na fase da candidatura ao PT2020. Resumidamente, esse planeamento inclui as seguintes actividades:

1. Análise de serviços SaaS e definição de modelo conceptual do sistema
2. I&D do sistema: Análise e concepção tecnológica do sistema
3. Criação de componentes complexos do sistema e execução de testes em contexto laboratorial
4. Desenvolvimento de componentes de suporte, instalação e validação experimental do sistema
5. Coordenação e gestão de projeto
6. Divulgação e promoção de resultados

Estas actividades dividiam-se em tarefas, distribuídas pelos dois parceiros do consórcio (um parceiro era o responsável pela execução de determinada tarefa, enquanto que o outro possui um papel de apoio a essa execução). O trabalho realizado no presente estágio incidiu em quase todas as actividades do projecto (à excepção das actividades 4 e 6), sendo apresentadas na Tabela 2 as tarefas concretas para as quais o estágio contribuiu.

Tarefa	Data de início	Data de fim
Actividade 1		
Investigação e definição de processo automático de migração de subscrições de serviços da mesma categoria	01/03/2017	30/05/2017
Definição de modelo concetual do sistema	01/04/2017	31/07/2017
Actividade 2		
Desenho da arquitectura do sistema	01/07/2017	31/01/2018
Elaboração da especificação detalhada dos componentes do sistema	01/08/2017	31/01/2018
Análise e especificação tecnológica dos componentes do sistema	01/09/2017	31/01/2018

Tarefa	Data de início	Data de fim
Actividade 3		
Construção e validação laboratorial dos componentes complexos	01/11/2017	31/09/2018
Testes e ensaios de laboratório	01/03/2018	30/10/2018
Actividade 5		
Coordenação técnico-científica e financeira	01/10/2016	30/06/2019

Tabela 2 - Planeamento do projecto Hotelcracy Apps (excerto)

O planeamento do estágio está intimamente relacionado com as actividades do planeamento do projecto Hotelcracy Apps. Sendo este um projecto de investigação e desenvolvimento, o seu planeamento não é tão previsível como o de um projecto de *software* típico, o que leva a variações nas actividades e tarefas inicialmente planeadas e na sua duração. Assim, o estágio está igualmente sujeito a essas variações, as quais se verificaram, e que serão apresentadas e justificadas nesta secção.

3.1.1. Planeamento do primeiro semestre

A Figura 1 apresenta o planeamento inicial das actividades (e respectivas tarefas) do estágio para o primeiro semestre, proposto pelo IPNlis.

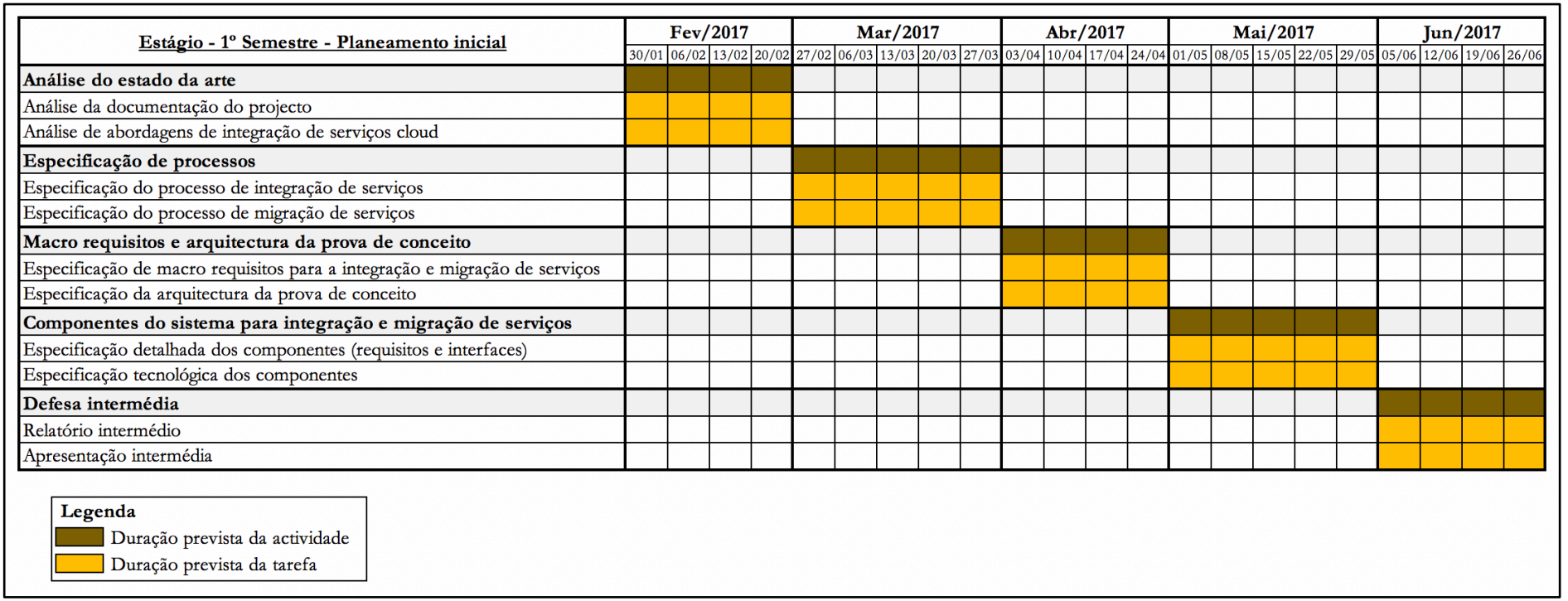

Figura 1 - Diagrama de Gantt inicial do 1º semestre do estágio, proposto pelo IPNlis

A Figura 2 apresenta a execução real das actividades durante o primeiro semestre do estágio, a qual possui diferenças relevantes em relação ao inicialmente planeado.

Do planeamento inicial, as seguintes actividades foram concluídas:

- **Análise do estado da arte:** esta actividade envolveu a análise da documentação do projecto, a análise de abordagens à integração de serviços *cloud*, a análise de plataformas de integração de serviços *cloud*, e a análise de uma tecnologia de orquestração de processos de negócio;
- **Especificação de processos:** esta actividade envolveu a especificação de processos genéricos para a integração e migração de serviços *cloud*;
- **Macro requisitos e arquitectura da prova de conceito:** esta actividade foi parcialmente concluída, tendo apenas sido executada e concluída a tarefa "Especificação de macro requisitos para integração e migração de serviços";
- **Defesa intermédia do estágio:** esta actividade envolveu a elaboração do relatório intermédio, e a elaboração e preparação da apresentação para a defesa intermédia do estágio.

As restantes actividades não foram concluídas:

- **Macro requisitos e arquitectura da prova de conceito:** a tarefa "Especificação da arquitectura da prova de conceito para os processos de integração e migração de serviços" não foi realizada devido a incertezas relacionadas com tecnologias e opções arquitecturais não estarem resolvidas aquando da sua execução. Deste modo, a tarefa foi adiada para o segundo semestre;
- **Componentes do sistema para integração e migração de serviços:** a complexidade associada a esta actividade só poderia ser endereçada com conhecimento que não estava disponível aquando da sua execução. Assim, o seu objectivo foi alterado, e a actividade adiada para o segundo semestre.

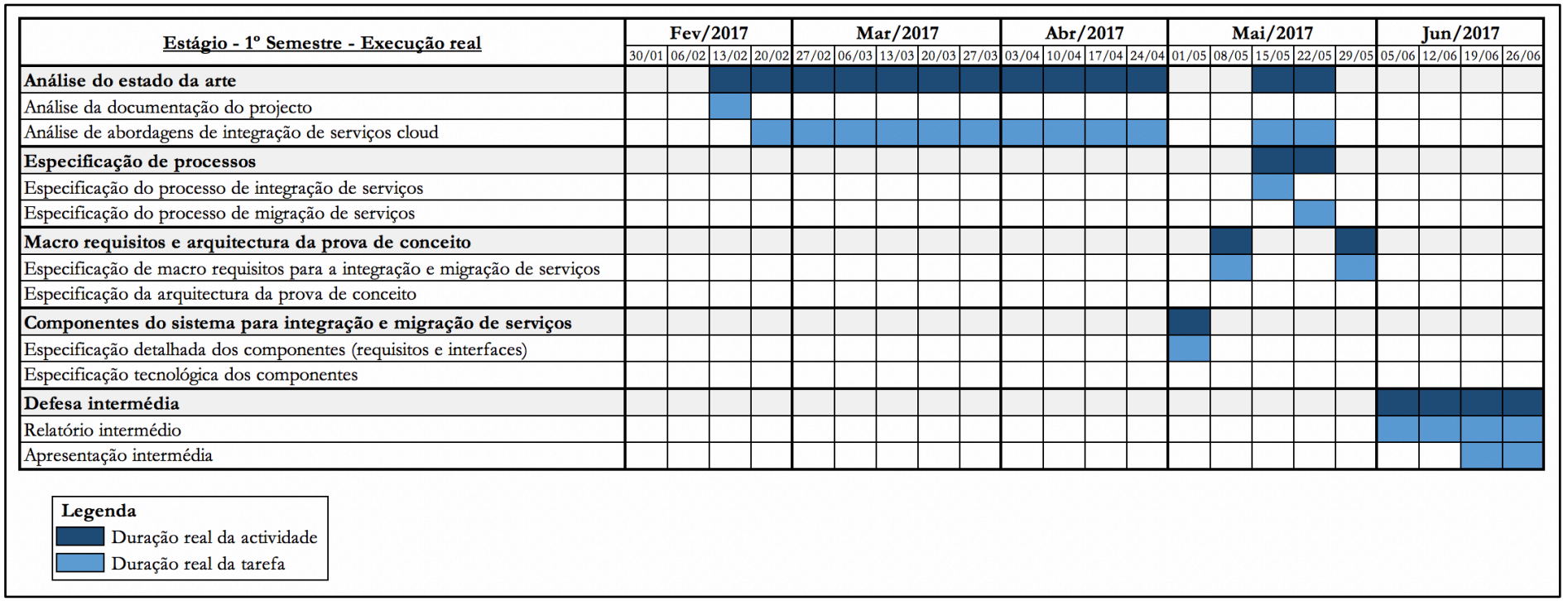

Figura 2 - Diagrama de Gantt relativo à execução do 1º semestre do estágio

De seguida, apresentam-se as justificações para as diferenças entre o planeamento inicial das actividades para o primeiro semestre e a sua execução real.

Actividade "Análise do estado da arte"

Na Figura 2 é possível verificar que a actividade relativa à análise do estado da arte teve uma duração muito superior ao planeado, tendo sido necessário alocar mais tempo para a execução de algumas das suas tarefas.

Para a análise de abordagens à integração de serviços *cloud*, a descoberta e recolha de documentos revelou ser um processo lento. A especificidade do tema pretendido resultou em poucos documentos relacionados com este, os quais foram de difícil descoberta. Alguns dos documentos analisados não estavam escritos de um modo claro e estruturado, o que dificultou a sua interpretação.

Relativamente às plataformas de integração de serviços *cloud*, a sua análise revelou ser uma tarefa morosa. A quantidade de plataformas identificadas foi aumentando ao logo da análise, com um total de 25 analisadas segundo um processo estruturado. Este processo implicou a análise de aspectos específicos das plataformas, cuja informação foi difícil de identificar. A análise focou-se na documentação e noutros artefactos destas, que, em alguns casos, era extensa e/ou desorganizada.

Em conclusão, a complexidade da análise de abordagens à integração, e a necessidade de avaliar um elevado número de plataformas de integração contribuíram de modo significativo para o aumento da duração desta actividade.

Actividade "Macro requisitos e arquitectura da prova de conceito"

A actividade "Macro requisitos e arquitectura da prova de conceito" foi parcialmente executada, uma vez que a tarefa "Especificação da arquitectura da prova de conceito para os processos de integração e migração de serviços" não foi concluída no primeiro semestre. A integração dos serviços *cloud* nos processos de negócio do sector do alojamento era um assunto que se encontrava ainda a ser discutido pelo consórcio durante a execução desta tarefa, e para o qual não existia ainda uma decisão de como iria ser tratado. Esta era uma decisão que teria impacto directo na arquitectura da prova de conceito, bem como na arquitectura do próprio projecto, daí o adiamento da tarefa para o segundo semestre.

Actividade "Componentes do sistema para integração e migração de serviços"

O planeamento proposto pelo IPNlis colocou esta actividade numa fase final do primeiro semestre do estágio, no entanto, uma vez que envolvia a colaboração com outro estágio, foi executada mais cedo, como é possível ver na Figura 2.

Inicialmente, a actividade envolvia uma tarefa de especificação de um modelo para a gestão de dados e suporte à interacção com os serviços SaaS. Esta especificação foi iniciada, em colaboração com o estagiário do outro estágio (o qual terminou em Julho de 2017). No entanto, foi considerado, por um dos *stakeholders* do projecto, que esta especificação não era possível de ser realizada na altura para a qual estava planeada. O conhecimento relacionado com os tipos de dados que a plataforma teria que lidar não era o suficiente, o que tornava impossível especificar um modelo correcto.

Por esta razão, a abordagem foi alterada, sendo considerado, como ponto de partida, a especificação de um mínimo denominador comum dos dados necessários à interacção com as

APIs dos serviços SaaS. Assim, o objectivo da tarefa para este estágio foi alterado, passando a ser a implementação de uma interface/protocolo interno à plataforma Hotelcracy Apps, como suporte às comunicações entre os seus componentes. A actividade foi, assim, adiada para o segundo semestre do estágio.

3.1.2. Planeamento do segundo semestre

A Figura 3 apresenta o planeamento inicial das actividades (e respectivas tarefas) do estágio para o segundo semestre, proposto pelo IPNlis.

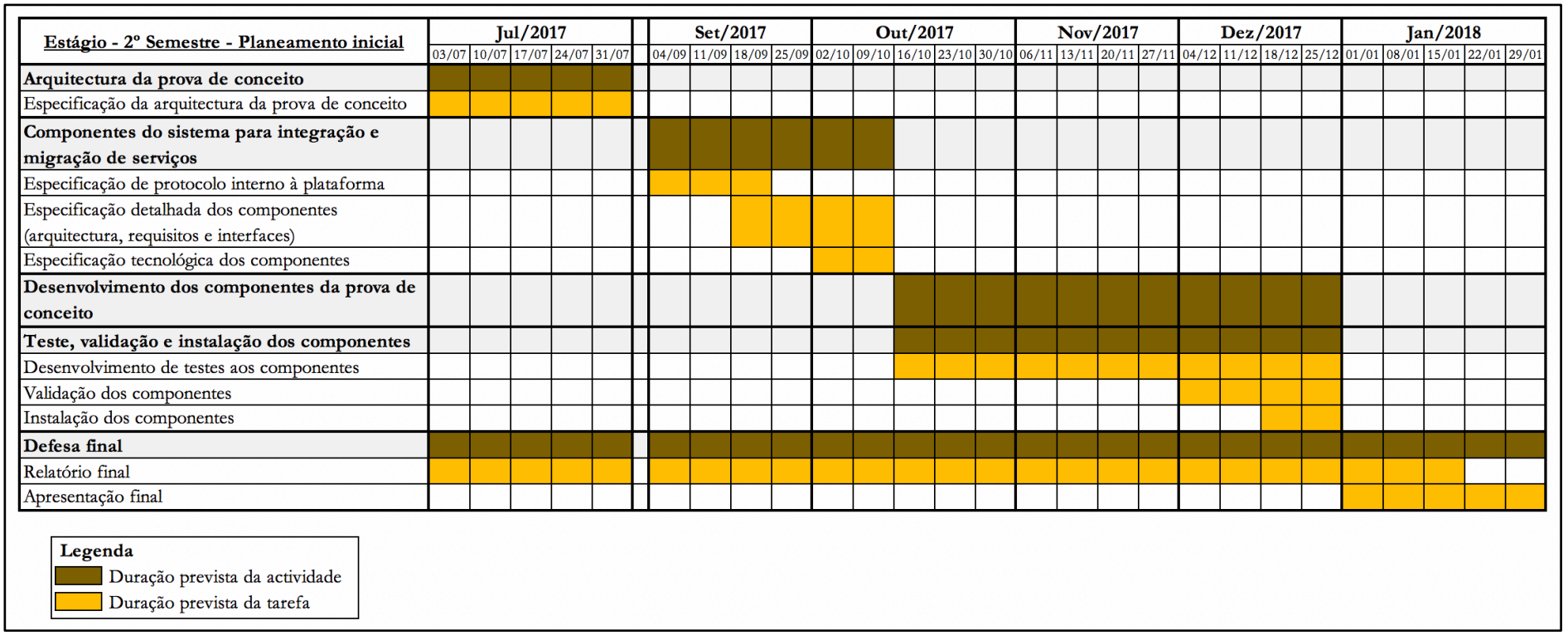

Figura 3 - Diagrama de Gantt inicial do 2º semestre do estágio, proposto pelo IPNlis

A Figura 4 e a Figura 5 apresentam a execução real das actividades durante o segundo semestre do estágio, a qual possui diferenças relevantes em relação ao inicialmente planeado. A diferença mais relevante é a extensão do tempo de execução do estágio, que compreendeu um prolongamento por um período de mais 6 meses. Outra diferença relevante é a alteração das actividades a executar, com a adição de novas actividades e a reorganização de algumas das que já estavam definidas. Estas duas diferenças são explicadas nos próximos parágrafos.

Incremento na duração do estágio

Aquando do início do segundo semestre do estágio, a equipa do IPNlis sofreu uma reestruturação, com a saída e entrada de elementos. Esta mudança levou a que o estagiário passasse a desempenhar uma posição mais proeminente no projecto, sendo-lhe atribuídas responsabilidades e tarefas mais próximas da gestão técnica do mesmo. Na secção 3.3 do relatório é apresentada em detalhe esta participação na gestão do projecto por parte do estagiário.

Com o decorrer do estágio, o trabalho a ser realizado neste tornou-se muito dependente das evoluções que se faziam nas actividades e tarefas do próprio projecto. Tal levou a que a execução das tarefas do estágio seguisse de modo muito próximo o calendário do projecto. Assim, tendo em consideração o atraso verificado no primeiro semestre e a participação na gestão técnica do projecto, houve a necessidade de estender o estágio durante mais um semestre, de modo a permitir a realização de todas as actividades.

Reorganização das tarefas do estágio

A proximidade às actividades e tarefas do próprio projecto levou também à reorganização de algumas das actividades e tarefas do estágio, em concreto nos seguintes casos:

- Foi adicionada a actividade "Arquitectura da plataforma Hotelcracy Apps", a qual aglutinou o trabalho a ser realizado na actividade inicial "Arquitectura da prova de conceito". Esta nova actividade incluiu a definição do modelo conceptual da plataforma, um estudo aprofundado da migração de serviços e do seu impacto na arquitectura da plataforma, e a especificação desta última;
- Foi adicionada a actividade "Especificação de protocolo interno à plataforma: Modelo Canónico", a qual aglutinou o trabalho a ser realizado na tarefa inicial "Especificação de protocolo interno à plataforma". Esta nova actividade incluiu o levantamento e estudo de entidades dos serviços SaaS do sector hoteleiro, e a especificação e evolução do modelo de dados definido;
- Foi adicionada a actividade "Gestão técnica do projecto", a qual incluíu a gestão de documentos técnicos do projecto, e a participação nas variadas reuniões do projecto e na co-orientação dos vários elementos da equipa do IPNlis, numa vertente mais técnica.

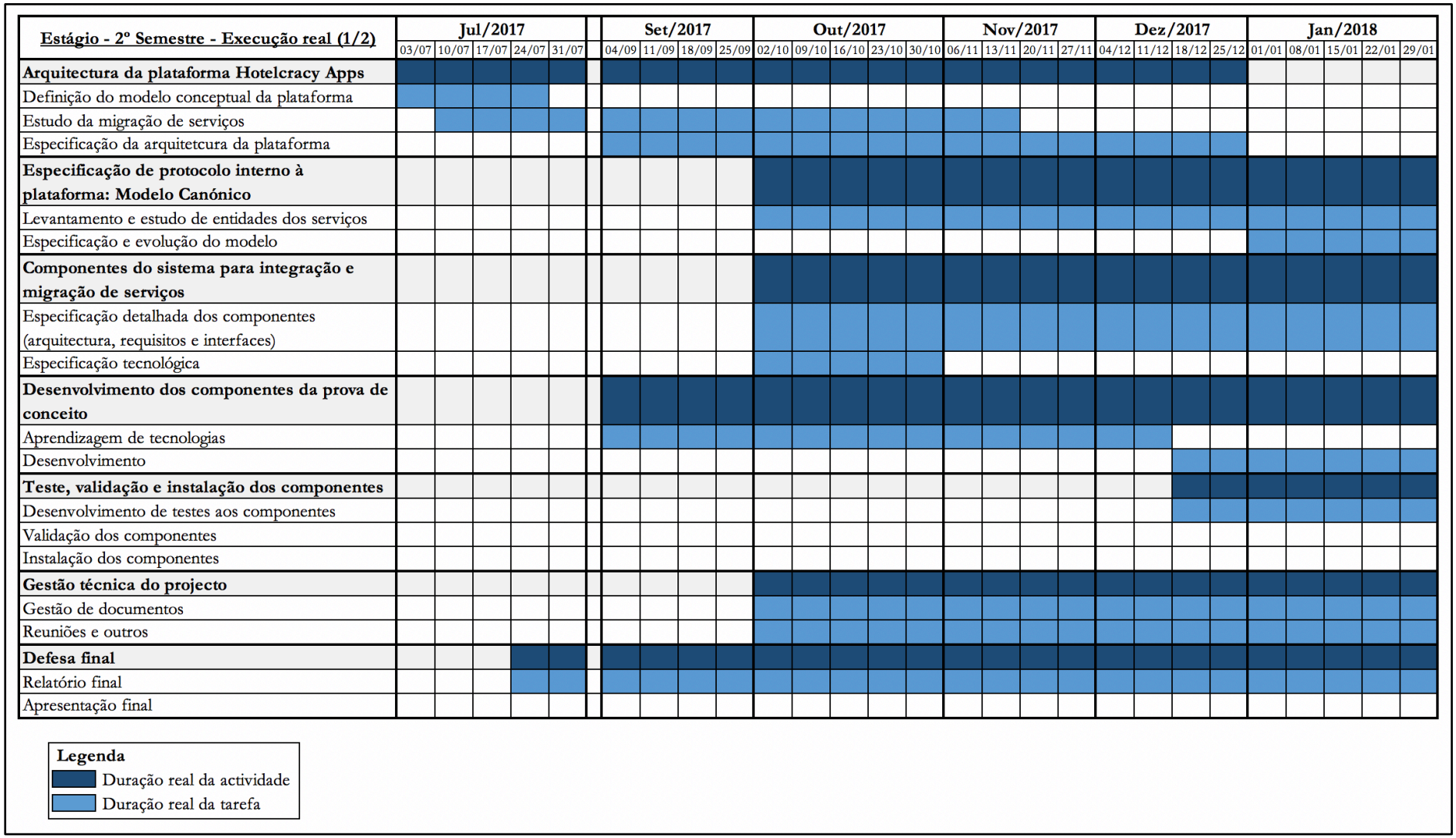

Figura 4 - Diagrama de Gantt relativo à execução do 2º semestre do estágio (parte 1)

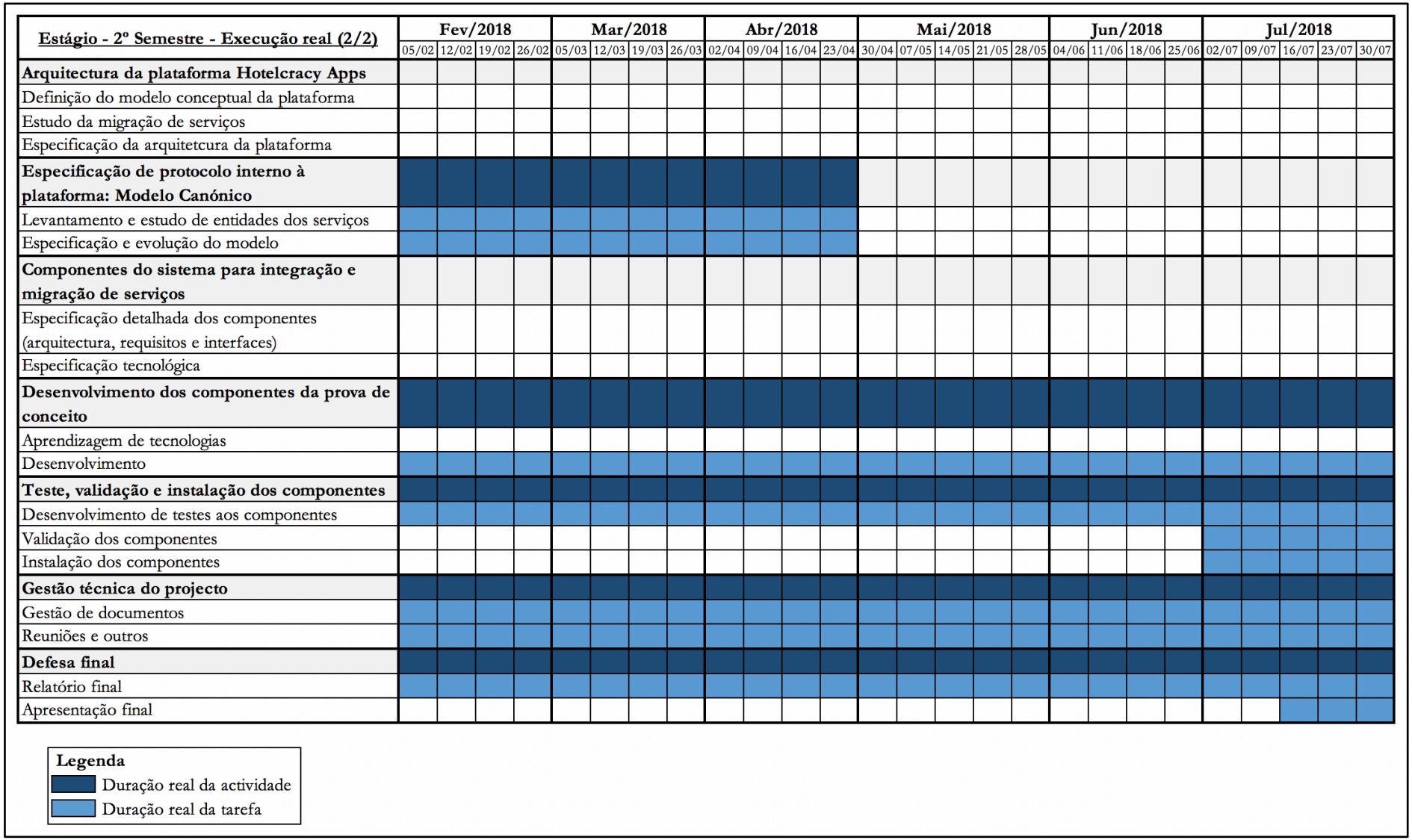

Figura 5 - Diagrama de Gantt relativo à execução do 2º semestre do estágio (parte 2)

3.2. Metodologia de trabalho

O projecto Hotelcracy Apps possui um planeamento geral, definido aquando da candidatura ao PT2020, e que segue uma estrutura baseada no Modelo em Cascata (Waterfall Model). Esta é uma abordagem linear e sequencial, aplicada na engenharia de software. Tende a ser pouco flexível e iterativa, uma vez que o progresso do desenvolvimento segue maioritariamente numa única direcção (de cima para baixo, como uma cascata) através de várias fases que compõem este modelo, como apresentado na Figura 6. O resultado de uma fase é, habitualmente, utilizado como artefacto de entrada da fase seguinte [8], [9]. Estas fases traduzem-se nas actividades que compõem o planeamento do projecto, e que foram enunciadas na secção anterior (secção 3.1).

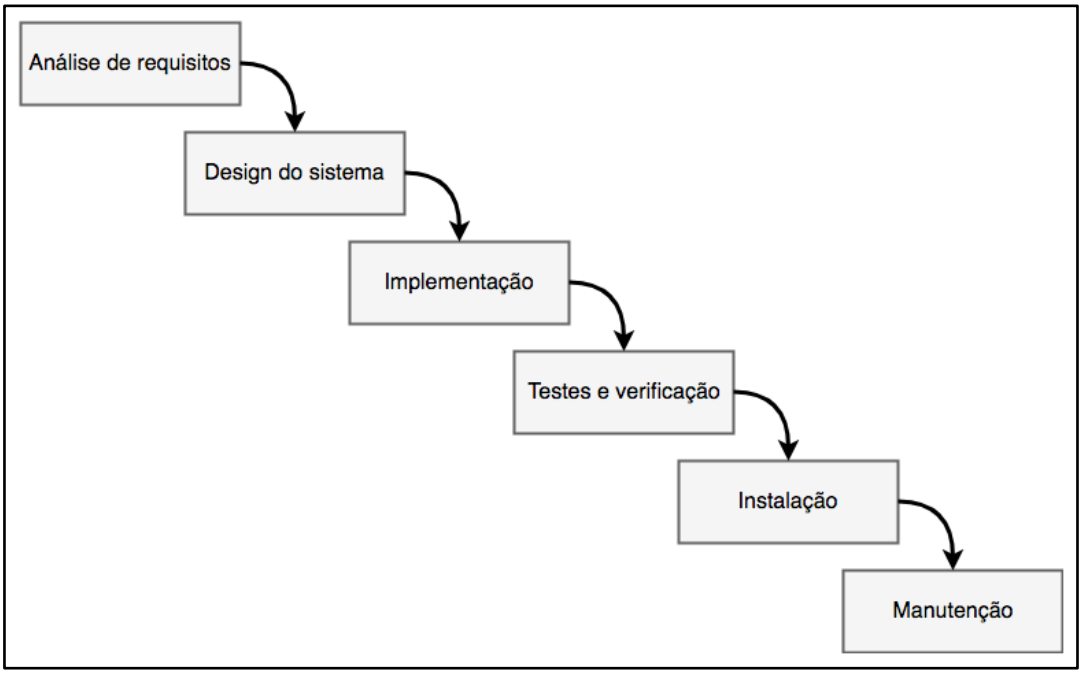

Figura 6 - Modelo em Cascata (baseado em [9]*)*

Como já referido, o projecto é desenvolvido por um consórcio composto por dois parceiros (a Hotelcracy Software, Lda., e o IPNlis) que seguem o mesmo planeamento geral, trabalhando em partes interdependentes. De modo a que os parceiros estejam sincronizados, são realizadas reuniões de ponto de situação do projecto, com uma regularidade de três a quatro semanas, onde estão presentes todos os membros das duas equipas. Nestas reuniões é apresentado o trabalho realizado por cada parceiro desde a última reunião, e são discutidos aspectos estruturais do projecto (por exemplo, lógica de negócio e arquitectura da plataforma) com consequentes tomadas de decisão. O estagiário participou activamente nestas reuniões, contribuindo para as discussões relacionadas com a concepção e o desenvolvimento da plataforma. Era também responsável pela preparação de documentos formais da reunião (agenda e acta), pela apresentação do trabalho realizado pelo IPNlis, fazendo uso de uma apresentação em slides quando necessário, e pela demonstração das evoluções realizadas no desenvolvimento da plataforma.

Para as actividades do projecto relativas ao desenvolvimento da plataforma, foi aplicada uma abordagem baseada na metodologia Scrum, por acordo entre os parceiros do consórcio. Esta é uma metodologia ágil para gestão de trabalho de um modo iterativo e incremental, que permite dar resposta à imprevisibilidade e aos problemas complexos normalmente associados

a algumas situações do desenvolvimento de software. O Scrum apoia-se num fluxo de trabalho bem definido (Figura 7) o qual se estende desde o planeamento do desenvolvimento até à disponibilização de software finalizado. Este fluxo tem por base a divisão do trabalho a realizar numa lista de tarefas mais pequenas, denominada de *product backlog*, de modo a que estas possam ser completadas em ciclos de tempo bem delimitados, de 30 dias ou menos, denominados de *sprints*. Cada ciclo pressupõe um ou vários objectivos ou marcos a serem atingidos, e começa com a identificação de uma lista de tarefas a serem realizadas nesse ciclo, denominada de *sprint backlog*. As tarefas a realizar são definidas e discutidas com recurso a uma reunião de planeamento (denominada de *sprint planning*) com foco no cumprimento dos objectivos definidos. Durante o *sprint* são também realizadas reuniões diárias, denominadas de *daily stand-ups*, onde cada elemento da equipa apresenta resumidamente o trabalho que realizou no dia anterior, qual o trabalho que vai realizar nesse dia, e quais os impedimentos que possam comprometer o seu progresso. A metodologia pressupõe a existência de três papéis base: *product owner*, representante da vontade do cliente e responsável por garantir que é obtido valor de negócio do trabalho realizado; a equipa de desenvolvimento, responsável por, a cada *sprint*, completar tarefas que incrementem o produto em desenvolvimento; e o *scrum master*, responsável por responder às dificuldades da equipa, resolvendo os constrangimentos que impeçam que esta atinja os objectivos definidos. Esta metodologia permite, assim, um rastreio constante da evolução do desenvolvimento, e a reorganização do trabalho planeado, de modo a dar resposta a alterações nos requisitos ou a dificuldades inesperadas [10].

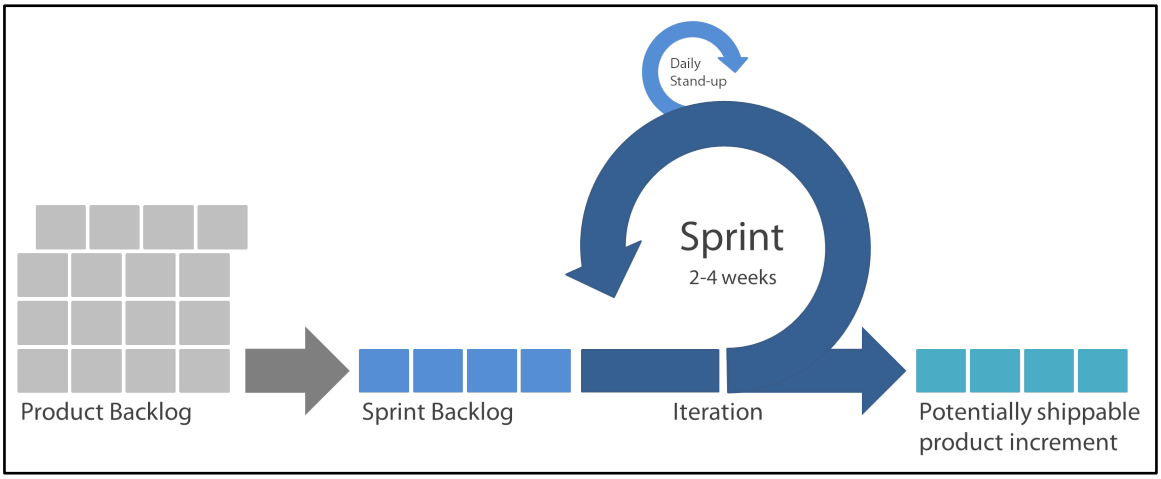

Figura 7 - Fluxo de trabalho da metodologia Scrum [11]

Neste projecto, a abordagem aplicada para o desenvolvimento não seguiu a metodologia Scrum de um modo rígido, sendo aplicado apenas o conceito de *sprints* de desenvolvimento e as reuniões associadas. Deste modo, foram considerados *sprints* de duas semanas, com a realização de uma reunião de *sprint planning* no final de cada um, e a realização de *daily stand-ups*. As reuniões seguiam a estrutura proposta pela metodologia, e nelas estavam presentes elementos dos dois parceiros do consórcio. Durante esta fase, as reuniões de consórcio, de ponto de situação do projecto, passaram a estar sincronizadas com os *sprints* de desenvolvimento, sendo realizadas a cada dois *sprints*. Nelas passou também a ser feita uma demonstração das evoluções realizadas no desenvolvimento da plataforma.

Para discussão de aspectos mais técnicos do projecto, como detalhes relativos às tecnologias ou às ferramentas a utilizar, eram realizadas reuniões técnicas com as equipas de desenvolvimento. Inicialmente, estas reuniões eram marcadas consoante a necessidade. Com o início da fase de desenvolvimento, as reuniões passaram a ser regulares, sendo realizadas a seguir às reuniões de *sprint planning*. Quando, nestas reuniões, se abordavam aspectos de fundo da plataforma, para os quais fosse necessário tomar decisões impactantes no projecto (por exemplo, alterações à arquitectura da plataforma), essa discussão e tomada de decisão

transitava para uma reunião de estado de projecto, de modo a ser discutida por todo o consórcio.

No decorrer do projecto houve a necessidade de se realizarem reuniões internas ao IPNlis, com a presença dos vários elementos do parceiro, onde se incluíam os orientadores do estágio (tanto o do IPNlis como o do Departamento de Engenharia Informática da Universidade de Coimbra). Estas reuniões eram focadas na reflexão e discussão de aspectos estruturais do projecto (por exemplo, arquitectura da plataforma) onde eram propostas decisões com impacto na sua evolução e desenvolvimento. Estas decisões eram depois alvo de discussão nas reuniões de ponto de situação do projecto. Aquando da fase de desenvolvimento, passaram também a ser realizadas semanalmente reuniões com os vários elementos da equipa de desenvolvimento do IPNlis. Nestas reuniões era dado a conhecer o estado do trabalho de cada elemento da equipa, eram discutidos aspectos técnicos do projecto e aspectos relacionados com o estágio.

Na Figura 8 é apresentado um esquema do planeamento das várias reuniões que eram realizadas aquando da fase de desenvolvimento do projecto. Nela são caracterizados dois *sprints* de desenvolvimento (*sprint* 1 e 2) e também parte da semana que os antecede e da que os sucede. A semana que os antecede possui as reuniões de preparação do *sprint* 1 (reunião interna do IPNlis, *sprint planning*, e reunião técnica) e a que sucede possui o culminar desses dois ciclos, a reunião de ponto de situação.

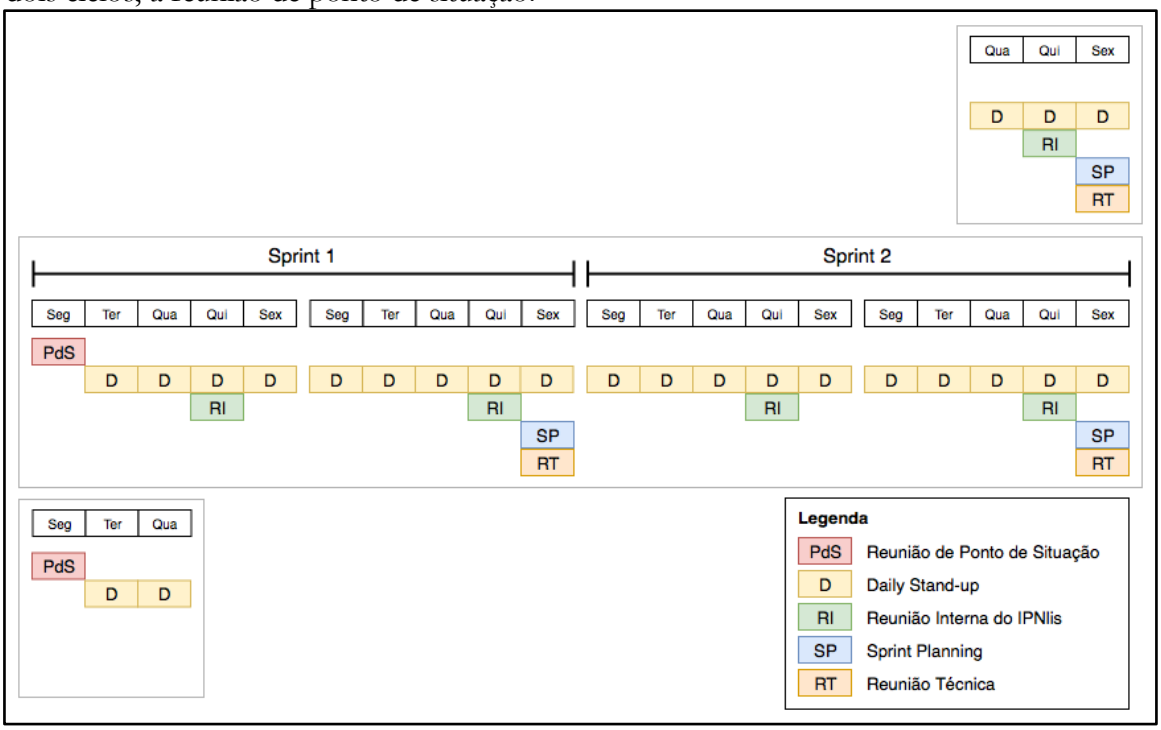

Figura 8 - Esquema do planeamento de reuniões, aquando da fase de desenvolvimento do projecto Hotelcracy Apps

Em determinadas fases do estágio, houve a necessidade de serem aplicados processos e metodologias específicos, que são apresentados em contexto ao longo deste relatório. Para a análise do estado da arte, foram especificados processos de recolha e análise de artefactos relacionados com as abordagens para a integração de serviços *cloud* (secção 4.2.1) e de artefactos relacionados com as plataformas de integração deste tipo de serviços (secção 4.3.2). Para o desenvolvimento dos componentes da plataforma em que o estágio incidiu foi aplicado um processo de gestão do ciclo de vida do código produzido (secção 7.1.1) e a técnica *Test-Driven Development* (secção 7.1.2).

3.3. Participação na gestão do projecto Hotelcracy Apps

Durante o 2.º semestre do estágio, houve a necessidade de o estagiário desempenhar uma posição mais proeminente no projecto. Esta necessidade deveu-se ao facto de a equipa do IPNlis ter sofrido uma reestruturação, com saída e entrada de elementos, passando o estagiário a ser um dos que possuía mais conhecimento e compreensão do projecto. Deste modo, para além das tarefas previamente definidas para o estágio, foram-lhe também atribuídas responsabilidades e tarefas mais próximas da gestão técnica do projecto.

Uma das principais responsabilidades do estagiário passou a ser participar no acompanhamento dos vários elementos da equipa do IPNlis, ao nível da concepção e desenvolvimento da plataforma, numa vertente mais técnica. Aquando do 2.º semestre do estágio tinha já decorrido mais de um ano de projecto, durante o qual vário conhecimento técnico tinha sido criado a partir da investigação, discussão e análise, sendo que nem todo estava vertido em documentos técnicos. De modo a garantir que esse conhecimento continuasse a ser aplicado, o estagiário passou a ser uma voz mais activa e preponderante na discussão de aspectos técnicos do projecto. Esta contribuição permitiu que a sua evolução respeitasse as decisões tomadas anteriormente, e que fossem atingidos os objectivos definidos.

Outra responsabilidade do estagiário estava relacionada com a produção dos documentos técnicos do projecto. Estes eram de grande importância, uma vez que serviam de meio de registo e de partilha do conhecimento gerado e das decisões tomadas. Para além disso, serviam de entregáveis para a entidade financiadora, como modo de comprovar e documentar o trabalho realizado. A sua produção era realizada pelos dois parceiros do consórcio, sendo realizadas várias iterações para evolução e revisão, com trocas de versões entre ambos. No 1.º semestre, o trabalho do estagiário incidiu na escrita e na revisão do conteúdo destes documentos. No 2.º semestre, este trabalho passou a envolver também a sua gestão. O estagiário ficou, assim, responsável pela orientação da evolução dos documentos técnicos, pela partilha de versões destes entre parceiros do consórcio e elementos da equipa do IPNlis, e pelo seu correcto arquivo.

Sob responsabilidade do estagiário estava também a gestão das reuniões do projecto, tanto as reuniões de estado, com todos os elementos dos parceiros, como as reuniões técnicas, reuniões de *sprint planning* e as reuniões internas. Para as reuniões de estado do projecto, as tarefas envolviam:

- **Antes da reunião:** preparação da reunião, com identificação de pontos de discussão e respectivas notas. Esta informação era introduzida numa ferramenta interna do IPNlis, dedicada à gestão de reuniões. Era também produzida uma agenda da reunião, um documento formal, com os pontos a serem discutidos (ver anexo HC_A9-Agenda_Reuniao). Esta agenda era depois partilhada com todo o consórcio. Caso fosse necessário, eram acrescentados ou retirados pontos à agenda, conforme indicação dos elementos dos parceiros do consórcio;
- **Durante a reunião:** condução da reunião, com apresentação de cada ponto a ser discutido, e tomada de notas. Esta condução e tomada de notas era realizada também com recurso à ferramenta interna de gestão de reuniões;
- **Após a reunião:** escrita da acta da reunião, com recurso às notas tomadas durante a mesma (ver anexo HC_A10-Acta_Reuniao). A acta era depois partilhada com o consórcio, que a validava e/ou indicava alterações a serem realizadas.

Para as reuniões técnicas, reuniões de *sprint planning* e reuniões internas não eram produzidos documentos formais. No entanto, eram também levantados os pontos de discussão, e tomadas notas dos vários aspectos discutidos, para consulta posterior.

Esta atribuição de responsabilidades de gestão ao estagiário teve também peso no tempo de execução do estágio, contribuindo para o seu prolongamento. Permitiu, porém, aplicar e aperfeiçoar competências de gestão de projecto num contexto real e complexo, revelando-se uma mais valia para o aumento de competências do estagiário.

Capítulo 4
Integração de serviços *cloud*

No presente capítulo são apresentadas as análises e estudos relativos à integração de serviços *cloud*, realizados no decorrer do estágio, e que se traduzem numa análise preliminar de tecnologias para integração de serviços SaaS, num estudo de abordagens teóricas de integração de serviços *cloud*, e numa análise de soluções proprietárias de integração de serviços SaaS.

Como referido anteriormente, a plataforma Hotelcracy Apps é um projecto de investigação e desenvolvimento, focado em dar resposta a um problema de integração *cloud* de serviços SaaS do sector do alojamento, cujo resultado se pretende que seja uma plataforma inovadora que permita o uso integrado destes serviços. Para responder à especificidade do problema foi necessário perceber o âmbito do mesmo, e identificar opções viáveis para a sua resolução. Para tal, foi realizada uma análise preliminar da integração de serviços SaaS do sector alojamento, a partir da qual foram tomadas decisões quanto ao rumo a seguir no desenvolvimento do projecto. Esta é apresentada na secção 4.1.

A especificidade do problema, integração de serviços *cloud*, levou à necessidade de averiguar quais as abordagens teóricas existentes que sejam direccionadas para esta temática, e cujo estudo do estado da arte é apresentado na secção 4.2.

Incluído no plano de trabalho do projecto e do estágio estava a realização de uma análise de soluções proprietárias de integração de serviços SaaS, as quais foram também identificadas como sendo uma abordagem à integração de serviços *cloud*. A análise do estado da arte deste tipo de soluções é apresentada na secção 4.3.

4.1. Análise preliminar de tecnologias para integração de serviços SaaS

Nesta secção é apresentada a análise preliminar de tecnologias levada a cabo, que teve como finalidade averiguar a aplicabilidade destas à integração de serviços SaaS. Nesta análise foi obtida inicialmente uma visão geral da utilização destes serviços nas operações do sector do alojamento, com base em exemplos de operações apresentados por *stakeholders* com experiência no sector. A partir desta visão, foi considerado o uso de uma tecnologia de gestão de processos de negócio para a gestão e execução dessas operações, e foi desenvolvida uma prova de conceito para averiguar a sua aplicabilidade no contexto do sector do alojamento. No final, foram tecidas conclusões e tomadas decisões quanto à utilização dessa tecnologia no projecto.

4.1.1. Utilização de serviços SaaS nas operações do sector do alojamento

O dia a dia de um hoteleiro é caracterizado pela execução de várias operações de gestão do hotel, tais como, por exemplo, a gestão de quartos quanto à sua disponibilidade e ao preço aplicado para a estadia, a gestão de reservas, o *check-in* e *check-out* de clientes, a gestão da limpeza dos quartos, e a gestão de recursos humanos. Estas e outras operações da rotina de um hotel podem envolver a execução de várias tarefas. Por exemplo, a gestão de reservas pode ser descrita num processo composto por:

1. Obter dados relativos às reservas registadas num determinado serviço de venda de quartos, como o Booking.com;

2. Validar as reservas, através da verificação dos cartões de crédito dos clientes associados às mesmas, com recurso a um terminal de pagamento;
3. Notificar os clientes de que a reserva está confirmada, e fornecer-lhes informações suplementares sobre a estadia.

Actualmente, existem disponíveis no mercado vários serviços SaaS que permitem a realização independente das tarefas indicadas no exemplo anterior. A obtenção de informação de reservas pode ser realizada com recurso a um serviço de gestão de canais de distribuição (serviços Channel Manager) como o Hotel-Spider (www.hotel-spider.com), responsável por recolher informação de reservas dos vários canais de distribuição (por exemplo, o Booking.com). A validação de um cartão de crédito pode ser realizada através de um terminal de pagamentos automáticos virtual, como o serviço disponibilizado pela Redunicre (www.redunicre.pt), que permite realizar pagamentos sem necessidade de um terminal físico. As tarefas das operações do hotel podem, assim, ser vistas como a realização de interacções com vários serviços SaaS.

4.1.2. Gestão de operações do sector do alojamento com recurso à gestão de processos de negócio

A análise de exemplos como o anterior, apresentados por *stakeholders* com experiência no sector hoteleiro, sugere a identificação de *workflows* ou processos de negócio, devido à sua decomposição em tarefas [12]. Considerando que a gestão de processos de negócio (BPM - *Business Process Management*) pode ser vista como a orquestração de tarefas, as quais podem consistir na realização de chamadas a serviços, e considerando que os serviços nelas invocados são serviços SaaS, assume-se que, neste caso, os processos de negócio podem traduzir-se na orquestração de chamadas a serviços SaaS. Com base nesta assumpção, ponderou-se a possibilidade de a plataforma fazer uso de tecnologias de BPM, em específico de um motor de execução de BPMN (*Business Process Model and Notation*)[1], para a realização das operações dos hotéis e orquestração das tarefas que compõem estas operações.

Para que as operações de um hotel possam ser executadas como processos de negócio, é necessário que estes contemplem tarefas de interacção humana e tarefas de interacção com serviços externos. As tarefas de interacção humana permitem que um utilizador realize acções como, por exemplo, submeter dados através de um formulário. As tarefas de interacção com serviços externos permitem estabelecer comunicações com serviços SaaS para, por exemplo, submeter ou obter dados desses serviços. Deste modo, para que seja possível a utilização de tecnologias BPM para execução das operações do sector, é necessário que estas tecnologias disponham de funcionalidades que permitam a execução de ambos os tipos de tarefas.

4.1.3. Prova de conceito da aplicação de tecnologias de BPM em operações do sector do alojamento

Para verificar a viabilidade da aplicação da gestão de processos de negócio ao contexto do sector do alojamento, foi desenvolvida uma prova de conceito com recurso a tecnologia de BPM, onde se simulou a execução de um processo de negócio que envolvia tarefas de interacção humana e de interacção com um serviço externo. A tecnologia escolhida foi a suite JBoss jBPM, sugerida pelos especialistas de TI do projecto, com base em conhecimento por estes adquirido num projecto anterior, onde foi realizado um estudo de motores de orquestração de processos. Esta é uma suite de BPM, cujo núcleo é um motor de fluxos de

[1] BPMN é uma notação para modelação gráfica dos processos de negócio de uma empresa, que facilita a compreensão destes e a sua comunicação de um modo standardizado [108].

trabalho leve e extensível, escrito em Java, que permite a execução de processos de negócio baseados na especificação BPMN 2.0 [13]. Das funcionalidades disponibilizadas pela suite, destacam-se algumas que foram consideradas serem mais valias para a sua eventual aplicação no projecto:

- Inclusão de tarefas de interacção humana nos processos de negócio, as quais podem ser associadas a um formulário [14];
- Inclusão de tarefas de execução de *scripts* nos processos de negócio, com suporte para as linguagens Java, Javascript e Mvel [15];
- Disponibilização de um servidor dedicado para execução dos projectos desenvolvidos[2] [13];
- Disponibilização de uma API REST (*Representational State Transfer*) no servidor dedicado, que permite colocar em execução os projectos, instanciar processos, gerir tarefas, entre outros [16];
- Disponibilização de um repositório de controlo de versões de código no servidor dedicado, para armazenamento e controlo de versões dos projectos [13].

O processo de negócio considerado na prova de conceito, e ilustrado na Figura 9, permite que um utilizador obtenha informação relativa a um determinado recurso alojado num serviço SaaS externo. O utilizador submete o identificador do recurso pretendido, sendo este identificador enviado num pedido realizado ao serviço SaaS, requerendo a informação relativa a esse recurso. A resposta do serviço SaaS é obtida, e a informação do recurso é disponibilizada ao utilizador.

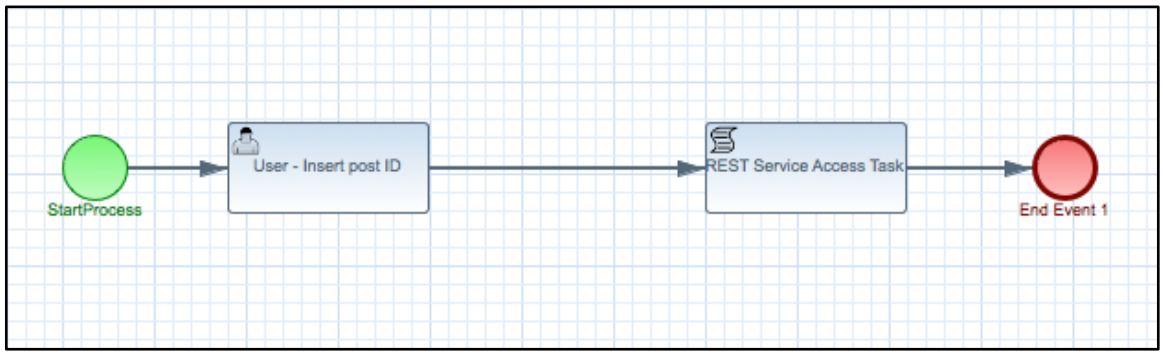

Figura 9 - Processo de negócio desenvolvido para a prova de conceito de utilização da tecnologia JBoss jBPM

O desenvolvimento da prova de conceito foi realizado com recurso ao IDE (*Integrated Development Environment*) Eclipse, e a um *plugin* disponibilizado pela suite JBoss jBPM para a criação de processos de negócio neste IDE. Para execução da prova de conceito, a suite foi colocada em execução num *container* Docker[3], com recurso à imagem jBPM Workbench Showcase (imagem Docker), e o projecto da prova de conceito armazenado num repositório Git presente na suite. A API REST disponibilizada foi utilizada para colocar o projecto em execução, instanciar o processo de negócio, aceder e completar a tarefa de interacção humana, e aceder aos dados obtidos a partir do serviço externo. O serviço utilizado como fonte de dados foi o JSONPlaceholder (jsonplaceholder.typicode.com), um serviço REST online gratuito, o qual disponibiliza uma API para acesso a dados para simulação. Para realizar os

[2] No JBoss jBPM, os processos de negócio são construídos no contexto de um projecto, podendo este último conter vários processos.

[3] Docker é uma plataforma de *containers* alternativa à virtualização tradicional, onde é feito uso do *kernel* do sistema hospedeiro, em vez de se virtualizar o sistema operativo completo. Os *containers* permitem que um *developer* empacote uma aplicação com todas as partes necessárias, tais como bibliotecas e dependências que esta necessita para funcionar, e disponibilizá-la num único artefacto [109], [110].

pedidos à API da suite jBPM e simular a interacção com o processo de negócio foi feito uso da ferramenta Postman.

4.1.4. Conclusões e decisões

A partir da prova de conceito concluiu-se que a utilização de tecnologias BPM para a realização de operações dos hoteleiros era tecnicamente possível, e que este tipo de tecnologias satisfazia as necessidades do projecto. A prova de conceito e as conclusões obtidas foram apresentadas ao consórcio, numa reunião de ponto de situação do mesmo.

Após demonstração e análise pelo consórcio, a empresa líder mostrou-se hesitante quanto ao uso de um motor de BPMN no contexto do sector, apresentando as seguintes justificações:

- Os processos de negócio do sector variam muito entre empresas/hotéis, o que inviabiliza a standardização de um conjunto específico de processos na plataforma que se pretende criar e que deve servir vários hotéis diferentes. Por outro lado, o facto de se pretender que a plataforma seja comercializada num modelo SaaS, inviabiliza também a personalização dos processos em função de cada empresa/hotel, pois este modelo de negócio exige simplificação e standardização para ser viável;
- Apesar de as operações do sector poderem ser modeladas em processos de negócio, o parceiro líder do consórcio favoreceu uma visão *ad-hoc* em que cabe ao utilizador despoletar a execução de cada tarefa nos momentos que considerar adequado;
- A empresa líder possui já conhecimentos na *framework* Ruby on Rails, os quais pretende capitalizar em vez de obter conhecimentos numa nova tecnologia.

O consórcio decidiu, assim, que a orquestração e integração dos serviços SaaS seria baseada em eventos *ad-hoc* iniciados a partir da interface com o utilizador, que despoletam chamadas às APIs dos serviços, em vez da execução de processos que definam a orquestração de chamadas a vários serviços, gerida por um motor de BPM. Com esta decisão, foi necessário obter conhecimento de abordagens mais tradicionais de integração de serviços, em específico de serviços *cloud*/SaaS. O estudo dessas abordagens é apresentado na secção seguinte, 4.2.

4.2. Estado da arte das abordagens de integração de serviços *cloud*

A integração de sistemas não é um assunto recente, tendo sido já endereçado por autores como Gregor Hohpe e Bobby Woolf no livro "Enterprise Integration Patterns", cuja primeira edição foi lançada no ano de 2003 [17]. Este apresenta padrões orientados para a integração de sistemas *on-premises*, ou seja, sistemas que estão fisicamente localizados nas instalações do seu proprietário. Do ponto de vista da integração, estes sistemas permitem um total acesso às suas características, sejam: as suas funcionalidades; os dados que possui armazenados, cujo acesso muitas vezes ocorre através de comunicações directas com a base de dados do sistema; e o controlo e gestão do próprio sistema. Para estes casos existem já ferramentas de EAI (*Enterprise Application Integration*) maduras e bem estabelecidas, que facilitam a sua integração.

Por outro lado, tem vindo a ganhar importância a integração de sistemas localizados na *cloud*, que coloca novos desafios. Estes sistemas impõem limitações que não se verificam nas variantes *on-premises*: o acesso e interacção nestes casos apenas pode ser realizado através de uma interface gráfica com o utilizador, ou com recurso a uma API; o acesso aos dados, quando permitido, é realizado através da dita API; a gestão e controlo do sistema está a cargo da entidade que o comercializa e não a cargo de quem o utiliza. Para estes sistemas existem

soluções de integração análogas às de EAI, mas que se encontram maioritariamente numa fase embrionária [18].

É com este tipo de sistemas, serviços *cloud*, que a plataforma Hotelcracy Apps vai interagir, existindo, assim, a necessidade de investigar novas abordagens para a integração, neste caso orientadas para a *cloud*.

4.2.1. Processo de pesquisa

A investigação de abordagens para a integração de serviços *cloud* teve como base a análise de artefactos relacionados com essa temática, tendo sido considerados os seguintes tipos: artigos científicos, teses de mestrado, relatórios técnicos empresariais, *white-papers* empresariais, *posts* de *blogs*, apresentações de conferências e *webinars*. Parte dos artefactos considerados foram indicados pelos orientadores do estágio e por outros *stakeholders* do projecto. Os restantes artefactos foram resultado de uma pesquisa realizada nas plataformas Google (www.google.com), Google Scholar (scholar.google.com), b-on (www.b-on.pt), e ResearchGate (www.researchgate.net). Numa primeira iteração da pesquisa, realizada na semana de 13 de Fevereiro de 2017, foram utilizados os termos "cloud integration" e "SaaS services integration". Como os primeiros resultados não abordavam aspectos relacionados com padrões de design e arquitectura, considerados importantes na construção de uma plataforma, foi realizada uma segunda iteração da pesquisa, na semana de 13 de Março de 2017, combinando as expressões iniciais com os termos "patterns" e "architecture". Numa terceira iteração, realizada na semana de 27 de Março de 2017, a pesquisa foi restringida a artefactos publicados entre o ano de 2010 e o ano de 2017, uma vez que os resultados anteriores a este período não eram relevantes, por não abordarem aspectos técnicos da integração de serviços.

4.2.2. Resultados da pesquisa

Do processo de pesquisa realizado resultaram artefactos que endereçam vários aspectos da integração de serviços *cloud*: a crescente necessidade de integração de serviços SaaS ([19]); a necessidade de as empresas adoptarem uma estratégia para implementarem a integração dos seus próprios serviços ([20]) e como desenvolver essa estratégia ([21]); técnicas ([22]) e *frameworks* para implementar a integração de serviços, abordando não apenas aspectos técnicos, mas também outras problemáticas associadas aos serviços SaaS, como segurança, privacidade, facturação, gestão, qualidade do serviço (*Quality of Service* – QoS) e Acordo de Nível de Serviço (*Service Level Agreement* – SLA) ([23]–[26]); o papel de soluções proprietárias na resolução da problemática da integração ([18], [27]–[29]) e a sua utilização em casos reais ([30]); e a comparação de possíveis soluções para a integração de serviços ([29], [31]).

Para facilitar a consulta, na Tabela 3 encontram-se listados estes vários aspectos da integração, e as respectivas referências bibliográficas.

Aspectos da integração de serviços *cloud*	Referências
A crescente necessidade de integração de serviços SaaS	[19]
A necessidade de adopção de uma estratégia para implementação da integração de serviços	[20]

Aspectos da integração de serviços *cloud*	Referências
Desenvolvimento de uma estratégia para implementação da integração de serviços	[21]
Técnicas e *frameworks* para implementação da integração de serviços	[22]–[26]
O papel de soluções proprietárias de integração na resolução da problemática da integração	[18], [27]–[29]
A utilização de soluções proprietárias de integração em casos reais	[30]
Comparação de soluções para a integração de serviços	[29], [31]

Tabela 3 - Aspectos da integração de serviços cloud, e respectivas referências

Para o presente estudo, consideraram-se os artefactos que abordavam aspectos técnicos da integração de serviços, em específico os que se relacionavam com o acesso às funcionalidades e aos recursos presentes nos serviços. Esses artefactos apresentavam *frameworks*, técnicas e arquitecturas para a integração de serviços *cloud*, nas quais são propostos conceitos como pontos de colaboração entre processos de negócio dos serviços ou conectores que abstraem o acesso aos serviços, e a utilização de soluções proprietárias de integração. Estas abordagens são apresentadas nas sub-secções seguintes.

Considera-se pertinente, para efeitos de completude, enunciar de forma breve os artefactos relacionados com aspectos técnicos da integração de serviços que não foram considerados para o presente estudo. Esta exclusão deve-se ao facto de estes artefactos não abordarem ou não apresentarem detalhes quanto ao acesso a funcionalidades ou recursos dos serviços *cloud*. W. Sun et al. [23] apresentam uma *framework* que possibilita a integração de serviços SaaS através da sua customização, onde estes são caracterizados com recurso a uma linguagem de descrição, e a partir da qual podem ser combinados para gerar um novo serviço. Não são, no entanto, apresentados detalhes de como esta combinação é realizada. G. Breiter et al. ([32]) apresentam uma *framework* cujo foco incide no controlo e gestão de uma rede de serviços integrados numa *cloud* híbrida – composta por serviços *on-* e *off-premises* [33] – sem, no entanto, disponibilizarem detalhes de como é implementada a integração entre serviços SaaS. F. Liu et al. ([34]) apresentam uma solução de "proxy-based firewall/NAT traversal" [34, p. 402] para a integração de serviços SaaS, mas que endereça a integração apenas ao nível da rede. S. Palanimalai et al. ([35]) sugerem uma arquitectura de integração híbrida, não disponibilizando, no entanto, detalhes relativos a esta. B. Wang et al. ([36]) propõem que os serviços SaaS podem ser integrados através da aplicação de técnicas de SOA (*Service Oriented Architecture*), publicando os serviços num *Service Registry*, a partir do qual é obtida informação com a finalidade de os combinar e criar novos serviços de *software*. No entanto, os autores não apresentam detalhes quanto à informação que deve estar presente no *Service Registry* e ao modo como a combinação de serviços deve ser realizada.

De seguida apresentam-se as abordagens que mais se enquadram no contexto do projecto e do estágio, e que endereçam a integração de serviços SaaS do ponto de vista do acesso às funcionalidades e aos recursos que estes disponibilizam.

4.2.2.1. *Framework* baseada em pontos de colaboração dos processos de negócio

Q. Li et al. [24] apresentam uma *framework* baseada em SOA, que aborda a integração de serviços ao nível dos seus processos de negócio, através da aplicação do conceito de pontos de colaboração. Este conceito identifica pontos onde é possível a colaboração entre processos de diferentes ambientes computacionais, como exemplificado na Figura 10 pelos círculos numerados: (1) colaboração entre a tarefa "Receive order" do ambiente "Supplier" e a tarefa "Purchase product" do "Trading company"; e (2) colaboração entre a tarefa "Receive order" do "Trading company" e a tarefa "Order product" do "Customer". O ponto de colaboração inclui operações lógicas complexas, como o acesso aos ambientes computacionais, ou a manipulação de dados.

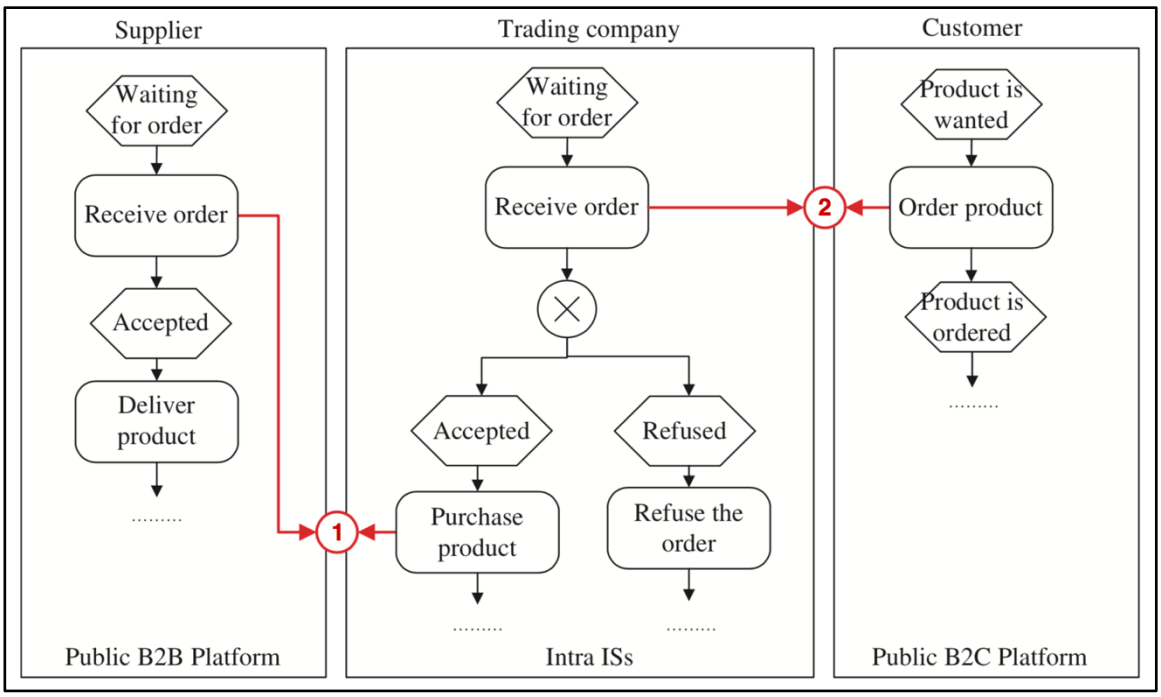

Figura 10 - Exemplo de pontos de colaboração entre processos de negócio (por Q. Li et al. [24])

Como representação da implementação do ponto de colaboração, a *framework* introduz também o conceito de agente de colaboração, o qual é utilizado pelos sistemas de informação *on-premises* para comunicarem com serviços *cloud* externos. Este agente possui lógica para a conexão, intercomunicação e interoperação entre sistemas *on-premises* e serviços *cloud* e pode conter também lógica de negócio aplicacional que permita a integração de diferentes serviços *cloud*.

Uma extensão desta *framework* é também proposta pelos autores para a integração de múltiplas *clouds* públicas [25], permitindo a implementação de um serviço *cloud* integrado, com recurso ao mesmo conceito de pontos de colaboração entre processos de negócio dos serviços públicos. Inclui componentes dedicados à gestão de relações entre o integrador e os fornecedores dos serviços *cloud* a três níveis: acesso aos serviços; gestão de contractos e licenças entre o integrador e os fornecedores; e troca de informação de negócio dos serviços, de modo a endereçar os objectivos de negócio dos fornecedores desses serviços.

Como indicado na secção 4.1, a plataforma Hotelcracy Apps não se irá basear numa utilização estruturada de processos do sector do alojamento. Deste modo, a abordagem aqui identificada não pode ser aplicada à plataforma.

4.2.2.2. Plataforma de integração de serviços *cloud* ao nível dos recursos

E. Pinho et al. [26] propõem a arquitectura de uma plataforma extensível para a integração de serviços *cloud* ao nível dos recursos, baseada na *Service Delivery Cloud Platform* (SDCP) [37], e desenhada para satisfazer requisitos e padrões de utilização de aplicações web. Esta funciona como camada de abstracção dos recursos *cloud*, e permite controlar o acesso a estes por parte das aplicações cliente.

A plataforma é composta por dois componentes principais: o *Cloud Controller*, entidade *middleware* que abstrai as especificidades do acesso aos recursos alojados em diferentes fornecedores de serviços *cloud*; e o *SDCP Client Runtime*, módulo JavaScript que executa no *browser* do cliente, e que interage com o *Cloud Controller* para acesso aos recursos *cloud*. Estes componentes são visíveis no diagrama do conceito SDCP, na Figura 11.

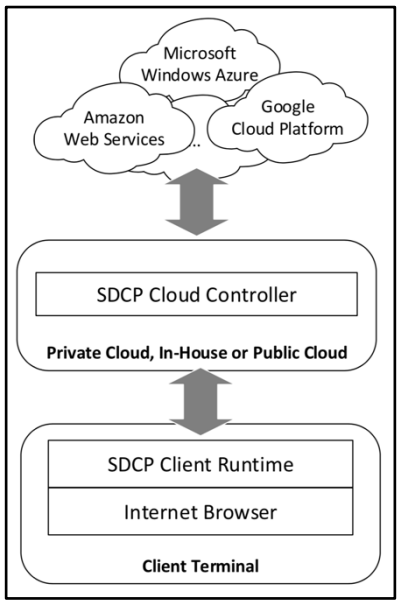

Figura 11 - Diagrama do conceito SDCP proposto (por E. Pinho et al. [22])

O *Cloud Controller* é responsável por agregar credenciais de utilizador, tratar de procedimentos de autenticação com os fornecedores *cloud*, e implementar o controlo de acesso aos recursos. Possui ainda funcionalidades que permitem a criação e uso de recursos *cloud* por clientes, em conjunto com a aplicação de políticas de controlo de acesso a esses recursos. Este componente especifica e implementa a abstracção do acesso aos recursos através da aplicação do seu próprio modelo de dados, atribuindo a cada recurso um identificador único. O *Cloud Controller* pode, a partir deste identificador, traduzir essa identidade num recurso concreto alojado num fornecedor *cloud* específico.

O componente *Cloud Controller* aplicado nesta plataforma possui características muito semelhantes às do padrão de design *Adapter* [38], uma vez que abstrai diferentes serviços *cloud* de modo a que uma terceira entidade aceda às funcionalidades e recursos destes seguindo um único protocolo.

Uma vez que o projecto Hotelcracy Apps pretende integrar vários serviços SaaS sob uma mesma plataforma, os quais disponibilizam APIs dispares entre si, esta abordagem (ou uma variação dela) foi considerada interessante e promissora para aplicação.

4.2.2.3. Integração de *software* e dados com recurso a *Master Tables*

W. Kim et al. [22] abordam a problemática da integração através da utilização do conceito de *Master Table*, onde informação relativa a utilizadores, *software* e sessões é armazenada, permitindo que os dados sejam processados e armazenados como um sistema integrado.

A *Master Table* proposta é constituída por uma tabela mestre principal e várias tabelas com informação pertencente aos utilizadores. A primeira contém atributos de informação básica para o *software* em uso pelos utilizadores. As segundas possuem um conjunto diferente de atributos, também eles para o *software* em uso, mas que podem ser modificados pelo administrador de sistema de modo a satisfazer os requisitos do *software*. Os dados são armazenados separadamente nas sub-tabelas correspondentes, e podem ser criados ou modificados em função das necessidades do utilizador e da utilização do *software* pretendido.

Um sistema de gestão de *software* integrado gere os serviços de *software* fornecidos, e concede aos utilizadores o acesso a esses serviços com base na informação armazenada na *Master Table*. Quando o utilizador termina a sua utilização do *software* fornecido, toda a informação gerada é guardada na *Master Table*.

Uma vez que não são disponibilizados detalhes acerca do modo de funcionamento interno do sistema de gestão referido, esta abordagem não foi considerada para aplicação no projecto.

4.2.2.4. Utilização de uma plataforma de integração iPaaS – *Integration Platform-as-a-Service*

O termo *Integration Platform-as-a-Service* (iPaaS) foi introduzido pela Gartner em 2011 [39], e representa uma suite de serviços *cloud* que permite o desenvolvimento, execução e gestão de fluxos de integração de serviços. Estes fluxos permitem combinar e conectar processos, serviços, aplicações e dados, localizados tanto em ambientes *on-premises* como em ambientes *cloud*, pertencentes a uma única organização, ou distribuídos por múltiplas organizações [40]. Sendo uma solução *as-a-Service*, estas plataformas de integração normalmente são adquiridas através de um plano de subscrição, que inclui o acesso a plataformas e/ou infraestruturas para execução dos fluxos de integração gerados, retirando aos clientes encargos e preocupações com a manutenção e gestão de *hardware* ou *middleware*. Em alguns casos, as plataformas de integração podem também disponibilizar ferramentas e ambientes orientados para o desenvolvimento dos fluxos, com níveis de customização a diferirem entre si.

Estas plataformas visam facilitar a integração de vários serviços e sistemas, resultando numa solução holística que permite agilizar, melhorar e automatizar processos empresariais, aceder a informação integrada, entre outros. Um dos aspectos que permite facilitar a integração de aplicações e serviços é a disponibilização de conectores pré-construídos. Normalmente, um conector é desenvolvido especificamente para um serviço *cloud* específico, e permite a comunicação e o acesso às funcionalidades desse serviço, necessitando o utilizador apenas de realizar configurações básicas no conector, tais como a introdução de credenciais de acesso.

Um termo semelhante, SaaS *Integration Platform* (SIP), tinha sido já apresentado pela Saugatuck Technology Inc. em 2006 [41], definindo uma solução centralizada que permite a partilha, distribuição e gestão de aplicações, considerando que este tipo de soluções de integração se tornaria crítico para que existisse uma ampla adopção de serviços SaaS pelo mercado. No relatório *Magic Quadrant for Enterprise Platform as a Service* para o ano de 2016, a Gartner considerou que este tipo de soluções serão a plataforma de eleição para novos projectos de integração até ao ano de 2019 [18], cimentando a importância destas soluções.

A adopção de uma plataforma de integração iPaaS foi avaliada por D. Merkel [31], comparando-a com a adopção de outros tipos de soluções num cenário de integração de serviços SaaS e de serviços localizados numa *cloud* privada. Neste estudo, a adopção de uma iPaaS foi comparada com: implementação de integração ponto-a-ponto, utilização de um *Enterprise Service Bus* (ESB) alojado numa *cloud* pública, e utilização de um ESB alojado numa *cloud* privada. A avaliação focou-se em aspectos de modificabilidade, extensibilidade, disponibilidade, complexidade, latência, nível de controlo interno, gestão centralizada, e escalabilidade das soluções em análise, tendo sido atribuída uma cotação a cada um destes aspectos para cada tipo de solução de integração.

Como resultado da avaliação realizada, concluiu-se que a adopção de uma plataforma iPaaS está a um nível semelhante ao da adopção de um ESB alojado numa *cloud* pública. Estas duas soluções possuem uma escalabilidade e modificabilidade superiores quando comparadas com as outras soluções de integração. No entanto, a necessidade de existir uma entidade intermediária que permite a interacção entre todos os serviços introduz um risco quanto à disponibilidade do sistema como um todo: ao falhar essa entidade intermediária, a interacção entre serviços fica indisponível. Ainda assim, estes tipos de soluções reduzem a complexidade de implementação e de modificação associadas a uma integração ponto-a-ponto, e permitem uma escalabilidade que pode não se verificar numa integração baseada num ESB alojado numa *cloud* privada, caso a infraestrutura desta última não tenha sido desenhada tendo em vista essa característica.

Apesar da escolha de qual solução aplicar depender de caso para caso, a avaliação realizada revela que a adopção de uma plataforma de integração iPaaS não deve ser descartada à partida, e que este tipo de soluções possui características que o podem tornar viável para aplicação num cenário de integração de serviços *cloud*.

4.2.3. Conclusões

Da análise apresentada neste capítulo destacam-se duas abordagens como promissoras para aplicação na plataforma Hotelcracy Apps. Em específico, o conceito *Cloud Controller* e as plataformas de integração iPaaS.

Uma variante do conceito *Cloud Controller*, apresentado por E. Pinho et al. [26], foi aplicada ao projecto para abstracção dos vários serviços SaaS a serem integrados. Esta variante será apresentada na arquitectura proposta para a plataforma Hotelcracy Apps.

Quanto às plataformas iPaaS, foi realizada uma análise das soluções deste tipo existentes no mercado e da sua aplicabilidade ao projecto. A informação recolhida relativa a estas plataformas de integração e a análise levada a cabo são apresentadas na secção seguinte.

4.3. Análise de plataformas de integração de serviços *cloud*

Como referido na secção anterior, as plataformas de integração iPaaS podem ser uma opção viável para cenários de integração de serviços *cloud* e para aplicação no projecto Hotelcracy Apps. Houve, assim, a necessidade de analisar estas plataformas e a sua possível aplicação no projecto. A informação recolhida e produzida neste estudo foi compilada num documento técnico, incluído neste relatório como anexo HC_A1-Analise_Solucoes_Integracao. Nesta secção é apresentado um resumo da análise realizada a este tipo de soluções de integração, e dos resultados obtidos.

4.3.1. Identificação de plataformas de integração

A identificação de plataformas de integração existentes no mercado foi um processo incremental, uma vez que, à medida que a análise do estado da arte era realizada, novas eram identificadas através de referências nos artefactos analisados. Das plataformas consideradas, as primeiras 12 foram identificadas a partir do relatório *Magic Quadrant for Enterprise Platform as a Service* produzido pela Gartner para o ano de 2016. As restantes foram identificadas através da análise de artefactos relacionados com abordagens para a integração de serviços *cloud*, nomeadamente *white papers*, *webinars* e *posts* de *blogs*, e através de pesquisas realizadas entre o dia 1 de Março e 30 de Abril de 2017 no motor de busca Google (www.google.pt), com os termos "ipaas", "integration platform" e "saas integration". Soluções identificadas por *stakeholders* do projecto foram também consideradas para a análise.

Um total de 25 plataformas de integração foram consideradas e sujeitas a um processo estruturado de análise. Estas encontram-se listadas abaixo:

- **Adaptris** Cirrus - www.adaptris.com/cirrus.php
- **Celigo integrator**.io - www.celigo.com/ipaas-integration-platform
- **Cloud Elements** - cloud-elements.com
- **Dell Boomi Integration Cloud** - boomi.com
- **Elastic.io** - www.elastic.io
- **Flow Integration** - www.flowsoftware.com/product/flow-integration-software
- **IBM App Connect** - appconnect.ibmcloud.com/personal
- **IBM App Connect Professional** - appconnect.ibmcloud.com/professional/
- **IBM Integration Bus on Cloud** - www-03.ibm.com/software/products/pt/ibm-integration-bus-on-cloud
- **Informatica Cloud** - www.informatica.com/products/cloud-integration.html
- **JitterBit Harmony Platform** - www.jitterbit.com
- **Magic xpi Integration Platform** - www.magicsoftware.com/magic-xpi-integration-platform
- **Microsoft Azure Logic Apps** - azure.microsoft.com/en-us/services/logic-apps/
- **MuleSoft Anypoint Platform** - www.mulesoft.com/platform/enterprise-integration
- **OneSaas** - www.onesaas.com
- **Oracle Integration Cloud Service** - cloud.oracle.com/integration
- **RunMyProcess** - www.runmyprocess.com
- **Scribe Online** - www.scribesoft.com/products/scribe-online
- **SnapLogic Cloud App Integration** - www.snaplogic.com/solutions/app-integration
- **Software AG webMethods Integration Cloud** - www.webmethodscloud.com/integration/index.html
- **Talend Application Integration** - www.talend.com/products/application-integration
- **Talend Cloud Integration** - www.talend.com/products/integration-cloud
- **TIBCO Cloud Integration** - cloud.tibco.com
- **WSO2 Integration Cloud** - wso2.com/integration
- **Zapier** - zapier.com

Algumas das plataformas de integração identificadas foram excluídas da análise. Essas estão listadas abaixo, juntamente com a justificação da sua exclusão:

- **Actian DataCloud** – é uma plataforma focada em *big data* [18];
- **Attunity CloudBeam** – é uma plataforma focada em integração de dados e na gestão de *big data* [18];
- **CloudWork** – é uma plataforma de descoberta de integrações de serviços SaaS, não disponibiliza funcionalidades para uma verdadeira implementação de integração [42];
- **DBSync** – é uma plataforma focada apenas na integração de dados [18];
- **Fujitsu Cloud Services Management** – a plataforma de integração não estava disponível como um serviço autónomo [18];
- **Innotas Integration Platform** – a integração de serviços é realizada pela própria empresa, sob a forma de prestação de serviço [43];
- **ItDuzzit** – passou a estar indisponível após ter sido adquirida pela Intuit [44];
- **Pipemonk** – passou a estar indisponível após ter sido adquirida pela Freshdesk [45];
- **SAP Cloud Platform Integration** – é uma plataforma focada na integração de aplicações SAP [18];
- **TerraSky SkyOnDemand** – é uma plataforma focada na integração de aplicações Salesforce [18];
- **Youredi** – possui uma estratégia vertical, focada no sector de mercado para o qual pretende endereçar a integração, tal como logística, cadeia de fornecimento e transacções financeiras [18].

4.3.2. Processo de análise e avaliação das plataformas de integração

A análise realizada focou-se na documentação e outros artefactos (*white-papers*, apresentações de ferramentas, entre outros) disponibilizados pelos fornecedores das plataformas de integração, e iniciou-se com uma análise preliminar de algumas das soluções consideradas, de modo a obter-se uma visão geral das características que estas possuíam. A partir desta análise preliminar foi identificado um conjunto de tópicos chave, sobre os quais incidiu uma análise aprofundada de cada plataforma de integração, permitindo que esta última fosse orientada e normalizada. A informação obtida a partir desta análise aprofundada foi compilada num conjunto de tabelas num documento técnico do projecto (o anexo HC_A1-Analise_Solucoes_Integracao), uma por cada solução de integração, com cada linha das tabelas a reflectir um tópico chave da análise.

Após a análise aprofundada, foi realizada uma avaliação da viabilidade da aplicação de cada solução de integração ao projecto. Para esta avaliação foi também definido um conjunto de tópicos chave, com base na análise conduzida previamente, de modo a normalizá-la e orientá-la, facilitando a decisão de aplicação ou não de uma solução ao projecto. A avaliação de cada solução de integração foi também inserida no documento técnico referido, com a adição de informação relativa a cada tópico chave.

A análise realizada para algumas das plataformas de integração não resultou em informação suficiente que permitisse realizar a sua avaliação, uma vez que os artefactos não abordavam todos os tópicos chave. Para esses casos, foram realizados contactos directos com representantes dos fornecedores das plataformas, por email e telefone, nos quais foi colocado um conjunto de perguntas com a finalidade de obter a informação necessária à realização da avaliação. Alguns dos contactos realizados não obtiveram resposta por parte dos fornecedores das plataformas de integração, tendo sido incluída nas respectivas avaliações uma indicação de tal facto.

De seguida apresentam-se os tópicos chave definidos para a análise aprofundada e para a avaliação, e também um exemplo de análise levada a cabo para uma das plataformas de

integração, com apresentação da tabela e da avaliação produzidas. A análise completa das 25 plataformas, bem como a avaliação de cada uma, encontra-se no anexo HC_A1-Analise_Solucoes_Integracao.

4.3.2.1. Definição dos tópicos chave da análise aprofundada

Os tópicos chave definidos para a análise aprofundada das plataformas de integração são listados abaixo, com uma descrição para cada um:

- **Nome:** nome da solução de integração;
- ***Uniform Resource Locator* (URL):** conjunto de endereços *web* relativos à plataforma de integração. Contém informação relevante para desenvolvimento, tal como documentação, e repositórios do código da plataforma;
- **Descrição:** breve descrição dos seguintes aspectos, quando disponível:
  - Tipo de plataforma disponibilizada (ex: iPaaS);
  - Locais de execução da plataforma (ex: numa *cloud*, numa infraestrutura *on-premises*);
  - Tipos de serviços que permite integrar (ex: serviços localizados na *cloud* ou aplicações *on-premises*);
  - Modelos de subscrição disponíveis (ex: mensal, anual);
  - Modos de implementação da integração entre serviços (ex: construção de fluxos, mapeamento de dados, transformação de dados);
  - Outras funcionalidades disponíveis (ex: implementação com recurso a ferramentas *in-browser* ou IDEs; construção e publicação de APIs; painéis de administração, gestão, e/ou monitorização).
- **Preços/Planos de subscrição:** breve descrição dos preços/planos de subscrição da plataforma de integração. Indica também se esta disponibiliza um período ou versão de avaliação;
- **Limitações de recursos:** breve descrição dos limites impostos pela plataforma de integração, os quais podem ser relativos a: recursos de infraestrutura, como a capacidade de processamento ou a quantidade de memória RAM disponibilizada; recursos de comunicação e execução, como a cardinalidade de conexões a serviços externos permitidas ou a cardinalidade de fluxos/instâncias permitidas em execução;
- ***Open source*:** indica se o código fonte da plataforma de integração está disponível. O tipo de licença não é tido em conta;
- **Funcionamento com o exterior:** breve descrição do funcionamento da plataforma de integração com os serviços SaaS externos;
- **Funcionamento interno:** breve descrição do funcionamento interno da plataforma de integração;
- **Modelo de dados:** informação relativa ao modelo de dados utilizado pela plataforma de integração;
- **Conectores para SaaS alvo da plataforma Hotelcracy Apps:** identifica os serviços SaaS alvo da Hotelcracy Apps, para os quais a plataforma de integração disponibiliza um conector pronto a utilizar. Na presente versão do documento, foram apenas considerados os serviços das categorias de *Channel Manager*, Facturação e TPA Virtual, por serem os serviços SaaS considerados prioritários para o projecto;
- **Comunicações personalizadas:** identifica se a plataforma de integração permite a comunicação com serviços SaaS para os quais não possui já um conector específico. Esta pode ser realizada através de conectores que possibilitem a comunicação através de protocolos populares, como o HTTP, ou através do desenvolvimento de conectores personalizados para os serviços pretendidos;

- **Documentação:** breve descrição do conteúdo da documentação disponibilizada pela plataforma de integração;
- **Repositório(s) de código da organização:** breve descrição do conteúdo do(s) repositório(s) de código da organização.

As tabelas compiladas com informação obtida na análise realizada possuem ainda os seguintes campos auxiliares:

- **Versão da tabela:** identifica a versão actual da informação contida na tabela que descreve a plataforma de integração. Sempre que haja alterações nessa informação, deverá ser incrementada a versão. As alterações realizadas, bem como o seu motivo, devem ser registadas no campo "**Controlo de versões**" da tabela respectiva;
- **ID:** identificador atribuído à plataforma de integração;
- **Controlo de versões:** registo das modificações efectuadas à informação presente na tabela.

4.3.2.2. Definição dos tópicos chave da avaliação

Os tópicos chave definidos para a avaliação da viabilidade da aplicação de cada solução de integração ao projecto são apresentados abaixo, com uma descrição para cada um:

- **Cardinalidade de conexões permitidas:** uma vez que o projecto pretende a integração de vários serviços, a solução de integração deve permitir instanciar uma quantidade arbitrária de conexões a esses serviços;
- **Cardinalidade de fluxos/instâncias em execução:** uma vez que os serviços SaaS que se pretende integrar possuem um leque alargado de operações para cada serviço, e que a integração entre eles permite uma elevada quantidade de combinações de operações, a solução de integração não deve possuir uma restrição quanto às execuções activas;
- **Conexões a serviços alvo:** a solução de integração deve permitir estabelecer conexões aos serviços alvo, quer através de conectores específicos para esses serviços, quer através de conectores que implementem o protocolo de comunicação de acesso ao serviço;
- **Operações de integração permitidas:** a solução de integração não deve restringir as operações de integração entre serviços;
- **Preço:** a solução de integração não deve acarretar um custo de utilização.

4.3.2.3. Exemplo de análise de uma plataforma de integração

Na Tabela 4 é apresentado um exemplo de uma das tabelas existentes no anexo HC_A1-Analise_Solucoes_Integracao, e que corresponde à análise realizada a uma das 25 plataformas de integração.

VERSÃO DA TABELA	1.2
ID	SI_01
NOME	MuleSoft Anypoint Platform
URL(S)	**URLs da plataforma:** https://www.mulesoft.com/platform/enterprise-integration **URLs de desenvolvimento:**

	https://developer.mulesoft.com/ https://docs.mulesoft.com/runtime-manager/cloudhub https://developer.mulesoft.com/download-mule-esb-runtime **Repositório da solução:** Motor Mule Runtime: https://github.com/mulesoft/mule
DESCRIÇÃO	Esta solução é uma plataforma que permite criar, integrar, modelar e colocar em execução serviços, APIs, e aplicações Mule. A integração de serviços é realizada com base em fluxos de mensagens, construídos com recurso ao Anypoint Studio, um IDE da MuleSoft baseado em Eclipse. Estes fluxos são construídos com recurso a elementos, os quais permitem receber, processar e encaminhar mensagens. Desses elementos, destacam-se os conectores (Anypoint Connectors), que permitem comunicações com serviços e recursos de terceiros. A plataforma apoia-se no Mule Runtime, um ESB (Enterprise Service Bus), que funciona como o motor de integração da MuleSoft. As integrações construídas podem ser colocadas em execução no MuleSoft Cloud Hub, a solução PaaS da MuleSoft, ou executadas *on-premises*, através do Mule Runtime [46], [47].
PREÇOS/PLANOS DE SUBSCRIÇÃO	**Planos para avaliação/teste:** Dispõe de uma versão de avaliação, com recursos reduzidos. **Planos para utilização:** Plano Base: preço sob consulta. Plano Platinum: preço sob consulta. A subscrição de planos permite, entre outros, o acesso a conectores *premium*, e a suporte pelo parceiro responsável pelo conector. **Outras informações:** Mais informações sobre preços/planos da solução podem ser consultadas em: https://www.mulesoft.com/anypoint-pricing
LIMITAÇÕES DE RECURSOS	**Planos para avaliação/teste:** Versão de avaliação possui recursos reduzidos. **Planos para utilização:** Limitações ao nível da capacidade de processamento.
OPEN SOURCE	A MuleSoft disponibiliza uma versão *community* do motor Mule Runtime. São também disponibilizados vários conectores, desenvolvidos e mantidos pela MuleSoft e pela MuleSoft Dev Community.
FUNCIONAMENTO COM O EXTERIOR	A plataforma faz uso de conectores para comunicações com serviços e recursos de terceiros, através dos protocolos de comunicação mais populares.
FUNCIONAMENTO INTERNO	A plataforma tem por base uma arquitectura orientada a eventos, com recurso a fluxos de mensagens para a recepção, processamento e encaminhamento das mesmas, suportada por um ESB, o Mule Runtime [48], [49].
MODELO DE DADOS	A plataforma faz uso de mensagens nos fluxos. Estas mensagens dividem-se em *header* (metadados) e *payload* (dados de negócio) [50].
CONECTORES PARA SERVIÇOS ALVO	Não possui conectores para os serviços de Channel Manager, Facturação e TPA Virtual alvos [51].
COMUNICAÇÕES PERSONALIZADAS	Dispõe de conectores para comunicações através dos protocolos de comunicação mais populares [52], [53]. Permite a implementação de conectores customizados [54].
DOCUMENTAÇÃO	Possui documentação relativa a: ferramentas presentes na plataforma, com informação relativa à instalação, configuração e utilização dessas ferramentas; implementação de fluxos, com exemplos práticos; conectores disponíveis na plataforma para implementação dos fluxos, e à implementação de conectores personalizados.

REPOSITÓRIO(S) DA ORGANIZAÇÃO	Possuem documentação da plataforma, conectores para serviços externos, *templates* para implementação. Os repositórios da organização podem ser consultados em: https://github.com/mulesoft
CONTROLO DE VERSÕES	**v1.0** (03/04/2017) - Primeira descrição da solução de integração. **v1.1** (06/07/2017) - Alteração do esquema da tabela. **v1.2** (18/07/2017) - Integração da informação de período/versão de avaliação na informação de preços/planos de subscrição.

Tabela 4 - Exemplo de tabela da análise de uma plataforma de integração

Na Tabela 5 é apresentado um exemplo de uma avaliação realizada a uma das plataformas de integração, presente no anexo HC_A1-Analise_Solucoes_Integracao.

MULESOFT ANYPOINT PLATFORM - AVALIAÇÃO DA APLICABILIDADE DA PLATAFORMA
A plataforma de integração MuleSoft Anypoint Platform é candidata a ser utilizada no projecto porque não possui restrições quanto às conexões a serviços externos, fluxos/instâncias de execução, e operações de integração permitidas, e permite comunicações com os vários serviços alvo. As limitações impostas pelos planos são referentes à capacidade de processamento disponibilizada. O custo de subscrição desta solução de integração só está acessível através de consulta. No dia 10/Maio/2017 foi enviado um pedido de esclarecimentos para o representante da plataforma, onde se questionava quais os recursos e limitações associados a cada plano de subscrição, e qual o seu custo. Até à data de actualização do documento, não foi obtida resposta às questões colocadas. Uma vez que a plataforma de integração possui um custo associado à sua subscrição, não é possível aplica-la ao projecto. A plataforma é baseada em *software* gratuito e *open-source*, a versão *community* do motor Mule Runtime, podendo ser explorada a utilização desse *software*.

Tabela 5 - Exemplo de avaliação de uma plataforma de integração

4.3.3. Conclusões

A análise de plataformas de integração permitiu a recolha de informação relativa às opções disponíveis no mercado, e, na maioria dos casos, permitiu a avaliação da sua aplicabilidade ao projecto. Algumas das plataformas revelaram-se mais restritivas do que outras, principalmente em relação à cardinalidade de conexões permitidas a serviços externos e à cardinalidade de fluxos de integração permitidos.

Um dos aspectos promissores destas plataformas de integração era o uso de conectores prontos a serem utilizados para comunicação com serviços externos. No entanto, na prática, estas dispunham de poucos conectores para os serviços alvo do projecto (serviços SaaS utilizados no sector do alojamento).

Todas as plataformas consideradas possuíam um custo associado, o qual poderia chegar à ordem dos milhares de Euros por mês, o que inviabiliza a sua aplicação no projecto. No entanto, algumas delas são baseadas em *software* gratuito e *open-source*, o que pode permitir a utilização de uma versão da plataforma de integração sem necessidade de a subscrever. Contudo, a utilização de *software open-source* não inclui o suporte fornecido pela organização, nem as capacidades de escalabilidade de uma solução subscrita, uma vez que o *software* teria de ser colocado em execução numa infraestrutura gerida e mantida pelo consórcio.

Da análise das 25 plataformas de integração consideradas:

- Oito plataformas foram consideradas tecnicamente candidatas a serem aplicadas no projecto, no entanto o custo que possuem associado inviabiliza essa aplicação;
- Treze plataformas não possuem características técnicas que as tornassem candidatas à aplicação no projecto;
- Quatro plataformas não disponibilizavam informação suficiente para serem avaliadas. Para estas plataformas foram realizados contactos junto das organizações por elas responsáveis, não tendo sido obtida resposta às questões colocadas.

O consórcio decidiu, assim, a não utilização de uma plataforma de integração, optando pelo desenvolvimento completo de uma solução à medida.

Capítulo 5
Arquitectura da plataforma Hotelcracy Apps

A arquitectura da plataforma Hotelcracy Apps resulta da análise e da discussão levadas a cabo durante as duas primeiras actividades do projecto. Na Actividade 1 foi definida uma arquitectura base, onde se identificaram os grandes grupos funcionais da plataforma, também denominados de macro-componentes. Com essa base em mente, no contexto deste estágio foi realizado um estudo relativo à migração de serviços *cloud*, onde se analisaram os desafios que esta migração impunha no contexto do projecto. A partir desta análise teceram-se considerações arquitecturais a ter em atenção, que levaram à identificação de alterações a aplicar à estrutura base definida. A actividade culminou com a especificação detalhada da arquitectura, onde se identificaram os *drivers* arquitecturais do projecto, se tomaram decisões, e se identificaram os componentes que compõem a plataforma.

Uma vez que estas análise e discussão foram um elemento preponderante no estágio, considera-se relevante que a arquitectura da plataforma seja apresentada seguindo essa linha de maturação, mesmo que de um modo resumido. Assim, o presente capítulo compreende três secções, iniciando com a contextualização da plataforma, onde é apresentada a arquitectura base definida. A secção seguinte corresponde à análise dos desafios da migração de serviços *cloud*, onde se enunciam as dificuldades identificadas e as considerações que daí surgiram. Na última secção é apresentada a especificação da arquitectura da plataforma, onde se enunciam os vários *drivers* arquitecturais identificados, as decisões tomadas, e os componentes da plataforma.

5.1. Arquitectura base

Nesta secção é apresentada a arquitectura base definida para a plataforma Hotelcracy Apps, numa visão de concepção da sua estrutura inicial. Será introduzido o contexto em que a plataforma funcionará, e serão apresentados os grandes grupos funcionais que a constituem, aqui denominados de macro-componentes, identificados pelos *stakeholders* e especialistas de TI do consórcio. Esta arquitectura serviu de ponto de partida para a análise no âmbito da migração de serviços e a consequente evolução, apresentadas nas secções posteriores. Mais detalhes acerca desta estrutura base podem ser consultados no anexo HC_A2-Modelo_Conceptual.

5.1.1. Contexto da plataforma

Como enunciado no enquadramento do estágio (secção 1.1) têm surgido vários serviços *cloud* orientados para o sector hoteleiro, que pretendem dar resposta às novas necessidades do sector. Estes serviços dividem-se em 15 categorias, tais como: facturação, gestão de canais de distribuição, gestores de propriedades, entre outros (ver secção 2.2). Para dar resposta às diferentes necessidades de negócio, as unidades hoteleiras subscrevem vários serviços SaaS, cuja adopção se pode tornar uma operação morosa e dispendiosa. Muitas vezes, a adopção dos serviços é realizada por pessoal não especializado, resultando em portfólios desorganizados e/ou sobrepostos, que podem levar à duplicação de operações e de dados e, por conseguinte, a prejuízos para o hotel.

De modo a dar resposta a estas dificuldades do sector hoteleiro, o projecto Hotelcracy Apps pretende desenvolver uma plataforma que integrará serviços SaaS das várias categorias

identificadas, e disponibilizar essa integração para uso pelos hotéis. Estes poderão subscrever, utilizar, migrar e cancelar os vários serviços SaaS de terceiros através de uma interface de utilizador centralizada e homogénea. A plataforma funciona, assim, como ponto central da interacção entre utilizadores e serviços SaaS, sendo a responsável por garantir a correcta integração destes serviços.

A partir deste contexto, é possível identificar os seguintes intervenientes:

- **Plataforma Hotelcracy Apps** – Plataforma a ser desenvolvida no âmbito do projecto;
- **Serviços SaaS de terceiros** – Serviços SaaS que a plataforma Hotelcracy Apps integra, disponibilizando o acesso às funcionalidades destes serviços através da sua interface;
- **Responsáveis e outros funcionários dos hotéis** – Utilizadores que possuem uma relação com um hotel, e que fazem uso da plataforma Hotelcracy Apps para acederem às funcionalidades dos serviços SaaS integrados, com a finalidade de realizarem as operações do dia-a-dia do hotel.

Deste conjunto de intervenientes destacam-se os principais actores do sistema: os serviços SaaS de terceiros e os responsáveis e outros funcionários dos hotéis. Consideram-se principais pois existe uma relação directa entre estes dois intervenientes (responsáveis e funcionários dos hotéis fazem uso dos serviços SaaS nas operações do dia-a-dia) e porque a plataforma se quer tornar no intermediário dessa relação e, para isso, terá que ser capaz de dar resposta às necessidades destes actores.

Na Figura 12 está ilustrado o contexto em que a plataforma Hotelcracy Apps funcionará, onde é possível observar o seu posicionamento e as relações com os restantes intervenientes.

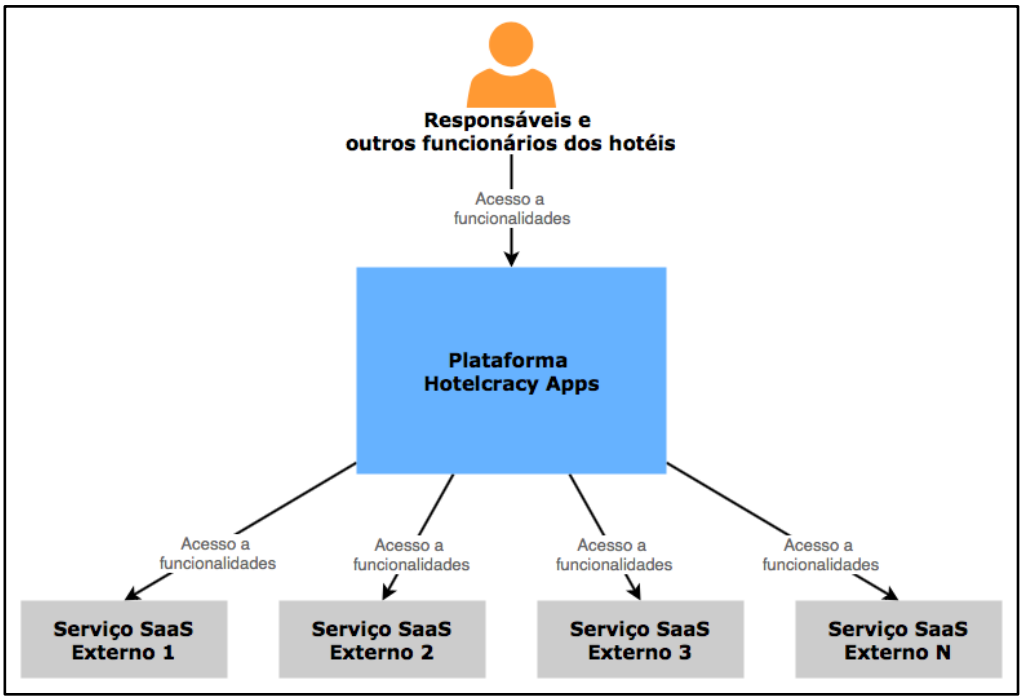

Figura 12 - Ilustração do contexto de funcionamento da plataforma Hotelcracy Apps

5.1.2. Macro-componentes da plataforma

Aquando da candidatura do projecto ao programa de financiamento PT2020, foi especificado um conjunto de grupos funcionais que compunham uma estrutura base da plataforma a

implementar, identificados com base na experiência dos *stakeholders* e dos especialistas de TI envolvidos no projecto. Da execução das tarefas da actividade 1 do projecto foi obtido conhecimento que permitiu refinar estes grupos e identificar outros novos, a partir dos quais se definiu uma estrutura de macro-componentes. Nesta estrutura, cada macro-componente possui responsabilidades distintas e complementares, que permitem dar resposta às necessidades dos intervenientes do sistema, servindo esta estrutura de ponto de partida para a evolução e especificação da arquitectura da plataforma.

Os macro-componentes definidos estão listados de seguida, sendo apresentada uma descrição breve das suas funções:

- ***User Interface*** – Disponibiliza uma interface web para o utilizador, a partir da qual este pode tirar partido da plataforma, e fazer uso das funcionalidades dos serviços SaaS, servindo como uma homogeneização das interfaces destes;
- ***Services Catalogue*** – Disponibiliza funcionalidades de gestão dos serviços SaaS integrados na plataforma. Inclui o acesso a informação comercial (por exemplo, descrição funcional e preçário) e informação técnica (por exemplo, detalhes do tipo de autenticação para acesso ao serviço) destes;
- ***Orchestrator*** – Tem como função orquestrar os processos de negócio do sistema Hotelcracy Apps, tais como a subscrição, migração e cancelamento de serviços SaaS, e ainda os processos relativos ao acesso às funcionalidades destes últimos.
- ***Integration Broker*** – Este macro-componente funciona como ponto de contacto com os serviços SaaS, servindo de intermediário entre o protocolo de cada um e o protocolo utilizado pelos componentes da plataforma. Possui associados componentes denominados de *Service Drivers*, um por cada serviço SaaS integrado. Cada um dos *Service Drivers* é responsável pela comunicação com um serviço específico, convertendo as mensagens do formato respondido pelo serviço SaaS para o modelo de dados interno da plataforma, e vice-versa.
- ***Identity & Access Manager*** – Fornece um serviço de gestão de utilizadores e de acessos às funcionalidades da plataforma, através da implementação de mecanismos de autenticação e autorização.
- ***Subscriptions Repository*** – Tem como função armazenar informação sobre as subscrições de serviços SaaS em vigor, realizadas pelos clientes da Hotelcracy Apps, em concreto as condições de subscrição e dados específicos do cliente (por exemplo, chaves de acesso do cliente à API do serviço SaaS).
- ***Control & Audit*** – Tem como função armazenar dados de controlo e de auditoria às operações realizadas na plataforma.
- ***Accounting & Billing*** – Tem como função armazenar dados de utilização dos serviços SaaS para efeitos de facturação, de modo a operacionalizar o modelo de negócio da plataforma.

5.1. Arquitectura base

Na Figura 13 é ilustrada a estrutura dos macro-componentes definidos, onde é possível obter uma visão da organização destes componentes na plataforma, e das relações entre si.

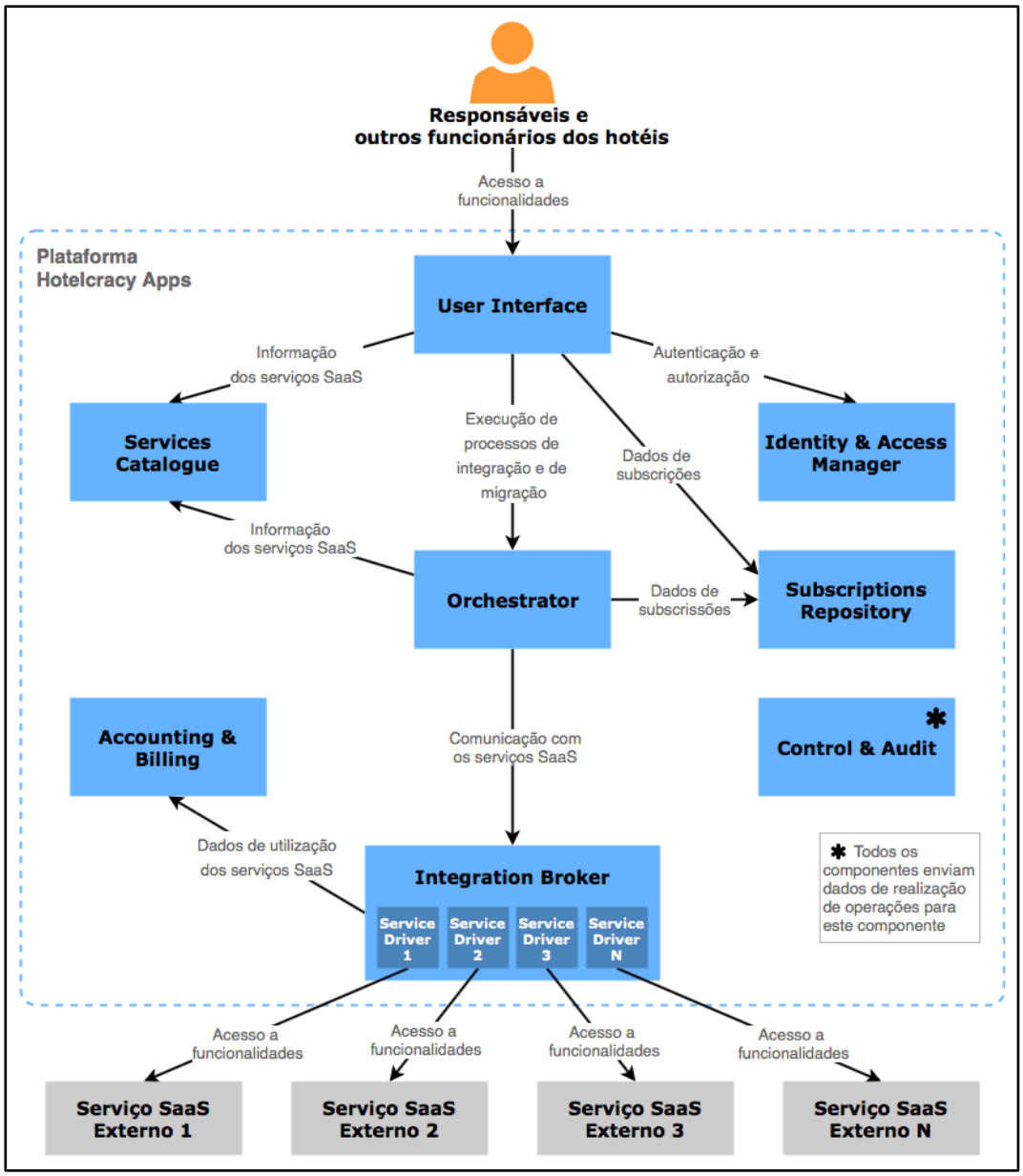

Figura 13 - Ilustração da organização dos macro-componentes da plataforma Hotelcracy Apps

Mais detalhes acerca dos macro-componentes definidos podem ser consultados no anexo HC_A2-Modelo_Conceptual. Para facilitar a consulta, na Tabela 6 é apresentado o mapeamento entre estes componentes e as secções correspondentes do anexo.

Macro-componente	Secção no anexo HC_A2-Modelo_Conceptual
User Interface	2.1
Services Catalogue	2.3
Orchestrator	2.4
Integration Broker	2.5
Identity & Access Manager	2.2
Subscriptions Repository	2.6
Control & Audit	2.7
Accounting & Billing	2.8

Tabela 6 - Mapeamento entre macro-componentes e secções do anexo HC_A2-Modelo_Conceptual

5.2. Análise dos desafios da migração de serviços *cloud*

A especificação da arquitectura de um sistema complexo de software deve envolver a análise e discussão das várias necessidades a que esta deve dar resposta, de modo a que os vários requisitos e constituintes necessários ao sistema sejam identificados, refinados e correctamente definidos. Deste modo, aumenta a garantia de que a arquitectura especificada é a adequada ao problema que se pretende solucionar.

No âmbito do projecto, a migração de serviços *cloud* é uma das necessidades a que a plataforma Hotelcracy Apps deve dar resposta. Para tal, procedeu-se à análise e discussão dos seus desafios da migração de serviços e do seu impacto na plataforma. Estas análise e discussão são apresentadas nesta secção, sendo dadas a conhecer as dificuldades identificadas, os problemas que delas advêm, e as várias soluções consideradas para estes problemas. A partir destas soluções, e tendo em conta a arquitectura base definida, foram tecidas algumas considerações arquitecturais, que se traduziram posteriormente em alterações à base definida.

A informação aqui apresentada envolveu a obtenção de conhecimento através da análise e realização de testes às APIs de vários serviços SaaS (enunciados abaixo), e o desenho e análise de propostas de processos e de sequências de execução das operações relacionadas com a migração. A informação recolhida e as propostas de processos desenhadas foram apresentadas e debatidas com os orientadores do estágio e outros especialistas de TI em reuniões de projecto, e permitiram a identificação de soluções e alterações a aplicar à arquitectura da plataforma.

5.2.1. Problemática da migração de serviços *cloud*

Uma das funcionalidades que a plataforma Hotelcracy Apps pretende disponibilizar é a de um mercado/catálogo de serviços SaaS, onde é apresentada informação dos vários serviços disponíveis para utilização de modo integrado na plataforma, permitindo aos hoteleiros analisar quais os que melhor se adequam à sua unidade hoteleira e subscrevê-los. Como indicado no anexo técnico da candidatura, o mercado hoteleiro está em crescente evolução [1]. Com esta evolução, as soluções SaaS escolhidas no passado podem não dar respostas às necessidades actuais dos hotéis, existindo assim a necessidade de se substituir os serviços em uso. Deste modo, a plataforma, para além de dar o poder de escolha no momento de aquisição de uma primeira solução de *software*, pretende que esse poder se mantenha ao longo da vida da unidade hoteleira e da utilização do serviço SaaS.

Normalmente, no sector hoteleiro, a aquisição de um novo serviço implica a alocação de recursos (como pessoas ou dinheiro) para realização da sua configuração e da migração dos dados presentes no serviço antigo. A falta de integração entre serviços implica que esta tarefa de migração seja executada por humanos, o que a torna uma tarefa morosa e com custos para além dos do preço do próprio serviço. Deste modo, a plataforma pretende disponibilizar a possibilidade de um hotel substituir um serviço subscrito por outro da mesma categoria, migrando do serviço antigo para o serviço novo as configurações e dados nele presentes, retirando ao hoteleiro a carga associada a essa tarefa, e colocando o serviço novo num estado pronto a ser utilizado. A existência da migração permitirá assim aos hoteleiros alterarem mais facilmente de serviço, subscrevendo aquele que melhor responda às suas necessidades no momento.

Do ponto de vista da plataforma, a migração deveria seguir um processo estruturado, que envolvesse as seguintes tarefas base: a subscrição do serviço novo; a transferência de dados do serviço antigo para o novo; e o cancelamento do serviço antigo. A partir destas considerações, identificaram-se as tarefas necessárias à correcta execução da migração de serviços, e estas foram organizadas no processo ilustrado na Figura 14.

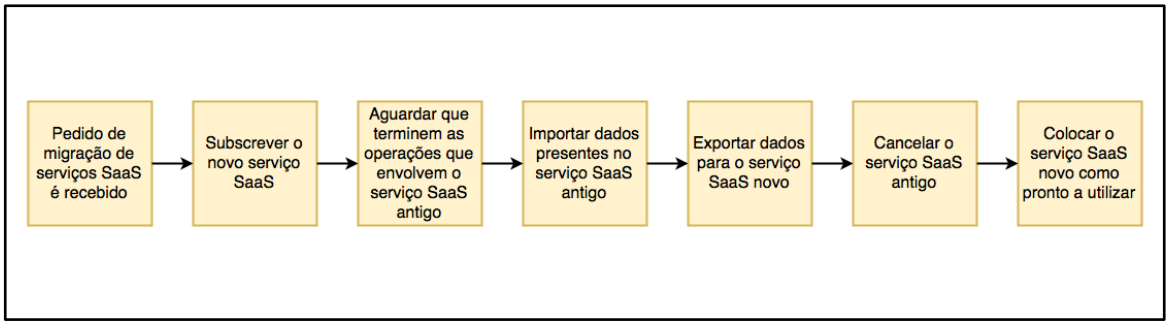

Figura 14 - Processo para a migração de serviços SaaS

O processo inicia-se com a recepção de um pedido para migração entre dois serviços SaaS. O novo serviço deve então ser subscrito, e a plataforma deve impedir o lançamento de novas operações que envolvam o serviço SaaS antigo. Deve também garantir-se que não existem operações pendentes que envolvam este último, e, caso haja, deve aguardar-se que essas terminem. De seguida, devem importar-se os dados presentes no serviço antigo, sendo esses dados utilizados na tarefa seguinte, a da exportação para o serviço novo. Depois da exportação dos dados, o serviço antigo deve ser cancelado, e o processo é finalizado com a colocação do serviço SaaS novo num estado que permita ser utilizado pelos funcionários da unidade hoteleira.

No caso da plataforma Hotelcracy Apps, a interacção com os serviços SaaS deve suportar-se nas APIs que estes serviços disponibilizam, tanto para o uso de um modo integrado como para a migração de serviços. Estas APIs são desenhadas e desenvolvidas em concreto para acesso e uso por máquinas (computadores), ao contrário da interface de utilizador, a qual é desenhada e desenvolvida para uso por um humano. Tendo este aspecto em mente, procedeu-se à análise das APIs de alguns serviços SaaS, com a finalidade de se perceber se o processo de migração desenhado era adequado, e de se averiguar em que moldes deveria ser executada a interacção entre a plataforma e os serviços, em específico para a realização da migração. Uma vez que as categorias de serviços SaaS consideradas prioritárias pelos *stakeholders* do projecto eram as de *channel manager* e de facturação, as APIs analisadas pertenciam aos serviços de *channel manager* Hotel-Spider (www.hotel-spider.com), HotelRunner (www.hotelrunner.com) e WooDoo by WuBook (wubook.net/channel-manager), e aos serviços de facturação InvoiceXpress (invoicexpress.com), InvoiceOcean (invoiceocean.com) e Zoho Invoice (www.zoho.eu/invoice).

Apesar do foco da análise ser nas APIs dos serviços, esta não se restringiu apenas à interacção directa com estas. Foram também realizados testes de interacção com o serviço através da sua interface de utilizador, o que permitiu obter mais conhecimento quanto à lógica de negócio associada, e foi ainda realizada a leitura da documentação das APIs, o que permitiu obter conhecimento quanto ao modo como estas devem ser utilizadas.

Os testes realizados permitiram identificar as entidades de negócio com que estes serviços lidam, e as características destas entidades, tais como atributos e relações entre si. Permitiram ainda compreender quais as limitações das APIs dos serviços e identificar várias dificuldades e problemas que essas limitações impõem ao projecto, principalmente ao nível da arquitectura da plataforma. Estas dificuldades, apesar de terem sido identificadas no âmbito da migração de serviços, estendem-se também à integração dos serviços para a sua utilização.

5.2.2. Dificuldades e problemas identificados e soluções consideradas

Da análise anterior, foram identificadas várias dificuldades, que se podem traduzir em problemas impactantes ao nível da arquitectura da plataforma. Essas dificuldades e problemas são apresentados de seguida, sendo também descritas as soluções consideradas para esses problemas.

5.2.2.1. Dificuldade 1 – Documentação incompleta ou incorrecta

Da análise da documentação dos serviços considerados, identificaram-se casos em que a documentação existente era incompleta ou incorrecta. Apesar de não possuir um impacto ao nível da arquitectura da plataforma, este é um ponto que se considera importante realçar, uma vez que dificulta a análise e integração dos serviços, o que influencia negativamente o tempo necessário para a implementação.

Em alguns casos a documentação era incompleta, possuindo apenas informação mínima quanto às configurações de acesso à API (identificação das credenciais de autenticação), e exemplos simples de acesso às funcionalidades. Estes incluíam apenas o URL e um exemplo reduzido do corpo do pedido, não sendo apresentada nem a explicação dos atributos do pedido nem a resposta esperada. Esta situação verificou-se na documentação do serviço InvoiceOcean [55].

Foram também identificados casos em que a documentação estava incorrecta. Nestes casos, a documentação possuía exemplos de respostas a serem devolvidas pela API, os quais não reflectiam a realidade, uma vez que existiam diferenças quanto aos atributos presentes nos

corpos das respostas. Esta situação verificou-se na documentação dos serviços Hotel-Spider [56] e Zoho Invoice [57].

Esta dificuldade não se traduziu num problema para o qual fosse necessário identificar soluções possíveis.

5.2.2.2. Dificuldade 2 – APIs não disponibilizam a totalidade das funcionalidades do serviço

O leque de funcionalidades presentes nas APIs dos serviços não engloba a totalidade de funcionalidades que estes disponibilizam através das suas interfaces de utilizador. Os serviços da categoria de *channel manager* são um dos exemplos onde este caso de verifica. Nestes serviços é possível realizar uma gestão dos quartos disponíveis para reserva, com a interface de utilizador a permitir a adição, consulta e edição dos dados. No entanto, as APIs dos serviços apenas permitem a consulta e a edição dos dados dos recursos, não existindo funcionalidades para criação. Este tipo de funcionalidades (criação de recursos) é crítico para que a migração de serviços seja possível, uma vez que a sua inexistência pode inviabilizar essa migração.

Problema a solucionar: execução de operações não disponibilizadas pelas APIs

O facto de existirem operações que não podem ser realizadas através das APIs dos serviços, implica identificarem-se abordagens que permitam contornar esta dificuldades, mesmo que apenas parcialmente. Assim, consideraram-se as seguintes soluções:

a) *Crawler* de interface de utilizador

Esta solução propõe a implementação de mecanismos de análise das interfaces de utilizador dos serviços SaaS, de modo a permitir executar as operações através da interface do serviço sem necessidade da interacção directa do utilizador. Implicaria identificar as interfaces correspondentes às operações a serem realizadas, e desenvolver o *crawler* de modo a enviar os dados necessários e a tratar possíveis erros.

Esta solução não é a ideal, uma vez que as interfaces de utilizador mudam com regularidade, e alterações nelas, mesmo que reduzidas, podem implicar o mau funcionamento do *crawler*, e inviabilizar a correcta execução das operações.

b) Introdução de dados pelo utilizador através da interface de utilizador do serviço

Esta solução propõe atribuir ao utilizador a responsabilidade de introduzir um conjunto mínimo de dados no novo serviço SaaS, através da sua interface de utilizador. Este conjunto mínimo seria o estritamente necessário, de modo a permitir que a plataforma realize o resto da migração de modo automático.

Apesar de não ser a ideal, esta é uma solução que permite lidar com a dificuldade identificada. Ainda assim, possui vários aspectos que podem dificultar a realização da migração, tais como: a plataforma ter que ser capaz de estabelecer uma relação com os dados que estão a ser inseridos no serviço; o utilizador introduzir os dados necessários de um modo incorrecto; a interface de utilizador do serviço sofrer alterações e impedir que o utilizador insira os dados necessários.

5.2.2.3. Dificuldade 3 - Inexistência de normalização das funcionalidades das APIs de serviços SaaS da mesma categoria

Da análise realizada, verificaram-se várias disparidades entre APIs de serviços de uma mesma categoria, ao nível das funcionalidades que disponibilizam. Estas disparidades revelam que, apesar de existirem *standards* de nomenclatura e de estruturas de dados definidos para o sector hoteleiro (e.g., OpenTravel Alliance [58] e AlpineBits [59]), estes não são aplicados extensivamente, levando à inexistência de normalização nas APIs. Estas disparidades verificam-se ao nível da nomenclatura das funcionalidades, das entidades/recursos que tratam, e dos atributos destas entidades.

No primeiro caso, identificaram-se funcionalidades semelhantes que possuíam uma denominação diferente de API para API. Por exemplo, a funcionalidade para obter dados de reservas num serviço de *channel manager*, no caso do serviço SaaS HotelRunner denomina-se "Retrieve Reservations" [60], enquanto que no serviço Hotel-Spider possui o nome "OTA_ReadRQ" [61].

No segundo caso, identificaram-se APIs que usam nomes diferentes para entidades semelhantes, e/ou consideram diferentes grupos de entidades nas suas mensagens. Por exemplo, nas APIs dos serviços SaaS de facturação, algumas consideram que a entidade que representa um produto facturável se denomina de "product" [62] enquanto que outras usam o nome de "item" [63]. Também nesta categoria de serviços, algumas APIs consideram a existência de duas entidades separadas para o produto e o imposto, estabelecendo uma relação entre elas [64], [65], enquanto que outras APIs consideram apenas a existência da entidade relativa ao produto, contendo esta atributos para os dados do imposto [62].

No terceiro caso, identificaram-se funcionalidades em que, para a execução de uma acção semelhante, eram considerados atributos com nomenclaturas e/ou obrigatoriedades diferentes de API para API. Por exemplo, para a criação de uma factura num serviço de facturação, alguns dos atributos a serem enviados são a data de emissão, a data de vencimento, e a lista de produtos facturados. No caso do serviço InvoiceXpress, estes atributos denominam-se "date", "due_date" e "items", respectivamente, com os três atributos a serem obrigatórios [66], enquanto que para o serviço InvoiceOcean estes possuem os nomes "issue_date", "payment_to" e "positions", com apenas o atributo "positions" a ser obrigatório [67].

Problema a solucionar: relacionamento e conversão de dados de diferentes APIs

A migração implica a transferência dos dados presentes no serviço SaaS de origem para o serviço SaaS de destino. Uma vez que a API de cada serviço usa estruturas próprias, que podem não ser iguais às de outras APIs, é necessário que se estabeleça uma relação entre as estruturas consideradas, e se proceda à conversão dos dados do formato compreendido pela API do serviço de origem para o formato compreendido pela API do serviço de destino. As soluções seguintes abordam o modo como essas relação e conversão podem ser realizadas.

a) Conversão ponto-a-ponto

Nesta solução, a conversão de dados é feita por pares de serviços, e implica que o formato compreendido por um determinado serviço seja convertido directamente para o formato de cada um dos serviços da categoria em que se insere, com o mesmo a acontecer para todos os outros serviços, de todas as categorias.

Esta solução não é ideal, uma vez que se torna incomportável, mesmo para um reduzido número de serviços. A Figura 15 ilustra a aplicação desta solução, com dois exemplos, onde é possível observar a dificuldade de aplicação. Na figura, cada seta de duplo sentido representa duas conversões, por exemplo, de A para B e de B para A. O exemplo 1 da figura representa uma determinada categoria que possui 3 serviços (serviços A, B e C), e para a qual é necessário implementar a conversão para 6 pares de serviços (de A para B, de A para C, de B para A, de B para C, de C para A, e de C para B). O exemplo 2 representa a mesma categoria, agora a possuir 5 serviços, e para a qual passa a ser necessário implementar a conversão de 20 pares de serviços. É evidente o crescimento de combinações necessárias que se registará se a quantidade de serviços também aumentar.

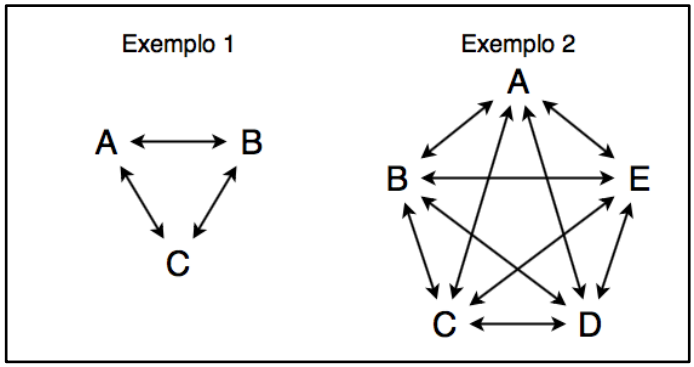

Figura 15 - Exemplos de aplicação da solução de conversão ponto-a-ponto

Com esta solução, a integração de um novo serviço fica também ela dificultada, uma vez que obriga à implementação de lógica nova nos componentes já existentes, de modo a que consigam lidar com a conversão de e para o formato do novo serviço.

b) Conversão com recurso a um formato intermédio

Nesta solução é usado um formato intermédio, para o qual os formatos utilizados por todos os serviços SaaS são convertidos, e a partir do qual é possível realizar a conversão para comunicar no formato esperado pelos serviços. A Figura 16 ilustra os mesmos exemplos da solução anterior, mas desta vez com a utilização de um formato intermédio, identificado pela letra "H". Nela podemos ver que para o exemplo 2 a quantidade de pares de conversão é inferior aos da solução anterior.

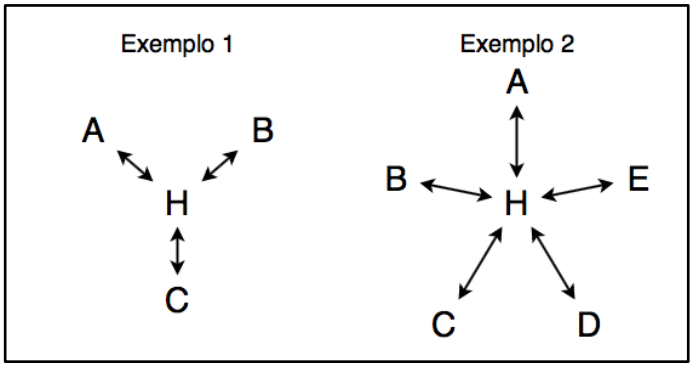

Figura 16 - Exemplos de aplicação da solução de conversão com recurso a um formato intermédio

Esta solução aparenta ser a ideal a aplicar, uma vez que permite que a lógica necessária à comunicação com um determinado serviço não seja influenciada pelos restantes serviços integrados, ou tenha que ser alterada quando é integrado um serviço novo. A solução reforça também uma ideia já considerada para a utilização dos serviços de um modo integrado: a

definição de um modelo de dados interno à plataforma, e que permitiria a combinação de dados de serviços de diferentes categorias.

5.2.2.4. Dificuldade 4 – Inexistência de funcionalidades nas APIs para transferência de dados em *bulk*

Do conjunto de funcionalidades que as APIs analisadas disponibilizam, não se identificaram funcionalidades que permitissem obter todos os dados presentes no serviço SaaS de uma só vez, operação habitualmente denominada de "obtenção de dados em *bulk*" [68]. Também não se identificaram funcionalidades que permitissem o tratamento de dados em *bulk*, como a criação de vários recursos com uma única chamada à API. Deste modo, a obtenção de dados e o envio destes para os serviços (como parte do processo de migração) terão que ser executados através de invocações sucessivas das funcionalidades das APIs.

Problema a solucionar: Local da lógica da migração

Desta necessidade, realização de várias chamadas para obtenção e envio de dados de e para os serviços SaaS, nasce o problema associado ao local onde reside a lógica para realização dessas interacções. A solução deste problema deve ter em atenção que as várias chamadas às APIs podem necessitar de seguir uma determinada rotina, e que esta rotina pode ser específica de cada serviço, devido às disparidades identificadas nas funcionalidades das APIs. De seguida, apresentam-se as soluções consideradas para este problema.

a) Lógica de migração presente nos *Service Drivers*

Nesta solução, a lógica para se migrar de e para um determinado serviço SaaS estaria presente no componente *Service Driver* (integrante do macro-componente *Integration Broker*) a ele correspondente, com cada um dos *Service Drivers* a possuir a lógica necessária ao seu serviço. Esta seria separada na rotina de obtenção de dados do serviço e na rotina de envio dos dados para esse serviço. O componente de orquestração (*Orchestrator*) seria responsável por indicar ao *Service Driver* do serviço de origem que executasse a lógica de obtenção dos dados e, quando este terminasse, indicaria ao *Service Driver* do serviço de destino que executasse a lógica de envio, passando-lhe os dados obtidos inicialmente. Esta solução está ilustrada na Figura 17, com apresentação de um excerto dos componentes da plataforma, e onde é possível visualizar o local onde residiria a lógica associada às rotinas de migração.

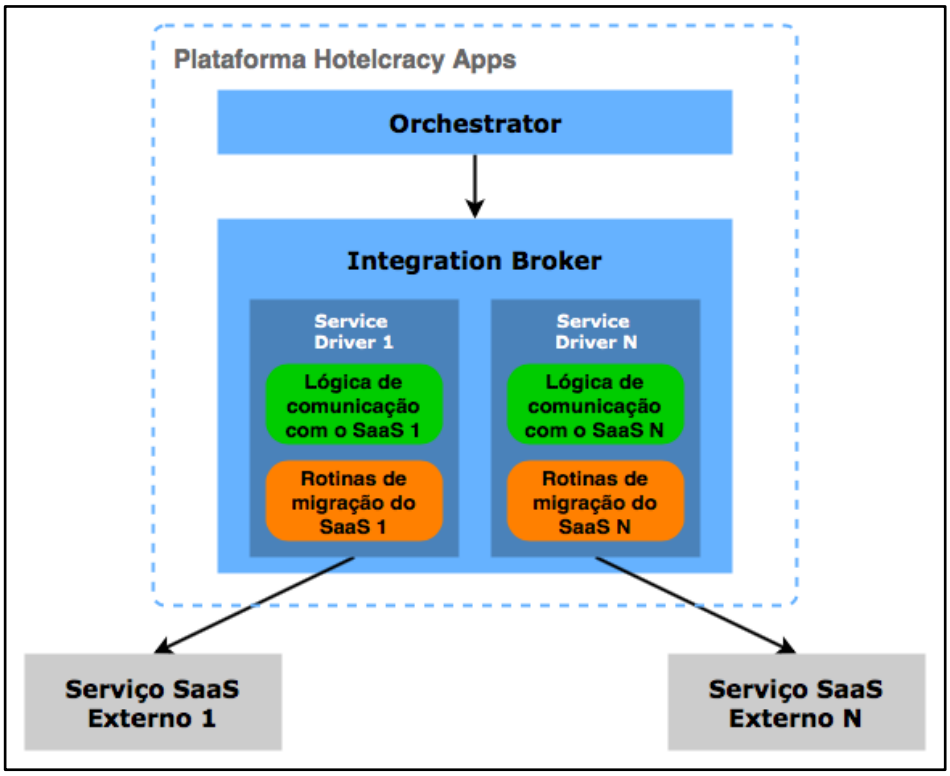

Figura 17 - Solução a) - Lógica de migração presente nos Service Drivers

Esta solução não é considerada ideal, uma vez que desvirtua o principal papel do *Service Driver*, o de converter uma mensagem do formato utilizado internamente pela plataforma para o formato compreendido pelo serviço SaaS, e realizar a comunicação dessa mensagem ao serviço. Esta solução pode implicar também um elevado esforço computacional nestes componentes aquando da realização das rotinas de migração, podendo levar a quebras na sua execução. Esta quebra teria impacto tanto na migração, como nas operações de outros clientes, uma vez que se considera que os *Service Drivers* são partilhados pelos vários utilizadores.

b) Componente dedicado à migração, suportado pelas funcionalidades dos *Service Drivers*

Nesta solução, a lógica de migração estaria presente num componente dedicado, denominado de *Migrator*, com as rotinas de obtenção e de envio de dados a realizarem as comunicações com os serviços SaaS através dos componentes *Service Driver*. Este componente dedicado seria responsável apenas pela execução do processo de migração, em específico das rotinas associadas aos serviços SaaS. Esta solução está ilustrada na Figura 18.

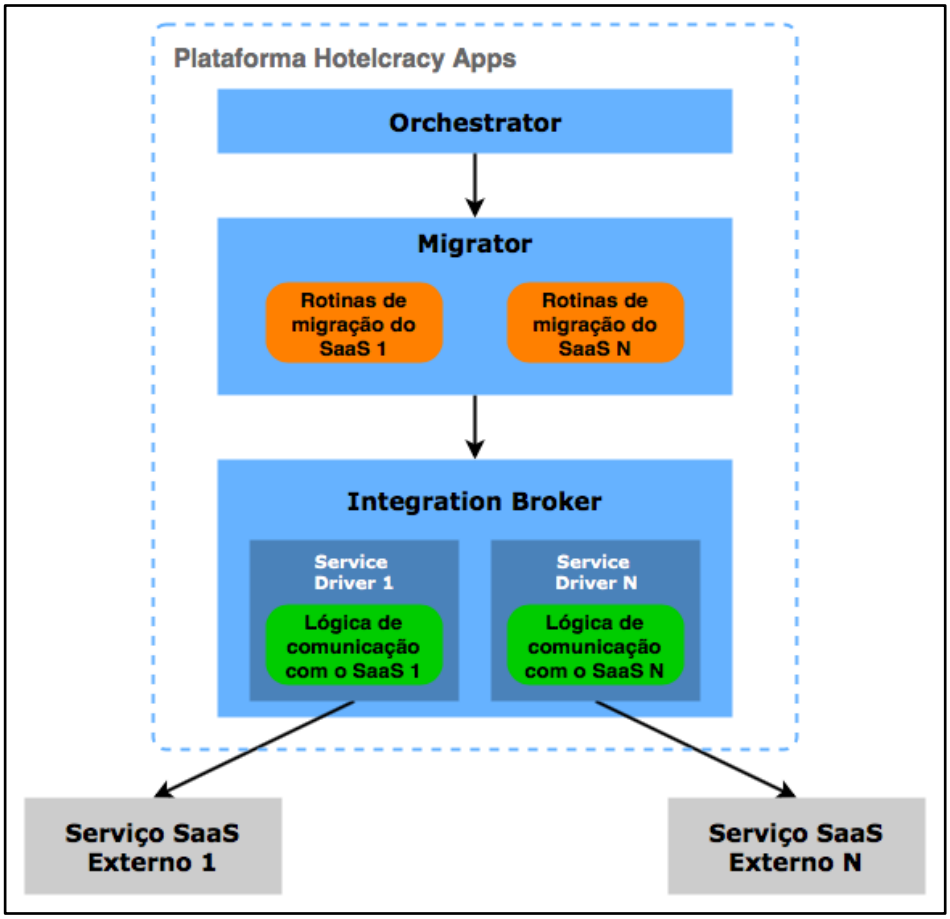

Figura 18 - Solução b) - Componente dedicado à migração, suportado pelas funcionalidades dos Service Drivers

A vantagem desta solução em relação à anterior é o facto de a execução das rotinas de migração estar localizada num componente dedicado, diminuindo o esforço computacional dos componentes *Service Drivers*.

No entanto, esta abordagem não é considerada ideal, uma vez que coloca a lógica de migração longe da fronteira da plataforma. Esta característica é relevante, uma vez que o projecto Hotelcracy Apps pretende que, no futuro, a integração de novos serviços SaaS na plataforma seja realizada pela empresa que os comercializa. Para que isto seja possível, a plataforma deve colocar toda a lógica associada aos serviços integrados o mais próximo possível da sua fronteira, para que as alterações impostas por essa nova integração tenham o menor impacto possível na estrutura da plataforma e nos componentes já existentes.

c) Utilização de componentes dedicados à migração

Esta solução pode ser vista como o aglutinar das duas soluções anteriores, deixando de fora a utilização dos componentes *Service Drivers* já existentes. A solução implica a criação de novos componentes, dedicados à migração de serviços, em específico às tarefas que envolvem transferências de dados.

Nos novos componentes a criar inclui-se uma gama dedicada às comunicações com os serviços SaaS para realização exclusiva da migração, denominados de *Migration Service Drivers*. Esta gama compreende a existência de um componente por cada serviço SaaS integrado, com cada componente a conter toda a lógica associada às rotinas de obtenção e de envio de dados de e para o serviço a que corresponde. Cada componente inclui ainda a lógica para a

comunicação com o serviço e para conversão de dados entre os protocolos do serviço SaaS e da plataforma.

O outro componente a criar, denominado de *Migrator*, será responsável pela orquestração das tarefas de transferência de dados presentes no processo de migração de serviços (a obtenção de dados do serviço de origem e o envio de dados para o serviço de destino) fazendo uso dos *Migration Service Drivers*. Para obter os dados presentes no serviço de origem, indica ao *Migration Service Driver* correspondente a esse serviço que execute a rotina de importação de dados. Com os dados importados, o *Migrator* comunica com o *Migration Service Driver* correspondente ao serviço de destino, para que execute a rotina de exportação de dados para o serviço.

Esta solução está ilustrada na Figura 19, onde se identificam os novos componentes: o *Migrator* e os *Migration Service Drivers*.

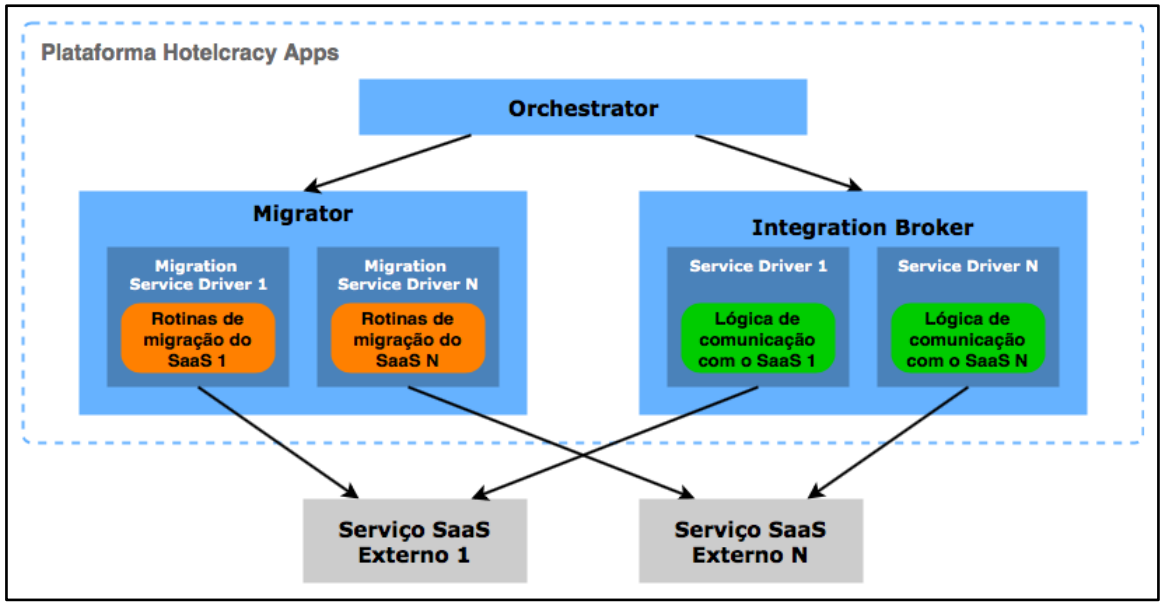

Figura 19 - Solução c) - Utilização de componentes dedicados à migração

As vantagens desta solução em relação à anterior são conseguir-se que a migração seja executada em componentes dedicados, localizados na fronteira da plataforma. Deste modo, reduz-se tanto o esforço computacional dos *Service Drivers* já existentes como o impacto da realização de alterações.

Esta abordagem também não é ideal, uma vez que pode implicar uma duplicação das funcionalidades necessárias à conversão de mensagens e à realização de chamadas às APIs dos serviços SaaS, as quais estão já presentes nos *Service Drivers* já existentes. No entanto, é a solução que aparenta melhor responder às necessidades do projecto, em específico a necessidade de colocar a lógica associada a cada serviço SaaS o mais próximo possível da fronteira da plataforma.

5.2.3. Considerações arquitecturais

Com base nas discussões realizadas no âmbito da migração de serviços SaaS e nas soluções consideradas para as dificuldades e problemas encontrados, foram tecidas algumas considerações arquitecturais. Nelas identificaram-se alterações/evoluções a aplicar à plataforma, as quais se traduziram posteriormente em decisões arquitecturais (que serão enumeradas na especificação da arquitectura, na secção seguinte). Essas considerações são apresentadas de seguida.

Utilização de um formato de dados intermédio para a conversão de mensagens

Esta consideração compreende a utilização de um formato de dados interno à plataforma, para a conversão e tratamento de mensagens utilizadas na comunicação com os serviços SaaS. Este formato intermédio terá o nome de "Modelo Canónico" e será construído com base nas estruturas de dados presentes nas mensagens das várias APIs dos serviços integrados. Deste modo, toda a comunicação de dados de negócio dos hotéis dentro da plataforma será baseada em mensagens que seguem este formato, para o qual e a partir do qual todas as mensagens recebidas e enviadas de e para os serviços são convertidas.

A utilização de um formato de dados intermédio vai ao encontro de uma necessidade identificada pelos *stakeholders* do projecto, a de se armazenar na plataforma os dados de negócio dos hotéis. Ao utilizar este formato intermédio para o armazenamento dos dados, a plataforma torna-se capaz de lidar com a disparidade de estruturas verificada nas APIs dos vários serviços, e de estabelecer relações entre dados de negócio de categorias de serviços diferentes.

Utilização de componentes dedicados à migração de serviços

Esta consideração compreende a utilização de componentes na plataforma que estão responsáveis exclusivamente pela execução de operações relacionadas com a migração de serviços. Esta decisão compreende também a divisão do processo de migração em três partes: uma relativa à subscrição e ao cancelamento de serviços SaaS, a qual fica a cargo do componente de orquestração já existente (o *Orchestrator*); uma relativa à orquestração das tarefas de transferência de dados do serviço de origem para o serviço de destino, que será tratada por um componente dedicado (o *Migrator*); e uma última relativa às rotinas associadas à obtenção e envio de dados de e para os serviços.

Ao nível arquitectural, esta consideração implica as seguintes alterações:

- Os componentes *Service Drivers* identificados inicialmente na arquitectura base da plataforma devem passar a denominar-se de "*Integration Service Drivers*", focando-se unicamente na integração de serviços para uso nas operações hoteleiras;

- Deve ser criado um componente responsável pela orquestração das tarefas de transferência de dados de um serviço para o outro, com o nome *Migrator*. O componente *Orchestrator* mantém-se como responsável por iniciar e terminar todo o processo de migração de serviços, mas delegará para o *Migrator* a parte do processo relativa à transferência dos dados entre serviços.

- Devem ser criados componentes onde está confinada a lógica associada às rotinas de obtenção e de envio de dados para um determinado serviço, sendo responsáveis pela execução destas rotinas quando requerido. Estes componentes terão o nome de "*Migration Service Drivers"*, e existirá um por cada serviço SaaS integrado na plataforma. O componente *Migrator* fará uso destes componentes na execução do processo de migração, em específico para a obtenção e o envio de dados de e para os serviços em causa.

5.3. Especificação detalhada da arquitectura

A especificação da arquitectura de um sistema de software é o conjunto de decisões de design que permitem compreender e justificar a estrutura aplicada a esse sistema. Estas baseiam-se em requisitos funcionais e comportamentais do software, nos utilizadores que farão uso dele,

nas restrições técnicas e de negócio (actuais e a longo prazo) que são impostas, e na experiência da equipa de projecto.

Para a especificação da arquitectura da plataforma Hotelcracy Apps seguiu-se a metodologia *Architecture Trade-off Analysis Method* (ATAM). Esta metodologia tem por base o levantamento de *drivers* ou requisitos significativos de arquitectura junto dos *stakeholders* do projecto, que se traduzem depois em cenários significativos, restrições técnicas e de negócio, e em atributos de qualidade. Estes *drivers* são depois combinados com abordagens e decisões arquitecturais, com a finalidade de se identificarem *trade-offs*, pontos sensíveis e riscos associados à arquitectura especificada [69].

Nesta secção é apresentada, resumidamente, a especificação da arquitectura considerada para a plataforma Hotelcracy Apps, levada a cabo pelos vários elementos dos parceiros do consórcio, onde se inseriu também o estagiário. São apresentados os *drivers* arquitecturais levantados, as decisões tomadas, e os riscos e fragilidades identificados, e também os diagramas que ilustram a estrutura associada à arquitectura especificada. Mais detalhes podem ser consultados no anexo HC_A3-Arquitectura_Plataforma.

5.3.1. *Drivers* arquitecturais

Um *driver* arquitectural é um requisito que tem um impacto profundo na arquitectura do sistema de software e pode, consequentemente, ter um alto significado para o negócio suportado pelo sistema. Para o levantamento dos drivers arquitecturais devem também ser considerados os actores que têm acesso ao sistema, de modo a compreender as restrições que estes possam provocar e entender as suas expectativas na utilização da plataforma.

Seguidamente, são apresentados os actores identificados e os drivers arquitecturais (cenários significativos, restrições, e atributos de qualidade) levantados no âmbito do projecto.

5.3.1.1. Actores

No âmbito do projecto, foram identificados os seguintes actores:

- **Anónimo**: utilizador não autenticado que pretende conhecer a plataforma, efectuar o registo gratuito e/ou autenticar-se na plataforma;
- **Utilizadores Hotelcracy**: grupo de utilizadores que pertencem à Hotelcracy Software Lda. Estes dividem-se em:
    - **Administrador da plataforma**: utilizador autenticado que colabora com a Hotelcracy Software Lda. e tem como função o controlo e gestão de toda a plataforma;
    - **Colaborador da empresa**: utilizador autenticado que colabora com a Hotelcracy Software Lda. e tem funções de apoio à gestão da plataforma. Este tipo de utilizador possuí permissões limitadas em relação ao Administrador.
- **Utilizadores Cliente**: grupo de utilizadores que pertencem a uma empresa no sector do alojamento (i.e., hotel, hostel, pensão, entre outros) que é cliente da Hotelcracy Software Lda. Estes dividem-se em:
    - **Administrador de cliente**: utilizador autenticado que é responsável por administrar os dados do cliente, por exemplo, gerir as subscrições de serviços do cliente;
    - **Colaborador de cliente**: utilizador autenticado que colabora com uma empresa no sector do alojamento e executa operações nos serviços subscritos pelo Administrador de cliente.

- **Parceiro**: utilizador autenticado que colabora com um parceiro da Hotelcracy Software Lda. para, por exemplo, gerir os seus serviços no *Marketplace* e aceder a informação de integração do seu serviço na plataforma;
- **Serviço SaaS BoB externo**: aplicação que fornece operações à plataforma Hotelcracy Apps.

Mais detalhes acerca destes actores podem ser consultados na secção 2 do anexo HC_A3-Arquitectura_Plataforma.

5.3.1.2. Cenários significativos

Os cenários considerados significativos são cenários de utilização/operação que possuem um elevado impacto na arquitectura da plataforma e no valor de negócio desta. No âmbito do projecto, estes cenários foram especificados a partir do trabalho desenvolvido aquando da candidatura do projecto ao PT2020, e no levantamento de macro-requisitos do sistema (tarefa inserida nas actividades de execução do projecto).

Os cenários especificados são apresentados seguidamente. Mais detalhes destes cenários significativos podem ser consultados na secção 3 do anexo HC_A3-Arquitectura_Plataforma.

CS_01 - Navegar num *Marketplace* de serviços SaaS BoB do sector do alojamento

O *Marketplace* pretende funcionar com um catálogo de serviços SaaS BoB, que suporta o processo de escolha de um serviço no poder da comunidade, expandindo assim a capacidade de decisão dos hoteleiros independentes.

CS_02 - Fácil integração de serviços SaaS BoB

A criação da plataforma deve ser focada na integração e escolha de soluções SaaS especializadas do sector do alojamento, e na fácil expansão do leque de soluções integradas.

CS_03 - Subscrição automática/semiautomática de serviço SaaS BoB

A subscrição de serviços será, quando possível, realizada pelo sistema de forma automática. Quando não for possível, será implementando um processo semiautomático que conduzirá o utilizador na subscrição do serviço.

CS_04 - Migração de serviços SaaS BoB da mesma categoria

Pretende-se aumentar a flexibilidade de uso da plataforma e o dinamismo do ecossistema de serviços SaaS, fomentando a capacidade e a flexibilidade de, quando possível, transferir as configurações e informações já introduzidas pelo utilizador. Deste modo, é dada ao utilizador a capacidade de facilmente reconfigurar o seu portfólio de serviços subscritos, migrando para o que melhor responda às suas necessidades.

CS_05 - Integração fluída de várias categorias de serviços SaaS BoB

Pretende-se integrar funcionalidades de diferentes serviços sob uma única interface com o utilizador, de modo a mascarar e abstrair as integrações a serem realizadas. Esta integração visa simplificar a realização de operações do sector, aumentado a fluidez da sua execução.

CS_06 - Cancelamento automático/semiautomático de serviço SaaS BoB

O cancelamento de serviços será, quando possível, realizado pelo sistema de forma automática. Quando não for possível, será implementando um processo semiautomático que conduzirá o utilizador no cancelamento do serviço.

CS_07 - Acesso autenticado e seguro ao sistema

Pretende-se que o acesso às funcionalidades do sistema seja realizado através do registo e autenticação de utilizadores.

CS_08 - Segurança de dados entre *tenants* do sistema

Pretende-se que o funcionamento SaaS do sistema não comprometa os dados dos seus clientes, isto é, que um utilizador apenas consiga aceder aos dados da sua organização.

CS_09 - Efectuar operação fornecida por um serviço SaaS BoB

Pretende-se que o utilizador aceda e execute operações fornecidas por um serviço SaaS através do sistema Hotelcracy Apps.

CS_10 - Gerir perfis e permissões dos utilizadores da empresa

Pretende-se que os clientes, enquanto empresas com permissões de acesso diversificadas, possam controlar autorizações distintas para os vários utilizadores e para os múltiplos serviços SaaS subscritos.

5.3.1.3. Restrições

As restrições impostas a um sistema de software limitam as opções a serem tomadas ao nível do desenho da arquitectura. Estas ajudam a equipa de projecto a compreender as dificuldades do projecto, permitindo que sejam tomadas decisões que não criem dificuldades a longo prazo para o sistema.

Seguidamente, apresentam-se as restrições identificadas para a plataforma, ao nível técnico, de negócio, e de qualidade, as quais podem ser consultadas em detalhe na secção 4 do anexo HC_A3-Arquitectura_Plataforma.

5.3.1.3.1. Restrições técnicas

As restrições técnicas são condições de design pré-existentes que moldam a arquitectura, não podendo ser contornadas. Um exemplo de restrição deste tipo é a necessidade de se fazer uso de APIs disponibilizadas pelos sistemas externos.

Seguidamente, descrevem-se as restrições técnicas identificadas para o projecto. Mais detalhes acerca destas restrições podem ser consultados na secção 4.1 do anexo HC_A3-Arquitectura_Plataforma.

RT_01 – Tecnologias de desenvolvimento

A plataforma deverá ser desenvolvida com recurso à *framework* Ruby-on-Rails para o *back-end* e à biblioteca React.js para o *front-end*, uma vez que são tecnologias já em uso pelo parceiro

líder do projecto. Deste modo, evita-se a curva de aprendizagem de novas tecnologias, e facilita ao parceiro líder a manutenção da solução no pós-projecto.

RT_02 - Dependência de protocolos de comunicação implementados pelos serviços SaaS externos ao projecto

A plataforma necessita comunicar com serviços SaaS que fazem uso de protocolos e linguagens diversas, nomeadamente, SOAP, REST, RPC e JSON ou XML.

Não existindo flexibilidade para a uniformização dos protocolos de comunicação entre os serviços, a arquitectura deverá permitir que exista um desacoplamento entre a lógica interna do sistema e os serviços com os quais comunica.

RT_03 - Limitações das API dos serviços SaaS ao nível das operações fornecidas (inserção e recolha de dados)

As funcionalidades a serem disponibilizadas pela plataforma Hotelcracy Apps estão dependentes das operações permitidas pelas APIs dos serviços SaaS externos, e pelas limitações que estas APIs impõem. Das análises realizadas identificaram-se casos de serviços que não disponibilizam através das suas APIs a totalidade de funcionalidades que permitem através das suas interfaces gráficas.

A acrescentar à problemática da limitação das APIs, existe a hipótese de os clientes poderem usar a interface de utilizador nativa dos serviços contratados. Como os serviços considerados não estão preparados para notificar a plataforma Hotelcracy Apps no caso de este evento acontecer, a plataforma poderá ter que lidar com problemas ao nível da coerência e integridade dos dados.

RT_04 – A integração entre operações de diferentes serviços deve ser realizada pelo componente de interface de utilizador da plataforma Hotelcracy Apps

Um dos objectivos da plataforma é disponibilizar a integração de operações de diferentes serviços SaaS, e disponibilizar essa integração através de uma única interface de utilizador. Na Actividade 1 do projecto concluiu-se que a grande maioria das operações do sector seriam simples/atómicas, não envolvendo um encadeamento de invocações a diferentes funcionalidades de diferentes serviços. Deste modo, foi decidido pelo consórcio que não existiria orquestração de processos de integração nos componentes internos da plataforma, e que, para cada funcionalidade de serviços SaaS a ser invocada, o componente de interface de utilizador deveria realizar um pedido aos componentes internos para que a funcionalidade seja invocada. A existir orquestração, esta terá que ser realizada pelo componente de interface de utilizador.

Considere-se o exemplo de um utilizador da plataforma que possui três serviços subscritos, os quais disponibilizam a funcionalidade para criação de um cliente. Neste exemplo, ao ser submetido pelo utilizador um formulário com dados para criação de um cliente nos três serviços, o componente de interface de utilizador da plataforma deve realizar um pedido para cada serviço, neste caso, três pedidos distintos, e deve lidar com as três respostas que receberá.

RT_05 - Fornecimento da plataforma num modelo SaaS

O parceiro líder do consórcio pretende que a plataforma seja comercializada segundo um modelo SaaS. Este constrangimento implica garantir um correcto isolamento de dados entre

os diversos clientes, e a implementação de um mecanismo de autorização que garanta que não existem acessos indesejados a dados de outras organizações.

5.3.1.3.2. Restrições de negócio

As restrições de negócio são condições incontornáveis relacionadas com a exploração do sistema de software ao nível de negócio (por exemplo, custo da equipa de desenvolvimento do sistema).

Seguidamente, descrevem-se as restrições de negócio identificadas para o projecto. Mais detalhes acerca destas restrições podem ser consultados na secção 4.2 do anexo HC_A3-Arquitectura_Plataforma.

RN_01 - Fornecimento da plataforma num modelo de SaaS

O fornecimento da plataforma num modelo de SaaS é também uma restrição de negócio porque implica uma análise dos custos de infraestrutura e manutenção, tendo um impacto significativo no modelo de negócio a desenhar para a plataforma. Esta será acedida como um serviço, através da Internet, devendo ser definido qual o nível de maturidade[4] do modelo de SaaS a ser aplicado, de modo a dar resposta às necessidades de configurabilidade, escalonamento e eficiência para vários clientes.

RN_02 - Custos de desenvolvimento

O esforço financeiro necessário para o desenvolvimento da plataforma deverá estar de acordo com a candidatura proposta ao programa PT2020, contemplando o valor total de € 656.841,83.

RN_03 - Inexistência de parcerias com fornecedores dos serviços SaaS

A imaturidade da API da maioria dos serviços SaaS estudados indica que será importante para o negócio o estabelecimento de parcerias, diminuindo o esforço de implementação das integrações e potenciando o sucesso da platforma. Os mecanismos para integração de novos serviços no ecossistema de aplicações fornecidas via plataforma Hotelcracy Apps deverá ser simples, removendo o bloqueio à entrada e valorizando a oferta da plataforma, potenciando o estabelecimento de parcerias.

RN_04 - Recursos humanos disponíveis para o desenvolvimento

A quantidade de recursos necessária para o desenvolvimento da plataforma deverá estar de acordo com a candidatura proposta ao programa PT2020. Os recursos considerados são: um coordenador de projecto; um gestor de projecto; dois investigadores seniores; quatro programadores seniores; e um programador júnior.

RN_05 - Período de desenvolvimento

A estimativa temporal necessária para o desenvolvimento da plataforma deverá estar de acordo com a candidatura proposta ao programa PT2020, a qual compreende um período de 33 meses.

[4] Níveis de maturidade das soluções SaaS segundo Carraro & Chong [72], que podem ser consultados no anexo HC_A3-Arquitectura_Plataforma, secção 4.2.1.

RN_06 - Custos em actividades de suporte ao cliente

A empresa Hotelcracy Software Lda. não pondera investir recursos em actividades de suporte ao cliente, nomeadamente, constituição de centros telefónicos de apoio ao cliente. A plataforma deverá guiar, sempre que possível, o utilizador nas operações mais complicadas e possuir um bom sistema de tolerância a falhas.

RN_07 - Criação e manutenção de ecossistema de serviços SaaS

A criação do ecossistema de aplicações com base num *Marketplace* aberto é essencial ao modelo de exploração pensado pela empresa Hotelcracy Software Lda. A plataforma deve, assim, permitir uma forma simples e eficaz de gerir e incluir novos serviços SaaS, considerando que, no futuro, se pretende que sejam equipas responsáveis pelos serviços SaaS externos a desenvolver as integrações desses serviços de forma segura e sem risco para a plataforma.

5.3.1.4. Atributos de qualidade

Os atributos de qualidade, são propriedades mensuráveis e testáveis do sistema de software, usados para assegurar que este satisfaz as necessidades dos seus *stakeholders* [70], e que em conjunto com as restrições condicionam muito mais a arquitectura do sistema do que os requisitos funcionais. Estes atributos de qualidade são quantificados com recurso à descrição de cenários, os quais são depois priorizados em relação à sua importância e impacto esperado na arquitectura do sistema.

Na secção 5 do anexo HC_A3-Arquitectura_Plataforma é possível consultar a descrição dos atributos de qualidade identificados para a plataforma Hotelcracy Apps, segundo a metodologia proposta por Len Bass no livro *Software Architecture in Pratice* [70]. Esta metodologia propõe que os atributos de qualidade sejam descritos com recurso a cenários bem definidos, que possuem as seguintes características:

- **Atributo de qualidade**: nome do atributo de qualidade relacionado com o cenário;
- **Fonte de estímulo**: identifica a origem do estímulo (por exemplo, um utilizador autenticado ou um sistema de computador);
- **Estímulo**: descreve um evento recebido pelo sistema. É uma condição que requer uma resposta do sistema (por exemplo, uma operação de um utilizador ou um ataque ao sistema);
- **Ambiente**: define as circunstâncias em que o cenário ocorre. O sistema pode estar numa condição de sobrecarga ou em funcionamento normal, ou nalgum outro estado relevante. Para muitos sistemas, a operação "normal" pode referir-se a um de um certo número de modos;
- **Artefacto**: o artefacto pode ser uma colecção de sistemas, todo o sistema, ou alguma parte ou partes do sistema, que foram estimulados;
- **Resposta**: descreve a forma como o sistema deve responder ao estímulo. A resposta consiste nas responsabilidades que o sistema (qualidades de *runtime*) ou programadores (qualidades de desenvolvimento) devem assumir aquando do estímulo.
- **Mensuração da resposta**: determina a forma como a resposta do sistema deve ser medida, de modo a que o requisito seja testado (por exemplo, medição da latência ou do tempo para executar, testar e instalar uma modificação ao sistema).

Na Tabela 7 é apresentado, a título de exemplo, um cenário de atributo de qualidade, retirado do anexo HC_A3-Arquitectura_Plataforma.

ID DO CENÁRIO	CQ_01
ATRIBUTO DE QUALIDADE	Escalabilidade
FONTE DE ESTÍMULO	Utilizadores
ESTÍMULO	Grande volume de novas contas, aumentando o número de pedidos ao sistema a um factor de 10 vezes.
ARTEFACTO	Sistema Hotelcracy Apps
AMBIENTE	O sistema encontra-se a funcionar correctamente.
RESPOSTA	O sistema consegue executar as operações e fornecer feedback ao utilizador.
MENSURAÇÃO DA RESPOSTA	Tempo de resposta do tratamento do pedido, contabilizado desde a sua recepção no sistema até à sua resposta.

Tabela 7 – Exemplo de tabela de um cenário de atributo de qualidade

Seguidamente, são apresentados os atributos de qualidade identificados para a plataforma, com recurso a uma descrição breve e resumida de alguns dos cenários considerados no documento anexo.

CQ_01 - Escalabilidade

A plataforma, num ambiente de operação normal, deve ser capaz de lidar com um incremento (de um factor de 10 vezes) no número de pedidos a serem processados, mantendo a execução normal das operações e fornecendo feedback ao utilizador.

CQ_02 - Disponibilidade

Num ambiente de operação normal, ao ocorrer uma quebra de serviço num dos componentes, a plataforma deve detectar a falha, comunicá-la ao administrador e manter o correcto funcionamento dos restantes componentes.

CQ_03 - Extensibilidade

Um parceiro da Hotelcracy Software Lda. integra um novo serviço SaaS na plataforma, disponibilizando o código dos componentes necessários e respectivas configurações. Num ambiente de operação normal, a plataforma deve configurar-se automaticamente para comunicar correctamente com o serviço, e manter o correcto funcionamento dos restantes componentes.

CQ_04 - Usabilidade

Um utilizador faz uso da plataforma para realização de uma operação do dia-a-dia da sua unidade hoteleira, submetendo um pedido através da interface de utilizador. A plataforma, num ambiente de operação normal, deve tratar o pedido e devolver *feedback* do resultado para o utilizador num período máximo de 10 segundos [71].

CQ_05 - Interoperabilidade

Um componente do sistema inicia um pedido para comunicação de informação com outro componente. Num ambiente de operação normal, onde os meios de comunicação entre

componentes (i.e., funcionalidades/operações que estes disponibilizam/permitem) são conhecidos à partida, o pedido é devidamente aceite e tratado, e a informação trocada com sucesso.

CQ_06 - Modificabilidade

A equipa do projecto pretende integrar um novo serviço SaaS na plataforma. A plataforma, num ambiente de desenvolvimento, deve permitir a integração do novo serviço, sendo realizadas modificações apenas nas fronteiras da plataforma (alterações nos componentes de interface com o utilizador, e adição de componentes dedicados à comunicação com o serviço SaaS) e/ou adicionadas ou alteradas configurações.

CQ_07 - Segurança

Um utilizador autenticado e identificado perante a plataforma, realiza uma operação de alteração de um conjunto determinado de dados. A plataforma, num ambiente de operação normal, deve detectar e registar a execução dessa operação, armazenando informação a ela relativa e ao utilizador que a realizou, de modo a ser possível determinar o seu autor (i.e., não-repudio da realização de operações).

CQ_08 - Manutenibilidade

A equipa de projecto pretende realizar uma correcção num componente bem definido da plataforma. A realização da alteração no componente em questão não deve implicar modificações adicionais nos restantes componentes da plataforma.

CQ_09 - Testabilidade

Após a realização de alterações ao código de componentes da plataforma, a execução da suite de testes deve permitir a detecção de erros ao nível da interacção entre componentes, e fornecer informação acerca do erro detectado e dos componentes em causa.

5.3.2. Decisões de desenho da plataforma

Nesta sub-secção são apresentadas as decisões arquitecturais tomadas para a especificação da arquitectura. Estas decisões foram tomadas com base nos *drivers* arquitecturais identificados e na priorização destes, de modo a dar resposta às necessidades evidenciadas por cada *driver* e em função da sua importância no âmbito do projecto. Seguidamente é apresentada a priorização de *drivers* considerada e, posteriormente, as decisões tomadas, cuja versão detalhada pode ser consultada na secção 6 do anexo HC_A3-Arquitectura_Plataforma.

5.3.2.1. Priorização dos *drivers* arquitecturais

Na Tabela 8 é apresentado um resumo da priorização de *drivers* arquitecturais considerada para o projecto, realizada pelos *stakeholders* do projecto. A tabela possui as seguintes colunas:

- **Driver arquitectural**: identificador e nome do *driver* arquitectural;
- **Importância**: quantificação da importância do *driver* arquitectural no sucesso do projecto;
- **Dificuldade**: quantificação da dificuldade de implementação do *driver* arquitectural.

As colunas "Importância" e "Dificuldade" da tabela utilizam a notação **B** – Baixa, **M** – Média, **A** – Alta, e **E** – Elevada.

Na secção 6.1 do anexo HC_A3-Arquitectura_Plataforma é possível consultar a tabela completa da priorização considerada.

***Driver* arquitectural**	**Importância**	**Dificuldade**
Cenários Significativos		
CS_01 - Navegar num *Marketplace* de serviços SaaS BoB no sector do alojamento	A	M
CS_02 - Fácil integração de serviços SaaS BoB	E	A
CS_03 - Subscrição automática/semiautomática de serviço SaaS BoB	M	A
CS_04 - Migração de serviços SaaS BoB da mesma categoria	A	E
CS_05 - Integração fluída de várias categorias de serviços SaaS BoB	E	A
CS_06 - Cancelamento automático/semiautomático de serviço SaaS BoB	M	A
CS_07 - Acesso autenticado e seguro ao sistema	E	B
CS_08 - Segurança de dados entre tenants do sistema	E	M
CS_09 - Efectuar operação fornecida por um serviço SaaS BoB	E	M
CS_10 - Gerir perfis e permissões dos utilizadores da empresa	A	B
Restrições Técnicas		
RT_01 – Tecnologias de desenvolvimento	A	B
RT_02 - Dependência de protocolos de comunicação implementados pelos serviços SaaS externos ao projecto	E	E
RT_03 - Limitações das API dos serviços SaaS ao nível das operações fornecidas (inserção e recolha de dados)	E	E
RT_04 – Integração entre operações de diferentes serviços deve ser realizada pelo componente de interface de utilizador da plataforma Hotelcracy Apps	E	M
RT_05 - Fornecimento da plataforma num modelo SaaS	A	A

Driver arquitectural	Importância	Dificuldade
Restrições de negócio		
RN_01 - Fornecimento da plataforma num modelo de SaaS	A	A
RN_02 - Custos de desenvolvimento	A	A
RN_03 - Inexistência de parcerias com fornecedores dos serviços SaaS	M	M
RN_04 - Recursos humanos disponíveis para o desenvolvimento	A	M
RN_05 - Período de desenvolvimento	A	A
RN_06 - Custos em actividades de suporte ao cliente	M	M
RN_07 - Criação e manutenção de ecossistema de serviços SaaS	A	A
Atributos de Qualidade		
CQ_01 - Escalabilidade	B	M
CQ_02 - Disponibilidade	E	A
CQ_03 - Extensibilidade	A	A
CQ_04 - Usabilidade	A	A
CQ_05 - Interoperabilidade	A	M
CQ_06 - Modificabilidade	A	M
CQ_07 - Segurança	E	M
CQ_08 - Manutenibilidade	A	A
CQ_09 - Testabilidade	A	B

Tabela 8 - Priorização dos drivers arquitecturais

5.3.2.2. Decisões arquitecturais

De seguida apresentam-se as decisões arquitecturais tomadas pela equipa de projecto, com uma descrição resumida de cada decisão. Estas são justificadas com base nos *drivers* arquitecturais a que respondem, sendo estes identificados no final de cada decisão. As descrições integrais das decisões, e o cruzamento com os *drivers* arquitecturais, podem ser consultadas na secção 6.2 do anexo HC_A3-Arquitectura_Plataforma.

Decisão 1 – Escolha do modelo SaaS

A equipa de projecto optou pela implementação inicial da plataforma num modelo de SaaS do nível 3, em que uma única instância serve os vários clientes, disponibilizando opções de configuração por cliente, e isolamento de dados entre si [72]. No entanto, deve ser considerado que a plataforma evoluirá para o modelo do nível 4, onde são utilizadas várias instâncias idênticas com balanceamento de carga, com a aplicação a ser configurável por cliente [72].

Esta decisão responde aos *drivers* arquitecturais RT_05 e RN_01.

Decisão 2 – Escolha de abordagem arquitectural a aplicar

A equipa de projecto optou pela utilização de uma arquitectura orientada para os micro-serviços, a qual permite a separação de responsabilidades por vários componentes da plataforma. Esta separação permite que a adição ou alteração de um componente não implique alterações significativas nos restantes componentes, e o teste de componentes tanto de modo isolado como integrado. A utilização desta abordagem arquitectural permite ainda que cada componente seja escalado de modo independente, adaptando a configuração da plataforma (i.e., instâncias de componentes em execução) em função da carga de utilização.

Esta decisão responde aos *drivers* arquitecturais RN_04, CQ_01, CQ_02, CQ_03, CQ_06, CQ_08 e CQ_09.

Decisão 3 – Normalização das estruturas de dados das APIs

De modo a dar resposta à inexistência de normalização entre APIs de diferentes serviços SaaS (ver Dificuldade 3, na secção 5.2.2.3), a equipa de projecto optou pela definição de um modelo canónico para estruturação dos dados a serem comunicados internamente na plataforma. Este modelo canónico servirá de ponto intermédio, para o qual os dados recebidos dos serviços SaaS serão traduzidos, e vice-versa. Permitirá estabelecer uma relação entre estruturas de dados de diferentes serviços, agilizando a integração e a migração destes, e será definido a partir das funcionalidades das APIs dos serviços SaaS, das entidades/recursos que estas APIs consideram, e dos atributos que constituem essas entidades/recursos. O modelo deverá ser flexível à sua própria evolução, induzida pela integração de novos serviços, e à acomodação de informação cuja obrigatoriedade varie de serviço para serviço.

O modelo canónico deve ainda ser interpretável pelos componentes da plataforma, de modo a que estes adeqúem o seu comportamento em função deste modelo e dos serviços subscritos pelo utilizador que requereu a execução de uma determinada operação. Por exemplo, quando o utilizador pede para visualizar o formulário de criação de uma determinada entidade, a interface de utilizador deve ser capaz de indicar os campos de preenchimento obrigatório em função dos serviços subscritos nesse contexto.

A utilização deste modelo permite uma normalização dos dados a serem tratados, facilitando o desenvolvimento e a alteração de componentes, e a interacção entre eles. Permite ainda que os vários componentes da plataforma se abstraiam das especificidades de cada serviço, facilitando o acesso às operações que estes serviços disponibilizam e a integração de novos.

Esta decisão responde aos *drivers* arquitecturais CS_02, CS_04, CS_05, CS_09, CQ_03, CQ_05 e CQ_06.

Decisão 4 – Localização da lógica de negócio inerente a cada serviço SaaS

A equipa de projecto optou pela aplicação do padrão de design "Adapter" [38] para definição do local da lógica de negócio associada a cada serviço SaaS, cuja aplicação na integração de serviços *cloud* foi identificada aquando da análise do estado da arte (ver secção 4.2.2.2). Este padrão considera que a lógica associada aos serviços externos deve estar concentrada em componentes localizados na fronteira do sistema de software, através dos quais se realiza a comunicação com esses serviços [38]. No contexto da plataforma, estes componentes devem ser denominados de *Service Drivers*, e devem dividir-se em *Integration Service Drivers* (componentes com lógica para a utilização das funcionalidades das APIs dos serviços na realização de operações do sector hoteleiro) e em *Migration Service Drivers* (componentes com lógica para a importação e exportação de dados de e para os serviços SaaS). Devem ser capazes de converter os dados recebidos dos serviços para o formato usado internamente (modelo canónico) e vice-versa, e de estabelecer a comunicação com as APIs dos serviços SaaS externos.

Esta decisão responde aos *drivers* arquitecturais CS_02, RT_02, RN_03, CQ_03 e CQ_06.

Decisão 5 – Armazenamento de informação na plataforma

De modo a agilizar o processo de migração de serviços, a equipa de projecto optou pelo armazenamento de informação na plataforma, com recurso a uma base de dados baseada no modelo canónico identificado. Este armazenamento permite que os dados sejam verificados e validados quanto à sua completude, e, para os casos em que se detecta informação incompleta, executar tarefas que permitam ao utilizador submeter os dados em falta. O armazenamento de informação ocorrerá não só durante o processo de migração, mas também durante a utilização da plataforma nas operações do dia-a-dia do hotel, com recolha dos dados submetidos pelos utilizadores aquando da utilização de funcionalidades da plataforma.

O componente onde serão armazenados os dados de negócio dos hotéis será denominado de *Business Data Manager*.

Decisão 6 – Acesso à plataforma e comunicação entre componentes

A equipa de projecto optou por definir o componente *User Interface* como único ponto de contacto dos utilizadores com a plataforma. O componente será responsável por garantir a autenticação e autorização dos utilizadores dos pedidos recebidos, antes de ser realizado o seu processamento.

Para a comunicação entre componentes da plataforma, cada componente possuirá apenas mecanismos que permitam verificar e validar que a mensagem recebida provem de um componente fidedigno, removendo a necessidade de cada um verificar ou validar o utilizador autor do pedido.

Esta decisão responde aos *drivers* arquitecturais CS_07, RQ_02 e CQ_07.

Decisão 7 – Construção da interface de utilizador

A equipa de projecto optou pelo desenvolvimento de uma interface de utilizador *responsive*, de modo a que esta possa ser utilizada em vários tipos de dispositivos com um reduzido impacto na usabilidade da mesma.

Esta decisão responde aos *drivers* arquitecturais RQ_01 e CQ_04.

Decisão 8 – Realização de operações não contempladas pelas APIs

A equipa de projecto optou por delegar para o utilizador a realização de operações que não estejam contempladas pelas APIs, através da interface de utilizador dos serviços. Esta decisão visa sobretudo os casos em que as APIs dos serviços não disponibilizam funcionalidades para adição de recursos, inviabilizando a migração de serviços.

Esta decisão não pretende fazer uso do utilizador como substituto da automatização do processo de migração, mas antes como meio de tornar essa automatização possível, mesmo que apenas parcial. Para que seja possível, é necessário que sejam bem definidas quais as interacções necessárias realizar pelo utilizador na interface gráfica do serviço, quais os dados a serem introduzidos, e como estabelecer uma relação entre os dados introduzidos e os dados já existentes na plataforma. Poderá ser feito uso de guias que ajudem o utilizador na correcta introdução dos dados, de modo a facilitar a tarefa.

Esta decisão responde ao *driver* arquitectural RT_03.

Decisão 9 – Execução do processo de migração

A equipa de projecto optou por definir componentes dedicados à migração de serviços SaaS, tanto para a orquestração do processo como para a execução de lógica específica de cada serviço. O processo de migração possuirá três níveis:

- O nível 1 compreende as tarefas relacionadas com a subscrição e cancelamento de serviços, e a invocação do processo relativo à transferência de dados entre os serviços SaaS alvo da migração;
- O nível 2 corresponde ao processo de transferência de dados referido no nível 1, e compreende as tarefas relacionadas com a obtenção e o envio de dados de e para os serviços, e tarefas de verificação e validação da completude dos dados a migrar;
- O nível 3 corresponde à lógica associada às rotinas para se obterem e enviarem dados para os serviços SaaS (tarefas referidas no nível 2), que compreenderá rotinas específicas por serviço.

Cada um destes níveis será da responsabilidade de componentes específicos:

- O nível 1 será da responsabilidade do componente *Orchestrator*. Este componente não é dedicado unicamente à migração de serviços, pois possuirá mecanismos para a subscrição e o cancelamento de serviços, e para tratar pedidos de utilização destes. Uma vez que a migração envolve a subscrição e o cancelamento de serviços, torna-se possível fazer uso destes mecanismos;
- O nível 2 será da responsabilidade do componente *Migrator*. Este componente receberá do *Orchestrator* pedidos para realizar o processo de transferência de dados entre os serviços considerados;
- O nível 3 será da responsabilidade da gama de componentes *Migration Service Drivers*, já enunciados na decisão 4. Estes componentes receberão do *Migrator* os pedidos para executarem as rotinas de obtenção ou envio de dados de e para os serviços SaaS.

Esta decisão implica que exista duplicação de funcionalidades ao nível da conversão de formatos de dados e da comunicação com os serviços externos, uma vez que esta necessita estar presente nos componentes *Integration Service Drivers* e nos componentes *Migration Service Drivers*.

5.3.3. Componentes da plataforma

Nesta sub-secção é apresentada a ilustração da arquitectura da plataforma, baseada nas decisões tomadas. Para tal, foram desenhados diagramas segundo a metodologia proposta por Simon Brown, o modelo C4 [73]. Esta metodologia propõe uma descrição da arquitectura com base em vários níveis de abstração, representando estruturas arquitecturais da plataforma. Neste relatório serão apresentados dois níveis de abstração:

- **Nível 1 - contexto da plataforma:** este diagrama tem como objectivo identificar os limites da plataforma, o actores externos com os quais esta interage e suas interfaces externas;
- **Nível 2 - componentes da plataforma:** este diagrama tem como objectivo identificar os componentes base que constituem a plataforma, identificando as interacções entre si e os actores externos.

Os diagramas apresentados seguidamente utilizarão os objectos gráficos representados na Figura 20.

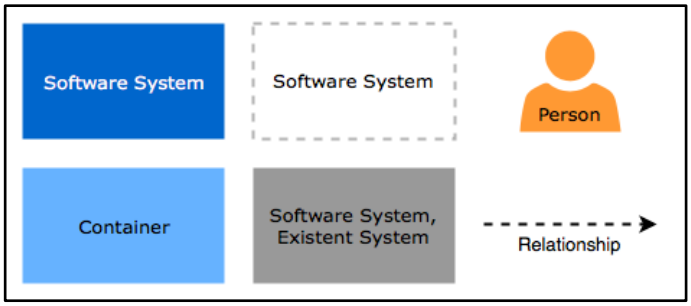

Figura 20 - Legenda dos diagramas de arquitectura da plataforma Hotelcracy Apps

Na secção 7 do anexo HC_A3-Arquitectura _Plataforma é possível consultar em detalhe a informação aqui apresentada.

5.3.3.1. Diagrama de contexto

Na Figura 21 apresenta-se o diagrama de contexto da plataforma Hotelcracy Apps. Neste diagrama é possível conferir os actores identificados para o sistema (ver secção 5.3.1.1), o posicionamento da plataforma, e as interacções entre esta e os actores identificados. É possível constatar que o diagrama aqui apresentado é uma versão evoluída e mais detalhada da ilustração de contexto apresentada na arquitectura base (ver secção 5.1.1) e onde se identificam novos intervenientes do sistema.

Na secção 7.1 do anexo HC_A3-Arquitectura_Plataforma é possível obter mais detalhes acerca do diagrama apresentado.

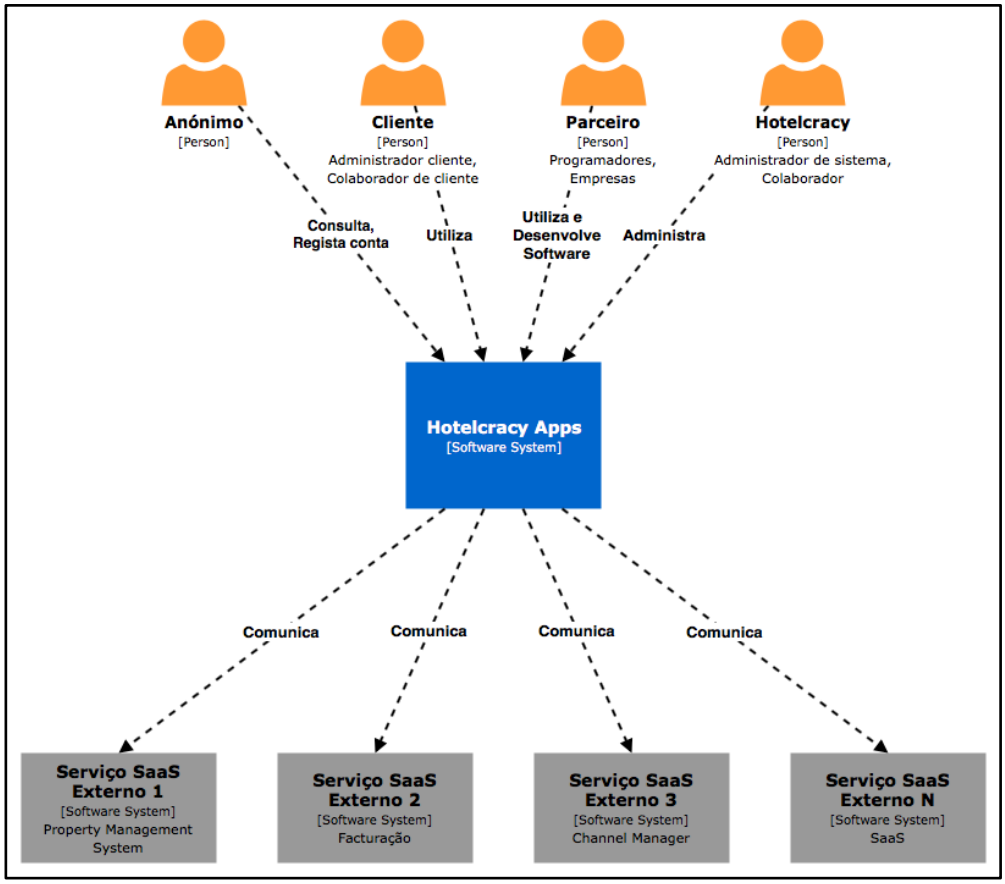

Figura 21 - Diagrama de contexto da plataforma Hotelcracy Apps

5.3.3.2. Diagrama de componentes

Na Figura 22 é apresentado o diagrama de componentes da plataforma. Neste diagrama é possível observar os vários componentes que constituem a plataforma, e as interacções entre esses componentes e os actores externos (i.e., utilizadores da plataforma e serviços SaaS externos). A plataforma pressupõe uma arquitectura orientada a serviços (*Service-Oriented Architecture* - SOA), com os vários componentes a representarem esses serviços, comunicando entre si utilizando o protocolo HTTPS e um formato de dados na notação JSON.

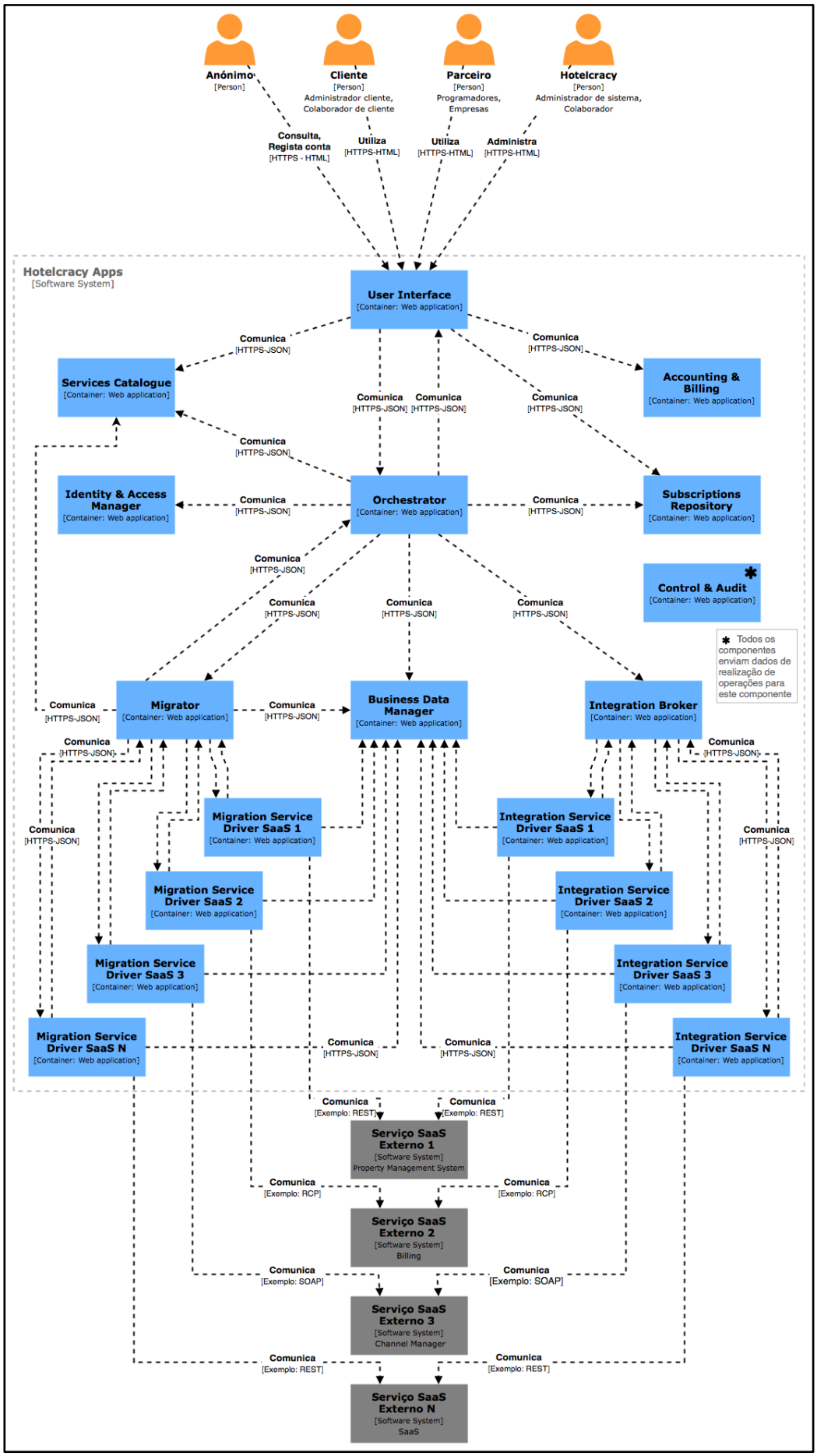

Figura 22 - Diagrama de componentes da plataforma Hotelcracy Apps

Os vários componentes identificados neste nível de descrição da arquitectura são baseados nos macro-componentes identificados na arquitectura base (ver secção 5.1.2). À semelhança desses macro-componentes, possuem responsabilidades distintas, dando resposta a necessidades específicas do projecto. Essas responsabilidades podem ser consultadas em detalhe na secção 7.2 do anexo HC_A3-Arquitectura_Plataforma.

De seguida, é apresentado um resumo do papel de cada componente identificado, das suas responsabilidades e das interacções com os outros componentes da plataforma:

- ***User Interface***: componente de interface com os utilizadores, responsável por lhes fornecer acesso às operações do sistema de uma forma centralizada, através de um *browser*. Comunica directamente com a maioria dos componentes da plataforma, de modo a aceder a informação e a operacionalizar acções realizadas pelos utilizadores;
- ***Services Catalogue***: responsável pela implementação funcional do *Marketplace* de serviços SaaS BoB. Fornece informação (comercial e/ou técnica) destes serviços para outros componentes da plataforma, utilizada na execução de operações destes;
- ***Orchestrator***: responsável por orquestrar e executar os processos de subscrição, utilização, migração e cancelamento de serviços SaaS, suportando-se noutros componentes da plataforma para a execução desses processos. O tratamento e a obtenção de informação de subscrições de serviços SaaS é realizado com recurso ao componente *Subscriptions Repository*, e a invocação de funcionalidades dos serviços é realizada com recurso ao componente *Integration Broker*. Na migração de serviços, a execução de tarefas relativas à transferência e ao tratamento de dados é delegada para o componente *Migrator*;
- ***Integration Broker***: responsável por redireccionar e distribuir mensagens provenientes do *Orchestrator* para os *Integration Service Drivers*, e vice-versa. É um componente intermédio entre este dois, que funciona como um *API gateway* [74] para as comunicações com os serviços SaaS externos, expondo para o *Orchestrator* a união das funcionalidades que os *Integration Service Drivers* permitem invocar nos serviços SaaS;
- ***Integration Service Drivers***: gama de componentes responsáveis pelo acesso às funcionalidades das APIs dos serviços SaaS, numa perspectiva de utilização dessas funcionalidades pelos utilizadores da plataforma Hotelcracy Apps. Compreende a existência de um componente por cada serviço SaaS considerado, possuindo lógica dedicada a esse serviço. Esta lógica inclui mecanismos necessários à comunicação com a API do serviço e à conversão de mensagens entre o formato/estrutura do serviço e o formato/estrutura da plataforma (o Modelo Canónico anteriormente enunciado);
- ***Migrator***: responsável por orquestrar as tarefas do processo de migração de serviços relativas à transferência e ao tratamento de dados, i.e., tarefas de obtenção e envio de dados de e para os serviços (suportadas pelos componentes *Migration Service Drivers*) e tarefas de verificação e validação dos dados a migrar;
- ***Migration Service Drivers***: gama de componentes responsáveis pela execução de lógica para obtenção e envio de dados de e para os serviços SaaS integrados, numa perspectiva de migração de serviços. Compreende a existência de um componente por cada serviço SaaS considerado, possuindo lógica dedicada a esse serviço. Esta lógica inclui rotinas específicas ao serviço: rotina para obter o conjunto máximo possível dos dados presentes no serviço; e rotina para enviar o conjunto máximo de dados possível para o serviço. Inclui ainda lógica para comunicação com a API do serviço, e lógica para conversão das mensagens entre o formato/estrutura do serviço e o formato/estrutura da plataforma;

- ***Business Data Manager***: componente responsável por armazenar os dados de negócio dos hotéis, tratados pela plataforma Hotelcracy Apps. É suportado numa base de dados cuja estrutura segue a definida pelo modelo canónico especificado;
- ***Identity & Access Manager***: responsável pela implementação dos mecanismos de autenticação e autorização da plataforma, de modo a permitir o acesso autenticado/autorizado e não-repudiável às funcionalidades desta;
- ***Subscriptions Repository***: responsável por armazenar informação sobre as subscrições de serviços SaaS dos utilizadores da plataforma, tais como as condições acordadas para a subscrição do serviço (preço, período de validade, entre outros) e as credenciais de acesso à API;
- ***Accounting & Billing***: responsável por contabilizar a utilização da plataforma por parte do utilizador, para efeitos de contabilidade e facturação;
- ***Control & Audit***: responsável por armazenar *logs* das operações dos utilizadores e de estado da plataforma.

5.3.4. Riscos e fragilidades da proposta arquitectural

Como referido, a especificação da arquitectura da plataforma seguiu a metodologia ATAM. As últimas fases desta metodologia incidem na avaliação da abordagem arquitectural definida, com foco na identificação de riscos e fragilidades que advêm das decisões tomadas.

Em virtude dos contornos do problema que a plataforma Hotelcracy Apps pretende abordar, parte dos riscos e fragilidades identificados são uma consequência inevitável de integrar sistemas externos não controlados pela equipa de projecto, e podem não se traduzir num impacto ao nível arquitectural. Ainda assim, considerou-se relevante que estes fossem identificados e incluídos no documento de arquitectura da plataforma.

Os riscos e fragilidades identificados são apresentados seguidamente, podendo também ser consultados na secção 8 do anexo HC_A3-Arquitectura_Plataforma.

5.3.4.1. Riscos

Foram identificados os seguintes riscos na abordagem arquitectural proposta:

- A execução de determinadas operações envolverá comunicações entre vários componentes e/ou o acesso a serviços SaaS externos, o que pode conduzir a latências indesejadas;
- A comunicação com os serviços SaaS externos é feita através das APIs que estes disponibilizam. Esta dependência leva a que seja necessária uma constante monitorização destas APIs para se detectar a ocorrência de evoluções, e agir em conformidade;
- Se existir a evolução da API de um serviço SaaS externo, será necessário implementar alterações aos componentes *Service Driver* (*Integration* e *Migration*) em uso para esse serviço, ou a implementação de novos componentes destes.

5.3.4.2. Fragilidades

Foram identificadas as seguintes fragilidades na abordagem arquitectural proposta:

- A disponibilização de funcionalidades dos serviços SaaS externos na plataforma Hotelcracy Apps está condicionada pelas limitações das APIs dos próprios serviços. Tal implica que a plataforma pode não disponibilizar todas as funcionalidades existentes na interface com o utilizador nativa de cada serviço SaaS externo;

- A delegação do desenvolvimento dos componentes *Service Driver* para os parceiros externos implica que estes obtenham conhecimento relativo ao funcionamento interno da plataforma, ao seu modelo de dados, e ao modo de acesso a dados armazenados na plataforma;
- A presente arquitectura não possui mecanismos sistemáticos que evitem o uso dos serviços SaaS externos através da sua interface com o utilizador nativa, existindo momentos em que os dados internos à plataforma possam não estar actualizados para a sua versão mais recente.

Capítulo 6
Modelo Canónico de dados

Neste capítulo é apresentado o Modelo Canónico de dados, o qual especifica as estruturas de dados de suporte ao negócio com que a plataforma Hotelcracy Apps lida. É feita uma descrição do modelo e dos artefactos que o representam, é enunciado o processo aplicado na sua criação e especificação, e são apresentados exemplos dos artefactos referidos.

6.1. Descrição do Modelo Canónico

O Modelo Canónico surge da decisão de se utilizar um formato interno para as comunicações entre os componentes de software da plataforma Hotelcracy Apps, proporcionando também um meio de resposta à disparidade entre as APIs dos serviços SaaS que se pretende integrar (decisão arquitectural 3, na secção 5.3.2.2). O propósito deste modelo é, assim, o de funcionar como formato intermédio, para o qual e do qual os dados de negócio dos hotéis (que fazem uso da plataforma) serão convertidos. Ao passar os dados recebidos dos serviços para o Modelo Canónico, é possível fazer o seu tratamento de modo independente dos serviços subscritos pelo hotel. Este modelo permite ainda que dados de um mesmo âmbito (por exemplo, dados de um hóspede) sejam partilhados entre serviços SaaS que deles façam uso, sem necessidade de se estabelecerem relações directas entre as estruturas de dados especificadas por cada serviço.

Uma vez que este modelo canónico será a base para a organização e estruturação, na plataforma, dos dados obtidos dos serviços SaaS externos, considerou-se que a sua definição deveria ter em conta as estruturas de dados especificadas pelas APIs destes serviços. Estas estruturas são utilizadas tanto na invocação das funcionalidades das APIs, como nas respostas que retornam. Com esta consideração em mente, foi possível desenhar um modelo no qual estas estruturas específicas encaixam e que permite uma organização lógica dos dados de negócio.

O modelo foi também definido com base em conhecimento da área de actuação de cada serviço SaaS (por exemplo, contabilidade e facturação), uma vez que a análise das estruturas especificadas pelas suas APIs não se revelou suficiente. Este conhecimento foi adquirido através de testes aos serviços, com recurso à sua interface de utilizador, e também junto de *stakeholders* do sector do alojamento. Assim, foi possível definir o modelo de acordo com a lógica de negócio do sector e com a área de actuação dos serviços considerados. Este é um aspecto relevante para a maturação do projecto, uma vez que se considera que o Modelo Canónico deve ser conhecido pelos parceiros da plataforma, de modo a facilitar o desenvolvimento de *Service Drivers* para integração dos serviços destes.

Com base no conhecimento obtido a partir da interacção com os serviços SaaS externos e dos *stakeholders* do projecto, foi possível identificar várias entidades de negócio, as quais compõem o Modelo Canónico. Estas entidades representam conjuntos de atributos e estruturam os dados que a plataforma vai tratar. No contexto do sector do alojamento, como exemplo destas entidades temos: clientes, quartos dos hotéis, planos de tarifa aplicados, reservas realizadas, facturas emitidas, entre outros. Como exemplos de atributos, para o caso da entidade que representa o cliente de um hotel, temos: nome, e-mail, número de contribuinte, entre outros.

O modelo desenhado do contexto do estágio foi baseado em serviços SaaS de facturação e de *channel management*, contendo as entidades que estes serviços consideram na sua utilização. Uma

vez que outras categorias de serviços serão posteriormente integradas na plataforma, o Modelo Canónico tem que permitir a sua própria evolução, de modo a acomodar novas entidades que não estejam ainda contempladas.

Apesar de o Modelo Canónico ser único, divide-se em três vertentes interdependentes: diagrama *Entity-Relationship* (ER); conversores de dados; e *schemas* (esquemas) de dados para invocação de funcionalidades. Estas vertentes são explicadas de seguida.

6.1.1. Diagrama *Entity-Relationship*

Um diagrama *Entity-Relationship* (ER) é um tipo de diagrama estrutural, usado habitualmente no desenho de bases de dados. Este ilustra as entidades presentes no contexto do sistema, e o modo como estas se relacionam entre si [75]. O diagrama pode reflectir um de três modelos de detalhe: conceptual, lógico e físico [76].

- **Modelo conceptual:** Modelo de mais alto-nível, que ilustra de modo pouco detalhado as várias entidades do âmbito do sistema e as relações entre si.
- **Modelo lógico:** Modelo que ilustra em detalhe as entidades e as relações entre estas, representando os requisitos de negócio e as estruturas de dados necessárias ao sistema. Este nível deve ser independente da tecnologia a aplicar, servindo de base para o modelo físico.
- **Modelo físico:** Este modelo representa visualmente o esquema utilizado para a estruturação física dos dados, e é específico em relação à tecnologia utilizada para esse fim. No caso de uma base de dados, este modelo representará as tabelas que a constituem e as colunas e tipos de dados a utilizar.

Para o projecto, considerou-se relevante organizar as entidades de negócio identificadas com recurso a um diagrama deste tipo, apresentando informação dos atributos de cada entidade. Para tal, foi construído um diagrama que segue as directrizes do modelo lógico, presente no anexo HC_A7-Modelo_Canonico_ER.

6.1.2. *Schemas* de dados para invocação de funcionalidades

Um *schema* é um esquema/representação de um formato de dados, segundo um conjunto de restrições quanto à sua estrutura e ao conteúdo que permite, e que é interpretável por computadores para construção e validação de mensagens [77]. Este pode ser construído com recurso a tecnologias como o XML Schema [78] ou o JSON Schema [79], para utilização com mensagens XML e JSON, respectivamente.

Para a plataforma Hotelcracy Apps considerou-se a utilização de uma tecnologia deste tipo, como resposta às disparidades das APIs dos serviços (identificada na secção 5.2.2.3 – Dificuldade 3), e para permitir um tratamento homogéneo dos dados trocados com estes. Para se compreender a aplicação deste conceito na plataforma, é necessário ter em atenção os seguintes aspectos:

- A plataforma Hotelcracy Apps disponibiliza um conjunto de funcionalidades, que se traduzem em invocações de funcionalidades de APIs de serviços SaaS;
- Cada funcionalidade de cada API (dos serviços SaaS) define um conjunto de dados obrigatórios para ser invocada, que pode ser diferente entre APIs, mesmo quando a funcionalidade é semelhante.

Assim, a definição dos *schemas* contempla as seguintes características:

- Para cada funcionalidade que a plataforma disponibiliza é construído um conjunto de *schemas* semelhantes, um por cada serviço SaaS em que a funcionalidade pode ser invocada;
- Os *schemas* definem estruturas de dados baseadas nas entidades contempladas no Modelo Canónico, e nos seus atributos;
- Cada *schema* identifica quais os atributos da estrutura que são obrigatórios para se poder realizar a invocação no serviço a que corresponde.

Deste modo, é possível construir e validar os dados a serem comunicados a cada serviço SaaS, os quais serão depois convertidos para o formato específico do serviço a que se destinam.

Outra vantagem da utilização dos *schemas* é a capacidade de permitirem a construção dinâmica de formulários para inserção de dados pelos utilizadores na plataforma. Através da utilização de bibliotecas de construção de interfaces gráficas (como a react-jsonschema-form [80]), é possível interpretar o *schema* definido, e ajustar o formulário a apresentar em função do portfólio de serviços que o utilizador possui. Como exemplo desta utilização, é possível destacar os campos obrigatórios para invocar a funcionalidade em questão num ou mais dos serviços subscritos.

6.1.3. Conversores de dados

Os conversores de dados são constituintes do código da própria plataforma. Estes encontram-se nos componentes *Service Driver* (*Integration* e *Migration*), programados em classes designadas por *Converters*. Estabelecem o mapeamento entre atributos de entidades do Modelo Canónico e entidades dos serviços SaaS, e permitem a conversão de dados entre os dois formatos, aplicando transformações quando necessário.

6.2. Processo de criação do Modelo Canónico

A criação do Modelo Canónico passou pela identificação e especificação das entidades que o constituem, realizada com recurso a um processo *ad-hoc*, que consistiu em:

1. Identificação de entidades de negócio do sector;
2. Para cada entidade, análise dos serviços SaaS ao nível das funcionalidades que com ela lidavam, para identificação dos atributos e lógica associados;
3. Mapeamento de atributos entre as estruturas que representavam a entidade em cada serviço;
4. Especificação dos atributos que a entidade possuiria no âmbito do Modelo Canónico, baseados no mapeamento realizado no passo 3.

Este era um processo iterativo, com as entidades de negócio a serem identificadas à medida que o projecto avançava e novas funcionalidades eram consideradas para inclusão na plataforma.

Como referido anteriormente, a identificação de entidades para o Modelo Canónico foi realizada com base no conhecimento obtido dos serviços SaaS considerados e dos *stakeholders* do sector do alojamento. Para a identificação dos atributos que cada entidade possui em cada serviço, foi analisada a documentação dos serviços considerados e foram realizados testes com recurso à API e à interface gráfica que disponibilizam.

Para o passo 3 e para o passo 4 foram utilizadas tabelas de mapeamento de atributos, como exemplificado na Tabela 9. Nesta, são mapeados os atributos da entidade 'Customer', onde as colunas 'InvoiceXpress' e 'Invoice Ocean' (serviços de facturação) possuem os atributos que

compõem a estrutura que cada serviço considera para esta entidade. Atributos que estejam na mesma linha da tabela permitem estabelecer uma relação entre si por representarem os mesmos dados ou dados semelhantes, por exemplo, os atributos 'name' e 'phone'. A última coluna contém os atributos definidos para a entidade no âmbito do Modelo Canónico, e que se relacionam com os atributos que os serviços SaaS consideram, e a primeira coluna ajuda a identificar o âmbito dos atributos. Algumas linhas possuem células em branco para os casos em que não se identificaram correspondências entre atributos.

Para além desta especificação, estas tabelas permitiram também definir uma base de conhecimento que foi depois aplicada na implementação dos conversores de dados utilizados na plataforma.

	InvoiceXpress	**Invoice Ocean**	**Modelo Canónico**
Identificação	code		organizational_id
		company	is_company
	name	name	name
		first_name	
		last_name	
		shortcut	
	fiscal_id	tax_no	fiscal_id
Contacto	email	email	email
	website	www	website
	phone	phone	phone
		mobile_phone	mobile_phone
	fax	fax	fax
		person	contact_person
Localização	address	street	street
		street_no	
	postal_code	post_code	postal_code
	city	city	city
	country	country	country

Tabela 9 - Tabela de mapeamento de atributos para a entidade 'Customer'

6.3. Resultados

Nesta secção descrevem-se os resultados obtidos em relação à especificação do Modelo Canónico. São apresentados excertos do diagrama ER, de um *Converter*, e de um *schema*, no seu estado à data de escrita deste relatório, e que pertencem a uma versão do modelo que compreende serviços de facturação e de *channel manager*. Estes excertos relacionam-se com a criação de um cliente (entidade 'Customer'), permitindo ver a interdependência das três vertentes do modelo.

6.3.1. Diagrama ER

Na Figura 23 é apresentado um excerto do diagrama ER desenhado, contendo as entidades que se relacionam com os dados dos clientes dos hotéis. O anexo HC_A7-Modelo_Canonico_ER apresenta o diagrama na sua totalidade, composto por 41 entidades.

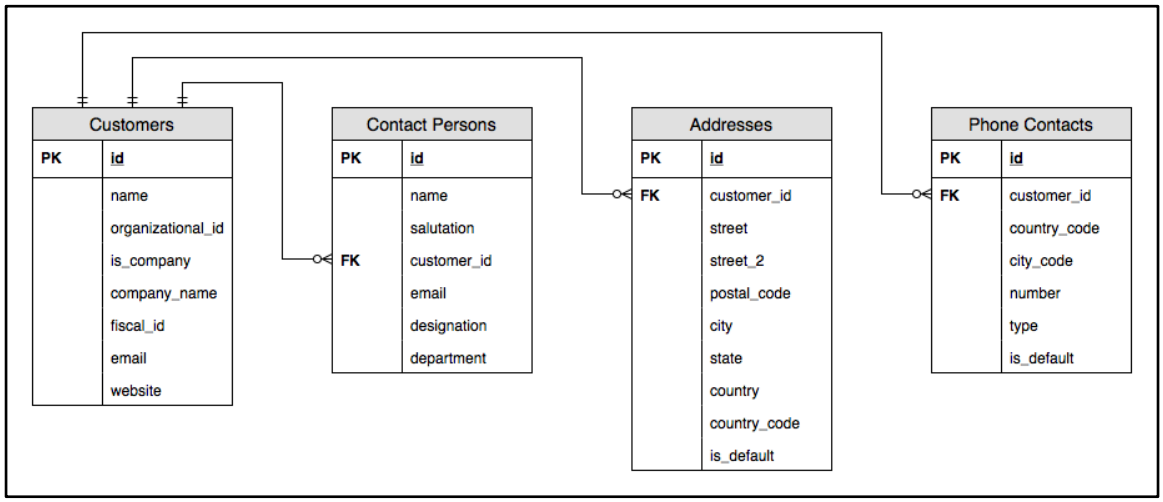

Figura 23 - Excerto do diagrama ER do Modelo Canónico, com detalhes da entidade 'Customer'

Na figura podem-se observar as tabelas que estruturam os dados na base de dados, as relações entre si, e os atributos que as compõem e que, neste caso, caracterizam um cliente. Estes atributos são os identificados nas tabelas de mapeamento referidas anteriormente, sendo possível observar que os atributos identificados na coluna 'Modelo Canónico' da Tabela 9 estão presentes na coluna 'Customers' da figura. De realçar que este excerto pertence a uma versão do Modelo Canónico que inclui já entidades e atributos usados noutros serviços que não apenas os da categoria de facturação, daí possuir mais informação do que a presente na Tabela 9.

6.3.2. Schema

No Quadro 1 é apresentado um excerto do JSON *schema* definido para a funcionalidade de criação de um cliente, específico para o serviço InvoiceXpress. Nele podem ser vistos os atributos que definem um cliente, listados na chave 'properties', e as restrições de tipo e/ou de quantidade aplicadas a cada um. De destacar os atributos 'name' e 'organizational_id' que estão marcados como obrigatórios (listados na chave 'required') caracterizando a obrigatoriedade imposta pelo serviço SaaS a que o *schema* corresponde. Ao se atentar nos nomes dos atributos, é possível, mais uma vez, estabelecer a relação com as tabelas de mapeamento de atributos, neste caso com a coluna 'InvoiceXpress' e a coluna 'Modelo Canónico' da Tabela 9.

```
{
  "title": "Create customer @ InvoiceXpress",
  "type": "object",
  "properties": {
    "addresses": {
      "items": {
        "$ref": "#/definitions/address"
      },
      "type": "array",
      "minItems": 0,
      "maxItems": 1
    },
    "company_name": {
      "type": "string"
    },
    "contact_persons": {
```

```
      "items": {
        "$ref": "#/definitions/contact_person"
      },
      "type": "array",
      "minItems": 0,
      "maxItems": 1
    },
    "email": {
      "format": "email",
      "type": "string"
    },
    "fiscal_id": {
      "type": "string"
    },
    "is_company": {
      "type": "boolean"
    },
    "name": {
      "type": "string"
    },
    "organizational_id": {
      "type": "string"
    },
    "phone_contacts": {
      "items": {
        "$ref": "#/definitions/phone_contact"
      },
      "type": "array",
      "minItems": 0,
      "maxItems": 1
    },
    "website": {
      "format": "uri",
      "type": "string"
    }
  },
  "required": [
    "name",
    "organizational_id"
  ],
  "definitions": {...}
}
```

Quadro 1 - Excerto do schema definido para a funcionalidade de criação da entidade 'Customer'

6.3.3. Converter

No Quadro 2 é apresentado um excerto de um *Converter* implementado para conversão de dados de um cliente. É possível ver as funções para converter do formato do serviço para o Modelo Canónico (função 'from_service') e para o inverso (função 'from_hotelcracy'), e que estabelecem uma relação entre os atributos considerados para o Modelo Canónico, e os atributos que o serviço SaaS considera (neste caso, o serviço InvoiceXpress). O resultado de cada função é uma estrutura de dados que pode ser interpretada pelos componentes da plataforma no caso da primeira, e uma estrutura que pode ser interpretada pelo serviço SaaS no caso da segunda função. Os nomes dos atributos são, mais uma vez, relacionáveis com os identificados na tabela de mapeamento (Tabela 9).

```
class CustomerConverter
  def self.from_service(client)
    client = ConverterHelpers.clean_empties client

    {
      external_id: client[:id],
      name: client[:name],
      organizational_id: client[:code],
      fiscal_id: client[:fiscal_id],
      is_company: false,
      company_name: nil,
      email: client[:email],
      website: client[:website]
    }
  end

  def self.from_hotelcracy(customer, address: nil, contact_person: nil,
phone_contacts: {})
    c = {
      name: customer[:name],
      code: customer[:organizational_id],
      email: customer[:email],
      fiscal_id: customer[:fiscal_id],
      website: customer[:website]
    }

    c.merge! PhoneContactConverter.from_hotelcracy(phone_contacts[:phone],
:phone)
    c.merge! PhoneContactConverter.from_hotelcracy(phone_contacts[:fax],
:fax)
    c.merge! ContactPersonConverter.from_hotelcracy(contact_person)
    c.merge! AddressConverter.from_hotelcracy(address)
  end
end
```

Quadro 2 - Excerto de código de um Converter da entidade 'Customer'

6.4. Conclusões

A utilização de um formato intermédio permitiu definir um conjunto de estruturas de dados negócio única a utilizar pela plataforma, e assim lidar com as disparidades encontradas nas APIs dos serviços SaaS a integrar. A utilização de tabelas de mapeamento de atributos serviu de base para a construção dessas estruturas, traduzindo-se no diagrama ER. A utilização de *schemas* permitiu associar parte das especificidades dos serviços SaaS às estruturas definidas, principalmente em relação à obrigatoriedade dos atributos, e os Converters permitiram a tradução de estruturas entre a plataforma e os serviços SaaS, lidando em específico com o mapeamento de atributos entre formatos.

Como contrapartida, a utilização deste modelo implica que sejam aplicadas alterações nas suas três vertentes sempre que é necessário realizar uma evolução das estruturas para incluir uma nova entidade, alterar alguma entidade ou algum atributo.

Capítulo 7
Desenvolvimento da prova de conceito

O desenvolvimento da prova de conceito compreendeu um conjunto de aspectos que se considera pertinente serem incluídos no presente relatório, uma vez que tiveram impacto no estágio a vários níveis, tais como a sua duração e os conhecimentos e boas práticas adquiridos. Esses aspectos são abordados neste capítulo, sendo apresentados os processos e técnicas aplicados no desenvolvimento e a preparação e estudos de tecnologias realizados. São ainda apresentados detalhes dos componentes desenvolvidos, decisões de desenvolvimento tomadas, os ambientes usados e os testes realizados.

7.1. Processos e técnicas aplicados no desenvolvimento

Para o desenvolvimento da prova de conceito foi aplicado um processo de gestão do ciclo de vida do código produzido, sugerido pelo parceiro líder para o desenvolvimento da plataforma Hotelcracy Apps, e também a técnica de desenvolvimento *Test-Driven Development* (TDD). Estes são apresentados de seguida.

7.1.1. Gestão do ciclo de vida do código produzido

O desenvolvimento de uma plataforma da dimensão da Hotelcracy Apps implica que este seja realizado por uma equipa de vários elementos, de modo a que possa ser concluída no prazo definido. A gestão do código desenvolvido por várias pessoas pode tornar-se uma tarefa complicada, uma vez que existe a necessidade de todo o código desenvolvido funcionar em conjunto, e por surgirem situações em que existe sobreposição, e consequentemente conflito, de linhas ou ficheiros de código. Associado a isto existe ainda a necessidade de todo o código, ou pelo menos parte dele, ter que ser partilhado ou acedido por várias pessoas, de modo a que o desenvolvimento seja possível, devendo este residir num local bem definido e acessível por todos. Se não existir um processo minimamente definido que permita governar ou realizar esta gestão, o desenvolvimento pode tornar-se difícil, moroso, ou até mesmo impossível de terminar ou de alcançar um estado estável e aceitável.

Uma vez que as equipas de desenvolvimento do projecto eram compostas por vários elementos, foi aplicado um processo que permitia a gestão do ciclo de vida do código produzido, e que envolveu também o código produzido no âmbito do estágio. O processo baseava-se na utilização de um repositório na plataforma GitHub [81], um serviço que permite o armazenamento, versionamento e revisão do código produzido num projecto de software, e com o qual é possível comunicar através da ferramenta Git [82]. Esta ferramenta permite o uso do conceito de *branches* (ramos), onde é definido um *branch* principal (denominado de *master*) com o código estável, e a criação de novos *branches* a partir desse ou de outros, para desenvolvimento isolado de partes do software, os quais residem tanto no repositório remoto como localmente no computador do programador. De modo a facilitar a gestão, o processo exige que cada funcionalidade a desenvolver (ou unidade de trabalho a realizar) possua o seu próprio *branch*. O programador deve depois enviar o código para o repositório, em específico para o *branch* correspondente.

O processo exige ainda que o código produzido seja revisto por outros elementos das equipas, antes de ser fundido no *branch master*. Assim, o autor original do código deve indicar ao resto dos elementos que existe uma nova funcionalidade desenvolvida e a necessitar de revisão. A

plataforma GitHub disponibiliza ferramentas de suporte a esta revisão, permitindo que outros elementos deixem comentários associados a linhas de código específicas, ou deixem avisos para o autor. Se forem detectados erros, ou melhorias necessárias, o programador pode realizar alterações e submete-las para o mesmo *branch*, para nova revisão. A realização destas revisões é uma mais valia para a qualidade do *software*, permitindo a identificação de erros numa fase ainda recente do desenvolvimento, aumentando assim a qualidade do código. Outro aspecto positivo é a partilha de conhecimento que se gera, e também o aumento e maturidade do conhecimento de todos os intervenientes, já que todos têm a ganhar com esta partilha.

A plataforma GitHub permite ainda a integração com a plataforma CircleCI [83] para automação dos testes implementados, executando-os num ambiente configurável pela equipa. O resultado dessa execução é depois comunicado ao GitHub, identificando os casos em que algum dos testes falhou.

Para se proceder à fusão do código produzido no *branch master* era necessário que todos os testes executados passassem, e que a revisão realizada pela equipa obtivesse a aprovação de pelo menos dois elementos.

7.1.2. Test-Driven Development

Um dos modos de atestar a qualidade do código desenvolvido no âmbito de um projecto de software é através da implementação de testes automáticos ao mesmo. Estes permitem avaliar se a execução do código desenvolvido produz os resultados esperados, e permite detectar casos em que tal não ocorre. Outro aspecto positivo está relacionado com a realização de alterações ao código previamente produzido, pois permite detectar casos em que tenham sido introduzidas falhas ou cometidos erros nas funcionalidades a que esse código corresponde, e actuar sobre eles.

Habitualmente, em projectos de software, a fase de testes é realizada após a fase de implementação (como no Modelo *Waterfall* ilustrado na Figura 6, na secção 3.2), e pode envolver o desenvolvimento de testes a vários níveis (por exemplo, testes unitários, de integração, ou de sistema) e de vários tipos (por exemplo, testes de regressão, de aceitação, ou de usabilidade). Com a aplicação da técnica de desenvolvimento *Test-Driven Development* (TDD), a fase de testes é realizada ao mesmo tempo que a fase de implementação das funcionalidades do software, com foco nos testes de mais baixo nível, como os unitários e de integração. Testes de mais alto nível, como os de sistema mantêm-se, habitualmente, numa fase posterior à da implementação, uma vez que abrangem o software num todo, em vez de apenas partes específicas deste.

Como referido, TDD é uma técnica de desenvolvimento de software, e que implica que a implementação de uma determinada funcionalidade deve ser iniciada com a especificação e desenvolvimento dos testes aos quais ela deve dar resposta, e só depois passar ao seu desenvolvimento. Esta técnica é abordada em detalhe por Kent Beck no livro "Test-Driven Development by Example" [84], afirmando o autor que o seu papel foi a "redescoberta" desta técnica e não a sua criação. Este refere que a descrição original do TDD está presente no livro de D. D. McCracken, "Digital Computer Programming" [84], de 1957, onde a técnica é identificada como "Program Checkout" [85].

TDD, essencialmente, inverte a sequência habitual associada à programação de uma funcionalidade, em que primeiro é implementada a funcionalidade e só depois os testes. A implementação de testes antes da implementação do código que os vai satisfazer oferece um conjunto de vantagens ao programador: permite que este trabalhe com confiança, uma vez que os testes existentes validam o código produzido, e permitem detectar erros ou falhas

introduzidas por alterações; o programador pode trabalhar numa série de passos alcançáveis, em vez de resolver um problema maior todo de uma só vez; ajuda a assegurar que o design do software é adequado e claro, focando-se na criação de operações que são invocáveis e testáveis; e deixa para trás uma suite de testes que ajuda a preservar a integridade do código ao longo do processo de desenvolvimento [86], [87].

Esta técnica assenta em ciclos de desenvolvimento curtos e repetitivos, compostos por três fases – *red*, *green*, e *refactor* – e encontra-se ilustrado na Figura 24 [88]. A fase *red* corresponde ao desenvolvimento dos testes da funcionalidade a implementar e à sua execução, com a finalidade de assegurar que os testes falham (daí a associação à cor vermelha (*red*)). A fase *green* corresponde ao desenvolvimento da funcionalidade em questão, produzindo o código mínimo necessário para que os testes criados na fase anterior passem (daí a associação à cor verde (*green*)). A fase *refactor* corresponde à melhoria do código produzido através da remoção de redundâncias, sendo garantido pelos testes produzidos anteriormente que não são introduzidos erros com as alterações realizadas.

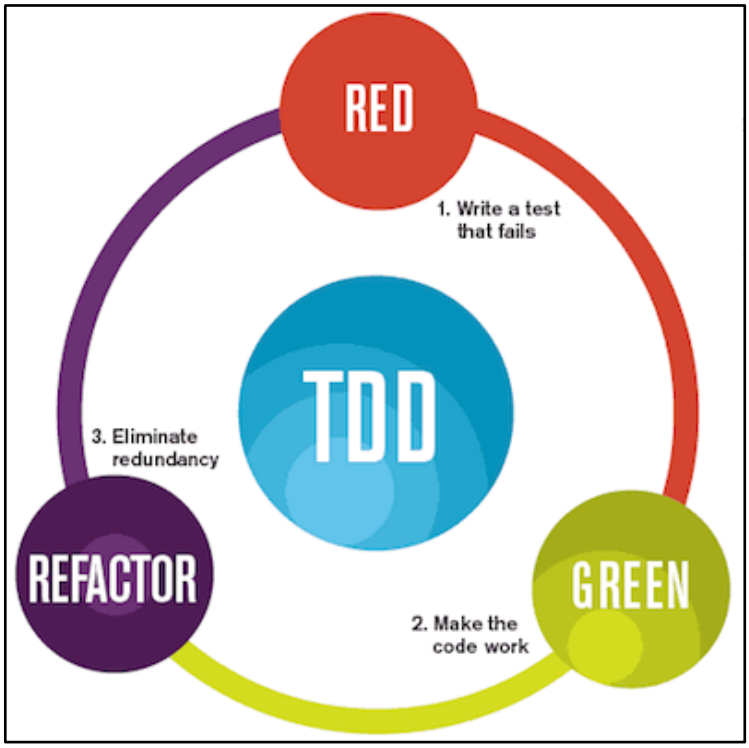

Figura 24 - Ciclo de vida da técnica Test-Driven Development [89]

7.2. Preparação para o desenvolvimento

O desenvolvimento do *back-end* da plataforma e dos seus componentes, como indicado na restrição técnica RT_01 (no Capítulo 5, secção 5.3.1.3.1), foi realizado com recurso à *framework* Ruby on Rails. Uma vez que o estagiário nunca tinha contactado com esta *framework* nem com a linguagem Ruby em que esta assenta, houve a necessidade de realizar uma fase de preparação, com um estudo teórico e prático destas tecnologias.

A preparação iniciou-se com o estudo da linguagem de programação Ruby [90], uma linguagem *open-source* e orientada a objectos [91]. Para tal, realizaram-se os cursos "Ruby Primer" [92] e "Ruby Primer: Ascent" [93], disponibilizados na plataforma de ensino interactivo Ruby Monk. O primeiro é um curso introdutório, onde se abordam aspectos básicos da linguagem, tais como a sua sintaxe, as estruturas de dados básicas, e os fluxos de

controlo, e aspectos da programação orientada a objectos. O segundo é um curso de nível intermédio, onde se abordam aspectos mais avançados da linguagem, tais como os conceitos de herança e de encapsulamento associados à programação orientada a objectos, o tratamento de excepções, e a manipulação de estruturas de dados.

Adquiridos os conhecimentos da linguagem Ruby, passou-se ao estudo da *framework* Ruby on Rails [94], uma *framework open-source* de desenvolvimento de aplicações web. O estudo iniciou-se com a leitura do artigo "The Rails Doctrine" [95], por David Heinemeier Hansson, criador da *framework*, onde é apresentada a doutrina que esta defende e os vários pilares em que se baseia. O estudo prático consistiu na leitura do livro "The Ruby on Rails Tutorial" [96], por Michael Hartl, e na resolução dos exercícios nele propostos. O livro aborda vários aspectos da *framework*, guiando o leitor no desenvolvimento incremental de uma aplicação web de exemplo.

Para a implementação de testes nos componentes foi escolhida a ferramenta RSpec [97], uma *framework* de desenvolvimento de testes orientada para o comportamento esperado das funcionalidades a implementar (técnica habitualmente denominada de *Behavior-Driven Development* [98]) e que simplifica a prática de TDD. Esta disponibiliza uma DSL (*Domain-Specific Language*) que permite a escrita de testes que se descrevem a si mesmos, e numa sintaxe compreensível por humanos. No Quadro 3 são apresentados dois testes de exemplo produzidos com esta ferramenta, relativos à validade de uma instância do modelo 'Service', com os títulos dos testes a indicarem que a instância não é válida se não possuir um nome ou se não possuir uma categoria. Para aquisição de conhecimentos nesta ferramenta foi lido o livro "Everyday Rails Testing with RSpec" de Aaron Sumner [99], que propõe uma abordagem prática à aplicação de TDD no desenvolvimento com Ruby on Rails e RSpec, e realizados os exercícios aí propostos.

```
RSpec.describe Service, type: :model do
  it "is not valid with empty name" do
    expect(build(:service, name: nil)).not_to be_valid
  end

  it "is not valid without a category" do
    expect(build(:service, category: nil)).not_to be_valid
  end
end
```

Quadro 3 - Exemplo de testes produzidos com recurso à ferramenta RSpec

A preparação incidiu ainda na aprendizagem de ferramentas de gestão associadas à linguagem Ruby e às bibliotecas que se podem utilizar para desenvolvimento no seu âmbito, denominadas de "gems". Uma das ferramentas foi o Bundler [100], um gestor de colecções de *gems* utilizadas em projectos Ruby, que disponibiliza funcionalidades para rastreio e instalação dessas colecções e controlo das versões instaladas. Outra ferramenta foi o RVM (Ruby enVironment Manager ou Ruby Version Manager) [101], um gestor de versões da linguagem Ruby, o qual permite configurar ambientes de execução isolados e autocontidos, em função da versão da linguagem em uso. Uma terceira ferramenta foi o Gemset [102], o qual está integrado no RVM, e que, em associação com os ambientes contemplados neste último, permite configurar ambientes de execução em função das colecções de *gems* instaladas. A utilização destas ferramentas tornou-se fulcral, uma vez que cada componente da plataforma era desenvolvido como um projecto isolado de Ruby on Rails, contemplando diferentes colecções de *gems*.

7.3. Componentes abordados no desenvolvimento

Nesta secção enunciam-se alguns dos componentes abordados no desenvolvimento, em específico os envolvidos na migração de serviços. São eles o *Services Catalogue*, o *Orchestrator* e o *Migrator*. Estes são classificados na candidatura ao PT2020 como componentes complexos, uma vez que implementam a execução dos principais fluxos de trabalho da plataforma (neste caso, a subscrição, migração e cancelamento de serviços) e operacionalizam a inclusão de novos serviços SaaS.

A informação aqui apresentada é um resumo da especificação detalhada realizada para cada componente, a qual foi registada em documentos técnicos do projecto. Cada documento inclui informação relativa a entidades de negócio do componente, funcionalidades que este disponibiliza, tecnologias aplicadas, contexto de execução, arquitectura interna, e especificação das suas interfaces. Estes documentos são anexos do presente relatório, e podem ser consultados para se obter mais detalhes de cada componente.

Nesta secção é apresentado um resumo dessa especificação detalhada para os componentes enunciados, com foco no papel do componente, nas suas entidades de negócio, no contexto de funcionamento e de interacção com outros componentes da plataforma.

7.3.1. Services Catalogue

O componente *Services Catalogue* tem como objectivo funcionar como um repositório de dados relativos aos serviços SaaS externos disponíveis para utilização de modo integrado na plataforma Hotelcracy Apps, e fornecer funcionalidades que permitam a gestão e acesso aos mesmos pelos restantes componentes da plataforma. Estes dados serão de dois foros: de foro comercial, cuja finalidade é serem apresentados aos utilizadores da plataforma, nomeadamente no *Marketplace*; e de foro técnico, cuja finalidade é serem interpretados pelos restantes componentes da plataforma, de modo a aplicarem-nos à sua própria execução.

No foro comercial, os dados relativos aos serviços disponibilizados compreendem a sua organização em função da categoria a que pertencem e, para cada serviço, o conjunto de informação relevante para a sua caracterização. Esta inclui o nome e descrição do serviço, custos associados, imagens, funcionalidades acessíveis através da plataforma, integrações com outros serviços, avaliações submetidas pelos utilizadores, entre outros. No foro técnico, também para cada serviço, compreende estruturas de dados que são compreensíveis pelos restantes componentes para operacionalizarem a subscrição, utilização, migração e cancelamento do serviço. Inclui listagens de entidades de negócio e de funcionalidades do âmbito do serviço, *schemas* que caracterizam a invocação das funcionalidades (referidos no capítulo do Modelo Canónico), entre outros.

Ao nível das interacções entre componentes, o *Services Catalogue* fornece ao *User Interface* informação para construção dos ecrãs a apresentar ao utilizador. Este último faz uso de dados do foro comercial para construção do *Marketplace*, e uso de dados do foro técnico para construção dos ecrãs de gestão dos serviços e para construção dos formulários de introdução de dados. Aos componentes *Orchestrator* e *Migrator*, o *Services Catalogue* fornece informação de foro técnico para validação de dados a submeter para os serviços e, em específico para o *Migrator*, para execução do processo de migração.

A especificação inicial deste componente esteve sobre responsabilidade do estagiário, sendo o desenvolvimento realizado pelo próprio e por outros elementos da equipa de projecto. No âmbito do estágio, o desenvolvimento neste componente incidiu na implementação de funcionalidades para acesso a detalhes dos serviços SaaS, em específico, listagens de entidades

a importar ou a exportar (entidades de negócio do sector hoteleiro, como, por exemplo, clientes, quartos e tarifas) e *schemas* de estruturas de dados, utilizadas na validação dos dados a exportar.

Mais detalhes acerca deste componente podem ser consultados no documento e especificação detalhada correspondente, o anexo HC_A4-Services_Catalogue.

7.3.2. Orchestrator

O componente *Orchestrator* tem como objectivo gerir a execução de processos complexos na plataforma. Estes envolvem a comunicação com vários componentes do sistema para o correcto processamento do pedido, sendo responsabilidade do *Orchestrator* a orquestração destas interacções, tornando-o um componente central da plataforma.

O *Orchestrator* interage com os vários componentes do sistema para execução tanto de lógica de negócio associada ao sector hoteleiro, como de lógica de negócio da própria plataforma. A primeira implica a recepção de pedidos dos utilizadores da plataforma (com origem no componente *User Interface*) a aplicação de validações aos pedidos, e a comunicação destes aos serviços SaaS externos para invocação das suas funcionalidades (realizada através do *Integration Broker* e respectivos *Integration Service Drivers*). A segunda implica a interacção com componentes de suporte da plataforma, para realização de acções de armazenamento de dados (com o *Business Data Manager*), contabilização e *logging* da utilização da plataforma (com o *Accounting & Billing* e o *Control & Audit*), entre outras.

A especificação inicial deste componente esteve sobre responsabilidade do estagiário, sendo o desenvolvimento realizado pelo próprio e por outros elementos da equipa de projecto. No âmbito do estágio, o desenvolvimento neste componente incidiu na implementação do processo de migração de serviços SaaS e de funcionalidades de suporte à execução deste. Incluiu funcionalidades para:

- Submissão e tratamento de um pedido de nova migração, envolvendo a invocação de mecanismos já desenvolvidos para a subscrição de serviços;
- Obter informação do estado da migração e de acções necessárias realizar pelo utilizador nesta;
- Submissão de dados em falta, identificados através de validações dos dados a exportar para o serviço novo (esta validação é uma acção realizada pelo *Migrator*, e não pelo *Orchestrator*);
- Alteração do estado da migração das subscrições dos serviços em questão.

Mais detalhes acerca deste componente podem ser consultados no documento e especificação detalhada correspondente, o anexo HC_A5-Orchestrator.

7.3.3. *Migrator* e *Migration Service Drivers*

O componente *Migrator* tem como objectivo orquestrar processos de migração de dados aquando da migração de serviços SaaS de uma mesma categoria, e fornecer funcionalidades para iniciar, actualizar e obter informação dessa migração de dados. A migração de serviços envolve um serviço SaaS de origem e um de destino, e o processo de migração em si envolve a subscrição do serviço SaaS de destino, a migração de dados do serviço de origem para o de destino, e o posterior cancelamento do serviço de origem. O componente *Orchestrator* é o responsável pela orquestração deste processo de migração de serviços, delegando para o componente *Migrator* a orquestração das operações relativas à migração de dados, também elas organizadas num processo, o qual está ilustrado na Figura 25.

Resumidamente, a migração de dados passa por: 1) obter os dados armazenados no serviço SaaS de origem; 2) actualizar a base de dados da plataforma com a informação obtida; 3) verificar quais os dados em falta, para que possa ser feita a exportação para o serviço SaaS de destino; 4) quando necessário, pedir ao utilizador os dados em falta, e aplicá-los; 5) exportar os dados para o serviço SaaS de destino; e 6) actualizar a base de dados da plataforma com nova informação de identificadores obtidos do serviço de destino. A necessidade de verificar os dados em falta (no ponto 3) está relacionada com a disparidade que se verificou nos serviços SaaS quanto aos dados que cada um considera como obrigatórios. A execução deste passo permite identificar os casos em que o serviço de destino da migração requer um determinado atributo que não se encontra ainda preenchido, e informar o utilizador de que deve fornecer um valor para tal.

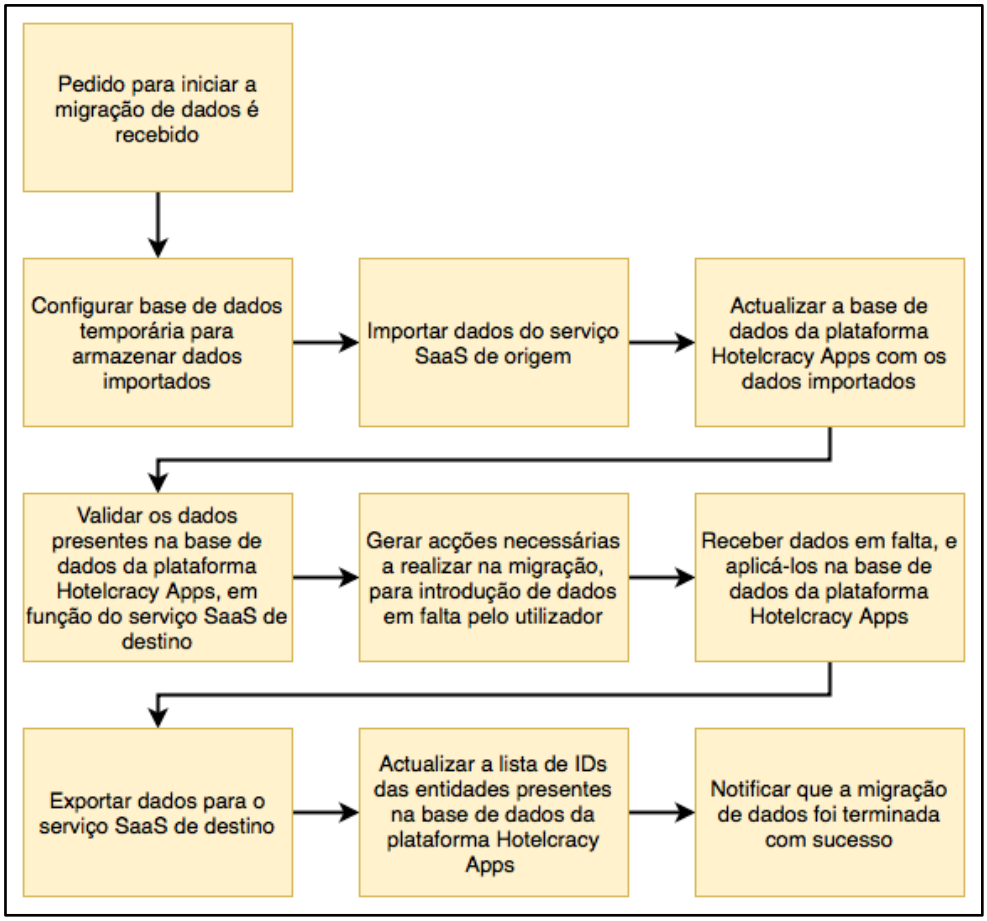

Figura 25 - Processo de migração de dados

Para a execução do processo de migração de dados o *Migrator* faz uso de outros componentes da plataforma, nomeadamente: os *Migration Service Drivers*, para que executem operações de importação e exportação de dados, de e para os serviços SaaS; o *Subscriptions Repository*, para obter credenciais de acesso às APIs dos serviços SaaS, e assim permitir aos *Migration Service Drivers* executarem as operações de importação e exportação; o *Business Data Manager*, para armazenamento, acesso e actualização de dados de negócio do sistema Hotelcracy Apps; e o *Services Catalogue*, para acesso a detalhes dos serviços SaaS, para aplicação na verificação de dados em falta. Apesar de, em termos arquitecturais, os *Migration Service Drivers* serem considerados componentes independentes, aquando do desenvolvimento estes foram implementados como parte integrante do *Migrator*. No entanto, foi tido em consideração que o código de cada *Migration Service Driver* deveria ser auto contido, de modo a permitir que seja

extraído do componente no futuro, sem necessidade de alterações significativas na estrutura de ambos.

A migração de dados pode implicar que a sua execução tenha uma duração temporal significativa, uma vez que pode envolver o tratamento de grandes quantidades de dados e a realização de várias comunicações aos serviços SaaS externos. De modo a manter a disponibilidade do componente aquando da execução das operações da migração, considerou-se que o processamento destas deveria ser realizado em *background*. Deste modo, a arquitectura interna do componente segue o padrão MVC (*Model-View-Controller*) associado à *framework* Ruby on Rails, e faz uso de uma estrutura de filas de tarefas e de *background workers*, já disponíveis de raiz na *framework*. Cada operação da migração de dados é traduzida numa tarefa, e colocada numa fila, a qual serve os *workers*. Estes são responsáveis pela execução integral das tarefas, comunicando com outros componentes da plataforma, e com os serviços externos (através dos *Migration Service Drivers*) para realização dessa execução.

Em relação à informação tratada e/ou disponibilizada pelo *Migrator*, este permite o acesso a migrações de dados em curso ou já realizadas, e de acções necessárias a serem realizadas na migração. A informação relativa à migração de dados é registada aquando do início da mesma, e a partir desta é possível saber qual a organização a que pertence, quais os serviços SaaS envolvidos, e qual o estado em que esta se encontra. A informação de acções necessárias a serem realizadas na migração é gerada através da verificação de dados em falta, a qual atesta se os dados de negócio do hotel estão completos para exportação para o serviço SaaS de destino. Estas acções necessárias indicam quais os dados em falta e são usadas para os requerer junto do utilizador. Estes dados são depois aplicados aos dados de negócio do hotel, para que a exportação possa ser realizada e a migração terminada.

A especificação inicial deste componente esteve sobre responsabilidade do estagiário, bem como a totalidade do seu desenvolvimento. Mais detalhes acerca deste componente podem ser consultados no documento e especificação detalhada correspondente, o anexo HC_A6-Migrator.

7.4. Decisões de desenvolvimento

Aquando da fase de desenvolvimento dos componentes, foram identificados alguns problemas e necessidades que não tinham sido ainda discutidos nas fases anteriores do projecto e do estágio. Por isto, de modo a que o desenvolvimento não fosse interrompido devido a esses problemas e necessidades, foi tomado um conjunto de decisões para lhes dar resposta, as quais são apresentadas nesta secção.

Política de actualização dos dados de negócio aquando da importação de dados na migração de serviços

Como referido anteriormente, o processo de migração de serviços implica a migração de dados entre os serviços SaaS em questão, a qual implica tarefas de importação dos dados de negócio presentes no serviço SaaS de origem e de actualização dos dados presentes na plataforma Hotelcracy Apps. Aquando a implementação da operação de actualização dos dados, constatou-se que não tinha ainda sido definida uma política para a sua execução. A definição desta política é fulcral, uma vez que possui um impacto grande na consistência e estabilidade dos dados, devendo conter regras que ditem quais os dados que são actualizáveis, em que condições essa actualização é permitida, e como resolver casos que não estejam ainda contemplados.

Após discussão do problema com a equipa de desenvolvimento, considerou-se que, para efeitos de desenvolvimento da prova de conceito, deveria ser apenas considerada a actualização de dados em falta na base de dados da plataforma Hotelcracy Apps, isto é, apenas são actualizados dados que possuem um valor nulo na base de dados da plataforma, e para os quais foi possível obter um valor a partir do serviço SaaS externo.

Base de dados de negócio dos hotéis em uso para a migração de serviços

A plataforma compreende na sua arquitectura um componente dedicado ao armazenamento dos dados de negócio dos hotéis que são tratados pela plataforma, o *Business Data Manager*. Apesar de este ser um componente de grande valor para o projecto, o seu desenvolvimento não foi considerado prioritário, dando-se primazia à implementação dos fluxos de negócio que permitem a interacção dos utilizadores da plataforma com as funcionalidades que os serviços SaaS disponibilizam.

Uma vez que a execução da migração de serviços implica o armazenamento destes dados de negócio localmente à plataforma, era necessário especificar e implementar uma base de dados para tal. Assim, a equipa de desenvolvimento considerou que, para efeitos de desenvolvimento da prova de conceito, essa base de dados poderia estar circunscrita ao âmbito da migração de dados, e ser acedida directamente pelos componentes que nela participam. Tal implicou a criação de um *script* para construção das estruturas da base de dados, baseado no Modelo Canónico definido, e que estaria presente no componente *Migrator*. Este *script* serviria depois de base para a definição da estrutura da base de dados a implementar no componente *Business Data Manager*, aquando do seu desenvolvimento.

7.5. Ambientes

Para o desenvolvimento da prova de conceito, foram considerados três ambientes tecnológicos: desenvolvimento, teste e *staging*. Não foi contemplado um ambiente para a fase de produção, uma vez que a passagem do projecto a esta fase não coincidia com o calendário do estágio.

7.5.1. Ambiente de desenvolvimento

O ambiente de desenvolvimento é relativo ao ambiente local em que foi realizada a implementação, mais propriamente o computador do estagiário. Este era também o ambiente utilizado para a execução de testes localmente.

Este ambiente tecnológico possuía as seguintes características:

- Sistema operativo: macOS High Sierra, versão 10.13.5
- Ferramenta de versionamento de código: Git, versão 2.15.0
- Linguagem de programação: Ruby, versão 2.4.1
- Gestor de ambientes Ruby: RVM, versão 1.29.3
- Gestor de colecções de *gems*: Bundler, versão 1.16.3
- *Framework* de desenvolvimento: Ruby on Rails, versão 5.1.4
- Base de dados: PostgreSQL, versão 10.1

7.5.2. Ambiente de teste

Para a execução dos testes implementados foram considerados dois ambientes: um local (no computador do estagiário) e que corresponde ao ambiente de desenvolvimento enunciado anteriormente; e um remoto, localizado na plataforma CircleCI. Esta é uma plataforma de *continuous integration*, uma estratégia de desenvolvimento de software que aumenta a velocidade de desenvolvimento enquanto assegura a qualidade do código [103], e que funciona de modo integrado com o repositório de código em uso. A palataforma permite que, a cada nova actualização do código presente no repositório, sejam despoletados automaticamente procedimentos para construção do projecto e execução da suite de testes. Se algum dos procedimentos falhar, a equipa de desenvolvimento é notificada, podendo actuar rapidamente sobre o erro e corrigi-lo, de modo a assegurar que o projecto se encontra num estado estável.

Este ambiente tecnológico possuía as seguintes características:

- Sistema operativo: Debian, versão 8
- Linguagem de programação: Ruby, versão 2.4.2
- Gestor de colecções de *gems*: Bundler, versão 1.16.3
- *Framework* de desenvolvimento: Ruby on Rails, versão 5.1.4
- Base de dados: PostgreSQL, versão 10.1

7.5.3. Ambiente de *staging*

O ambiente de *staging* serve de fase de transição entre o ambiente de desenvolvimento e teste e o ambiente de produção, permitindo o teste da plataforma em condições iguais às deste último. Deste modo, garante-se que as alterações realizadas em desenvolvimento não têm impacto no correcto funcionamento da plataforma em ambiente de produção.

Este ambiente tecnológico possuía as seguintes características:

- Sistema operativo: Debian GNU/Linux, versão 9
- Linguagem de programação: Ruby, versão 2.4.1
- Gestor de ambientes Ruby: RVM, versão 1.29.4
- Gestor de colecções de *gems*: Bundler, versão 1.16.3
- *Framework* de desenvolvimento: Ruby on Rails, versão 5.1.4
- Base de dados: PostgreSQL, versão 10.1

7.6. Testes

A implementação de testes automáticos num projecto de *software* pode ser realizada a vários níveis. A Figura 26 ilustra essa divisão com recurso a uma pirâmide, um conceito proposto por Michael Cohn, composta por três camadas: testes unitários na base, testes de serviço na camada intermédia e testes de interface de utilização no topo [104]. Este conceito propõe que testes que pertencem a camadas inferiores devem abordar âmbitos menores, mais isolados e executar mais rapidamente do que testes que pertencem a camadas superiores, os quais devem ser mais abrangentes quanto ao código que é executado, abordar a integração de vários módulos do projecto e, por isso, possuírem um tempo de execução superior.

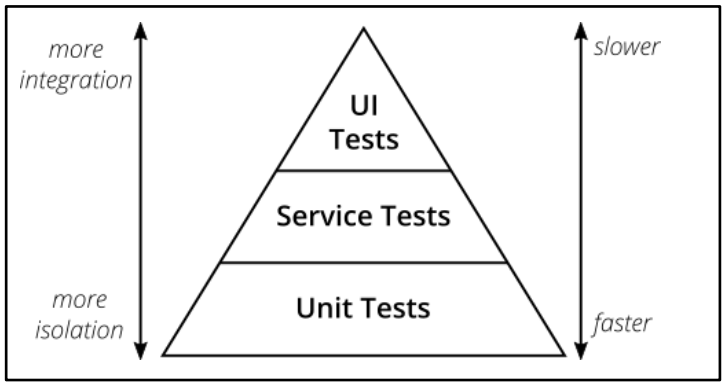

Figura 26 - "The test pyramid", por Michael Cohn [104]

O conceito proposto por Cohn é uma visão simplista, cuja divisão de camadas e nomenclatura aplicada pode, de um ponto de vista mais moderno, ser considerada enganadora [105]. No entanto, serve bem de guia para o desenvolvimento de uma suite de testes de um projecto de *software*, através de duas regras: implementar testes com diferentes granularidades; e quanto mais alto o nível (isto é, quanto maior for o âmbito) do que se está a testar, menos testes devem ser implementados. Ou seja, a suite de testes deve seguir uma forma em pirâmide, possuindo muitos testes unitários pequenos e de rápida execução, alguns testes de granularidade intermédia, e poucos testes de alto nível que testem o projecto de ponta-a-ponta.

A nomenclatura aplicada para os vários níveis de testes é ainda hoje um tema de debate, com autores a considerarem diferentes ordens ou diferentes granularidades de testes para os níveis, como apresentado em [105], [106], e [107]. Assim, para o desenvolvimento consideraram-se os seguintes níveis, com base em [107], da maior granularidade (base da pirâmide) para a menor (topo da pirâmide):

- **Testes unitários (*unit tests*):** validam que a funcionalidade de uma secção específica de código executa conforme o desenhado e esperado. Em programação orientada a objectos, estes testes ocorrem habitualmente ao nível dos métodos das classes ou da classe num todo;
- **Testes de integração (*integration tests*):** validam a interacção entre objectos e interfaces, estando circunscritos ao âmbito da aplicação a ser testada. Neste nível, as interacções com bases de dados ou aplicações externas são simuladas com recurso a bibliotecas para tal;
- **Testes de sistema (*system tests*):** testam a execução do sistema num todo, de um modo integrado, com o propósito de avaliar se cumpre os requisitos especificados.

Os testes implementados com a aplicação de TDD num projecto são os relativos aos níveis mais baixos - unitários e de integração – devido às características da própria técnica, que se foca na implementação de testes a unidades de código mais pequenas. No âmbito do estágio, e tendo em conta a arquitectura MVC que a *framework* Ruby on Rails segue, os testes desenvolvidos com recurso ao TDD incidiram nos modelos e controladores definidos, e, em casos específicos como o do *Migrator*, nos *jobs*. As vistas não foram testadas, uma vez que os componentes abordados não possuíam a vertente de interacção com o utilizador, disponibilizando acesso apenas através de uma API. Posteriormente, foram também realizados testes de sistema, utilizados para validar as interacções entre componentes da plataforma.

Na Tabela 10 é apresentado um resumo da suite de testes produzida. A tabela possui as seguintes colunas:

- **Componente:** identifica o componente a que se refere a linha da tabela;

- **Número de testes:** quantidade de testes desenvolvidos no âmbito do estágio, englobando os diferentes tipos referidos anteriormente. Como alguns dos componentes referidos foram desenvolvidos por outros elementos para além do estagiário, é apresentado entre parêntesis a quantidade total de testes para esses casos;
- **Tempo de execução:** tempo decorrido, em segundos, entre o início e o fim da execução da suite completa de testes ao componente. De salientar que este é um valor meramente indicativo, e não um valor médio;
- **Cobertura:** percentagem de código do componente que é executado aquando da execução da suite de testes.

Componente	Número de testes	Tempo de execução	Cobertura
Services Catalogue	41 (143)	6,13 s	97,62 %
Orchestrator	48 (380)	8,69 s	99,54 %
Migrator	390	24,38 s	96,90 %

Tabela 10 - Resumo da execução da suite de testes

No anexo HC_A8-Testes é possível consultar a listagem completa de testes implementados, por componente. Esta listagem é obtida automaticamente a partir do *output* da execução dos testes desenvolvidos, uma característica da *framework* RSpec utilizada na implementação dos testes e da DSL que esta compreende.

Ainda no âmbito dos testes, foi necessário validar o resultado da migração. Para tal, realizaram-se testes de validação que consistiram na realização de operações com os serviços SaaS externos de destino da migração, depois de esta última ser realizada. Estes testes compreendiam a execução de *scripts* que invocavam as funcionalidades dos serviços (através das APIs) e também a realização de acções através das interfaces de utilizador que disponibilizam. Deste modo era possível validar que os dados tinham sido exportados correctamente para o serviço SaaS, e que estes poderiam ser utilizados para realizar operações do âmbito do serviço.

Capítulo 8
Conclusões

O presente relatório descreve o trabalho realizado no âmbito do estágio inserido no projecto Hotelcracy Apps, o qual pretende desenvolver uma plataforma integradora de serviços *cloud* SaaS utilizados no sector do alojamento. O principal objectivo do estágio foi o desenvolvimento de componentes que permitissem a migração entre serviços SaaS através da plataforma, e que servisse de prova de conceito da viabilidade dessa migração. Como complemento a este objectivo, o trabalho realizado no estágio permitiu ainda estabelecer uma base sólida para a evolução da plataforma, com contributos fortes na especificação da sua arquitectura, dos componentes que a compõem, e dos processos que compreende. Nos parágrafos seguintes é apresentada uma reflexão do trabalho realizado, dos resultados obtidos, e do trabalho futuro no projecto.

Sumário do trabalho realizado

Em jeito de resumo, o trabalho realizado incidiu nos seguintes aspectos:

- Foi realizada a análise do estado da arte de abordagens de integração de serviços cloud, onde se identificaram casos aplicáveis ao projecto Hotelcracy Apps para acesso às funcionalidades de serviços SaaS. Uma das abordagens identificadas foi a utilização de um iPaaS, uma plataforma de integração, que motivou a realização de um estudo das plataformas deste tipo existentes no mercado, das suas características, e da sua aplicabilidade ao projecto. Esta acabou por ser descartada por não reunir as condições necessárias à sua aplicação. Outra abordagem identificada foi a aplicação do padrão de design *Adapter* [38], a qual foi aplicada na plataforma, em específico ao nível arquitectural.
- Foi realizado um estudo relativo à migração de serviços SaaS utilizados no sector do alojamento, que envolveu a interacção com esses serviços, a aprendizagem de conceitos de negócio do sector e a estruturação do processo a aplicar na migração. A partir deste estudo identificou-se um conjunto de dificuldades que a migração colocava, para as quais se propuseram soluções, que se traduziram na tomada de decisões com impacto ao nível arquitetcural da plataforma.
- Foram realizados contributos na especificação da arquitectura da plataforma a desenvolver no âmbito do projecto, onde se reflectiram as decisões tomadas a partir do estudo da migração de serviços. Esta especificação envolveu a identificação formal de *drivers* arquitecturais (cenários significativos, restrições, e atributos de qualidade), a tomada de decisões e o desenho da arquitectura.
- Foi desenhado o modelo de dados aplicado na plataforma, o Modelo Canónico, que permite lidar com os dados de negócio dos hotéis para a sua estruturação e validação, e para a construção das interfaces de utilizador.
- Foram especificados componentes da plataforma que estão envolvidos tanto na migração de serviços como na execução de outras operações: o *Services Catalogue*, o *Orchestrator*, o *Migrator* e os *Migration Service Drivers*.
- Foi desenvolvida uma prova de conceito para realização da migração de serviços, onde se reflectiram os vários conhecimentos adquiridos ao longo do estágio, tais como: a aplicação de abordagens de integração de serviços SaaS para acesso às suas funcionalidades; a aplicação de técnicas de gestão de projectos de software; e a

aplicação de técnicas e tecnologias de desenvolvimento de software. Através desta prova de conceito foi possível validar a realização da migração de serviços, e a identificação dos desafios que esta coloca.

Limitações do trabalho realizado

Em relação aos estudos e especificações realizadas, estas foram extensivas e permitiram estabelecer uma base sólida para a evolução da plataforma. O modo como o processo de migração de serviços foi definido e estruturado permite que este seja aplicado não só na migração, mas também nos seguintes casos:

- No caso de uma organização subscrever um serviço SaaS novo, a plataforma pode usar parte do processo de migração para enviar para esse serviço um conjunto de dados iniciais que permitem ou facilitam a sua utilização;
- No caso de a plataforma Hotelcracy Apps disponibilizar um novo serviço SaaS para utilização integrada, organizações que anteriormente fazia já uso desse serviço podem associar essa subscrição à plataforma, e importar desse serviço dados que sejam utilizados nas comunicações com os restantes serviços do portfólio da organização.

Em relação ao desenvolvimento da prova de conceito da migração, os serviços e tipos de dados abrangidos são reduzidos, muito por limitações das APIs. No entanto, a especificação e estruturação dos componentes envolvidos permitiu o seu desenvolvimento numa base sólida, que se considera poder ser mantida com a evolução da plataforma, e a partir da qual se acredita ser fácil aumentar o leque de serviços e de tipos de dados abrangidos.

Ainda em relação à migração de serviços, é necessário evoluir a política de actualização de dados em uso, de modo a torná-la mais abrangente e, assim, permitir o tratamento automárico de casos de conflito de valores, tentando reduzir ao mínimo o recurso à interacção do utilizador na migração.

Em relação à interacção entre componentes da plataforma, é necessário implementar a comunicação com os componentes *Control & Audit*, *Accounting & Billling* e *Business Data Manager*, uma vez que estes não estavam desenvolvidos aquando da realização do estágio. É também necessário implementar a comunicação com o componente User Interface, uma vez que aquando do estágio não estavam ainda implementados os ecrãs necessários à migração de serviços.

Trabalho futuro

Como trabalho futuro na plataforma, existem vários aspectos a serem abordados. São eles:

- À medida que novos serviços SaaS forem considerados para integração na plataforma, é necessário evoluir o Modelo Canónico (e os vários artefactos que o compõem) para que seja possível a interacção com esses serviços segundo os moldes definidos para a plataforma;
- É necessário aumentar o leque de serviços SaaS e de tipos de dados contemplados para migração;
- É necessário implementar a comunicação entre os componentes envolvidos na migração de serviços e os componentes *Control & Audit*, *Accounting & Billling* e *Business Data Manager*;
- É necessário criar ecrãs no componente *User Interface* que permitam ao utilizador despoletar uma migração de serviços, obter informação desta e actuar sobre ela quando necessário.

Considerações finais

Como conclusão, considera-se que o estagiário cumpriu e ultrapassou os objectivos do estágio, uma vez que conseguiu ir além do simples estudo, especificação e implementação da prova de conceito que foi proposta. Foi possível ao estagiário a aplicação de vários conhecimentos adquiridos durante o curso, bem como o seu aprofundamento. Possibilitou lidar com aspectos muito importantes relacionados com a engenharia de software, tais como a especificação formal da arquitectura de um sistema de software, e a discussão e análise de soluções possíveis para as várias dificuldades identificadas. Permitiu ainda a aprendizagem e aplicação de técnicas e de processos relacionados com a gestão e o desenvolvimento de projectos de software, bem como o desenvolvimento de competências ao nível do trabalho em equipa (a equipa do IPNlis) e do trabalho com outras equipas (a equipa do parceiro Hotelcracy Software, Lda., e o consórcio no seu todo), aspectos que se considera serem basilares na formação de um engenheiro informático.

Tabela de anexos

Anexo 1: HC_A1-Analise_Solucoes_Integracao	**Confidencial:** Sim
Título: Análise de soluções de integração de serviços *cloud*	
Ficheiro: HC_A1-Analise_Solucoes_Integracao.pdf	
Anexo 2: HC_A2-Modelo_Conceptual	**Confidencial:** Sim
Título: Descrição do modelo conceptual do sistema	
Ficheiro: HC_A2-Modelo_Conceptual.pdf	
Anexo 3: HC_A3-Arquitectura_Plataforma	**Confidencial:** Sim
Título: Documento de arquitectura da plataforma Hotelcracy Apps	
Ficheiro: HC_A3-Arquitectura_Plataforma.pdf	
Anexo 4: HC_A4-Services_Catalogue	**Confidencial:** Sim
Título: Documento de especificação detalhada do componente Services Catalogue	
Ficheiro: HC_A4-Services_Catalogue.pdf	
Anexo 5: HC_A5-Orchestrator	**Confidencial:** Sim
Título: Documento de especificação detalhada do componente Orchestrator	
Ficheiro: HC_A5-Orchestrator.pdf	
Anexo 6: HC_A6-Migrator	**Confidencial:** Sim
Título: Documento de especificação detalhada do componente Migrator	
Ficheiro: HC_A6-Migrator.pdf	
Anexo 7: HC_A7-Modelo_Canonico_ER	**Confidencial:** Sim
Título: Diagrama *Entity-Relationship* do Modelo Canónico	
Ficheiro: HC_A7-Modelo_Canonico_ER.pdf	

Tabela de anexos

Anexo 8: HC_A8-Testes	**Confidencial:** Sim
Título: Listagem de testes implementados	
Ficheiro: HC_A8-Testes.pdf	
Anexo 9: HC_A9-Agenda_Reuniao	**Confidencial:** Sim
Título: Exemplo de agenda de reunião de consórcio	
Ficheiro: HC_A9-Agenda_Reuniao.pdf	
Anexo 10: HC_A10-Acta_Reuniao	**Confidencial:** Sim
Título: Exemplo de acta de reunião de consórcio	
Ficheiro: HC_A10-Acta_Reuniao.pdf	

Referências

[1] Hotelcracy Software Lda. and Instituto Pedro Nunes, "Proposta de Candidatura - Parte B (Anexo Técnico) - Sistema De Incentivos À Investigação E Desenvolvimento Tecnológico (SI I&DT)," pp. 1–68, 2015.

[2] Reuters, "Success in Direct Bookings Necessary for Independent Hotel Survival," *Reuters U.S.*, NewYork, 04-Nov-2015.

[3] The Economist, "The market for booking travel online is rapidly consolidating," 2014. [Online]. Available: https://www.economist.com/news/business/21604598-market-booking-travel-online-rapidly-consolidating-sun-sea-and-surfing. [Accessed: 01-Jan-2017].

[4] Techopedia, "What is a Best of Breed System? - Definition from Techopedia." [Online]. Available: https://www.techopedia.com/definition/23200/best-of-breed-system. [Accessed: 29-Aug-2017].

[5] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," *Nist Spec. Publ.*, vol. 145, p. 7, 2011.

[6] B. Almeida and J. Miranda, "Relatório de análise de serviços SaaS no sector do alojamento - v5.0," 2017.

[7] Techterms, "API (Application Programming Interface) Definition," 2016. [Online]. Available: https://techterms.com/definition/api. [Accessed: 30-Aug-2017].

[8] M. Rouse, "What is waterfall model? - Definition from WhatIs.com," 2017. [Online]. Available: https://searchsoftwarequality.techtarget.com/definition/waterfall-model. [Accessed: 10-Aug-2018].

[9] Tutorials Point, "SDLC Waterfall Model." [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. [Accessed: 10-Aug-2018].

[10] Scrum.org, "What is Scrum?" [Online]. Available: https://www.scrum.org/resources/what-is-scrum. [Accessed: 10-Aug-2018].

[11] Boğaziçi Üniversitesi, "Scrum: A new perspective on agile development - bounswe/bounswe2016group6," 2016. [Online]. Available: https://github.com/bounswe/bounswe2016group6/wiki/Scrum:-A-new-perspective-on-agile-development. [Accessed: 10-Aug-2018].

[12] TechTarget, "What is business process? - Definition from WhatIs.Com," 2016. [Online]. Available: http://searchcio.techtarget.com/definition/business-process. [Accessed: 19-Dec-2017].

[13] JBoss, "jBPM - Open Source Business Process Management - Process engine." [Online]. Available: https://www.jbpm.org/. [Accessed: 13-Jun-2017].

[14] JBoss, "jBPM Documentation - Chapter 7. Human Tasks." [Online]. Available: https://docs.jboss.org/jbpm/release/6.5.0.Final/jbpm-docs/html/ch07.html.

[Accessed: 13-Jun-2017].

[15] JBoss, "jBPM Documentation - Chapter 6. Processes." [Online]. Available: https://docs.jboss.org/jbpm/release/6.5.0.Final/jbpm-docs/html/ch06.html#d0e3711. [Accessed: 13-Jun-2017].

[16] JBoss, "jBPM Documentation - Chapter 17. Remote API." [Online]. Available: https://docs.jboss.org/jbpm/release/6.5.0.Final/jbpm-docs/html/ch17.html. [Accessed: 13-Jun-2017].

[17] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, 1st ed. Addison-Wesley Professional, 2003.

[18] Gartner, "Magic Quadrant for Enterprise Integration Platform as a Service, Worldwide," 2016.

[19] J. Garofalo, "Why SaaS is Broken (and how we're going to fix it)," 2014. [Online]. Available: https://blitzen.com/blog/why-saas-is-broken/. [Accessed: 10-Jun-2017].

[20] N. V. S. Kolluru and N. Mantha, "Cloud Integration - Strategy to Connect Applications to Cloud," *2013 Annu. IEEE India Conf. INDICON 2013*, pp. 2–7, 2013.

[21] M. Johnson, "4 Steps to Consider in a Cloud Integration Framework.," *CIO Insight*, p. 1, 2015.

[22] W. Kim, J. H. Lee, C. Hong, C. Han, H. Lee, and B. Jang, "An innovative method for data and software integration in SaaS," *Comput. Math. with Appl.*, vol. 64, no. 5, pp. 1252–1258, 2012.

[23] W. Sun, K. Zhang, S.-K. Chen, X. Zhang, and H. Liang, "Software as a Service: An Integration Perspective," *Serv. Comput. - ICSOC 2007*, pp. 558–569, 2007.

[24] Q. Li, Z. Wang, W. Li, J. Li, C. Wang, and R. Du, "Applications integration in a hybrid cloud computing environment: modelling and platform," *Enterp. Inf. Syst.*, vol. 7, no. 3, pp. 237–271, 2013.

[25] Q. Li, Z. Wang, W. Li, Z. Cao, R. Du, and H. Luo, "Model-based services convergence and multi-clouds integration," vol. 64, no. 7 OP-In Computers in Industry September 2013 64(7):813-832, p. 813, 2013.

[26] E. Pinho, L. B. L. B. Silva, and C. Costa, "A cloud service integration platform for web applications," *High Perform. Comput. Simul. (HPCS), 2014 Int. Conf.*, pp. 366–373, 2014.

[27] H. Smith, "Software as a Service (SaaS) Sightings, Directory and Showcase: SaaS 2.0 relies on SaaS Integration Platforms (SIPs)," 2006. [Online]. Available: http://saassightings.blogspot.pt/2006/05/saas-20-relies-on-saas-integration.html. [Accessed: 10-Jun-2017].

[28] SutiSoft, "The Benefits of SaaS Integration Platforms (SIPs) | SutiSoft," 2016. [Online]. Available: http://www.sutisoft.com/blog/the-benefits-of-saas-integration-platforms-sips/. [Accessed: 10-Jun-2017].

[29] C. Young, "Mediation Models for iPaaS and EAI," 2015. [Online]. Available: http://geekswithblogs.net/cyoung/archive/2015/01/04/mediation-models-for-ipaas-and-eai.aspx. [Accessed: 10-Jun-2017].

[30] J. Lamont, "SaaS: Integration in the Cloud," *KMWorld*, vol. 19, no. January, pp. 12, 13,

22, 2010.

[31] D. Merkel, "Cloud Integration Patterns," Karlsruhe University of Applied Sciences, 2014.

[32] G. Breiter and V. K. Naik, "A Framework for Controlling and Managing Hybrid Cloud Service Integration," *2013 IEEE Int. Conf. Cloud Eng.*, pp. 217–224, 2013.

[33] TechTarget, "What is hybrid cloud? - Definition by WhatIs.com." [Online]. Available: http://searchcloudcomputing.techtarget.com/definition/hybrid-cloud. [Accessed: 11-Dec-2017].

[34] F. Liu, W. Guo, Z. Q. Zhao, and W. Chou, "SaaS Integration for Software Cloud," *Proc. - 2010 IEEE 3rd Int. Conf. Cloud Comput. CLOUD 2010*, pp. 402–409, 2010.

[35] S. Palanimalai and I. Paramasivamb, "An Enterprise Oriented View On The Cloud Integration Approaches - Hybrid Cloud And Big Data," *Procedia Comput. Sci.*, vol. 50, pp. 163–168, 2015.

[36] B. Wang, H. Liu, and J. Song, "SaaS-based enterprise application integration approach and case study," *J. Supercomput.*, vol. NA, no. 7, pp. 2833–2847, 2016.

[37] L. A. B. Silva, C. Costa, and J. L. Oliveira, "A common API for delivering services over multi-vendor cloud resources," *J. Syst. Softw.*, vol. 86, no. 9, pp. 2309–2317, 2013.

[38] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[39] Gartner, "Gartner Reference Model for Integration PaaS," 2011. [Online]. Available: https://www.gartner.com/doc/1729256/gartner-reference-model-integration-paas. [Accessed: 11-Jun-2017].

[40] Gartner, "iPaaS - integration Platform-as-a-Service - Gartner." [Online]. Available: http://www.gartner.com/it-glossary/information-platform-as-a-service-ipaas/. [Accessed: 11-Jun-2017].

[41] Saugatuck Technology Inc., "SaaS 2.0: Software-as-a-Service as Next-Gen Business Platform - Executive Summary." 2006.

[42] CloudWork, "Cloud Business App Integrations - CloudWork." [Online]. Available: https://cloudwork.com/. [Accessed: 23-Jun-2017].

[43] Innotas, "Integration Platform Solutions | Innotas." [Online]. Available: https://www.innotas.com/solutions-integration-platform. [Accessed: 03-Jul-2017].

[44] Intuit, "Cloud Integration | Intuit Developer." [Online]. Available: https://developer.intuit.com/itduzzit. [Accessed: 03-Jul-2017].

[45] Pipemonk, "Pipemonk: we have joined Freshdesk." [Online]. Available: https://www.pipemonk.com/. [Accessed: 03-Jul-2017].

[46] MuleSoft, "Anypoint Platform - MuleSoft Videos." [Online]. Available: https://videos.mulesoft.com/watch/HJFShJodg7ajEGj6h8bbaW. [Accessed: 02-May-2017].

[47] MuleSoft, "Mule Runtime Engine | Anypoint Platform | MuleSoft." [Online]. Available: https://www.mulesoft.com/platform/mule. [Accessed: 02-May-2017].

[48] MuleSoft, "Mule Runtime // MuleSoft Documentation." [Online]. Available: https://docs.mulesoft.com/mule-user-guide/v/3.8/. [Accessed: 02-May-2017].

[49] MuleSoft, "Key Concepts // MuleSoft Documentation." [Online]. Available: https://docs.mulesoft.com/mule-user-guide/v/3.8/mule-concepts. [Accessed: 02-May-2017].

[50] MuleSoft, "Mule Message Structure // MuleSoft Documentation." [Online]. Available: https://docs.mulesoft.com/mule-user-guide/v/3.8/mule-message-structure. [Accessed: 02-May-2017].

[51] MuleSoft, "Anypoint Exchange | connector | MuleSoft." [Online]. Available: https://www.mulesoft.com/exchange#!/?types=connector. [Accessed: 02-May-2017].

[52] MuleSoft, "HTTP/HTTPS Connector | Anypoint Exchange." [Online]. Available: https://www.mulesoft.com/exchange#!/http-integration-connector?types=connector&searchTerm=http. [Accessed: 02-May-2017].

[53] MuleSoft, "Web Service Consumer Connector | Anypoint Exchange." [Online]. Available: https://www.mulesoft.com/exchange#!/webservice-consumer-integration-connector?types=connector&searchTerm=webservice. [Accessed: 02-May-2017].

[54] MuleSoft, "Anypoint Connector DevKit // MuleSoft Documentation." [Online]. Available: https://docs.mulesoft.com/anypoint-connector-devkit/v/3.8/. [Accessed: 02-May-2017].

[55] InvoiceOcean, "InvoiceOcean/API: InvoiceOcean API," 2017. [Online]. Available: https://github.com/InvoiceOcean/API. [Accessed: 10-Aug-2018].

[56] Hotel-Spider, "Open Travel Alliance (OTA) - Hotel-Spider - Hotel-Spider Connectivity Solutions," 2016. [Online]. Available: https://tourisoft.atlassian.net/wiki/spaces/HOT/pages/6520882/Open+Travel+Alliance+OTA. [Accessed: 10-Aug-2018].

[57] Zoho, "Zoho Invoice | API doc." [Online]. Available: https://www.zoho.eu/invoice/api/v3. [Accessed: 10-Aug-2018].

[58] Open Travel, "Open Travel - Enabling the Future." [Online]. Available: https://opentravel.org/. [Accessed: 10-Aug-2018].

[59] AlpineBits, "AlpineBits | Open Data Exchange in the Alpine tourism." [Online]. Available: https://www.alpinebits.org/. [Accessed: 10-Aug-2018].

[60] HotelRunner, "HotelRunner - API." [Online]. Available: https://developers.hotelrunner.com/custom-apps/rest-api/reservations/retrieve-reservations. [Accessed: 10-Aug-2018].

[61] Hotel-Spider, "OTA_ReadRQ / OTA_HotelResNotifRQ or OTA_ResRetrieveRS - Hotel-Spider - Hotel-Spider Connectivity Solutions." [Online]. Available: https://tourisoft.atlassian.net/wiki/spaces/HOT/pages/6520943/OTA+ReadRQ+OTA+HotelResNotifRQ+or+OTA+ResRetrieveRS. [Accessed: 10-Aug-2018].

[62] InvoiceOcean, "InvoiceOcean/API: InvoiceOcean API # Products." [Online]. Available: https://github.com/InvoiceOcean/api#products. [Accessed: 10-Aug-

2018].

[63] InvoiceXpress, "Items - InvoiceXpress API v2.0.0 Docs - Invoicexpress API v2." [Online]. Available: https://developers.invoicexpress.com/docs/versions/2.0.0/resources/items. [Accessed: 10-Aug-2018].

[64] Zoho, "Zoho Invoice | API Doc # Items." [Online]. Available: https://www.zoho.eu/invoice/api/v3/#Items. [Accessed: 10-Aug-2018].

[65] Zoho, "Zoho Invoice | API doc # Taxes." [Online]. Available: https://www.zoho.eu/invoice/api/v3/#Taxes. [Accessed: 10-Aug-2018].

[66] InvoiceXpress, "Invoices - InvoiceXpress API v2.0.0 Docs - Invoicexpress API v2." [Online]. Available: https://developers.invoicexpress.com/docs/versions/2.0.0/resources/invoices/endpoints/list-invoices-6be1b76f-ef0a-428e-97e0-f465fc474982. [Accessed: 10-Aug-2018].

[67] InvoiceOcean, "InvoiceOcean/API: InvoiceOcean API # Invoices." [Online]. Available: https://github.com/InvoiceOcean/api#6. [Accessed: 10-Aug-2018].

[68] Open Knowledge International, "Open Data Handbook | Bulk." [Online]. Available: http://opendatahandbook.org/glossary/en/terms/bulk/. [Accessed: 10-Aug-2018].

[69] Carnegie Mellon University, "Reduce Risk with Architecture Evaluation." Software Engineering Institute, p. 2, 2018.

[70] L. Bass, P. Clements, and K. Rick, "Software Architecture In Practice," 3rd ed., Addison-Wesley, 2012.

[71] J. Nielsen, "Nielsen Norman Group - Website Response Times," 2010. [Online]. Available: https://www.nngroup.com/articles/website-response-times/. [Accessed: 10-Aug-2018].

[72] F. Chong and G. Carraro, "Software as a Service (SaaS): An Enterprise Perspective," 2006. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa905332.aspx. [Accessed: 23-Jun-2017].

[73] S. Brown, *The Art of Visualising Software Architecture*. 2016.

[74] C. Richardson, "API gateway pattern," 2016. [Online]. Available: http://microservices.io/patterns/apigateway.html. [Accessed: 10-Aug-2018].

[75] Visual Paradigm, "What is Entity Relationship Diagram (ERD)?" [Online]. Available: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/. [Accessed: 10-Aug-2018].

[76] J. C. Rayan, "Data modelling using ERD with Crow Foot Notation - CodeProject," 2015. [Online]. Available: https://www.codeproject.com/Articles/878359/Data-modelling-using-ERD-with-Crow-Foot-Notation. [Accessed: 10-Aug-2018].

[77] Techopedia, "What is Schema? - Definition from Techopedia." [Online]. Available: https://www.techopedia.com/definition/1242/schema. [Accessed: 10-Aug-2018].

[78] W3C, "W3C XML Schema." [Online]. Available: https://www.w3.org/XML/Schema. [Accessed: 10-Aug-2018].

[79] JSON Schema, "JSON Schema | The home of JSON Schema." [Online]. Available: http://json-schema.org/. [Accessed: 10-Aug-2018].

[80] M. Services, "mozilla-services/react-jsonschema-form: A React component for building Web forms from JSON Schema," 2018. [Online]. Available: https://github.com/mozilla-services/react-jsonschema-form. [Accessed: 10-Aug-2018].

[81] GitHub, "GitHub." [Online]. Available: https://github.com/. [Accessed: 10-Aug-2018].

[82] "Git." [Online]. Available: https://git-scm.com/. [Accessed: 10-Aug-2018].

[83] CircleCI, "Continuous Integration and Delivery - CircleCI," 2018. [Online]. Available: https://circleci.com/. [Accessed: 10-Aug-2018].

[84] K. Beck, *Test Driven Development: By Example*, 1st Editio. Addison-Wesley Professional, 2002.

[85] Quora, "Why does Kent Beck refer to the 'rediscovery' of test-driven development? What's the history of test-driven development before Kent Beck's rediscovery?," 2012. [Online]. Available: https://www.quora.com/Why-does-Kent-Beck-refer-to-the-rediscovery-of-test-driven-development-Whats-the-history-of-test-driven-development-before-Kent-Becks-rediscovery. [Accessed: 10-Aug-2018].

[86] S. W. Ambler, "Introduction to Test Driven Development (TDD)." [Online]. Available: http://www.agiledata.org/essays/tdd.html#TDDAMDD. [Accessed: 10-Aug-2018].

[87] T. P. Bookshelf, "Test Driven Development by Kent Beck | The Pragmatic Bookshelf." [Online]. Available: https://pragprog.com/screencast/v-kbtdd/test-driven-. [Accessed: 10-Aug-2018].

[88] Codecademy, "Red, Green, Refactor | Codecademy." [Online]. Available: https://www.codecademy.com/articles/tdd-red-green-refactor. [Accessed: 10-Aug-2018].

[89] M. Lewandowski, "Three levels of TDD | Craftsmanship Archives," 2017. [Online]. Available: http://lewandowski.io/2017/02/thre-levels-of-tdd-1/. [Accessed: 10-Aug-2018].

[90] "Linguagem de Programação Ruby." [Online]. Available: http://www.ruby-lang.org/pt/. [Accessed: 10-Aug-2018].

[91] "About Ruby." [Online]. Available: http://www.ruby-lang.org/en/about/. [Accessed: 10-Aug-2018].

[92] S. Ponnappa and J. A. Basheer, "RubyMonk - Ruby Primer." [Online]. Available: https://rubymonk.com/learning/books/1-ruby-primer. [Accessed: 10-Aug-2018].

[93] J. A. Basheer, "RubyMonk - Ruby Primer: Ascent." [Online]. Available: https://rubymonk.com/learning/books/4-ruby-primer-ascent. [Accessed: 10-Aug-2018].

[94] "Ruby on Rails." [Online]. Available: https://rubyonrails.org/. [Accessed: 10-Aug-2018].

[95] D. H. Hansson, "Doctrine | Ruby on Rails," 2016. [Online]. Available: https://rubyonrails.org/doctrine/. [Accessed: 10-Aug-2018].

[96] M. Hartl, *The Ruby on Rails Tutorial*, 4th editio. 2016.

[97] "RSpec: Behaviour Driven Development for Ruby." [Online]. Available: http://rspec.info/. [Accessed: 10-Aug-2018].

[98] A. Alliance, "BDD: Learn about Behavior Driven Development | Agile Alliance." [Online]. Available: https://www.agilealliance.org/glossary/bdd. [Accessed: 10-Aug-2018].

[99] A. Sumner, *Everyday Rails Testing with RSpec*. Leanpub, 2017.

[100] "Bundler: The best way to manage a Ruby application's gems." [Online]. Available: https://bundler.io/. [Accessed: 10-Aug-2018].

[101] "RVM: Ruby Version Manager." [Online]. Available: https://rvm.io/. [Accessed: 10-Aug-2018].

[102] "RVM: Ruby Version Manager - RVM Gemsets." [Online]. Available: https://rvm.io/gemsets. [Accessed: 10-Aug-2018].

[103] CircleCI, "Continuous Integration - CI | CircleCI." [Online]. Available: https://circleci.com/continuous-integration/. [Accessed: 10-Aug-2018].

[104] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*, 1st Editio. Addison-Wesley Professional, 2009.

[105] H. Vocke, "The Practical Test Pyramid." [Online]. Available: https://martinfowler.com/articles/practical-test-pyramid.html. [Accessed: 10-Aug-2018].

[106] OpenConcept, "Software testing cheat sheet." .

[107] Software Testing Fundamentals, "Software testing Levels - Software Testing Fundamentals." [Online]. Available: http://softwaretestingfundamentals.com/software-testing-levels/. [Accessed: 10-Aug-2018].

[108] Object Management Group, "BPMN Specification - Business Process Model and Notation." [Online]. Available: http://www.bpmn.org/. [Accessed: 02-May-2017].

[109] Docker, "Why Docker | Docker." [Online]. Available: https://www.docker.com/why-docker. [Accessed: 10-Aug-2018].

[110] Opensource.com, "What is Docker? | Opensource.com." [Online]. Available: https://opensource.com/resources/what-docker. [Accessed: 10-Aug-2018].