UNIVERSIDADE D
COIMBRA

Francisco José Artur Azedo

SMART METERING PARSE & CORRELATE

Dissertação no contexto do Mestrado em Engenharia Informática, Especialização em Engenharia de Software orientada pelo Prof. Dr. Nuno Antunes e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Setembro de 2021

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

Smart Metering Parse & Correlate

Francisco José Artur Azedo

Dissertação no contexto do Mestrado em Engenharia Informática, Especialização em Engenharia de Software orientada pelo Prof. Dr. Nuno Antunes e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Setembro de 2021

1 2 9 0

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Agradecimentos

Ao longo da realização desta tese de mestrado contei com o apoio e incentivo de várias pessoas, o que contribuiu também para ultrapassar momentos menos bons e ajudar a chegar à conclusão deste projeto. A todas estas pessoas o meu muito obrigado.

Em primeiro lugar queria agradecer aos meus orientadores, ao Prof. Nuno Antunes e ao engenheiro Sénior César Lourenço, pela sua orientação, pelos desafios colocados que me ajudaram superar a mim próprio, e por todo o acompanhamento prestado e tempo despendido ao longo deste ano.

Em segundo lugar queria mostrar a minha gratidão para com os membros que fazem parte do júri, o Prof. Carlos Bento e o Prof. Nuno Lourenço, pelas orientações dadas durante a defesa intermédia e também pelo tempo despendido na leitura e avaliação deste documento.

Agradeço ainda aos colaboradores da empresa Critical Software, Raquel Azevedo e Sérgio Pastilha, por todo o apoio prestado no esclarecimento de dúvidas relativas ao projeto e também pelas revisões feitas a este documento.

Por último mas não menos importante, queria fazer um agradecimento especial aos meus pais, Carlos Azedo e Lúcia Artur, e à minha namorada, Francisca Carvalho, por todo o apoio prestado, por todo ânimo dado, muitas vezes em fases críticas, e por toda a paciência que tiveram ao longo deste projeto. A conclusão deste estágio talvez não tivesse sido possível sem a vossa ajuda.

A todos os outros que não foram identificados pessoalmente mas que contribuíram de certa forma neste projeto, amigos e restantes familiares, aqui fica o meu sincero agradecimento.

This page is intentionally left blank.

Abstract

Currently, with the advancement of technology and the arrival of industry 4.0, several areas have changed in the way they operate. The energy area was one of those that underwent significant changes, specifically when it comes to meters. This type of devices started do acquire intelligent functionalities, originating Smart meters. These allow communications with energy suppliers to be established and, through a screen, to present relevant data to their owners.

PARSEC is a project developed by the company Critical Software in order to help in the communication process between authorized entities and smart meters. However, the fact that this project was developed using Java version 7 means that it is not optimized with the current technologies. With this in mind, we identified the necessity to update the tools used on the project, in order to maintain it's levels of security, trust and performance, through the usage of a version that offers long-term support.

This thesis aims to update the project to version 11 of Java and, later, to analyse the impact this had on it. This migration aims to update the technologies used, reduce the amount and complexity of code and optimize its performance. Since no new features will be introduced, validating this thesis' success will envolve, after identifying code and performance requirements, the execution of load tests as well as a metrics' evaluation, in order to check if these are in agreement with the project's requirements.

This document contains information about the research process, including the analysis of the project's initial state, the investigation about migration processes, and the survey of changes introduced in the new versions of the technologies used. Besides, it also contains the requirements needed for the migration process, as well as their validation model. There are also contemplated the risks inherent to this project, the planning of the tasks to be developed throughout the internship, the work performed in the process, and the discussion of the obtained results.

Keywords

Energy, Smart Meters, Java, Migration, Optimization

This page is intentionally left blank.

Resumo

Atualmente, com o avanço da tecnologia e com a chegada da indústria 4.0 várias foram as áreas que sofreram alterações no seu modo de funcionamento. A área da energia foi uma das que sofreu grandes alterações, nomeadamente ao nível dos contadores. Este tipo de dispositivos passou a possuir funcionalidades inteligentes, dando origem aos contadores inteligentes. Estes permitem estabelecer comunicações os fornecedores de energia e também, através de um ecrã, disponibilizar vários dados aos seus proprietários.

O PARSEC é um projeto desenvolvido pela empresa Critical Software que tem como objetivo ajudar no processo de comunicação entre entidades autorizadas e os contadores inteligentes. No entanto, o facto deste projeto ter sido desenvolvido recorrendo à versão 7 do Java faz com que este esteja bastante desatualizado face à atualidade. Posto isto foi identificada a necessidade de atualização das ferramentas utilizadas no projeto para manter os níveis de segurança, confiabilidade e performance do mesmo, através da utilização de uma versão da linguagem que oferece suporte a longo termo.

O objetivo principal desta tese é atualizar o projeto para a versão 11 do Java e, posteriormente, analisar os impactos que esta teve no mesmo. Com esta migração pretende-se atualizar as tecnologias utilizadas, reduzir a quantidade e complexidade de código e otimizar o seu desempenho. Uma vez que não vão ser introduzidas novas funcionalidades, a validação do sucesso desta tese passará por, após identificar requisitos a nível de código e performance, realizar testes de carga e avaliar métricas para perceber se estas estão em conformidade com os requisitos do projeto.

Este documento contém informação acerca do processo de pesquisa, que contempla a análise do estado inicial do projeto, a investigação acerca de processos de migração e o levantamento de alterações introduzidas nas novas versões das tecnologias utilizadas. Além disto contém ainda os requisitos necessários para o processo de migração, assim como um modelo de validação para os mesmos. Estão ainda contemplados os riscos inerentes ao projeto, o planeamento das tarefas a desenvolver ao longo de todo o estágio, o trabalho desenvolvido no mesmo e a discussão dos resultados obtidos.

Palavras-Chave

Energia, Contadores Inteligentes, Java, Migração, Otimização

This page is intentionally left blank.

Conteúdo

This page is intentionally left blank.

Acrónimos

Comms Hub Communications Hub. 8, 9, 35, 36, 39

CSP Communication Service Provider. 32, 35, 39

DCC Data Comunications Company. 1, 8–10, 27–29, 31

DSP Data Service Provider. 34, 36, 39

DUIS DCC User Interface Specification. 8–10

GBCS Great Britain Companion Specification. 8–10, 27–30, 33, 34, 38, 39

HAN Home Area Network. 8, 31, 32

IHD In Home Display. 8

JAXB Java Architecture for XML Binding. 45, 46

JDK Java Development Kit. 4, 42, 43, 45, 48, 57, 63

JRE8 Java SE Runtime Environment 8. 10

JVM Java Virtual Machine. 10, 12, 67

MAC Message Authentication Code. xiii, xvi, 10, 28, 30, 32, 33, 37, 41

MMC Message Mapping Catalogue. 10, 27

NCSC National Cyber Security Centre. 8

PARSEC Parse & Correlate. 1, 2, 4, 5, 7, 10, 11, 22, 27–29, 31, 32, 34–38, 40–42, 45, 46, 51, 53, 55, 59–61, 65, 72, 75

SMETS Smart Meter Equipment Technical Specifications. 9

SMIP Smart Metering Implementation Programme. 1, 7, 9, 31

WAN Wide Area Network. 8

XML eXtensible Markup Language. 45

XSD XML Schema Definition. 46

This page is intentionally left blank.

Lista de Figuras

This page is intentionally left blank.

Lista de Tabelas

This page is intentionally left blank.

Capítulo 1

Introdução

Este documento tem como propósito a apresentação do relatório de estágio desenvolvido no âmbito da unidade curricular de Dissertação/Estágio em Engenharia de Software do Mestrado em Engenharia Informática da Faculdade de Ciências e Tecnologias da Universidade de Coimbra. Foi desenvolvido na empresa Critical Software sob supervisão do Engenheiro Sénior César Lourenço e do Professor Nuno Antunes.

1.1 Contextualização

Atualmente, o avanço da tecnologia é uma coisa constante e transversal a todas as áreas de negócio. Uma das áreas que tem sofrido grandes alterações a nível tecnológico é a área da energia. Uma das principais mudanças que tem vindo a ser feita nos últimos tempos é a substituição dos contadores tradicionais de energia, luz e gás, por contadores com funcionalidades inteligentes.

No seguimento desta evolução, e com o objetivo de melhorar o seu sistema energético, o governo do Reino Unido iniciou um programa chamado Smart Metering Implementation Programme (SMIP). Este programa tem como objetivo substituir, em todas as habitações domésticas e pequenos negócios, os contadores tradicionais por contadores inteligentes (*smart meters*) até ao final do ano de 2020 [1].

A dimensão deste programa é bastante grande uma vez que tem um elevado impacto na transição do Reino Unido para uma economia com baixas emissões de carbono e permite atingir alguns dos seus objetivos, tais como garantir um acesso à energia de forma acessível, segura e sustentável [1]. De modo a conseguir implementar um programa com esta dimensão e complexidade foi necessária a intervenção de várias entidades, sendo uma delas a empresa Critical Software com o projeto Parse & Correlate (PARSEC).

O PARSEC é um projeto que tem como objetivo responder a necessidades de comunicação existentes entre as entidades autorizadas e os contadores inteligentes. Apesar de todo o suporte à comunicação com os contadores inteligentes ser assegurado pela Data Comunications Company (DCC), existiam algumas necessidades específicas como por exemplo verificação de comandos críticos, como alterações de tarifários, a tradução de respostas dos contadores e a validação do destinatário das mensagens. A resposta para estas necessidades foi a criação da biblioteca PARSEC para que pudesse ser utilizada pelas entidades que pretendam comunicar com os contadores de energia.

O PARSEC é considerado um projeto crítico uma vez que é utilizado por várias pessoas e

empresas do Reino Unido, dando suporte diariamente à comunicação de milhões de pedidos entre contadores e outras entidades, e qualquer anomalia no seu funcionamento poderá ter impacto direto e significativo na vida de milhões de pessoas.

1.2 Âmbito do Estágio

A biblioteca PARSEC foi desenvolvida pela empresa Critical Software e é utilizada na área das comunicações com contadores de energia inteligentes. O objetivo desta é permitir que certas entidades, como fornecedores de energia, operadores de rede e entidades externas autorizadas, consigam validar mensagens críticas, verificar os destinatários das mensagens enviadas pelos contadores e efetuar a tradução das mesmas.

Uma vez que este projeto foi desenvolvido utilizando a versão 7 do Java e que, atualmente, suporta a versão 8, que deixou de ter suporte no final de 2020, surgiu a necessidade de fazer a migração da plataforma para uma versão mais atualizada que garanta não só um novo prazo de suporte da sua plataforma oficial, como também uma atualização do projeto que se traduz em melhorias tanto a nível de código como a nível da execução.

1.3 Objetivos

Como referido no ponto anterior, atualmente a biblioteca PARSEC encontra-se desenvolvida utilizando a versão 7 da linguagem de programação Java. O principal objetivo deste estágio é a atualização da biblioteca para que consiga suportar a versão 11 desta linguagem, tirando partido de novas funcionalidades que foram lançadas entre as duas versões.

Além da atualização ao nível da linguagem utilizada, outro dos objetivos deste estágio é a atualização de todas as dependências, *plugins* e servidores utilizados pela biblioteca e que garantem o seu funcionamento.

O terceiro objetivo deste estágio é realizar um refactoring ao código existente de modo a tornar este mais conciso, claro, otimizado e menos susceptível a bugs.

Por fim, pretende-se ainda estudar o impacto que a atualização dos diferentes componentes deste projeto tem na performance do mesmo, mais precisamente no tempo de processamento de pedidos.

1.4 Metodologia

O ponto de partida deste projeto foi a realização de uma reunião com pessoas do projeto PARSEC onde foram abordados temas como o contexto do projeto, análise da proposta de estágio e definição do plano de trabalhos. A abordagem escolhida para a realização deste estágio passou por dividir o mesmo por várias fases sequenciais e ir executando as tarefas correspondentes a cada fase com o decorrer do estágio. Esta estratégia encontra-se representada na Figura 1.1.

A primeira fase do projeto foi a fase de *setup* do mesmo. Esta consistiu em efetuar todas as configurações necessárias para ter o projeto funcional, acesso a repositórios, configurações de editores de código e de servidores. Para finalizar foram feitos testes funcionais à ferramenta com alguns casos de exemplo.

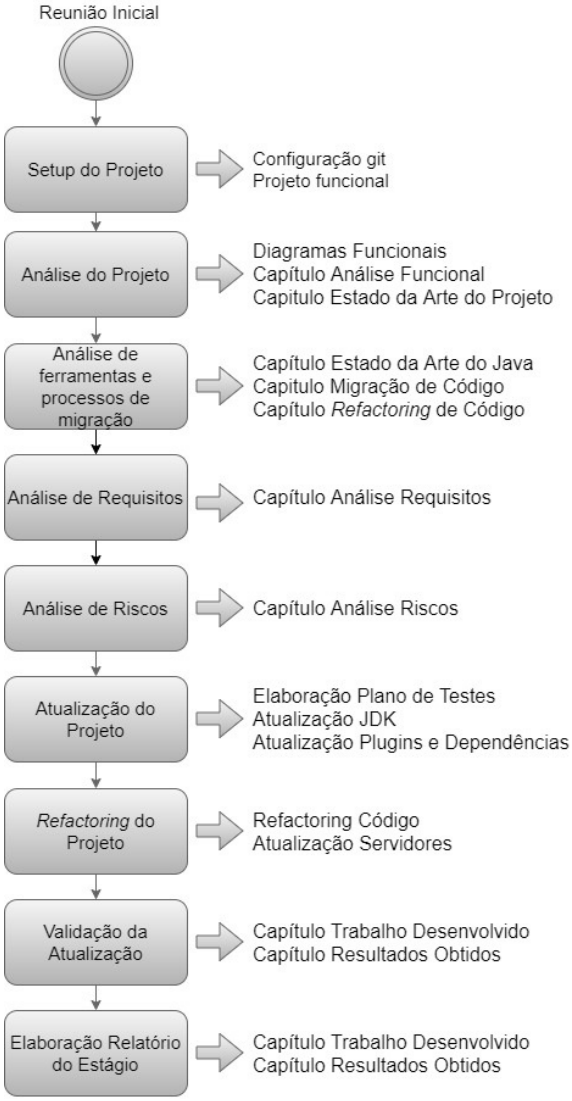

Figura 1.1: *Metodologia utilizada*

A fase que se seguiu foi a análise do projeto onde foram aprofundados alguns aspetos referentes ao projeto PARSEC como o contexto, os seus objetivos, o estado atual e as suas funcionalidades. Como resultado desta fase foram produzidos vários diagramas funcionais e elaborados os capítulos da análise funcional e do estado da arte do projeto.

Seguiu-se a fase da análise de ferramentas e processos referentes à migração do projeto. Nesta fase foram pesquisadas informações acerca de versões do Java, métodos e boas práticas de migração e *refactoring* de código. No final desta fase foram produzidos os capítulos de estado da arte do Java e migração e *refactoring* de código.

Depois disto, a fase seguinte foi a fase dos requisitos. Nesta foram identificados os requisitos correspondentes ao processo de atualização da biblioteca PARSEC e definidos os respetivos critérios de validação. Resultante desta fase foi elaborado o capítulo dos requisitos.

Seguidamente iniciou-se a fase da análise de riscos onde, com base nos requisitos definidos anteriormente, foram identificados possíveis riscos presentes neste projeto. Esta fase terminou com a produção do capítulo de riscos.

Sucedeu-se a fase de atualização do projeto onde, inicialmente, foram realizados testes de performance, passando depois para a atualização da versão do Java Development Kit (JDK), *plugins* e dependências necessárias para garantir a compatibilidade com a mesma.

Depois disto foram repetidos novamente os testes para verificar o estado do projeto e deu-se início à fase de *refactoring*. Nesta fase procedeu-se à implementação de novas funcionalidades oferecidas pela nova versão da linguagem e à otimização de código. Por fim foi ainda feita a atualização dos servidores e foram realizados, novamente, os testes de performance para avaliar o impacto da atualização do projeto.

Por último deu-se início à fase de validação da atualização. Nesta fase foram comparados os resultados dos diferentes testes de performance e foi utilizada a ferramenta SonarQube para recolher métricas e perceber os impactos da atualização no projeto.

Paralelamente ao desenvolvimento destas fases, foi sendo produzido um relatório para ser apresentado contendo toda a informação relativa a este estágio.

1.5 Estrutura do Relatório

Este documento, está estruturado da seguinte forma:

- **Capítulo 2 - Estado da arte**, que será composto com a análise de conceitos, tecnologias e metodologias utilizadas ao longo projeto.
- **Capítulo 3 - Análise Funcional**, onde serão descritas e analisadas todas as funcionalidades presentes na biblioteca PARSEC.
- **Capítulo 4 - Análise de Requisitos**, onde irão estar documentados os requisitos deste projeto tendo em conta os seus objetivos.
- **Capítulo 5 - Processo de Atualização**, que irá conter uma descrição das tarefas realizadas ao longo deste estágio.
- **Capítulo 6 - Planeamento**, onde será demonstrado o plano de trabalhos para ambos os semestres e identificados os riscos associados ao projeto.

- **Capítulo 7 - Resultados obtidos**, contendo a análise dos resultados obtidos da atualização da biblioteca PARSEC e respetiva validação.
- **Capítulo 8 - Conclusões**, onde serão expostas as conclusões retiradas deste estágio através de uma reflexão acerca de tudo o que aconteceu no decorrer deste projeto.

This page is intentionally left blank.

Capítulo 2

Estado da Arte

Neste capítulo vão ser analisados e descritos os principais conceitos relativos ao projeto. A informação apresentada de seguida é resultante de pesquisa sobre o estado atual dos mesmos, desde conceitos da área da energia até às tecnologias tecnologias que serão utilizadas no projeto.

De seguida vão ser apresentadas 4 secções, sendo que a primeira irá documentar alguns conceitos chave relacionados com o projeto Parse & Correlate (PARSEC), mais precisamente com a área da energia e dos contadores inteligentes, a segunda incidirá sobre a evolução e o estado atual da linguagem Java, a terceira terá informação relacionada com práticas relacionadas com a migração de código e, por fim, a quarta secção terá uma análise acerca dos processos de *refactoring* de código.

2.1 Projeto *Parse & Correlate*

Nesta secção serão apresentados e discutidos alguns conceitos essenciais para uma melhor compreensão da necessidade do projeto PARSEC e também vão ser analisados alguns aspetos acerca do seu estado atual, nomeadamente sobre o seu funcionamento.

2.1.1 *Smart Meters*

Anteriormente ao projeto Smart Metering Implementation Programme (SMIP), os contadores de energia utilizados no Reino Unido eram contadores tradicionais, que ainda são utilizados na grande maioria do mundo. Estes contadores começaram por ser analógicos e passaram, posteriormente, para contadores digitais. No entanto, todas as operações relacionadas com leituras de dados e alterações quer fosse de tarifa ou de fornecedor de energia necessitavam de uma interação direta, neste caso física, com o contador e, em alguns casos, era mesmo necessária a sua substituição, por exemplo na troca de fornecedor.

Com o início do SMIP, os contadores passaram a possuir alguma autonomia e a suportar comunicações vindas do exterior. Surgiram então os denominados contadores inteligentes. Estes novos contadores possuem várias funcionalidades inteligentes tais como [2] [3]:

- A capacidade de comunicação, troca de mensagens, entre contadores e com os fornecedores de energia, operadores de rede e entidades externas autorizadas.
- A contagem de energia de forma automática, isto é, a energia que é consumida

é enviada para o fornecedor que cobra o preço da mesma em vez de cobrar uma estimativa.

- A possibilidade de mudar de fornecedor de energia ou de tarifa através do envio de um comando para o contador.

Além dos contadores, existe também outro dispositivo, In Home Display (IHD), que é instalado juntamente com os mesmos. Este dispositivo estabelece uma ligação com todos os contadores inteligentes instalados na habitação e permite visualizar, em tempo real, a quantidade de energia gasta e o preço da mesma. Isto faz com que os consumidores de energia, que tenham estes equipamentos, tenham mais controlo sobre os seus gastos energéticos, permitindo-lhes poupar energia e dinheiro [4].

2.1.2 DCC

A principal preocupação com os contadores, antes mesmo de serem instalados nas casas dos consumidores de energia, era garantir que os mesmos conseguiam comunicar entre si, com o IHD e com os fornecedores de energia, operadores de rede e entidades externas autorizadas. Para produzir uma solução para esta necessidade, o governo escolheu a empresa Data Comunications Company (DCC).

A DCC é a empresa responsável por fornecer toda a estrutura de comunicações do Reino Unido para contadores inteligentes. Esta rede tem como objetivo estabelecer a ligação entre os contadores e os *DCC Service Users*, que são os fornecedores de energia, os operadores de rede e as entidades externas autorizadas. A nível de segurança, todas as ligações têm de cumprir standards de segurança estabelecidos pelo National Cyber Security Centre (NCSC). Atualmente esta estrutura conta com vários milhões de contadores já instalados [5] [6].

2.1.3 Comunicação

Após ter sido resolvida a necessidade da existência de uma rede de comunicação para os contadores, surgiu a questão de que forma se iria processar a comunicação entre os vários dispositivos e os *DCC Service Users*. A origem deste problema provém do facto de existirem vários fornecedores de energia, vários fabricantes de contadores e do facto de, por se tratar de empresas diferentes, existir a necessidade de criar um mecanismo de comunicação transversal a todos eles.

Perante este problema, a solução adotada passou pela definição de uma linguagem que todos os contadores percebessem e que pudessem usar para comunicar, neste caso foi especificado Great Britain Companion Specification (GBCS). Foi também definida uma especificação, DCC User Interface Specification (DUIS), para ser utilizada pelos *DCC Service Users* para comunicar com os contadores.

Além disto, foi também instalado na casa de cada um dos consumidores de energia um dispositivo chamado Communications Hub (Comms Hub). Este aparelho está ligado a uma rede, Home Area Network (HAN), que está associada a uma propriedade e que suporta as comunicações dos vários dispositivos ligados à rede. Para comunicações com o exterior é utilizada uma rede diferente denominada Wide Area Network (WAN). Assim, o Comms Hub comunica com todos os contadores da propriedade através da HAN e com os *DCC Service Users* através da WAN. A responsabilidade deste dispositivo é fazer o reencaminhamento de pedidos e respostas em ambos os sentidos.

2.1.4 Responsabilidades da DCC

Com base na solução descrita anteriormente, a DCC construiu e disponibilizou uma *gateway* para que os *DCC Service Users* pudessem comunicar com os dispositivos presentes nas habitações dos consumidores de energia. Nesta estão contemplados três *endpoints* distintos: `SendCommand`, `Transform` e `NonDevice` [7].

O *endpoint* `transform` é utilizado para transformar os comandos críticos envidas pelos *DCC Service Users* de DUIS para GBCS para que os contadores de energia consigam entender o pedido que lhes está a ser feito.

O *endpoint* `sendCommand` é utilizado para enviar comandos para um contador. No caso de se tratar de um comando crítico este já vem convertido para GBCS, caso contrário é necessário fazer a tradução do comando antes de enviar o mesmo para o Comms Hub que, posteriormente, trata do reencaminhamento para que chegue até ao contador.

Além destas duas funcionalidades, a DCC encarrega-se também de fornecer um *endpoint* para recepção de respostas. Este disponibiliza tanto as respostas assíncronas dos contadores como as respostas síncronas da própria *gateway* da DCC.

2.1.5 Necessidade

No arranque do programa SMIP os contadores, que inicialmente foram instalados nas habitações, eram eram da primeira geração de contadores inteligentes, denominados Smart Meter Equipment Technical Specifications (SMETS). A comunicação entre os contadores e os *Service Users* era feita através de uma rede 3G. Eram utilizados vários sistemas para gerir o funcionamento deste tipo de equipamentos, cada produtor de contadores tinha o seu sistema. Esta diversidade levou a que surgissem problemas quando existia a necessidade de mudar de fornecedor de energia. O facto de nem todos os fornecedores terem compatibilidade com todos os contadores dificultava esta operação. Na maior parte dos casos era necessário substituir o equipamento devido ao facto deste não conseguir comunicar com o novo fornecedor de energia, perdendo assim as suas principais funcionalidades inteligentes [8].

Para conseguir resolver este problema, foi lançado em 2018 um novo tipo de contadores, SMETS2. Este novo modelo utiliza uma rede de comunicação desenvolvida especificamente para ser utilizada para comunicar com todos os *DCC Service Users*, agilizando as operações de mudança de fornecedor e mantendo os contadores inteligentes.

Com a atualização para este novo tipo de contadores e para o novo modelo de comunicação surgiram algumas necessidades:

- A conversão das respostas GBCS dos contadores para um XML que os fornecedores conseguissem entender.
- A verificação da conversão de comandos críticos feita pela DCC, validando todos os campos traduzidos.
- A verificação dos campos que eram necessários assinar por parte do fornecedor num comando crítico.
- A validação do destinatário das respostas enviadas pelos contadores.

2.1.6 Solução Implementada

Com o objetivo de conseguir resolver todos os problemas identificados anteriormente, foi desenvolvido, pela empresa Critical *Software*, o projeto PARSEC.

Este projeto consiste numa biblioteca que contém as seguintes funcionalidades:

- Tradução das respostas dos contadores de GBCS para um XML denominado Message Mapping Catalogue (MMC) [9].
- Validação da tradução feita pela DCC de um pedido DUIS com um comando crítico para um GBCS denominado *Pre-Command*.
- Extração da informação do *Pre-Command* que necessita de ser assinada.
- Extração da informação necessária para validar o Message Authentication Code (MAC) presente na resposta GBCS do contador, caso seja uma resposta não crítica.
- Extração da informação necessária para validar a assinatura da resposta GBCS do contador, caso seja uma resposta crítica.

2.1.6.1 Funcionalidades da Solução

Como referido no capítulo anterior, o projeto PARSEC consiste numa biblioteca com um leque de funcionalidades necessárias no processo de comunicação com um contador. O objetivo desta solução é ser utilizada pelos *DCC Service Users* quando pretendem comunicar com os contadores. Dentro da biblioteca existem 3 módulos principais que agregam as funcionalidades disponibilizadas: `parse`, `correlate` e `macSignatureInfo`.

O módulo `parse` tem como propósito efetuar a tradução das respostas vindas dos contadores de GBCS para XML, neste caso MMC [10].

O módulo `correlate` possibilita que se faça uma comparação entre o comando crítico que foi enviado e a conversão deste para GBCS devolvida pela DCC.

Por fim, o módulo `macSignatureInfo` permite realizar várias operações. Em primeiro lugar permite verificar os campos do *Pre-Command* que são necessários assinar, antes deste ser enviado para o contador. Em segundo permite extrair da resposta os componentes necessários para efectuar as validações criptográficas. No caso de se tratar de uma resposta crítica devolve a informação necessária para validar a assinatura, caso seja não crítica, para validar o MAC.

2.1.6.2 Características da Solução

Apesar de se tratar de uma biblioteca, de forma a agilizar a sua utilização por parte dos *DCC Service Users*, juntamente com o projeto PARSEC é também disponibilizada uma aplicação Web com uma interface que permite a utilização de cada um dos módulos referidos na secção anterior. No desenvolvimento deste projeto foi definido que a linguagem utilizada seria Java. Atualmente, este projeto é executado numa Java Virtual Machine (JVM) com Java SE Runtime Environment 8 (JRE8), apesar de ter sido desenvolvido utilizando a versão 7. Uma vez que a versão 8 do Java foi lançada depois do início do

desenvolvimento do projeto, grande parte das novidades introduzidas nesta versão não foram ainda implementadas no projeto.

Além disto, são também disponibilizadas configurações pré-definidas de que permitem que a aplicação Web e os *endpoints* disponibilizados pelo PARSEC possam correr em dois servidores distintos. Uma das configurações é compatível com a versão 6.1 do *Red Hat JBoss*. A outra configuração é relativa à versão 7.0 do servidor *Apache TomCat*.

2.2 Linguagem de Programação Java

Nesta secção irá ser feita uma análise à linguagem Java. Inicialmente será apresentado uma breve descrição de como surgiu esta linguagem, de seguida serão analisadas algumas características referentes à mesma e, por fim, serão percorridas todas as versões, desde o lançamento da linguagem até à última versão, e serão analisadas as principais *features* lançadas em cada uma delas.

2.2.1 História

A história do Java começa em 1991 numa empresa chamada Sun Microsystems. Nesta altura, uma equipa de três engenheiros, Patrick Naugthon, Sun Fellow e James Gosling, juntaram-se a dois fundadores da empresa Sun, Andy Bechtolsheim e Bill Joy, para realizar um projeto denominado Projeto Green.

O objetivo deste projeto era criar programas portáteis que pudessem ser executados em vários dispositivos e que possibilitasse a comunicação entre eles. Por exemplo, no caso de uma cozinha em que houvesse um fogão, deixava-se a comida a assar e quando estivesse pronta o forno enviava uma mensagem para ligar o micro-ondas e, posteriormente, o micro-ondas enviava uma mensagem para ligar a televisão que depois mandava tocar o despertador.

Pouco tempo depois de iniciar o projeto a equipa percebeu que um dos desafios seria contornar o facto de cada fabricante ter o seu próprio sistema operacional. Devido ao facto de ser incomportável fazer um programa para cada sistema operacional, deste projeto rapidamente surgiu um novo sistema chamado GreenOS.

Este novo sistema utilizava uma nova linguagem que, inicialmente, foi denominada de *oak*, a origem do nome vem devido ao facto do chefe o projeto, James Gosling, observar um carvalho pela janela enquanto pensava numa estrutura de diretorias. No entanto, esse nome já se encontrava registado. Surge então, numa cafetaria local onde a equipa costumava tomar café, o nome Java, visto ser o nome da terra de onde era importado o café consumido por toda a equipa.

Até 1994, o Java ainda não tinha uma aplicação bem definida, então Jonathan Payne e Patrick Naughton criaram um navegador para Web que oferecia a possibilidade de executar programas escritos em Java (*applets*) ao qual foi dado o nome de Web Runner e que teve um enorme sucesso e uma aceitação por parte dos browsers populares na altura, como por exemplo o Netscape Navigator.

Pouco depois, em 1996, numa iniciativa inédita a empresa Sun Microsystems decide disponibilizar gratuitamente um kit de desenvolvimento de *software* para poder ser utilizado pela comunidade, denominado Java Developer´s Kit (JDK) nome que se mantém mesmo nos dias de hoje. Este produto teve uma enorme aceitação e o desenvolvimento de apli-

cações em Java cresceu exponencialmente. Posteriormente foi lançada uma versão 1.1 do JDK com melhoramentos para o desenvolvimento de aplicações gráficas e distribuídas.

Em Abril de 2009, a Oracle fez uma proposta de compra à Sun Microsystems que viria a ser aceite, passando os produtos desta empresa, entre eles o Java e o sistema operacional Solaris, a pertencer à gigante americana. Esta transição permitiu ao Java ter um salto qualitativo bastante significante [11] [12].

2.2.2 Características

Algumas das principais características da linguagem de programação Java vão ser descritas de seguida [13]:

- É uma linguagem multiplataforma uma vez que, após compilado, o código Java gera *bytecode* que, posteriormente, é executado por uma JVM. Esta pode ser executada em diversas plataformas.
- É uma linguagem orientada a objetos, permitindo a utilização de funcionalidades como herança e reutilização de código.
- É inspirada das linguagens C e C++, sendo por isso familiar para os programadores, e retira a responsabilidade da gestão da utilização da memória do programador, facilitando a sua tarefa.
- Permite conferir um nível de segurança extra às aplicações, uma vez que possui uma ferramenta que efetua uma verificação do programa antes do mesmo ser executado de modo a garantir que este não contém nenhum tipo de vírus ou funcionalidades com acessos não autorizados. Esta verificação é também efetuada nos *applets* que correm nos browsers.
- Possui um elevado número de bibliotecas e *frameworks* que oferece um leque maior de possibilidades aos programadores.

2.2.3 Histórico de Versões

Neste ponto vão ser analisadas todas as versões do Java desde o seu lançamento até à versão 15, que é a versão mais recente, salientando as principais novidades lançadas ao longo de todas estas versões. Contudo, como o projeto vai ser migrado para a versão 11 do Java, devido ao facto de ser a última versão lançada com *long-term support*, apenas vão ser aplicadas as alterações correspondentes até esta versão [14] [15].

2.2.3.1 JDK 1.1

A versão 1.1 do Java foi lançada em Fevereiro de 1997. De todas as alterações introduzidas nesta versão há algumas que são necessárias salientar e que são [16] [17] [18]:

Inner Class

O conceito de *Inner Class* resume-se à possibilidade de declarar classes dentro de outras classes ou de interfaces.

As vantagens de adotar esta prática são [19] [20]:

- Agrupar classes e interfaces que funcionam juntas, o que torna o código desenvolvido mais legível e facilita a manutenção.
- Permite o acesso aos métodos e variáveis privados da classe exterior e aumenta o nível de encapsulamento.
- Permite melhorar a organização e reduzir a quantidade de código, uma vez que evita a criação de uma classe separada.

JDBC

JDBC (*Java Database Connectivity*) é uma API que permite fazer a gestão da ligação de uma aplicação em Java com uma base de dados, tipicamente relacional. Esta API disponibiliza vários métodos com diversos propósitos, desde a criação da conexão com a base de dados até a operações efetuadas sobre a mesma [21] [22].

Algumas das vantagens que esta ferramenta trouxe são [23]:

- Permite ao programadores abstraírem-se do protocolo DBMS.
- Permite a conexão a sistemas de bases de dados previamente existentes sem que estes tenham de ser alterados.
- Conexão à base de dados é feita através de um URL identificador que faz com que não seja necessário efetuar mais nenhuma configuração além desta.
- Permite efetuar todas as ações possíveis a uma base de dados, quer ao nível de persistência de dados quer a alterações estruturais na base de dados.

2.2.3.2 J2SE 1.2

A versão 1.2 do Java foi lançada em Dezembro de 1998. Do conjunto de alterações introduzidas nesta versão, é de salientar uma delas [16] [24] [25].

Collections

A ferramenta Collections do Java é um conjuntos de classes que permite agregar e manipular conjuntos instâncias de objetos. Permitem efetuar todo operações sobre o grupo de objetos como adicionar, retirar, alterar, pesquisar e ordenar. São disponibilizadas várias interfaces de modo a que o mesmo conjunto de objetos possa ser representado de formas diferentes, por exemplo: `Set` e `List` [26] [27].

Esta solução disponibilizada trouxe algumas vantagens tais como [28] [29]:

- Os programadores não necessitam de gastar tempo a produzir as suas próprias APIs ou a aprender outras que permitam manipular conjuntos de objetos.
- Aumenta a qualidade do código produzido, uma vez que se passou a usar uma ferramenta standard da linguagem devidamente testada.
- Permitem um aumento de performance, uma vez que os algoritmos utilizados nesta ferramenta são mais eficientes que os utilizados anteriormente.

2.2.3.3 J2SE 1.4

Na versão 1.4 do Java, que foi lançada em Fevereiro de 2002, foram introduzidas várias alterações. Deste leque de alterações existem algumas que merecem a pena ser destacadas [16] [30] [31].

Assertions

A *keyword* `Assertion` foi introduzida no Java com o propósito de ser utilizada principalmente nos métodos de teste, embora possa ser usada também para o desenvolvimento de algum componente. Permite testar a veracidade de uma expressão [32] [33].

Quando a condição que está a ser verificada é inválida, o programa lança uma exceção automaticamente. No entanto, quando o programa passa para produção, estas anotações devem ser retiradas para que o mesmo não fique dependente destas.

Algumas das vantagens que está *keyword* trouxe foram [34]:

- Maior eficiência na deteção e correção de *bugs*, pois veio agilizar o processo de testes.
- Otimização do código, uma vez que apenas uma *keyword* substitui uma verificação e um lançamento de uma exceção como era feito anteriormente.

***Logging* API**

Logging é uma API que permite em vários pontos do código colocar várias mensagens para que depois, quando estiver a ser executado, o programa possa gerar um relatório com todas as mensagens por onde passou. Isto é bastante útil para tanto para o programador como para o utilizador da aplicação ou sistema pois permite que estes consigam identificar com bastante facilidade todo o caminho percorrido pelo programa e perceber se aconteceram ou não anomalias.

Esta ferramenta tem vários níveis, por exemplo *debug* e erro, que são utilizadas para colocar mensagens consoante o seu tipo. Estas níveis permitem que haja uma melhor separação das mensagens do programa, o que acaba por facilitar a parte de monitorização do comportamento do programa. Por exemplo, sempre que o programa passa por um determinado ponto são colocadas mensagens que permitem verificar que o programa passou por ali, facilitando assim o processo de *debug*. Já no caso dos erros, sempre que o programa chega a um ponto de erro emite uma mensagem com uma descrição do mesmo, o que facilita o processo de identificação e correção de erros [35] [36].

Algumas das vantagens trazidas por esta funcionalidade são [36]:

- Agilização do processo de *debug*, pois é gerado um relatório com todas as mensagens por onde o programa passou.
- Maior legibilidade e qualidade do código, pois passou a existir um mecanismo específico para o tratamento de mensagens.
- Melhor manutenção dos programas, uma vez que sempre que ocorre um erro é fácil identificar o mesmo e o tempo de identificação e resolução do problema é otimizado.
- Permite armazenar mensagens num ou mais repositórios, mediante configuração, sem que o programa em si tenha que ser alterado.

2.2.3.4 J2SE 5.0

Com o lançamento da versão 5 do Java, em Setembro de 2004, foram introduzidas várias alterações. Deste leque de alterações existem algumas que merecem a pena ser destacadas [16] [37] [38].

Foreach Loop

Uma das grandes mudanças introduzidos nesta versão do Java foi a introdução dos ciclos *foreach*. Esta funcionalidade é um melhoramento dos ciclos previamente existentes.

Antes desta versão, toda a vez que se pretendia percorrer um conjunto de objetos, tipicamente, era feito um ciclo com uma variável, que por norma servia de índice, que ia incrementando para percorrer os diversos objetos.

Com esta novidade, passa a ser possível declarar o tipo de objeto do conjunto que se vai iterar e deixa de existir índice, passando a utilizar o objeto em si para iterar [39] [40].

Através desta abordagem é possível utilizar o objeto que está a servir de iterador para fazer também operações, deixando de ser necessário fazer um `get` pelo índice para conseguir obter este mesmo. As vantagens desta novidade são [41]:

- A redução de erros no código, como por exemplo operações com índices que não sejam válidos.
- Redução da quantidade de código, uma vez que, como já foi referido, deixa de ser necessário fazer operações para obter o objeto pelo seu índice dentro dos ciclos e deixa de ser necessário ter um contador a iterar.
- Melhora a legibilidade do código, pois torna-se mais clara a iteração pelos objetos do conjunto.

Enum

Uma enumeração é uma nova forma de representar dados em Java. Guarda um conjunto de vários dados, que são constantes e do mesmo tipo, e permite que uma variável tenha um valor do conjunto de valores predefinidos. À semelhança de uma classe uma enumeração pode ter variáveis e métodos. Podem ser guardados valores apenas ou podem ser usadas como uma espécie de dicionário em que para cada entrada existe uma chave e um valor [42].

Algumas das vantagens deste tipo de dados são [43]:

- É um tipo de dados *type-safe*, isto é, uma variável apenas pode assumir valores que estejam presentes na enumeração, caso contrário irá ocorrer um erro na compilação do programa.
- Limita o tipo de inputs para operações, como por exemplo um `switch`, permitindo ter um maior controlo sobre o conjunto de possibilidades que o valor de uma variável pode assumir.
- Permite agrupar valores que sejam usados para o mesmo propósito, permitindo fazer iterações sob estes e tornando o código mais limpo.
- Permite associar valores de variáveis a nomes ou expressões que podem ser utilizadas no código no lugar dos valores. Isto torna o código mais legível.

Annotations

Muitas vezes no código produzido surge a necessidade de inserir informação adicional, embora esta não faça, diretamente, parte do programa. De forma a conseguir introduzir estas informações adicionais, foram disponibilizadas anotações para serem utilizadas na linguagem Java. Estas surgem antes de variáveis, métodos ou classes e são identificadas pelo símbolo @. Uma das utilizações mais comum deste recurso é quando se pretende fazer uma implementação de um método diferente da que estava originalmente definida. Para isto é utilizada a anotação `@Override` que permite ao compilador saber que aquela implementação vai substituir a implementação original daquele método [44] [45].

Algumas das vantagens que esta abordagem trouxe são:

- Permite contornar algumas limitações sintácticas do Java, tal como a verbosidade excessiva de *getters* e *setters* que podem ser evitada com anotações do projecto *Lombok*.
- Facilita a parte das configurações da aplicação uma vez que uma anotação pode substituir um conjunto de linhas de configuração para o compilador.
- Permite reduzir código pois existem anotações que fazem com que, no processo de compilação, sejam gerados ficheiros *bytecode* com métodos, como construtores, `gets` e `sets`, mesmo que estes não estejam definidos no código.

2.2.3.5 Java SE 7

Na versão 7 do Java, que foi lançada em Julho de 2011, foram lançadas algumas funcionalidades, de entre as quais uma que merece ser analisada mais ao pormenor [16] [46] [47].

Try-with-resources

Uma das novidades introduzidas na versão 7 do Java foi a funcionalidade *try-with-resources*. Esta nova funcionalidade permite iniciar um bloco de `try catch` com a instanciação de um recurso, por exemplo um objeto para abrir um ficheiro. Anteriormente, sempre que eram instanciados objetos para leitura de ficheiros, era boa prática fechar os mesmo no bloco `finally` do `try catch`, de forma a libertar recursos. Com esta nova funcionalidade isto deixa de ser preciso pois sempre que um recurso é iniciado com um *try-with-resources*, assim que o programa sair deste bloco de código o recurso aberto é automaticamente fechado [48] [49].

Este mecanismo traz vantagens como:

- Melhora a eficiência do programa, uma vez que liberta recursos que não estejam a ser utilizados automaticamente, em vez de estar dependente do programador.
- Diminui do número de erros do programa, uma vez que garante que não são deixados por exemplo ficheiros abertos que depois possam causar interferência em tentativas futuras de alterar o mesmo ficheiro.
- Melhora a otimização de código pois já não é necessário existirem blocos `finally` de propósito para fechar recursos.

2.2.3.6 Java SE 8

No lançamento da versão 8 do Java, que ocorreu em Março de 2014, devido a se tratar de uma versão com suporte a longo prazo foram lançadas várias novidades importantes [16] [50] [51].

Interface Funcional

A maior mudança que foi introduzida com o lançamento da versão 8 do Java foi a introdução das interfaces funcionais. Estas interfaces permitem aproximar esta linguagem, que tipicamente é imperativa, das linguagens de programação funcionais, como *Haskell* ou *Erlang*. Para isto foram introduzidas duas grandes ferramentas: *Lambda Expressions* e *Stream* API.

Lambda Expressions

As *Lambda Expressions* são expressões que representam funções anónimas, isto é, podem ser criadas sem pertencerem a nenhuma classe em específico. Além disto, podem ser passadas como parâmetro de uma função para serem executadas posteriormente.

Geralmente, este tipo de expressões é utilizado para implementar eventos, como `listeners` ou `callbacks`, ou para efetuar operações funcionais como por exemplo no uso da `Stream` API [52] [53].

***Stream* API**

A `Stream` API é uma interface que fornece um conjunto de métodos que possibilitam ao programador efetuar operações sobre um determinado conjunto de objetos.

O objetivo desta novidade é permitir a iteração sobre todos os elementos do conjunto e, em cada elemento, permitir a realização de operações de filtragem, mapeamento, transformação, entre outras [54] [55].

As vantagens obtidas com a introdução das interfaces funcionais foram [56]:

- Redução de substancial do número de linhas de código necessárias para a realização de uma operação sobre um conjunto de dados
- Melhora a legibilidade do código, pois além de reduzir a quantidade de código aumentam também a clareza das operações do mesmo.
- Aumento da eficiência do programa, pois reduzem o tempo de execução de uma operação.

2.2.3.7 Java SE 9

Em Setembro de 2017 foi lançada a versão 9 do Java que trouxe, entre outras novidades, uma nova abordagem à organização dos projetos [16] [57] [58].

Java Platform Module System

Uma das maiores alterações lançadas no Java 9 foi o seu novo sistema de módulos. Até à versão 8 o topo da hierarquia de um projeto Java era a *package*. Com o lançamento desta nova versão essa posição passou a ser ocupada pelo módulo, sendo possível ter vários *packages* dentro do mesmo módulo. Isto permite que, em cada módulo, sejam especificadas

quais as dependências que existem de outros módulos e também quais os *packages* que vão ser expostos, através do ficheiro "module-info.java"[59].

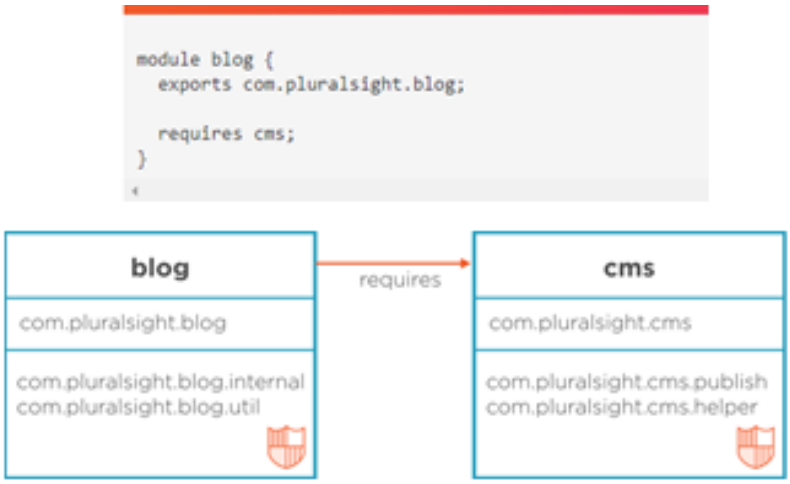

Figura 2.1: *Java Platform Module System*

Apenas os *packages* expostos conseguem ser acedidos por outros módulos. Como é possível visualizar nas imagens acima, através deste novo sistema é possível tornar públicos apenas os *packages* que tenham operações que necessitem de ser acedidas por outros módulos, tornando os outros inacessíveis o que oferece uma camada extra de proteção.

As vantagens desta nova abordagem são [60]:

- Melhorar a organização do código, de modo a conseguir uma melhoria na separação de funcionalidades públicas com funcionalidades internas da aplicação.
- Agilizar o processo de gestão de dependências e facilitar o processo de escalamento de uma aplicação.

2.2.3.8 Java SE 10

No lançamento da versão 10 do Java, em Março de 2018, foram lançadas algumas novidades importantes que vão ser analisadas de seguida [16] [61] [62].

Local-Variable Type

A maior novidade lançada na versão 10 do Java foi o *Local-Variable Type*. A partir desta versão é possível utilizar a inferência na declaração de variáveis. Isto faz com que, mesmo não especificando o tipo da variável, o compilador consiga autonomamente identificar este tipo. Esta característica foi introduzida com a limitação de só pode ser utilizada em três contextos: quando a variável é inicializada na sua declaração, em ciclos *for* ou em ciclos *foreach* [63].

2.2.3.9 Java SE 11

Na versão 11 do Java, cujo lançamento ocorreu em Setembro de 2018, apesar de ser uma versão com *long-term support*, não houve nenhuma novidade que tivesse especial destaque. No entanto, tendo em conta o projeto e a sua necessidade de processamento de strings, considero importante salientar a seguinte novidade [16] [64] [65].

Java String Methods

Nesta versão do Java foram adicionados alguns métodos novos à classe String de modo a conseguir obter mais funcionalidades. Alguns dos métodos adicionados foram [64]:

- `isBlank()` que valida se uma string está vazia ou se só contém espaços em branco;
- `lines()` que retorna uma *stream* com todas as várias linhas contidas na string,
- `repeat()` que retorna a uma string constituída pela repetição da string original um determinado número de vezes,
- `stripLeading()`, `stripTrailing()` e `strip()` que servem para remover os espaços em branco no inicio de uma string, no final e em ambos os casos.

As vantagens da introdução destes novos métodos são:

- Redução da quantidade de código, uma vez que, existindo funções com esse propósito, deixa de ser necessário fazer código para fazer certos tipos de operações com strings.
- Redução de erros de código, uma vez que os programadores deixam de ter de elaborar funções para estes propósitos.
- Melhoramento da legibilidade do código pois os novos métodos têm nomes bastante sugestivos das suas funcionalidades, tornando assim mais fácil entender a operação que está a ser executada.

2.2.3.10 Java SE 12

Da versão 12 do Java, lançada em Março de 2019, existem algumas novidades, que no fundo são melhorias a funcionalidades previamente existentes, que merecem ser destacadas [16] [66] [67].

Switch

A expressão *Switch* sofreu algumas melhorias com o lançamento desta versão. A primeira foi que passou a permitir retornar valores. Até então apenas era possível fazer processamento de dados e igualar os resultados a uma variável que, posteriormente, era utilizada para comunicar os mesmos.

Foram disponibilizadas duas formas para retornar dados. A primeira é através da *keyword* `break`. A utilização desta expressão seguida de um valor ou uma expressão permite que esse valor seja retornado [67].

A segunda forma de retornar valores é outras das novidades introduzidas. A expressão *Switch* passou a suportar o operador *arrow* (`->`) também utilizado nas expressões lambda. Através da utilização deste operador, por substituição da *keyword* `break`, possibilita o retorno de valores.

As vantagens desta novidade são:

- Otimização de código pois deixa de ser necessário utilizar variáveis auxiliares para conseguir obter o resultado de uma operação de um `switch`.

- Melhorias na legibilidade do código, a utilização do operador *arrow* proporciona uma leitura de código muito mais fluída para o programador.

Teeing Collectors

Outra das grandes alterações introduzidas nesta versão foi a adição do método `teeing`. Este método permite a junção do resultado de dois `collects` diferentes através de uma função especificada. Geralmente a operação de `collect` é aplicada a uma `stream` para obter um conjunto de dados resultante de uma operação, por exemplo de uma filtragem. Este método permite cruzar esses dois conjuntos de dados através de uma função que pode gerar várias coisas, desde um novo conjunto de dados até objetos de vários tipos [68].

```
public static void main(String[] args)
{
    List<Employee> employeeList = Arrays.asList(
                                    new Employee(1, "A", 100),
                                    new Employee(2, "B", 200),
                                    new Employee(3, "C", 300),
                                    new Employee(4, "D", 400));

    HashMap<String, Employee> result = employeeList.stream().collect(
                        Collectors.teeing(
                                Collectors.maxBy(Comparator.comparing(Employee::getSalary)),
                                Collectors.minBy(Comparator.comparing(Employee::getSalary)),
                                (e1, e2) -> {
                                    HashMap<String, Employee> map = new HashMap();
                                    map.put("MAX", e1.get());
                                    map.put("MIN", e2.get());
                                    return map;
                                }
                        ));

    System.out.println(result);
}
```

Figura 2.2: *Teeing Collectors*

A vantagem que esta novidade traz é:

- Otimização de código pois leva a uma redução de código visto que permite realizar uma operação definida pelo programador sem que este tenha de recorrer a uma função auxiliar.

2.2.3.11 Java SE 14

Lançada em Março de 2020, a versão 14 do Java trouxe um conjunto de melhorias e novas funcionalidades. Deste grupo de novidades existe uma funcionalidade nova que merece ser destacada [16] [69] [70].

Records

A principal novidade introduzida na versão 14 do Java foi a adição da *keyword* `Record`. Ao longo das versões, um dos aspetos principais que tem sido sempre alvo de melhoria é a otimização do código para que cada vez seja necessário escrever menos código para efetuar uma operação. Foi neste sentido que surgiu esta nova funcionalidade.

Anteriormente a esta versão, sempre que se pretendia declarar um novo tipo de objeto era necessário criar uma classe com os respetivos métodos. De forma a substituir todo este código surgiu esta *keyword*. Um *record* é uma classe que possui construtor com todos os parâmetros e os métodos de `get`, `hashcode`, `equal` e `toString`, sendo estes gerados pelo compilador. No entanto, este tipo de dados tem a particularidade de apenas poder ser utilizado para dados finais, isto é, que sejam imutáveis [71] [72].

```
public record Person (String name, String address) {}
```

Figura 2.3: *Records*

Através de uma simples linha de código é possível ter um novo tipo de dados. É possível também adicionar alguns métodos ou variáveis, no entanto, uma vez que se trata de uma classe que se destina a guardar dados não mutáveis as variáveis adicionadas são estáticas e os métodos não podem alterar o conteúdo dos atributos existentes.

A grande vantagem da implementação desta abordagem são:

- Melhoria na otimização do código, uma vez que ajuda a reduzir o número de linhas de código necessárias em larga escala.
- Melhoria na legibilidade do código, pois existindo menos código torna-se mais fácil ler e conseguir entender o mesmo.

2.2.3.12 Java SE 15

Por fim, na versão mais recente do Java, que é a versão 15, que foi lançada em Setembro de 2020 foram lançadas algumas novidades, de entre as quais há a destacar a seguinte funcionalidade [16] [73] [74].

Sealed

A principal novidade introduzida na versão 15 do Java foi a introdução de um novo tipo de classes e interfaces que é o `Sealed`. O tipo `Sealed` permite às classes e interfaces ter um maior controlo sobre os subtipos que estas suportam, isto é, conseguem restringir as classes que as podem estender ou implementar. Sempre que é utilizada esta anotação é necessário identificar todas as classes às quais se pretende dar permissão, através da *keyword* `permits`, como demonstrado no exemplo abaixo [75] [76].

```
sealed interface Shape
    permits Circle, Rectangle {

      record Circle(Point center, int radius) implements Shape { }

      record Rectangle(Point lowerLeft, Point upperRight) implements Shape { }
}
```

Figura 2.4: *Sealed*

As vantagens desta nova funcionalidade são:

- Aumenta o controlo da segurança do código, uma vez que permite limitar a utilização de classes e interfaces a classes específicas.

- Auxilia a tarefa da modelação pois é garantido que a utilização das classes ou interfaces é apenas para um propósito específico.

2.3 Processo de Migração de código

Uma vez que um dos objetivos deste estágio consiste na atualização da versão da linguagem Java utilizada no projeto PARSEC, um dos principais aspetos a ter sido em conta no que diz respeito ao estudo do estado da arte é o processo de migração de código e técnicas aplicadas ao mesmo.

Neste ponto será feita uma descrição daquilo que é o processo de migração de código, em que consiste e as principais razões que levam um *software* a ter que passar por este processo, e serão também analisadas técnicas que permitem facilitar a realização do mesmo.

2.3.1 Conceito

A área do desenvolvimento de *software* encontra-se em constante evolução. Desde o lançamento dos primeiros *softwares* até aos dias de hoje foram lançadas novas plataformas, ferramentas e até mesmo novas linguagens de programação. Todos os *softwares*, após desenvolvidos, necessitam de manutenção e, muitas vezes, necessitam de passar por um processo de migração de código, consequente da constante evolução desta área.

O processo de migração de código é, como o próprio nome sugere, um processo que contempla todas as tarefas necessárias para que ocorra a mudança do código de um *software*, sem que esta comprometa o seu funcionamento.

2.3.2 Tipos de processos de migração

Existem três tipos principais de migrações de código: migrar a versão da linguagem que está a ser utilizada, migrar o código para uma nova linguagem e migrar o código para uma nova plataforma ou sistema operativo.

O primeiro caso, migração da versão da linguagem, é o que se aplica a este estágio e é também o processo menos complexo dos três. Neste processo as principais alterações ocorrem ao nível da sintaxe de código, sendo que, por vezes, podem existir partes do código que não necessitam de sofrer alterações.

No segundo caso, migração para uma nova linguagem, o processo já é um pouco mais complexo que o anterior. Apesar de ser conhecido o âmbito e os requisitos do projeto não existe qualquer tipo de reaproveitamento de código. Por mais semelhantes que sejam as duas linguagens em casa cada uma tem as suas especificações e, por isso, torna-se necessária a criação de um novo projeto.

Por fim, o terceiro caso, migração para uma nova plataforma ou sistema operativo, é o mais complexo de todos. Aqui, além da necessidade de alteração de código, também presente nos casos anteriores, para adaptar o *software* à nova plataforma existe também a necessidade adicional de configuração. Podendo ser mais ou menos complexa, esta necessidade extra é o principal fator para que este processo seja o mais complexo dos três [77].

2.3.3 Motivos

Após terem sido explicados o conceito de processo de migração e tipos de processos existentes, irão ser descritos de seguida alguns dos principais motivos para a realização de processos de migração.

Um dos motivos que promovem os processo de migração de código é a atualização do *software*. Ao longo do tempo os *softwares* acabam por ficar desatualizados e, apesar de continuarem a funcionar, as tecnologias e ferramentas utilizadas por eles possuem novas versões, deixando as mais antigas de ter suporte.

Outro dos motivos para que aconteça um processo de migração de código é a performance do software. Com o passar do tempo, além das tecnologias utilizadas por este ficarem desatualizadas o *software*começa a perder performance comparativamente a outros que utilizem ferramentas mais recentes. Isto acontece não porque o *software*em si não se torna mais lento ou mais dispendioso mas porque surgem novas ferramentas que têm um maior grau de eficiência comparativamente com as mais antigas.

A adaptabilidade e o suporte a novas plataformas ou sistemas é outra das principais razões que tornam os processo de migração de código necessários. Em muitos casos, o *software*é pensado e estruturado inicialmente para um propósito e, com o desenvolvimento e utilização do mesmo, surgem novas condições de utilização do mesmo. Nestes casos é necessário reestruturar o código oportunidades e, por vezes, repensar a arquitetura do *software*em si para que possa suportar novas funcionalidades, plataformas ou sistemas [78].

2.3.4 Boas práticas

O processo de migração de código possui um conjunto de boas práticas, assim como o processo de desenvolvimento, que devem ser aplicadas para que este decorra da melhor forma possível. Posto isto, de seguida irão ser apresentadas um conjunto de boas práticas relativas a este processo.

Em primeiro lugar deve ser feita uma análise do software, devem ser avaliados a sua estrutura, as suas dependências e as ligações com outras aplicações ou sistemas. Esta fase é crucial para todo o processo de migração, uma vez que permite identificar o estado atual do projeto e definir requisitos com base neste.

Em segundo, é importante garantir a integridade das bases de dados de produção, caso estas existam. Para o processo de migração deve ser utilizada uma cópia da base de dados, mantendo sempre os dados de produção de backup para, caso seja necessário, utilizar.

Um ponto importante neste processo será também, caso existam alterações de dados, preparar a estrutura da base de dados para receber os novos tipos de informação. Como dito anteriormente deve ser preservada uma versão de backup e deve ser preparada a migração de forma a efectuar o mínimo de alterações necessárias.

Outra dos aspetos a ter sido em conta durante o processo de migração é a separação entre este e as operações diárias utilizadas pelos clientes. Com isto pretende-se que, mesmo durante este processo. o utilizador consiga efetuar as tarefas que costuma realizar no seu dia a dia. Este aspeto é importante para garantir que, mesmo durante a migração, o *software*continua a ser funcional e que os seus utilizadores conseguem tirar partido do mesmo.

Outra boa prática do processo de migração de código é que deve ser dada prioridade ao

funcionamento do programa e, posteriormente, proceder a alterações estruturais. Esta abordagem quase que divide o processo de migração em duas fases, uma primeira onde é o garantido que o programa continua a funcionar de maneira correta e uma segunda onde são feitos melhoramentos, *refactoring*.

Por fim, o último aspeto a destacar é o processo de testes. Como em qualquer operação relacionada com código é de extrema importância realizar testes para garantir o bom funcionamento e integridade do programa [79] [80].

2.4 Processo de *Refactoring* de código

No seguimento do contexto da secção anterior surge o tema *Refactoring* de código. Na grande maioria dos casos os processos de migração e de *refactoring* de código estão relacionados. Ambos compreendem alterações feitas a nível de código, sendo a migração para efeitos de atualização, performance ou funcionalidades enquanto que o *refactoring* se aplica para melhorar organização e legibilidade de código.

2.4.1 Conceito

Na área do desenvolvimento, assim como em muitas outras áreas, é difícil elaborar um produto que chegue a uma versão final logo à primeira. Tipicamente é concebida uma primeira versão funcional e, posteriormente, esta vai sofrendo algumas alterações, como correções de *bugs* ou melhoramentos de código.

O conceito de *refactoring* consiste na alteração da estrutura interna do código sem alterar o funcionamento externo do software. Permite melhorar a organização do código e a sua legibilidade, afetando assim indiretamente a tarefa de manutenção do mesmo. Em alguns casos pode ainda melhorar atributos como performance, escalabilidade e segurança ou até mesmo descobrir *bugs* que estavam escondidos até então [81] [82] [83].

2.4.2 Motivos

Os processos de *refactoring* são bastantes importantes na área da programação, uma vez que permitem efetuar melhorarias ao *software* que podem trazer bastantes benefícios, como referido no ponto anterior.

Posto isto, de seguida irão ser indicadas algumas das principiais razões que levam os programadores a efectuar *refactoring* ao código de um *software*.

A primeira razão pela qual se deve fazer *refactoring* do código é para manter o mesmo o mais claro possível, evitando repetições de código, métodos muito longos e demasiados parâmetros nos métodos. Isto faz com que o código seja mais fácil de ler e manter.

A segunda razão é para poupar tempo e dinheiro. Uma vez tendo um código bem estruturado e claro torna mais fácil no futuro a adição de novas funcionalidades ou correção de bugs, uma vez que o programador não terá de perder muito tempo para entender o código.

Outra das razões pelas quais se faz *refactoring* de código é para tornar este mais rápido e mais eficiente. A eliminação de classes, métodos ou variáveis que sejam desnecessárias contribui não só para a simplicidade e legibilidade do código mas também para o seu desempenho.

Por fim, um dos grandes motivos de se efectuar um processo de *refactoring* de código é para atualizar o mesmo. Quanto mais tempo passa sem que um software seja atualizado maiores são as probabilidades do seu código estar desatualizado e da linguagem e das ferramentas utilizadas deixarem de ter suporte. Nestas situações ou para evitar as mesmas é necessário efetuar periodicamente atualizações ao código desenvolvido [83] [84].

2.4.3 Técnicas

Para finalizar a análise aos processos de *refactoring* de código, vão ser analisadas de seguida algumas das principais técnicas utilizadas neste tipo de processo.

Uma das principais técnicas utilizadas nos processos de *refactoring* é a técnica *Red-Green-Refactor*. É tipicamente utilizada em desenvolvimento *Agile*. Consiste em particionar o processo em três passos. Inicialmente é verificado o que precisa de ser desenvolvido. De seguida é desenvolvido o código estritamente necessário e o mais simples possível que permita passar nos testes das funcionalidades desenvolvidas. Por fim, surge a ação de *refactoring* em que o código produzido é melhorado e otimizado sem adicionar nenhuma funcionalidade ou comprometer o funcionamento anterior.

Outra das técnicas utilizadas nestes processos é *Refactoring By Abstraction*. É tipicamente utilizada quando o processo de *refactoring* tem uma dimensão considerável e tem como propósito principal a remoção de duplicações de código. Para isto está técnica utiliza como recurso principal a migração de métodos ou variáveis da classe super para as suas descendentes e vice-versa.

Uma das técnicas mais utilizadas nestes processos é *Movig Features Between Objects*. Esta técnica é utilizada quando existem classes sobrecarregadas de código. Consiste na criação de novas classes e na repartição do código pelas mesmas de forma a que se mantenha o funcionamento e que cada classe fique com um propósito bem definido.

Por fim, a ultima técnica a ser analisada é *Preparatory Refactoring*. Esta técnica é utilizada quando no processo de desenvolvimento de uma nova funcionalidade o programador se depara com a necessidade de fazer *refacoring* ao código. Neste caso é desenvolvida a funcionalidade e feito o *refactor* ao código utilizado pela mesma. Assim, apesar de continuar a existir código que necessite de alterações, fica já parte do processo de *refactoring* realizado e a nova funcionalidade desenvolvida fica já feita dessa forma também [85] [86] [87].

This page is intentionally left blank.

Capítulo 3

Análise Funcional

Neste capítulo irá ser feita uma análise, a nível funcional, à biblioteca Parse & Correlate (PARSEC). Com esta análise pretende-se identificar os componentes que constituem esta biblioteca, a forma como eles interagem entre si e de que forma é que cada um deles é utilizado em cada uma das funcionalidades disponibilizadas para que, posteriormente, no processo de migração exista um conhecimento prévio do funcionamento da biblioteca.

De modo a conseguir representar as funcionalidades desta biblioteca, os componentes que a constituem e o fluxo de dados ao longo das várias operações vão ser elaborados vários diagramas, recorrendo à ferramenta *Enterprise Architect*.

3.1 Funcionalidades

O PARSEC é uma biblioteca que disponibiliza várias funcionalidades. Para garantir uma melhor organização, a nível estrutural, estão divididas em vários módulos. Os módulos principais do projeto são: *parse*, *correlate* e *macSignatureInfo*.

3.1.1 Funcionalidades do módulo *Parse*

O módulo *parse* é utilizado sempre que um *DCC Service User* recebe uma resposta vinda de um contador. Todos os contadores utilizam a mesma linguagem, Great Britain Companion Specification (GBCS), para comunicar, quer seja entre si ou com outras entidades. Por sua vez, os *DCC Service Users*, utilizam uma linguagem diferente para comunicar, que neste caso é XML.

Tendo em conta o que foi descrito anteriormente, existe uma necessidade de tradução de pedidos tanto de XML para GBCS como vice-versa. No primeiro caso, essa conversão é assegurada pela Data Comunications Company (DCC). Para conseguir obter a tradução no sentido inverso é necessário utilizar a funcionalidade *parse* da biblioteca PARSEC.

Sempre que um *DCC Service User* recebe uma resposta de um contador, envia a mesma para o *endpoint* `parse` do PARSEC e a resposta recebida é a conversão do GBCS enviado para um *XML*, denominado de Message Mapping Catalogue (MMC), utilizado pelo *DCC Service User*.

Caso a resposta obtida contenha algum dos campos encriptado, é necessário realizar esta operação por duas vezes. Na primeira, irá ser retornado um XML com os campos en-

criptados assinalados. Com esta resposta, o *DCC Service User* desencripta os conteúdos assinalados, substitui os mesmos pelo resultado da sua desencriptação e volta a submeter o mesmo pedido, agora com a informação desencriptada. Como resultado deste segundo pedido, o *DCC Service User* irá receber um XML com todos os campos preenchidos com informação que ele consiga entender.

3.1.2 Funcionalidades do módulo *Correlate*

O módulo *correlate* é utilizado sempre que o *DCC Service User* pretende enviar um comando crítico para o contador, como por exemplo alterar o preço de uma tarifa.

A operação de envio de um comando crítico para um contador, por parte de um *DCC Service User*, é dividida em duas partes. Inicialmente o comando crítico é enviado para o *endpoint* `transform` da DCC, para que possa ser convertido para GBCS, denominado de *Pre-Command*. Quando o *DCC Service User* recebe o *Pre-Command*, resultante da transformação inicial, necessita de validar que a operação de conversão do comando foi feita corretamente e que todos os valores presentes nos diversos campos do pedido estão corretos.

Para efetuar esta validação, o *DCC Service User* envia o pedido original e o *Pre-Command* recebido para o *endpoint* `correlate` do PARSEC. Este vai fazer a comparação entre os dois pedidos e verificar se eles são equivalentes ou se ocorreu algum erro durante a conversão e devolver o resultado desta operação.

3.1.3 Funcionalidades do módulo *MacSignatureInfo*

O módulo *macSignatureInfo*, ao contrário dos dois anteriores, não tem apenas uma, mas sim três finalidades para os *DCC Service Users*. Este módulo é utilizado no processo de assinatura de comandos críticos e como auxiliar no processo de validação criptográfica das respostas vindas dos contadores, sendo que este segundo ponto se divide em dois casos: pedido crítico e não crítico.

No caso da assinatura de pedidos críticos, este módulo é utilizado pelo *DCC Service User* logo depois deste fazer a validação do *Pre-Command*. Uma vez validado, é enviado para o *endpoint* `macSignatureInfo` do PARSEC de forma a conseguir obter o conjunto de propriedades necessárias para a assinatura.

Nas respostas vindas dos contadores, os *DCC Service Users* utilizam este módulo para extrair os componentes necessários para efectuar as validações criptográficas da assinatura ou do Message Authentication Code (MAC), consoante a resposta seja crítica ou não crítica.

No caso de ser uma resposta a um pedido não crítico, são extraídos os componentes necessários para efectuar a validação criptográfica do MAC. Caso seja uma resposta a um comando crítico, é identificada a assinatura dos campos do *Pre-Command*. Após obter esta informação o *DCC Service User* procede à validação criptográfica da mesma.

3.2 Diagramas Funcionais

Com o objetivo de complementar a análise às funcionalidades feita no ponto anterior, foram elaborados diagramas funcionais para ajudar na compreensão e perceção dos comportamentos do programa. De seguida vão ser apresentados os vários tipos de diagramas

elaborados.

3.2.1 Diagrama de Casos de Uso

O primeiro passo realizado na análise funcional à biblioteca PARSEC foi a identificação das funcionalidades disponibilizadas pela mesma. Como resultado desta análise foi elaborado o diagrama de casos de uso que será apresentado de seguida.

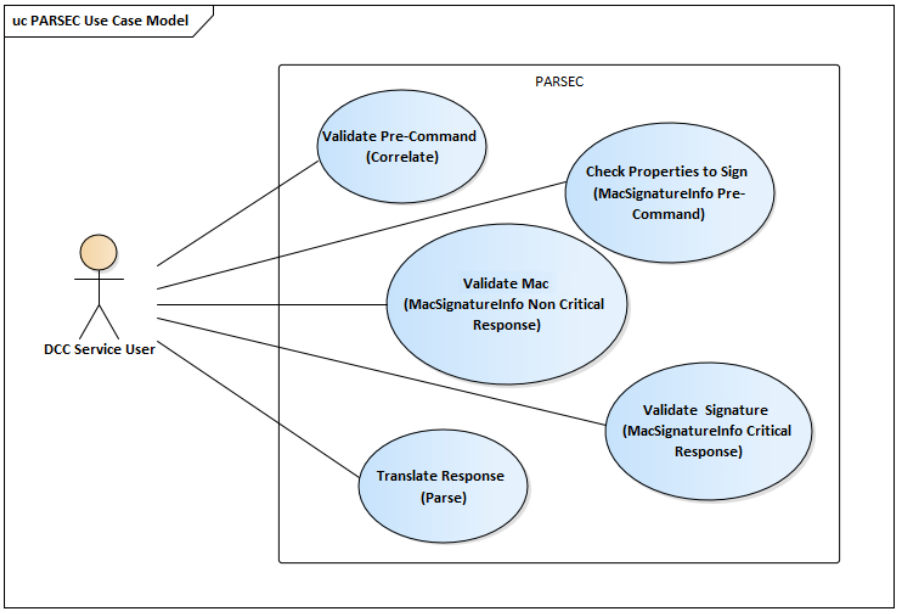

Figura 3.1: Diagrama de Casos de Uso

Analisando o diagrama apresentado em cima, o primeiro aspeto a ser tido em consideração é o facto de que o projeto PARSEC tem apenas um tipo de utilizador, *DCC Service User*. Este corresponde a todas as entidades autorizadas que pretendem comunicar com os contadores, fornecedores de energia, operadores de rede e entidades externas autorizadas.

Além disto, é possível visualizar que esta biblioteca disponibiliza aos seus utilizadores 5 funcionalidades diferentes. Estas estão representados no diagrama através dos seguintes casos de uso: validar *Pre-Command*, extrair a informação do *Pre-Command* necessária para assinar, validar o *mac* da resposta, verificar a assinatura da resposta e fazer *parse* da resposta.

De seguida serão apresentados os casos de uso, desta vez no formato de tabela.

Caso de Uso FR-1	Validar tradução de um comando crítico
Ator	*DCC Service User*
Descrição	O *DCC Service User*, após ter recebido a tradução GBCS do comando crítico efetuada pela DCC, envia o comando inicial e a sua tradução para o *endpoint* correlate. Como resposta este receberá o resultado da comparação entre os dois.

Tabela 3.1: Caso de Uso Validar tradução de um comando crítico

Caso de Uso FR-2	Extrair a informação do *Pre-Command* necessária para assinar
Ator	*DCC Service User*
Descrição	Depois de validar a tradução de um comando crítico e antes de o enviar, a mensagem GBCS tem que ser assinada. Para isso, o *DCC Service User* envia o comando para o *endpoint macSignatureInfo*. Como resposta ao pedido, este irá receber a informação necessária para assinar a mensagem. Depois de assinar a mensagem, o *DCC Service User* submete o comando.

Tabela 3.2: Caso de Uso Extrair a informação do *Pre-Command* necessária para assinar

Caso de Uso FR-3	Extrair informação para validar o MAC
Ator	*DCC Service User*
Descrição	Após receber uma resposta vinda de contador, o *DCC Service User* envia esta para o *endpoint macSignatureInfo*. Para uma resposta não crítica, este *endpoint* devolve a informação necessária para validar o MAC. Depois de receber essa informação, o *DCC Service User* utiliza o material criptográfico relevante para validar o MAC da mensagem.

Tabela 3.3: Caso de Uso Extrair informação para validar o MAC

Caso de Uso FR-4	Extrair informação para validar a assinatura
Ator	*DCC Service User*
Descrição	Depois de receber uma resposta crítica, o *DCC Service User* necessita de validar a assinatura da mensagem. Para isto, este envia a resposta recebida para o *endpoint macSignatureInfo*. Como resposta a este pedido, o *DCC Service User* recebe a informação necessária para validar a assinatura. Depois disto utiliza o material criptográfico relevante para efectuar a validação.

Tabela 3.4: Caso de Uso Extrair informação para validar a assinatura

Caso de Uso FR-5	Traduzir resposta de um contador
Ator	*DCC Service User*
Descrição	O *DCC Service User* envia o GBCS recebido do contador para o *endpoint* parse. Como resposta este receberá a tradução da resposta para um formato XML. Caso este XML contenha campos identificados como encriptados, o *DCC Service User* necessita de fazer a tradução dos mesmos. Depois disto substitui o conteúdo encriptado pela respetiva desencriptação e volta a submeter o pedido. A resposta a este segundo pedido será a conversão da mensagem para XML com todos os campos traduzidos.

Tabela 3.5: Caso de Uso Traduzir Resposta

3.2.2 Diagrama de Componentes

Depois de identificadas as funcionalidades presentes na biblioteca PARSEC a tarefa seguinte passou por identificar os componentes que a constituem. De seguida será apresentado um diagrama de componentes de modo a ilustrar os componentes que intervenientes no projeto bem como as interações entre eles.

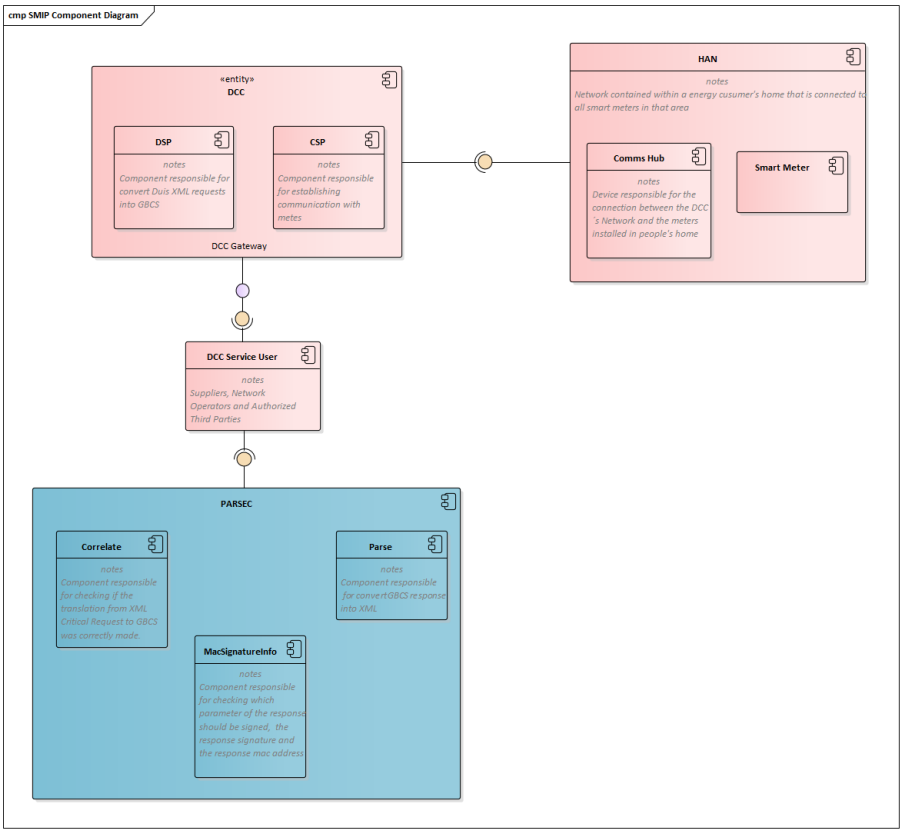

Figura 3.2: Diagrama de Componentes

Começando por analisar o PARSEC, é possível visualizar que este é composto por 3 componentes que são: *parse, correlate* e *macSignatureInfo*. Estes correspondem aos módulos que agrupam as diversas funcionalidades disponibilizadas.

Além do PARSEC, estão também representados neste diagrama outros componentes que pertencem ao Smart Metering Implementation Programme (SMIP) que são: *DCC Service User*, DCC e Home Area Network (HAN).

Posto isto, o PARSEC apenas interage com o *DCC Service User*. Este por sua vez interage também com a DCC que, por fim, estabelece a comunicação com a HAN, chegando assim à comunicação ao contador.

3.2.3 Diagramas de Atividade

Uma vez identificados os componentes que constituem o projeto PARSEC e as interações entre os mesmos, a tarefa seguinte passou pela elaboração de diagramas de atividade que contemplassem os casos de uso identificados.

Inicialmente serão apresentados os diagramas com os componentes respetivos ao SMIP passando depois para uma análise apenas ao âmbito do PARSEC.

3.2.3.1 Receber mensagem

Abaixo encontra-se representado o diagrama de atividade correspondente à operação de receber uma resposta por parte de um *DCC Service User*. No mesmo, a azul, encontram-se representadas as atividades correspondentes a funcionalidades da biblioteca PARSEC. Estas serão apresentadas e analisadas em mais detalhe nos pontos seguintes.

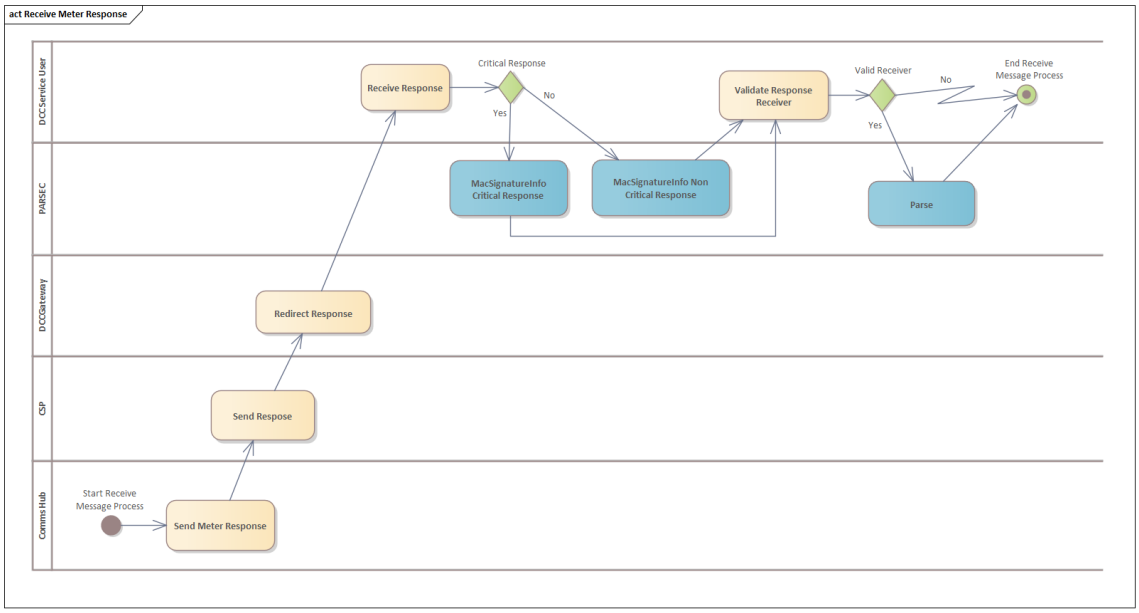

Figura 3.3: Diagrama de Atividade SMIP Receber Resposta

Esta operação começa com o envio da resposta por parte da HAN. Passa pelo Communication Service Provider (CSP) que através da *DCC Gateway* faz com que esta chegue ao *DCC Service User*.

Após receber a resposta, o *DCC Service User* verifica o tipo da mesma. Consoante seja uma resposta crítica ou não, este recorre à funcionalidade *macSignatureInfo*, na vertente resposta crítica ou não crítica, do PARSEC.

De seguida é verificado se a entidade que recebeu a mensagem corresponde ao destinatário presente na resposta recebida. Caso não seja este processo é interrompido de imediato. Caso seja, o *DCC Service User* utiliza a funcionalidade *parse* do PARSEC, obtendo a tradução da resposta como resultado desta ação.

3.2.3.2 Extrair informação para validar o MAC

De seguida vai ser mostrado o diagrama de atividade correspondente à operação de estração de informação para validar o MAC de uma mensagem não crítica, disponibilizado pelo PARSEC.

Este diagrama surge no seguimento do contexto do diagrama apresentado anteriormente tendo, por isso, como base que o *DCC Service User* recebeu uma resposta não crítica vinda do contador.

Neste caso, a primeira tarefa a ser executada é enviar a mesma para o *endpoint macSignatureInfo* do PARSEC. Este devolve a informação necessária para validar o MAC.

Quando recebe o resultado da operação realizada, o *DCC Service User* utiliza o material

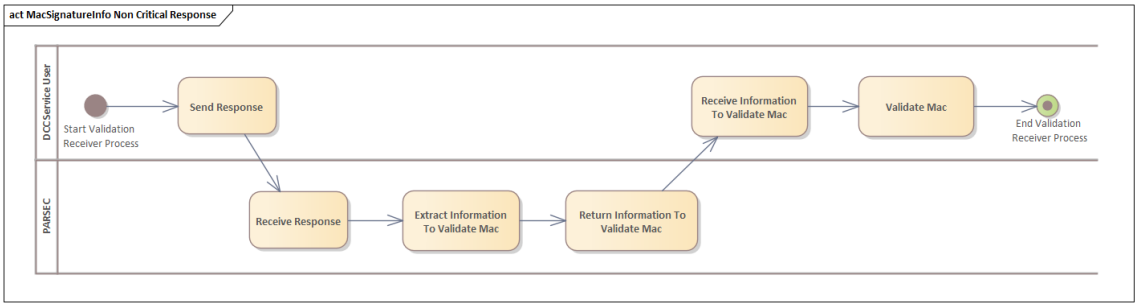

Figura 3.4: Diagrama Atividade Extrair informação para validar o MAC

criptográfico recebido para validar o MAC da mensagem.

3.2.3.3 Extrair informação para validar a assinatura resposta crítica

Tal como o diagrama anterior, este também se insere no mesmo contexto da receção de uma resposta, desta vez crítica, por parte de um *DCC Service User*. Posto isto, de seguida será apresentado o diagrama de atividade correspondente à operação de extrair informação necessária para validar a assinatura uma mensagem crítica.

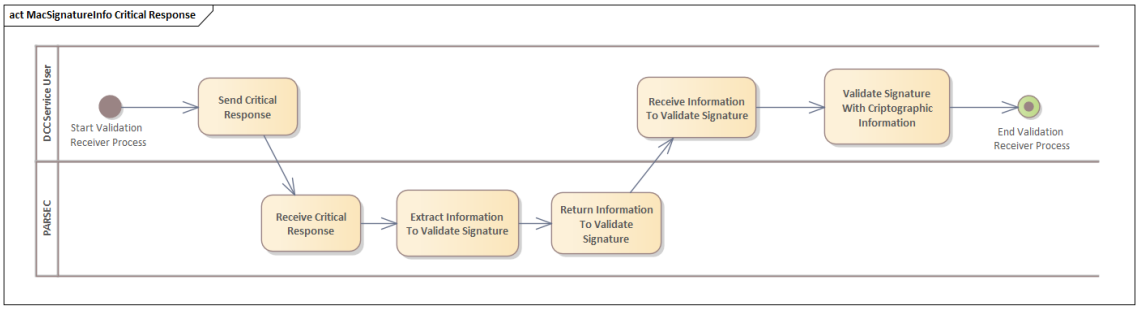

Figura 3.5: Diagrama Atividade Extrair informação para validar a assinatura resposta crítica

Analisando o diagrama apresentado acima e comparando o mesmo com o apresentado no ponto anterior é possível verificar que estes são tudo semelhantes. A diferença entre estes dois diagramas é que, neste caso, como se trata de uma resposta critica o parâmetro a validar é a assinatura. Depois de obtida a informação criptográfica, esta é utilizada para validar a assinatura presente na resposta.

3.2.3.4 Traduzir resposta

Uma vez aplicadas as validações criptográficas à resposta, a operação que se segue é a de traduzir a resposta. Seguidamente irá ser representado e analisado o diagrama de atividade correspondente a esta operação.

A primeira tarefa respetiva à operação de tradução de uma resposta de um contador é o envio da mesma para o *endpoint parse*, para que seja feita tradução de GBCS para XML.

Depois de feita a tradução, o *DCC Service User* verifica a presença de campos encriptados. Caso não existam, o processo é concluído com sucesso. No entanto, se existirem campos encriptados, o *DCC Service User* necessita de proceder a desencriptação do conteúdo dos

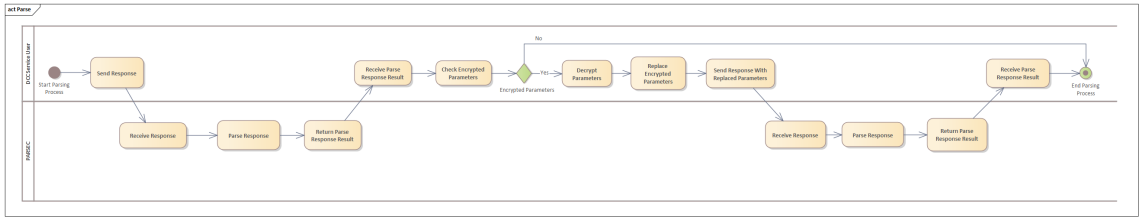

Figura 3.6: Diagrama Atividade Traduzir Resposta

mesmos. De seguida, substitui a informação encriptada pelo resultado obtido e volta a submeter a resposta recebida, desta vez sem campos encriptados.

Como resultado da segunda submissão, o *DCC Service User* irá receber a tradução da resposta completa e, então, o processo é concluído com sucesso.

3.2.3.5 Enviar comando crítico

Seguidamente vai ser apresentado o diagrama de atividade correspondente à operação de enviar um comando crítico. Neste é possível encontrar, marcadas a azul, as atividades correspondentes a utilização de funcionalidades disponibilizadas pela biblioteca PARSEC. Nos pontos seguintes será feita uma análise das mesmas.

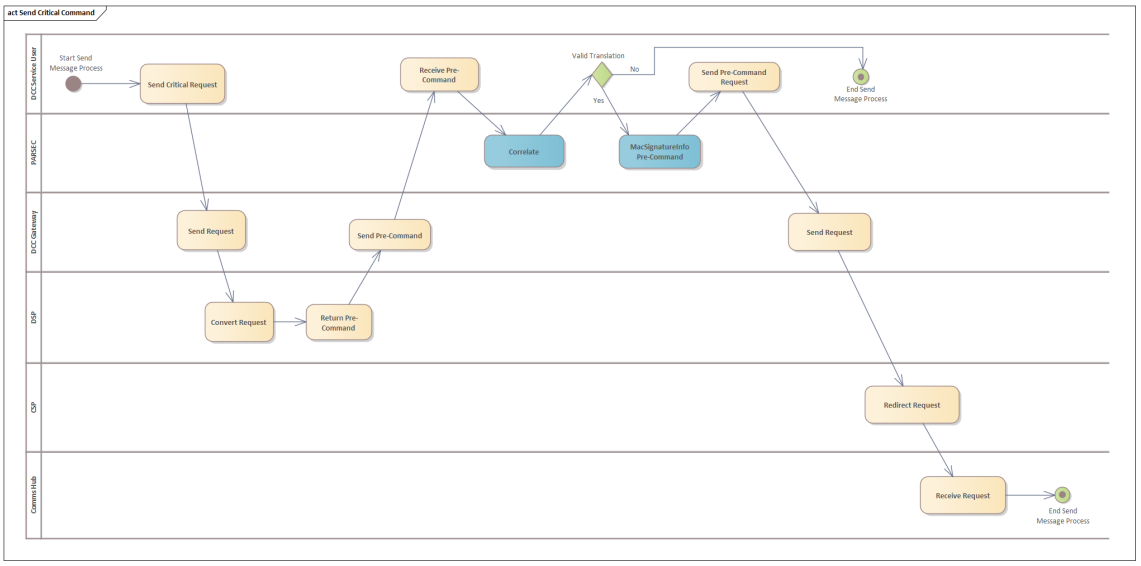

Figura 3.7: Diagrama Atividade SMIP Enviar Comando Crítico

O processo de enviar um comando crítico começa com o envio do mesmo por parte do *DCC Service User* para a *DCC Gateway* que o reencaminha para o Data Service Provider (DSP). Este converte o comando enviado em GBCS e envia o resultado desta operação de volta.

Após receber a tradução do comando enviado, o *DCC Service User* verifica, através da utilização da funcionalidade *correlate* do PARSEC, se a tradução foi feita de forma correta. Se o resultado desta tarefa for negativo, o processo para de imediato. Caso contrário, o *DCC Service User* procede à verificação dos campos que necessitam de assinatura.

Tendo recebido a informação dos campos que precisam de assinatura, o *DCC Service User* procede à assinatura dos mesmos e, por fim submete o comando crítico assinado para a

DCC Gateway que o reencaminha para o CSP. Este por sua vez faz com que o pedido chegue ao Communications Hub (Comms Hub).

3.2.3.6 Verificar tradução comando crítico

Abaixo encontra-se representado o diagrama de atividade correspondente à operação de verificar a tradução de um comando crítico, presente na biblioteca PARSEC.

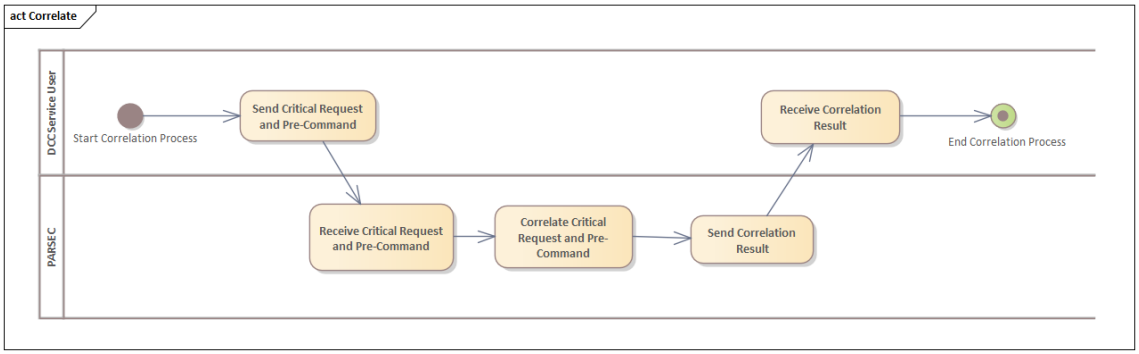

Figura 3.8: Diagrama Atividade Verificar Tradução Comando Crítico

Uma vez que esta operação ocorre no seguimento do contexto fornecido pelo diagrama anterior, é tido em conta que o *DCC Service User* recebeu o *Pre-Command* correspondente à tradução do comando crítico previamente enviado.

Posto isto, a primeira tarefa a ser executada no processo de verificação da tradução é enviar o pedido original e o *Pre-Command* para o *endpoint correlate* do PARSEC.

Este verifica se a tradução corresponde ao comando original ou se existem campos que não foram traduzidos corretamente. O resultado desta verificação é devolvido para o *DCC Service User*.

3.2.3.7 Verificar propriedades para assinar comando crítico

Sendo o resultado da operação de verificar a tradução de um comando crítico positivo, o processo avança e segue-se a operação de verificar os campos do comando que precisam de assinatura. O diagrama de atividade correspondente a esta operação vai ser analisado de seguida.

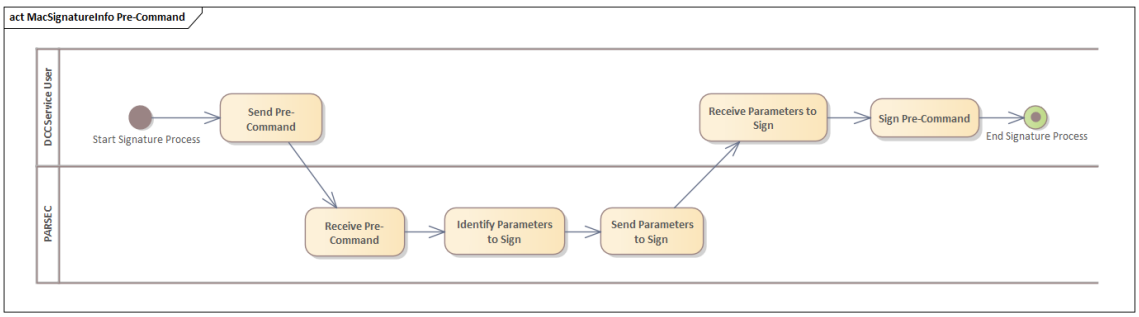

Figura 3.9: Diagrama Atividade Verificar Propriedades Para Assinar Comando Crítico

Inicialmente, o *DCC Service User* submete o *Pre-Command* ao *endpoint macSignatureInfo* para verificar os atributos que são necessários assinar. Este analisa o *Pre-Command* e

devolve os campos que necessitam de assinatura.

Após receber a resposta ao pedido efectuado, o *DCC Service User* assina os campos assinalados e submete o comando devidamente assinado.

3.2.4 Diagramas de Sequência

Após terem sido elaborados os diagramas de Atividade, a tarefa efetuada de seguida foi a elaboração de diagramas de Sequência correspondentes. Estes permitem analisar com mais detalhe o fluxo de informação ao longo do sistema PARSEC.

3.2.4.1 Receber mensagem

Seguidamente irá ser apresentado o diagrama de sequência correspondente à operação de receber uma resposta por parte de um *DCC Service User*. À semelhança do que acontecia nos diagramas de atividade, também nestes irão ser representadas as funcionalidades da biblioteca PARSEC recorrendo à cor a azul.

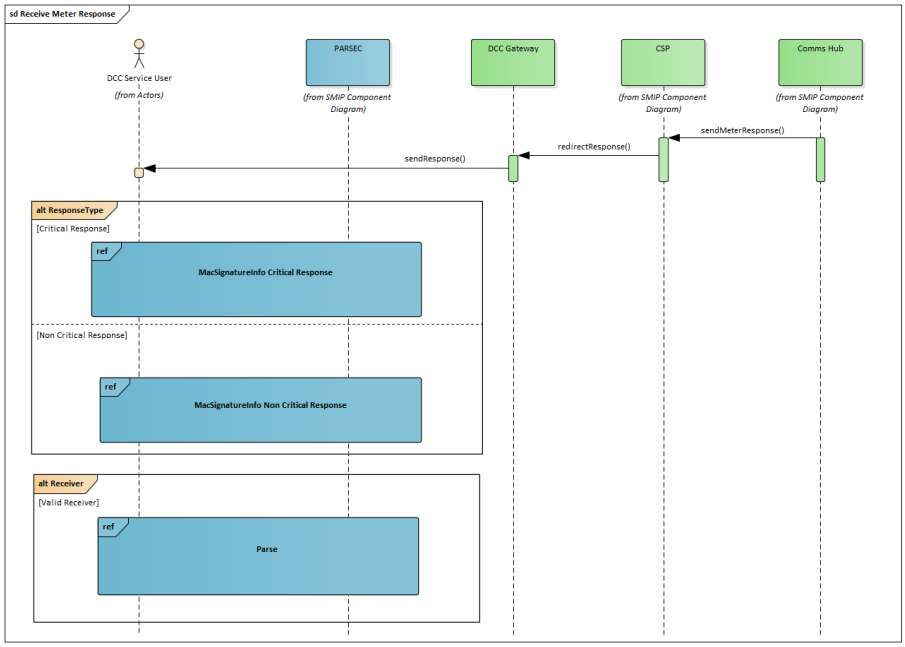

Figura 3.10: Diagrama de Sequência SMIP Receber Resposta

Inicialmente o Comms Hub envia a resposta vinda do contador para a DSP que através da *DCC Gateway* faz com que esta chegue até ao *DCC Service User*.

Uma vez recebida a resposta,a primeira operação a efetuar é a validação do destinatário da mesma, que pode variar consoante se trate de uma resposta crítica ou não crítica.

Depois de validado o destinatário da resposta, e para terminar o processo de receber uma mensagem, o *DCC Service User* utiliza o PARSEC para obter a tradução da mesma.

3.2.4.2 Validar MAC numa resposta não crítica

Após ter sido feita uma análise à tarefa de receber uma mensagem, de seguida irão ser analisados os diagramas de sequência relativos a tarefas da biblioteca PARSEC. Abaixo encontra-se representado o diagrama relativo à tarefa de validar o MAC de uma mensagem não crítica.

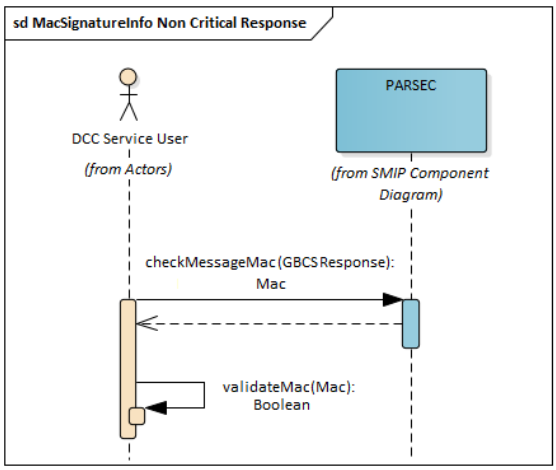

Figura 3.11: Diagrama de Sequência Validar MAC Resposta Não Crítica

Analisando o diagrama acima, é possível visualizar que este começa com o envio da resposta recebida para o PARSEC. Este faz o processamento da mesma e extrai a informação necessária para validar o MAC. Para finalizar, o *DCC Service User* utiliza o material criptográfico relevante para validar o MAC da mensagem..

3.2.4.3 Validar assinatura numa resposta crítica

Neste ponto será analisado o diagrama de sequência correspondente ao processo de validar a assinatura numa resposta crítica.

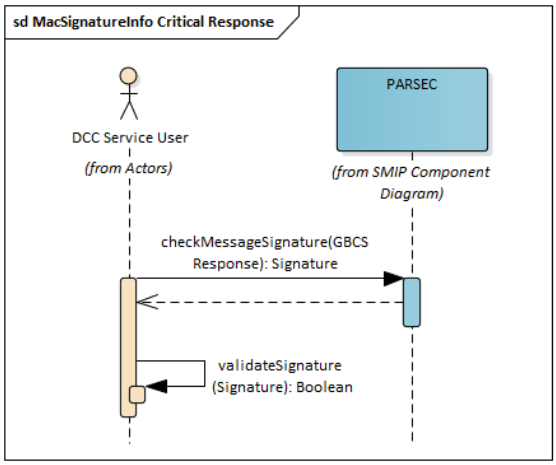

Figura 3.12: Diagrama de Sequência Validar Assinatura Resposta Crítica

O diagrama representado acima é, na sua grande maioria, semelhante ao diagrama apresentado anteriormente. Neste caso o a biblioteca PARSEC é utilizada extrair a informação

necessária para validar a assinatura da mensagem. Após esta operação, o *DCC Service User* utiliza o material criptográfico relevante para validar a mesma.

3.2.4.4 Traduzir resposta

Após terem sido aplicadas as validações criptográficas à resposta segue-se, por fim, a operação de efetuar a tradução da mesma. Abaixo é possível encontrar a representação do diagrama de sequência relativo a essa tarefa.

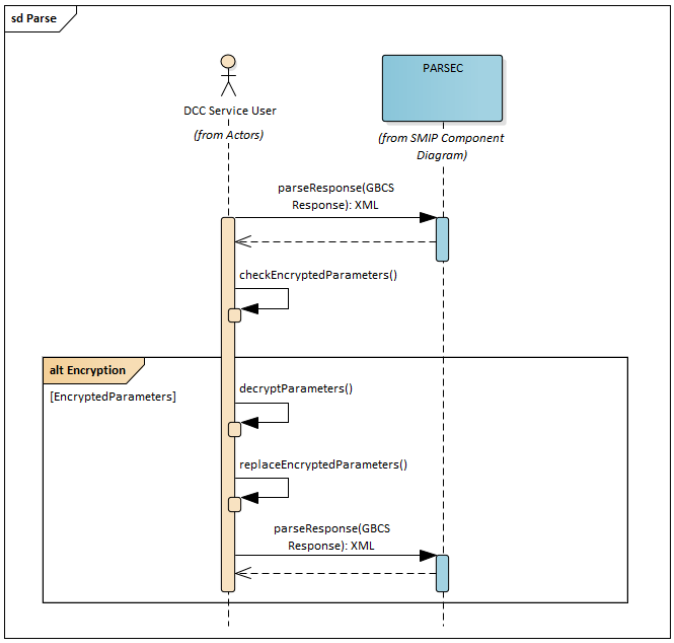

Figura 3.13: Diagrama de Sequência Traduzir Resposta

Na operação de tradução de uma mensagem, a primeira tarefa a acontecer é o envio da resposta ao PARSEC. Este faz a tradução de GBCS para XML e retorna o resultado desta tradução.

Depois disto, o *DCC Service User* verifica se existem campos encriptados e, caso não existam, o processo termina aqui. No entanto, caso existam campos encriptados é necessário proceder à sua desencriptação.

Terminada esta tarefa, é feita a substituição dos valores encriptados pelo resultado obtido da operação anterior e é efectuada a submissão da resposta recebida com todos os campos desencriptados.

Por fim, como resultado da segunda submissão, o *DCC Service User* recebe a tradução da mensagem enviada pelo contador.

3.2.4.5 Enviar comando crítico

O último processo a ser analisado é o de enviar um comando crítico. De seguida irá ser mostrado o diagrama de sequência correspondente a este mesmo.

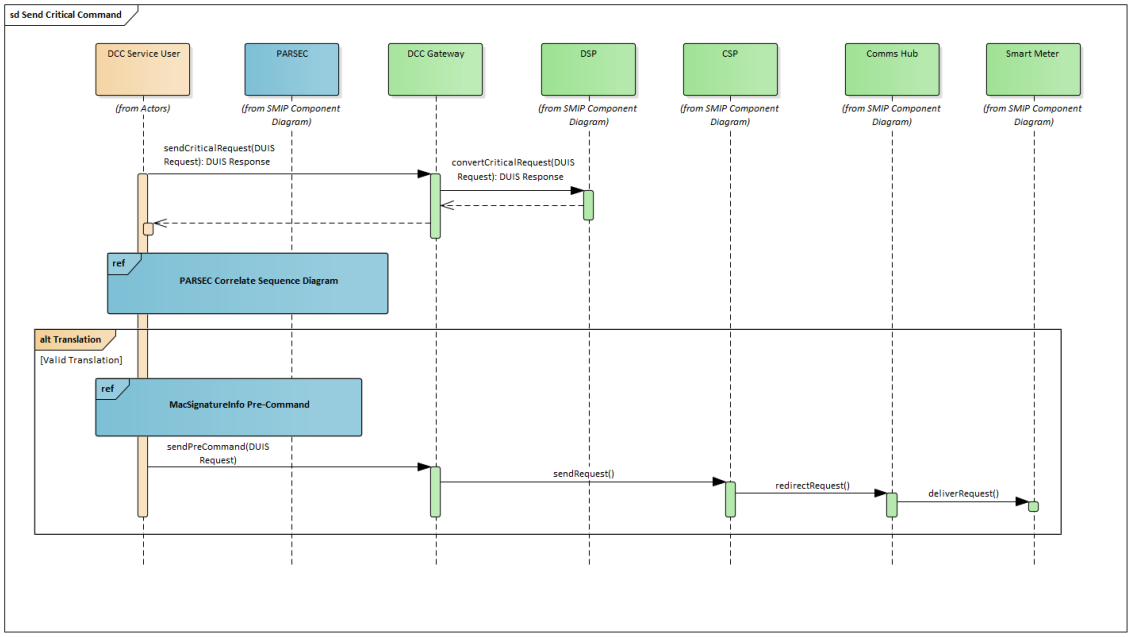

Figura 3.14: Diagrama de Sequência SMIP Enviar Comando Crítico

A primeira tarefa a ser executada é o envido do pedido crítico que, através da *DCC Gateway* chega ao DSP. Aqui é feita a conversão para GBCS que, posteriormente, e devolvida de volta.

Quando é recebido o resultado da tradução, a tarefa seguinte passa por verificar se esta foi efectuada corretamente.

Se existirem erros o processo é interrompido, caso contrário a próxima tarefa é a verificação dos campos do pedido crítico que necessitam de assinatura. Tendo esta informação, o *DCC Service User* procede à sua assinatura e, finalmente, este que passa pela *DCC Gateway*, CSP e Comms Hub até chegar ao contador

3.2.4.6 Verificar tradução comando crítico

Neste ponto irá ser analisado o diagrama de sequência correspondente à operação de verificar a tradução de um comando crítico.

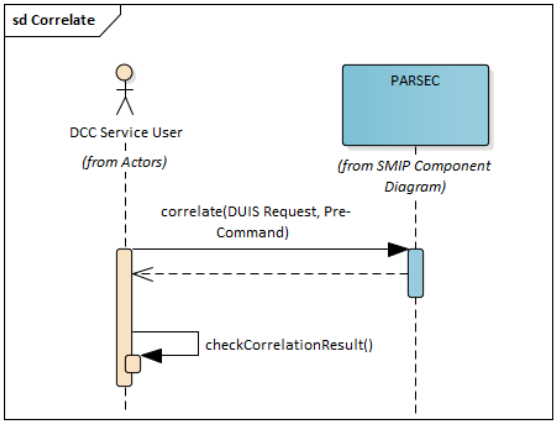

Figura 3.15: Diagrama de Sequência Tradução Comando Crítico

Analisando o diagrama apresentado acima é possível visualizar que a primeira tarefa realizada nesta operação é o envio do pedido com o comando crítico original e a sua tradução para o PARSEC.

Este, por usa vez, vai fazer a comparação de ambos e devolver o resultado dessa comparação. Com este resultado o utilizador verifica se a tradução bem ou mal sucedida,

3.2.4.7 Verificar propriedades para assinar comando crítico

Tendo sido validada a tradução efectuada, a tarefa que se segue é a verificação das propriedades do comando que requerem assinatura.

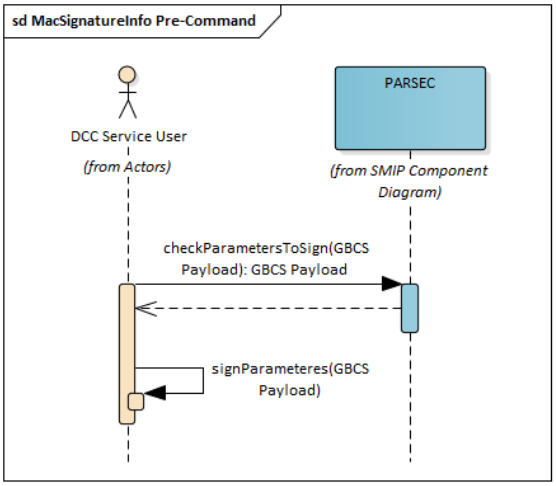

Figura 3.16: Diagrama de Sequência Verificar Propriedades Para Assinar Comando Crítico

Nesta operação, a primeira tarefa corresponde ao envio do *Pre-Command* ao PARSEC. Este analisa todos os campos e retorna todos que requerem assinatura por parte do *DCC Service User*.

Depois disto, o *DCC Service User* assina os campos assinalados e submete o comando para que este seja reencaminhado para o contador.

Capítulo 4

Análise de Requisitos

Nesta será feita uma análise dos requisitos relativos a este projeto. Vão ser identificados e descritos os requisitos funcionais e não funcionais e será ainda apresentado um plano de validação que terá como objetivo a validar, no final do trabalho desenvolvido, se todos os requisitos foram cumpridos com sucesso.

4.1 Requisitos Funcionais

Como referido anteriormente, o objectivo principal deste estágio é atualizar a versão da linguagem utilizada na biblioteca Parse & Correlate (PARSEC). Este processo não contempla quaisquer alterações ao nível de funcionalidades, sendo o principal objetivo fazer com que o projeto PARSEC continue a usufruir de uma versão do Java que forneça *long-term support*.

Para alcançar este objetivo é necessário garantir que no final do processo de atualização as funcionalidades disponibilizadas inicialmente continuam funcionais, sendo estas:

- Traduzir a resposta
- Validar tradução de um comando crítico
- Extrair a informação necessária para validar o Message Authentication Code (MAC)
- Extrair a informação necessária para validar a assinatura
- Extrair a informação do *Pre-Command* necessária para assinar

4.2 Requisitos não funcionais

Os requisitos não funcionais, também denominados de atributos de qualidade, têm como objetivo definir uma gama de valores de um determinado parâmetro que, por norma, pode ser utilizado para efetuar uma avaliação de um software.

Tendo em conta o facto de o PARSEC se tratar de uma biblioteca e que o principal objetivo é a sua atualização foram considerados alguns requisitos não funcionais que se aplicam a este projeto e que irão ser explicados de seguida.

4.2.1 Desempenho

O principal objetivo deste estágio é a atualização da biblioteca PARSEC. Esta atualização tem como principal objetivo prolongar o período de suporte da linguagem. No entanto, espera-se que com esta atualização esta biblioteca consiga aumentar a sua performance, sendo esperado que os tempos de processamento da mesma desçam pelo menos entre os cinco e os dez porcento. Um resultado abaixo destes valores mas que não melhore a performance é visto como abaixo do esperado, sendo apenas considerado que o requisito não foi cumprido se o tempo de processamento aumentar.

4.2.2 Compatibilidade

Outro dos requisitos não funcionais identificados para esta atualização é a compatibilidade. Uma vez que a biblioteca vai ser atualizada da versão 8 para a versão 11 do Java, no final deste processo é necessário garantir que a mesma consegue funcionar de forma correta num ambiente/servidor que possua a versão 11 do Java Development Kit (JDK).

4.2.3 Confiabilidade

Uma das características a ter em atenção em qualquer tipo de software é a sua confiabilidade, ou seja, garantir que não surgem anomalias durante o seu funcionamento. Uma vez que se trata de um projeto crítico, existem critérios definidos que necessitam de ser cumpridos e um deles é garantir que não existem bugs na sua implementação. Para isto, a biblioteca PARSEC é submetida a uma análise, através da ferramenta SonarQube, para garantir que este requisito é cumprido. No caso do processo de atualização é necessário garantir que este requisito continua a ser satisfeito.

4.2.4 Segurança

Continuando na linha dos requisitos de um software crítico, temos a segurança. Neste aspeto é necessário garantir que o código do projeto não apresenta vulnerabilidades, isto é, pontos da aplicação em que seja possível alterar o seu comportamento. A forma de validação deste requisito é a mesma que a apresentada anteriormente e o critério que será utilizado para validar o mesmo no final do processo será também a ausência de vulnerabilidades, perante as regras estabelecidas para o projeto.

4.2.5 *Maintainability*

Um dos requisitos não funcionais importantes para qualquer processo de migração de código é a a capacidade de manutenção do mesmo. Para este requisito, o atributo de validação será a ausência de *code smells*, isto é, de partes do código em que seja identificada alguma lacuna que poderá levar à ocorrência de um problema durante a execução do mesmo. Este atributo consegue também ser monitorizado recorrendo à configuração utilizada na ferramenta SonarQube.

4.2.6 Cobertura de código

Por fim, o último atributo de qualidade a ser tido em conta neste processo de atualização é a cobertura de código que os de testes do projeto fornecem. Tratando-se de um projeto crítico definiu-se que o valor mínimo aceitável para considerar este requisito satisfeito seria garantir uma cobertura de código de pelo menos oitenta porcento.

4.3 Plano de validação de requisitos

Uma vez identificados os requisitos funcionais e não funcionais, de seguida vai ser apresentado o processo que vai ser realizado no final do processo de migração de forma a poder validar que todos os requisitos foram cumpridos.

Para o caso dos requisitos funcionais, antes da migração cada um deles vai ser testado com um caso de uso exemplo que será repetido no final da mesma. Para validar estes requisitos o resultado terá de ser igual nos dois testes. Além disto, os testes unitários codificados para estas funcionalidades podem ter de ser sofrer alguns ajustes no processo de migração mas, aquando a sua conclusão, devem continuar a ser executados com sucesso.

No caso dos requisitos não funcionais, para validar o requisito de desempenho será executada uma bateria de testes de performance, recorrendo à ferramenta JMeter, e será feita a comparação do tempo de processamento antes e depois da atualização. Para o requisito da compatibilidade, será configurado um servidor Tomcat ou JBoss, utilizando a versão 11 do JDK e serão executados exemplos de teste para verificar se existem algumas anomalias no funcionamentoda biblioteca.

Por fim, para os requisitos de confiabilidade, segurança, *maintainability* e cobertura de código será executada uma verificação ao código do projeto, recorrendo à ferramenta SonarQube, e serão retiradas métricas que permitirão identificar se os critérios definidos para cada um dos requisitos foram alcançados com sucesso.

This page is intentionally left blank.

Capítulo 5

Processo de Atualização

Neste capítulo será feita uma análise ao processo de atualização da biblioteca Parse & Correlate (PARSEC). Vão ser identificadas e descritas as tarefas realizadas, o seu objetivo e as principais dificuldades e desafios encontrados em cada uma.

5.1 Atualização do JDK

A tarefa que deu início ao processo de atualização foi a atualização do Java Development Kit (JDK) utilizado no projeto, alterando a sua versão de 8 para 11. Sendo uma das maiores mudanças introduzidas por esta versão a exclusão da biblioteca Java Architecture for XML Binding (JAXB) do JDK e atendendo ao facto de que o a biblioteca PARSEC faz o processamento de dados no formato eXtensible Markup Language (XML), entre outros, foram notórios de imediato os impactos desta alteração, deixando de ser possível executar a compilação do projeto.

5.1.1 Atualização da dependência JAXB

De forma a conseguir produzir um novo artefacto, iniciou-se a pesquisa de possíveis soluções para resolver este problema. Desta surgiram algumas opções que serão expostas de seguida.

Permitir a utilização de módulos descontinuados

Uma das possibilidades para a resolução deste problema seria permitir a utilização de módulos descontinuados. Desta forma, através de uma configuração aplicada ao projeto, dar-se-ia permissão à biblioteca PARSEC para continuar a utilizar módulos apesar destes terem sido descontinuados.

Esta opção foi rejeitada de imediato devido ao facto da utilização de ferramentas descontinuadas, sejam elas quais forem, acarretar o risco da ocorrência de problemas no futuro, seja anomalias de comportamento ou limitação de funcionalidades, além do facto de ser considerada também uma má prática.

Adicionar o módulo JAXB como dependência externa

Outra das soluções encontradas durante o processo de pesquisa foi a adicionar do módulo JAXB isolado como dependência externa. À semelhança da solução anterior também esta foi rejeitada devido ao facto de se tratar de uma solução da mesma natureza, o que faria com que o PARSEC ficasse a utilizar ferramentas que já tinham sido descontinuadas e

exposto ao risco que estas acarretam.

Utilização da ferramenta *EclipseLink MOXy*

Uma abordagem já mais ponderada seria a utilização da ferramenta *EclipseLink MOXy*. Através da utilização desta seria possível com de algumas configurações fazer com que o PARSEC continuasse a suportar as funcionalidades oferecidas pelo módulo JAXB.

Apesar desta solução não ter o problema de utilizar ferramentas descontinuadas, como no caso das anteriores, optou-se por não recorrer a esta solução devido ao facto do processo de configuração desta ferramenta não ser claro e intuitivo de se fazer.

Utilização da dependência *Jakarta EE*

Por fim, a última solução apresentada foi a utilização da dependência *Jakarta EE*. Esta dependência foi originada a partir de um projeto que resultou da migração do Java EE da *Oracle* para a *Eclipse Foundation*. Para que o projeto continuasse a usar as funcionalidades oferecidas pelo módulo JAXB apenas seria necessário importar esta dependência para o projeto, não sendo necessário qualquer tipo de configuração.

Esta acabou por ser a opção utilizada no projeto pela facilidade em configurar, por ter ser uma ferramenta que que teve intervenientes no seu desenvolvimento ligados diretamente à linguagem Java, como é o caso da empresa e também por ser a continuação oficial vertente empresarial do Java (Java EE), que foi passado para a empresa *Eclipse Foundation*, o que confere uma vertente de confiança e estabilidade a longo prazo para este projeto.

Após a adição da nova dependência ao projeto foi necessário percorrer todos os ficheiros onde esta estava a ser utilizada e atualizar o caminho do *import* antigo utilizado para o correspondente à nova versão. Este processo foi executado para todo o tipo de classes do projeto, incluindo testes unitários.

5.1.2 Atualização de *plugins*

Após a configuração da nova dependência foram eliminados os erros de compilação dos ficheiros, no entanto, apenas com estas alterações, ainda não era possível construir um executável do projeto. Neste ponto os problemas de compilação residiam nos *plugins*, nomeadamente `jaxb-maven-plugin`, utilizados no módulo *"gen"*. Este tem como finalidade gerar classes de suporte baseado em ficheiros XML Schema Definition (XSD).

Uma vez que o *plugin* utilizado não suportava Java 11, foi necessário encontrar o *plugin* correspondente que suportasse esta alteração. Assim sendo foi configurado o `jaxb2-maven-plugin` para conseguir gerar classes com a nova dependência do JAXB.

Além deste, foi ainda necessário alterar os ficheiros XSD e xjb para que ficassem de acordo com as especificações do novo *plugin*.

Esta atualização acabou por consumir mais tempo do que o inicialmente previsto devido à quantidade de ficheiros que foram necessários atualizar e, principalmente, derivado à dificuldade em encontrar documentação acerca da configuração deste novo *plugin* para funcionar com este tipo de ficheiros e também acerca das novas propriedades dos mesmos.

```
<!-- Generate JAXB Objects from XSD's -->
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>jaxb2-maven-plugin</artifactId>
    <version>2.5.0</version>
    <executions>
        <!-- Generate DUIS/MMC -->
        <execution>
            <id>xjc-DUIS-MMC</id>
            <goals>
                <goal>xjc</goal>
            </goals>
            <configuration>
                <xjbSources>
                    <xjbSource>${resources.final.directory}/bindings.xjb</xjbSource>
                </xjbSources>
                <sources>
                    <source>${xsd.patch.final.directory}/parsec.xsd</source>
                    <source>${xsd.patch.final.directory}/ServiceUserGateway.xsd</source>
                    <source>
                        ${xsd.patch.final.directory}/DCC_Service_User_Response_and_Alert_Common_Interface.xsd
                    </source>
                    <source>${xsd.patch.final.directory}/xmldsig-core-schema.xsd</source>
                </sources>
                <catalog>${resources.original.directory}/catalog.xml</catalog>
                <extension>true</extension>
                <outputDirectory>${basedir}/target/generated-sources/duis-mmc</outputDirectory>
                <noPackageLevelAnnotations>true</noPackageLevelAnnotations>
            </configuration>
        </execution>
        <!-- Generate Parse Responses Descriptors -->
        <execution>
            <id>xjc-message-descriptor</id>
            <goals>
```

Figura 5.1: Atualização para o plugin *plugin* `jaxb2-maven-plugin`

```
<!-- Common Types -->

<xs:element name="Request" type="sr:Request"/>

<xs:element name="Response" type="sr:Response"/>

<xs:element name="SMETS1SignedResponse" type="sr:SMETS1SignedResponse"/>

<xs:element name="S1SPAlert" type="sr:S1SPAlert"/>

<xs:complexType name="SMETS1ResponseMessage">
    <xs:sequence>
        <xs:element name="ServiceReference" type="sr:ServiceReference" minOccurs="0"/>
        <xs:element maxOccurs="1" minOccurs="0" name="ServiceReferenceVariant"
                    type="sr:ServiceReferenceVariant"/>
        <xs:element minOccurs="0" name="DSPScheduleID" type="sr:scheduleID"/>
        <xs:element name="ThrottledAlertSequenceId" type="xs:unsignedInt" minOccurs="0"
                    maxOccurs="1"/>
        <xs:element name="ThrottledAlertCount" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"/>
        <xs:element minOccurs="1" maxOccurs="1" ref="sr:SMETS1SignedResponse"/>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="range_1_3">
    <xs:restriction base="xs:positiveInteger">
        <xs:minInclusive value="1"/>
        <xs:maxInclusive value="3"/>
    </xs:restriction>
</xs:simpleType>
```

Figura 5.2: Exemplo de um ficheiro de configuração xsd

```
<bindings xmlns="https://jakarta.ee/xml/ns/jaxb"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema"
                version="3.0">

    <bindings schemaLocation="xsd/ServiceUserGateway.xsd">
        <schemaBindings>
            <package name="com.criticalsoftware.smartdcc.parsec.gen.duis.request" />
        </schemaBindings>
    </bindings>

    <bindings schemaLocation="xsd/DCC_Service_User_Response_and_Alert_Common_Interface.xsd">
        <schemaBindings>
            <package name="com.criticalsoftware.smartdcc.parsec.gen.mmc.alertresponse" />
        </schemaBindings>
    </bindings>

    <bindings schemaLocation="xsd/parsec.xsd">
        <schemaBindings>
            <package name="com.criticalsoftware.smartdcc.parsec.gen.parsec.results" />
        </schemaBindings>
<!--        <globalBindings localScoping="toplevel"/>-->
    </bindings>

</bindings>
```

Figura 5.3: Exemplo de um ficheiro de configuração xjb

5.2 *Refactoring* do código

Após terminar o processo de atualização que permitisse gerar uma nova versão do projeto com o novo JDK, a tarefa que se seguiu foi iniciar o *refactoring* do código existente. A técnica utilizada na abordagem a este processo foi dividir o projeto em cada uma das suas funcionalidades e efetuar as alterações em cada uma separadamente.

Dentro de cada funcionalidade o primeiro ficheiro a ser alterado foi correspondente ao *endpoint* disponibilizado. Este ficheiro foi analizado e foram feitas as modificações identificadas como melhorias e de seguida foi-se repetindo este processo nos ficheiros dos quais o inicial fosse dependente até chegar a um ficheiro que não tivesse dependência de outros ficheiros.

As primeiras alterações a serem feitas foram a identificação de métodos, de classes maioritariamente utilitárias ou internas do próprio Java, que estivessem descontinuados, figura 5.4.

```
    try {
        inputState = Integer.valueOf(inputAndOutpustStates.get(0).toString());
        inputState = Integer.parseInt(inputAndOutpustStates.get(0).toString());
    } catch (final NumberFormatException ex) {
        throw new ParseWithPartialContentException(ParseErrorMessages.JSON_ELEMENT_PARSE_ERROR,
                "OperationalUpdateDeviceAlert.inputState");
    }
    if ((inputState >= 0) && (inputState <= 100)) {
        this.operationalUpdateDeviceAlert.setInputState(inputState);
    } else {
        throw new ParseWithPartialContentException(
                ParseErrorMessages.ELEMENT_VALUE_DIFFERENT_THAN_EXPECTED_RANGE,
                "OperationalUpdateDeviceAlert.inputState", EXPECTED_STATE_RANGE, inputState);
    }
}
if (inputAndOutpustStates.get(1) != null) {
    try {
        outputState = Integer.valueOf(inputAndOutpustStates.get(1).toString());
        outputState = Integer.parseInt(inputAndOutpustStates.get(1).toString());
    } catch (final NumberFormatException ex) {
        throw new ParseWithPartialContentException(ParseErrorMessages.JSON_ELEMENT_PARSE_ERROR,
                "OperationalUpdateDeviceAlert.outputState");
```

Figura 5.4: Exemplo de uma atualização de métodos descontinuados

Depois de se ter concluído a atualização de métodos descontinuados, a tarefa de *refactoring* seguinte foi fazer a análise das expressões condicionais utilizadas com o objetivo de perceber se é possível remover parcialmente as mesmas ou então tornar estas mais legíveis, como apresentado na figura 5.5.

```
@Override
public boolean checkStatus(final GbcsAccessRequestType accessRequestPayload) {
    final ActivateFirmwareResponsePayload asn1Payload = accessRequestPayload
            .getAsn1Payload(ActivateFirmwareResponsePayload.class);

    if (asn1Payload.executionOutcome.isInitialized) {

        return ActivateImageResponseCode.success.equals(asn1Payload.executionOutcome.activateImag
                .getValue());

    } else if (asn1Payload.commandAccepted.isInitialized) {

        return true;
    }

    return false;
    } else return asn1Payload.commandAccepted.isInitialized;

}
```

Figura 5.5: Exemplo atualização de expressões condicionais

A atualização que se seguiu foi a implementação da principal característica introduzida na versão 11 do Java que foram as interfaces funcionais. Nesta tarefa foram analizadas todas as utilizações de ciclos para que se procedesse à substituição das mesmas por expressões lambda ou através da utilização da interface `Streams`. Exemplos destas atualizações são as figuras 5.6 5.7 5.8.

```
/**
 * @param value status value
 * @return the status
 */
public static StatusASN1Enum getByAsn1Value(final long value) {
    for (final ActivateFirmwareResponseCodeEnum obj : values()) {
        if (obj.asn1Value == value) {
            return obj.getMmcValue();
        }
    }
    return StatusASN1Enum.NOT_KNOWN;
    return Arrays.stream(values()).filter(obj -> obj.asn1Value == value).findFirst().map(ActivateFirmwareResponseCodeEnum::getMmcValue).orElse(StatusASN1Enum.NOT_KNOWN);
}

/**
 * @return the mmcValue
 */
private StatusASN1Enum getMmcValue() {
    return this.mmcValue;
```

Figura 5.6: Exemplo de uma atualização de ciclos com condições

```
final List<PublishDayProfile.DayProfileEntryFriendlyCredit> listEntries = new ArrayList<>();
for (int j = 0; j < zigBeeDayProfile.getListFrames().size(); j++) {
    final Frame frameDayEntry = zigBeeDayProfile.getListFrames().get(j);

zigBeeDayProfile.getListFrames().forEach(frameDayEntry -> {
    final HexUnsignedInteger16 zigBeeStartTime = zigBeeDayProfile.getAttributeContent(frameDayEntry,
            FriendlyCreditSwitchFrameDescriptor.START_TIME);
    final Boolean zigBeeFriendlyCreditEnable = zigBeeDayProfile.getAttributeContent(frameDayEntry,
            FriendlyCreditSwitchFrameDescriptor.FRIENDLY_CREDIT_ENABLE);
```

Figura 5.7: Exemplo de uma atualização de ciclos com índices

```
this.startMessageCodeIndexReservation(ERRORS_P21_P23);
for (int i = 0; i < this.gbcsTrustAnchorCellRemotePartyRoleList.size(); i++) {
    this.addUseCaseRule(
            REPLACEMENTS_REPLACEMENT_CERTIFICATE + (i + 1) + INVALID_TAC_REMOTE_PARTY_ROLE_DESCRIPTION,
            this.gbcsTrustAnchorCellRemotePartyRoleList.get(i), expectedTACTrustAnchorCellRemotePartyRole,
            AssertionType.EQUALS);
}
IntStream.range(0, this.gbcsTrustAnchorCellRemotePartyRoleList.size()).forEach(i -> this.addUseCaseRule(
        REPLACEMENTS_REPLACEMENT_CERTIFICATE + (i + 1) + INVALID_TAC_REMOTE_PARTY_ROLE_DESCRIPTION,
        this.gbcsTrustAnchorCellRemotePartyRoleList.get(i), expectedTACTrustAnchorCellRemotePartyRole,
        AssertionType.EQUALS));
this.endMessageCodeIndexReservation();
}
```

Figura 5.8: Exemplo de outra atualização de ciclos com índices

Por fim, a última tarefa realizada no processo de *refactoring* foi verificar as duplicações de código existentes e resolver as mesmas através da abstração de variáveis e métodos para classes `super` ou utilitárias.

```
685  686        final GasDayProfiles dayProfiles = tariffSwitchingTable.getDayProfiles();
686  687        Assert.assertNotNull(dayProfiles);
687       -     Assert.assertEquals(dayProfiles.getDayProfile(), dayProfileList);
     688  +     TestClassesComparator.assertEqualsListsOfGasDayProfiles(dayProfiles.getDayProfile(), dayProfileList);
688  689
689  690        final GasWeekProfiles weekProfiles = tariffSwitchingTable.getWeekProfiles();
690  691        Assert.assertNotNull(weekProfiles);
691       -     Assert.assertEquals(weekProfiles.getWeekProfile(), weekProfileList);
     692  +     TestClassesComparator.assertEqualsListsOfGasWeekProfiles(weekProfiles.getWeekProfile(), weekProfileList);
692  693
693  694        final GasSeasons seasons = tariffSwitchingTable.getSeasons();
694  695        Assert.assertNotNull(seasons);
695       -     Assert.assertEquals(seasons.getSeason(), seasonList);
     696  +     TestClassesComparator.assertEqualsListsOfGasSeasons(seasons.getSeason(), seasonList);
696  697

1204 1205        final GasDayProfiles dayProfiles = tariffSwitchingTable.getDayProfiles();
1205 1206        Assert.assertNotNull(dayProfiles);
1206      -     Assert.assertEquals(dayProfiles.getDayProfile(), dayProfileList);
     1207 +     TestClassesComparator.assertEqualsListsOfGasDayProfiles(dayProfiles.getDayProfile(), dayProfileList);
1207 1208
1208 1209        final GasWeekProfiles weekProfiles = tariffSwitchingTable.getWeekProfiles();
1209 1210        Assert.assertNotNull(weekProfiles);
1210      -     Assert.assertEquals(weekProfiles.getWeekProfile(), weekProfileList);
     1211 +     TestClassesComparator.assertEqualsListsOfGasWeekProfiles(weekProfiles.getWeekProfile(), weekProfileList);
1211 1212
1212 1213        final GasSeasons seasons = tariffSwitchingTable.getSeasons();
1213 1214        Assert.assertNotNull(seasons);
1214      -     Assert.assertEquals(seasons.getSeason(), seasonList);
     1215 +     TestClassesComparator.assertEqualsListsOfGasSeasons(seasons.getSeason(), seasonList);
1215 1216
```

Figura 5.9: Exemplo de redução de duplicação de código

```
public static void assertEqualsDates(Date date1, Date date2) {
    Assert.assertEquals(date1.getYear().getSpecifiedYear(), date2.getYear().getSpecifiedYear());
    Assert.assertEquals(date1.getMonth().getSpecifiedMonth(), date2.getMonth().getSpecifiedMonth());
    Assert.assertEquals(date1.getDayOfMonth().getSpecifiedDayOfMonth(), date2.getDayOfMonth().getSpecifiedDayOfMonth());
    Assert.assertEquals(date1.getDayOfWeek().getSpecifiedDayOfWeek(), date2.getDayOfWeek().getSpecifiedDayOfWeek());
}

public static void assertEqualsListsOfGasDayProfiles(List<GasDayProfile> dayProfiles1, List<GasDayProfile> dayProfiles2) {
    Assert.assertEquals(dayProfiles1.size(), dayProfiles2.size());

    IntStream.range(0, dayProfiles1.size()).forEach(index -> assertEqualsGasDayProfiles(dayProfiles1.get(index), dayProfiles2.get(index)));
}

public static void assertEqualsGasDayProfiles(GasDayProfile gasDayProfile1, GasDayProfile gasDayProfile2) {
    Assert.assertEquals(gasDayProfile1.getDayName(), gasDayProfile2.getDayName());
    Assert.assertEquals(gasDayProfile1.getTOUTariffAction(), gasDayProfile2.getTOUTariffAction());
    Assert.assertEquals(gasDayProfile1.getBlockTariff(), gasDayProfile2.getBlockTariff());
}

public static void assertEqualsListsOfGasWeekProfiles(List<GasWeekProfile> gasWeekProfiles1, List<GasWeekProfile> gasWeekProfiles2) {
    Assert.assertEquals(gasWeekProfiles1.size(), gasWeekProfiles2.size());

    IntStream.range(0, gasWeekProfiles1.size()).forEach(index -> assertEqualsGasWeekProfiles(gasWeekProfiles1.get(index), gasWeekProfiles2.get(i
}
```

Figura 5.10: Exemplo de criação de classe utilitária

Todo este processo acabou por se alongar devido ao facto do PARSEC ser um projeto com uma dimensão bastante considerável, contando com inúmeros ficheiros quer ao nível de implementação de funcionalidades quer ao nível de testes unitários.

5.3 Atualização das restantes dependências e *plugins*

Terminado o processo de *refactoring*, uma das últimas tarefas do processo de atualização a ser executada foi atualizar os *plugins* e dependências que não tinham sido alteradas até então. Este processo passou por verificar manualmente dependência a dependência, procurar a data da versão atual e se existiam versões mais atualizadas e, caso existissem, atualizar para uma versão mais recente, tendo em atenção para não escolher versões que não fossem finais.

Para garantir que esta atualizações não introduziam problemas a nível das funcionalidades ou de compilação do projeto, no final de cada uma foi executada a *build* do projeto, compilando o mesmo, correndo os testes unitários e gerando um novo executável. A lista de dependências e *plugins* alterados e as respetivas mudanças de versões podem ser consultadas nas tabelas 5.1 e 5.2.

Em alguns casos, o facto de as versões utilizadas serem bastante antigas fez com que fosse necessário proceder a algumas alterações a nível de configuração, tipicamente com *plugins*, para que o projeto ficasse de acordo com nova versão e fosse possível gerar um novo executável do mesmo.

Dependência	**Versão Anterior**	**Versão Atual**
Testng	6.1.0	7.4.0
Log4j-api	2.12.1	2.14.1
Log4j-core	6.1.0	7.4.0
Log4j-web	6.1.0	7.4.0
Jaxb-impl	2.2.11	3.0.2
Commons-codec	1.10	1.15
Commons-lang3	3.6	3.12.0
Orika-core	1.5.1	1.5.4
Jdlms	0.9.0	1.7.1
Joda-time	2.9.9	2.10.10
Ant	1.10.1	1.10.11
Commons-vfs2	2.1	2.9.0
Commons-net	3.6	3.8.0
Httpclient	4.5.3	4.5.13
Httpcore	4.4.6	4.4.14
Commons-beanutils	1.9.3	1.9.4
Metrics-core	3.2.6	4.2.3
Metrics-jmx	3.2.6	4.2.3
Metrics-healthchecks	3.2.6	4.2.3
Jackson-databind	2.12.3	2.12.4
Jsch	0.1.54	0.1.55
Json	20190722	20210307
maven-scm-provider-jgit	1.9.5	1.11.3

Tabela 5.1: Tabela de dependências atualizadas

Dependência	**Versão Anterior**	**Versão Atual**
Maven-jaxb2-plugin	0.13.2	0.14.0
Maven-war-plugin	3.1.0	3.3.1
Maven-ejb-plugin	2.5.1	3.1.0
Maven-dependency-plugin	3.0.1	3.2.0
Maven-assembly-plugin	3.0.0	3.3.0
Build-helper-maven-plugin	3.0.0	3.2.0
Exec-maven-plugin	1.6.0	3.0.0
Maven-antrun-plugin	1.8	3.0.0
Jmeter-maven-plugin	2.2.0	3.4.0
Maven-release-plugin	2.5.3	3.0.0-M4
Maven-clean-plugin	3.0.0	3.1.0
Maven-patch-plugin	1.1.1	1.2
Jacoco-maven-plugin	0.7.9	0.8.7
Maven-surefire-plugin	2.20	3.0.0-M5
Maven-source-plugin	3.0.1	3.2.1
Maven-javadoc-plugin	2.10.4	3.3.0
Maven-deploy-plugin	2.8.2	3.0.0-M1
Maven-jar-plugin	3.0.2	3.2.0

Tabela 5.2: Tabela de *plugins* atualizados

5.4 Atualização dos servidores

Finalmente, a última tarefa respeitante ao processo de atualização do projeto foi a de atualizar dos servidores utilizados. Estes são disponibilizados em conjunto com o projeto através de um módulo, `dist`, que tem como objetivo, através de *plugins* e ficheiros configuração, disponibilizar arquivos `zip` com todos os ficheiros do servidor já devidamente adaptados para conseguir suportar a biblioteca PARSEC. A atualização deste módulo envolveu atualizar as versões dos servidores disponibilizadas, para que conseguissem suportar a versão 11 do JDK, e também alterar os ficheiros de configuração para estarem de acordo com estas.

```
<security-realm name="ManagementRealm">
    <authentication>
        <local default-user="$local"/>
        <local default-user="$local" skip-group-loading="true"/>
        <properties path="mgmt-users.properties" relative-to="jboss.server.config.dir"/>
    </authentication>
    <authorization map-groups-to-roles="false">
        <properties path="mgmt-groups.properties" relative-to="jboss.server.config.dir"/>
    </authorization>
</security-realm>
<security-realm name="ApplicationRealm">
    <server-identities>
        <ssl>
            <keystore path="application.keystore" relative-to="jboss.server.config.dir" keystore-pass
        </ssl>
    </server-identities>
    <authentication>
        <local default-user="$local" allowed-users="*"/>
        <local default-user="$local" allowed-users="*" skip-group-loading="true"/>
        <properties path="application-users.properties" relative-to="jboss.server.config.dir"/>
    </authentication>
    <authorization>
        <properties path="application-roles.properties" relative-to="jboss.server.config.dir"/>
```

Figura 5.11: Exemplo atualização de ficheiros de configuração de servidores

Finalizando esta tarefa, concluiu-se o processo de atualização do projeto. Além das atualizações, este processo contou também com a realização de testes de performance que foram sendo intercalados com a execução das outras tarefas, porém a descrição destes assim como os seus resultados apenas serão apresentados no capítulo 7.

This page is intentionally left blank.

Capítulo 6

Planeamento

Neste capítulo será feita uma análise ao planeamento deste projeto. Serão apresentados os planos para a cada um dos semestres, contendo informação das tarefas inicialmente previstas e das tarefas efetivamente realizadas. Desta forma, é possível avaliar se existiram atrasos nas tarefas ou mudanças ao plano inicial e identificar as causas das mesmas. Além disto, neste capítulo estará ainda contida uma secção de riscos onde serão apresentados os riscos levantados inicialmente, bem como aqueles que se vieram a verificar ao longo do estágio.

6.1 Primeiro semestre

Nesta secção irá ser realizada uma análise às atividades relativas ao primeiro semestre deste estágio. De modo a tornar mais perceptível o trabalho desenvolvido, de seguida será mostrado um diagrama de *Gantt* contendo informação das tarefas planeadas e das tarefas realizadas durante este período. Assim, será mais fácil perceber em que contexto as tarefas realizadas foram desenvolvidas face ao planeamento inicial, isto é, se houve atrasos ou alterações do plano.

No diagrama representado na figura 6.1 é possível encontrar, a cinzento, as tarefas planeadas no inicio do estágio e, a azul, as tarefas correspondentes que efetivamente foram realizadas durante o primeiro semestre. Estão também representadas as datas de início e de conclusão das mesmas.

A primeira tarefa planeada consistia em analisar sistemas e processos existentes e definir o âmbito do projeto. Na realidade esta tarefa acabou por se dividir em duas, fazer *setup* do projeto, ou seja, descarregar o repositório e configurar os servidores de modo a conseguir correr a aplicação com as funcionalidades da biblioteca Parse & Correlate (PARSEC), e analisar o contexto do mesmo, de modo a perceber como surgiu e onde este se insere.

De seguida, a segunda tarefa requeria uma análise da ferramenta PARSEC. Na prática esta tarefa acabou também por se dividir em duas que foram a análise funcional do projeto e o estado da arte do mesmo. Estas acabaram depois do tempo previsto, uma vez que foram feitas por várias iterações, que eram validadas com elementos da empresa Critical Software mediante a sua disponibilidade, e por isso, até chegar a uma versão final, acabaram por ultrapassar o tempo previsto.

Nos intervalos entre cada uma das iterações da tarefa anterior e a sua validação, foi desenvolvida a tarefa da análise do estado da arte do Java. Nesta tarefa foram incluídos os

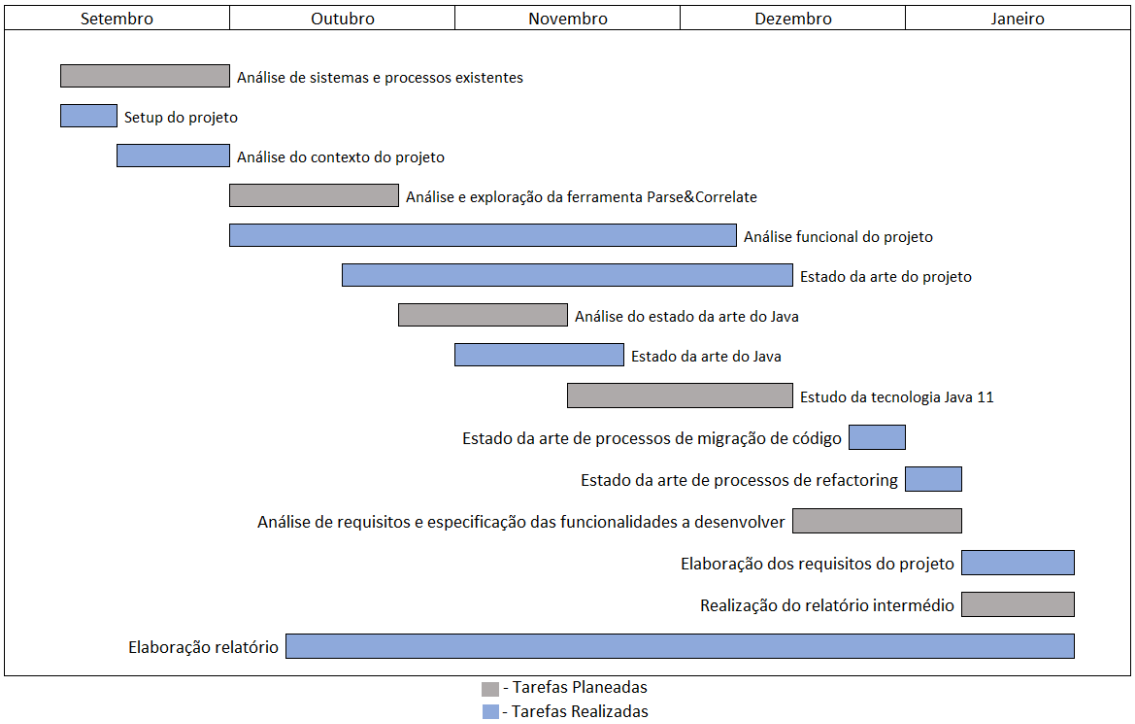

Figura 6.1: Diagrama de Gantt das tarefas do 1º Semestre

âmbitos das tarefas de análise do estado da arte do Java e de estudo da tecnologia Java 11.

Para completar o capítulo do estado da arte foram ainda feitas pesquisas relacionados com os temas de migração e *refactoring* de código.

Finalmente foram levantados requisitos do projeto e da respetiva migração e definidos métodos de validação dos mesmos.

Paralelamente ao desenvolvimento das tarefas acima descritas foi sendo desenvolvido o relatório intermédio. Para esta tarefa foi também adotada a estratégia de desenvolvimento por iterações.

6.2 Segundo semestre

Após ter sido feita a análise ao trabalho realizado no primeiro semestres, de seguida será analisado o trabalho realizado ao longo do segundo semestre. No diagrama representado na figura 6.2 encontram-se representadas as tarefas que estavam inicialmente planeadas e as que foram efetivamente realizadas durante o período de tempo referido.

Analisando o diagrama é possível identificar que a primeira tarefa planeada era a atualização do código do projeto. No entanto, após uma apresentação intermédia foram identificados alguns pontos em falta, sendo um deles a identificação dos riscos inerentes ao projeto. Posto isto, os trabalhos do segundo semestre iniciaram com a tarefa de análise de riscos.

Após terem sido identificados os riscos do projeto, a tarefa que se seguiu foi a realização de testes de performance. A execução destes testes numa fase anterior ao início da atualização do código do projeto tem como objetivo estabelecer uma base de comparação para posteriormente ser utilizada para validar os impactos da atualização.

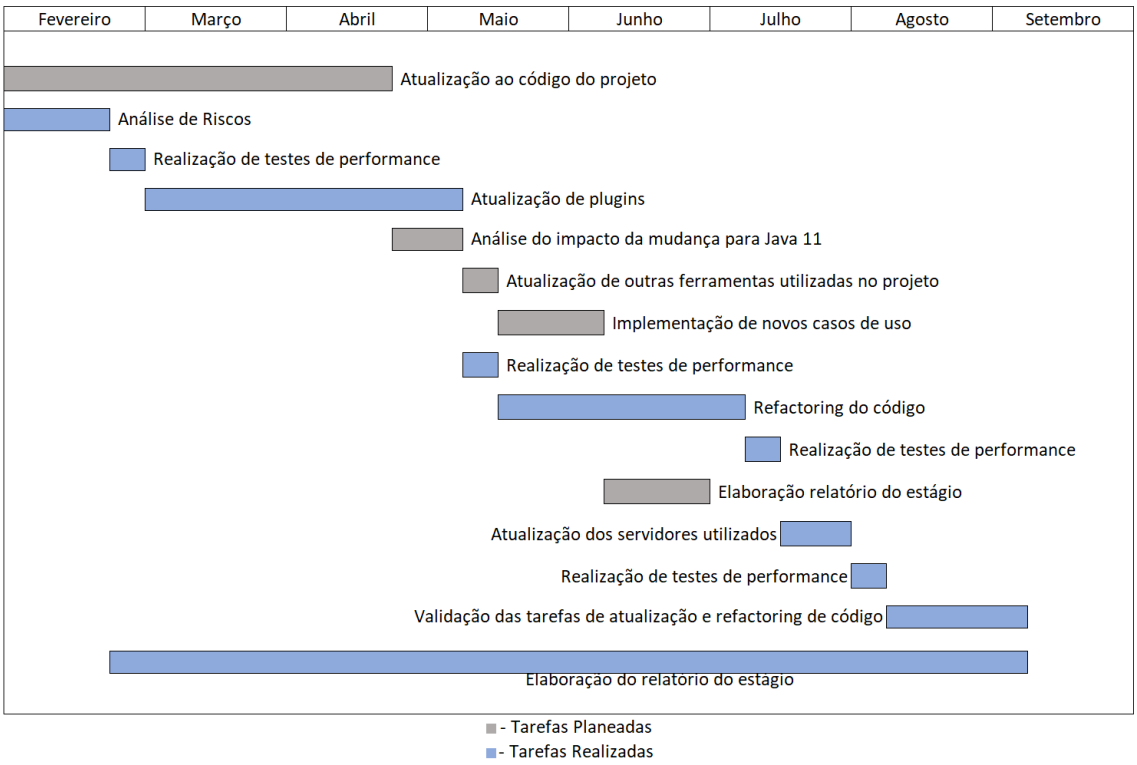

Figura 6.2: Diagrama de Gantt das tarefas do 2º Semestre

De seguida deu-se início ao processo de atualização. A primeira tarefa desta fase passou pela atualização de *plugins* utilizados pelo projeto. Esta tarefa acabou por demorar mais do que o esperado devido ao facto da versão dos *plugins* utilizados no projeto ser muito antiga, o que fez com que grande parte das configurações existentes tivesse de ser refeita, e também devido ao facto de estes serem utilizados para gerar classes de suporte ao funcionamento da biblioteca e terem de passar a gerar classes que suportassem a versão 11 da linguagem Java.

Após isto, as tarefas planeadas para esta altura seriam analisar os impactos da atualização, atualizar outras ferramentas, como os servidores utilizados para suportar o funcionamento da biblioteca, e implementar novos casos de uso. Esta última tarefa acabou por não se chegar a concretizar devido a uma necessidade de um cliente da empresa que fez com que estes casos de uso tivessem de ser implementados e entregues com urgência.

Posto isto e tendo em conta o facto de que as tarefas se encontravam atrasadas, decidiu-se reajustar o âmbito do estágio, passando o foco deste a incidir sobre a atualização do projeto e de todas as ferramentas utilizadas pelo mesmo, sobre o refactoring do código do projeto e sobre a validação dos resultados produzidos com esta mudança.

Assim, a tarefa que se seguiu foi repetir os testes realizados anteriormente. Neste ponto estes testes serviram para verificar os estado do projeto após a atualização da versão do Java Development Kit (JDK) para 11 e de todos os *plugins* e dependências de modo a que ficassem em conformidade com esta alteração.

A fase seguinte foi a fase de refactoring ao código. No final desta foram mais uma vez executados os testes de performance de forma a verificar a conseguir ter um comparativo do estado do projeto nas diferentes fases.

A última fase referente a alterações a aplicar no projeto foi a fase de atualização dos

servidores utilizados no mesmo. Neste ponto foram atualizados os dois tipos de servidores utilizados e foram também executados os testes para verificar se esta atualização produzia algum tipo de impacto a nível de poder de processamento.

Depois de terminada a fase de atualização iniciou-se a fase de validação do trabalho realizado. Nesta fase foram comparados os resultados dos testes realizados ao longo de diferentes pontos de atualização do projeto e foi utilizada a ferramenta SonarQube, devido ao facto de ser uma das ferramentas utilizadas na empresa Critical Software, para retirar métricas acerca do projeto que serviram para o processo de validação do impacto produzido pela atualização do projeto.

Por fim, paralelamente às tarefas práticas que foram desenvolvidas neste semestre, foi sendo redigido um relatório contemplando toda a informação referente a este estágio.

6.3 Análise de Riscos

Uma vez finalizada a análise as todas tarefas constituintes deste projeto, de seguida será apresentada aquela que é uma das principais componentes de qualquer projeto de software, a análise de riscos. Neste ponto serão apresentados e analisados os riscos inerentes a este projeto.

6.3.1 Identificação

Com base nas tarefas e objetivos definidos para este estágio foram identificados vários riscos. Estes encontram-se representados na tabela que se segue sendo possível visualizar para cada um as suas condições, consequências e respetivos planos de mitigação.

Id	Condições	Consequências	Plano Mitigação
R-1	Elevada dimensão do projeto	Incumprimento de objetivos	Monitorização do desenvolvimento Alteração dos objetivos
R-2	Elevada complexidade do projeto	Incumprimento de objetivos	Monitorização do desenvolvimento Alteração dos objetivos
R-3	Requisitos não identificados	Redefinição de requisitos	Redefinição dos objetivos
R-4	Incompatibilidade de dependências existentes	Atraso no processo de atualização	Monitorização do desenvolvimento Alteração dos objetivos
R-5	Introdução de *bugs*	Atraso no processo de atualização	Execução plano de testes unitários Utilização de um controlador de versões
R-6	Aumento do tempo de processamento	Incumprimento de objetivos	Execução plano de testes de performance Utilização de um controlador de versões

Tabela 6.1: Lista de riscos do projeto

6.3.2 Classificação

Após terem sido apresentados, de seguida será feita uma classificação dos riscos identificados tendo em conta dois fatores: a probabilidade de ocorrência e o impacto que o mesmo terá no desenvolvimento do projeto. Na figura 6.3 é apresentado o diagrama resultante da classificação dos riscos identificados.

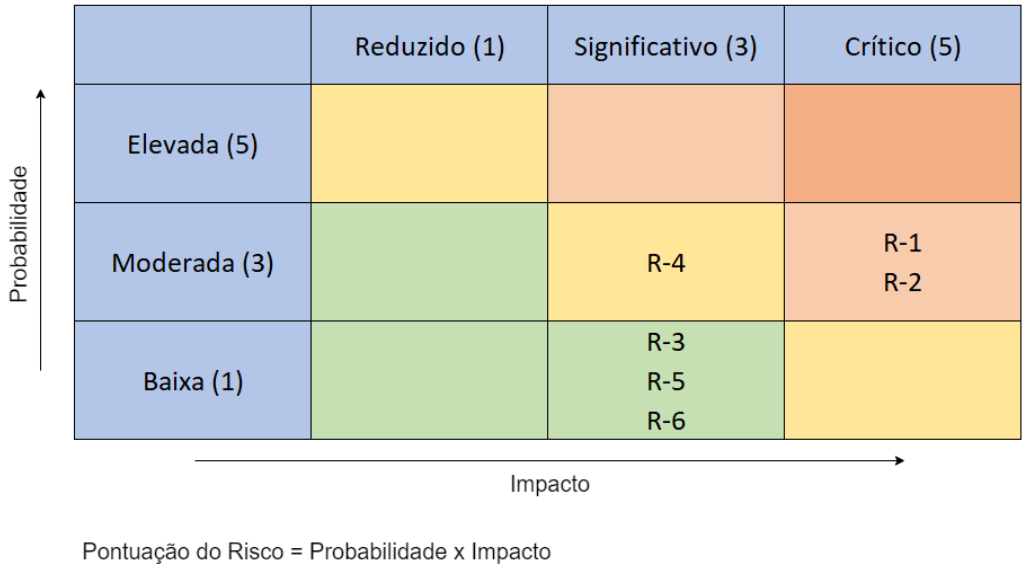

Figura 6.3: Tabela de Classificação de Riscos

Através da visualização da tabela apresentada é possível verificar que existem dois riscos classificados com uma pontuação de quinze pontos que são os risco da dimensão e complexidade do projeto PARSEC.

Começando pela probabilidade, ambos os riscos foram classificados como tendo probabilidade moderada, devido ao facto do projeto em questão apresentar alguma extensibilidade no que diz respeito a código e configurações e pelo facto de ser utilizado num contexto crítico. Quanto ao impacto foi considerado crítico, uma vez a ocorrência de um destes riscos pode comprometer os objetivos deste estágio.

Com uma pontuação um pouco mais baixa surge o risco de incompatibilidade de dependências. Este é considerado como tendo uma probabilidade moderada de ocorrer, uma vez que são várias as dependências usadas pelo projeto e as versões utilizadas são antigas, e tem um impacto significativo, uma vez que pode provocar atrasos significativos na realização das tarefas planeadas.

Por último surgem ainda, com a pontuação mais reduzida, os riscos de requisitos não identificados e introdução de *bugs* e de aumento do tempo de processamento. Para estes foi considerada uma probabilidade de ocorrer baixa, uma vez que os requisitos foram planeados e validados com pessoas experientes da empresa Critical Software, porque se trata de um código que envolve muita validação através de testes unitários e porque a mudança para uma versão mais recente da linguagem trará melhoramento de funcionalidades e como tal o tempo de processamento será melhorado. Quanto ao impacto foram classificados como significativo pois, considerando um cenário em que a resolução destes problemas consuma muito tempo, é possível que tenham de ser feitas alterações no processo definido para a realização deste estágio.

6.3.3 Dificuldades

Apesar de terem sido identificados alguns riscos referentes ao projeto, durante o desenvolvimento do mesmo surgiram algumas contrariedades que afetaram, de uma forma direta ou indireta, a realização deste estágio. Nesta secção encontram-se descritas as contrariedades que se verificaram durante o decorrer deste estágio.

6.3.3.1 Alteração da alocação de tempo ao projeto

Uma das dificuldades que mais condicionou a realização deste estágio foi o facto de no início deste segundo semestre ter sido feito uma alteração à alocação de tempo destinada ao projeto. Esta alteração deveu-se a ter sido atribuído um segundo projeto pela empresa com a alocação total do horário laboral o que fez com que a alocação de tempo destinada à realização do estágio passasse a ser exclusivamente em horário pós-laboral.

A solução encontrada para este problema foi o reajuste do âmbito do estágio, causado também por outros motivos descritos anteriormente, que passou a focar-se no processo de atualização e *refactoring* da biblioteca PARSEC.

6.3.3.2 Realização do estágio em horário pós-laboral

No seguimento da dificuldade identificada anteriormente, outra das dificuldades sentidas foi o facto da alteração do desenvolvimento do estágio para um horário pós-laboral condicionar a produtividade devido ao apoio ao projeto, seja para esclarecimento de dúvidas ou para outras tarefas, ser dado em horário laboral e também devido ao cansaço acumulado pelas diversas horas de trabalho. Esta combinação fez com que cada vez mais a produtividade fosse decaindo ao longo do semestre, o que fez com que algumas tarefas viessem a levar mais tempo do que o inicialmente planeado.

A solução apresentada anteriormente veio-se a revelar adequada também a este contratempo, não sendo por isso necessário voltar a reajustar o âmbito do projeto.

Capítulo 7

Resultados obtidos

De forma a perceber o impacto da migração a vários níveis, foram realizados testes de performance em várias fases da migração, de forma a acompanhar os progressos nas diferentes fases, e foram utilizadas métricas de código, recorrendo à ferramenta SonarQube. Posto isto, neste capítulo vão ser apresentados os resultados obtidos.

7.1 Plano de Testes

Com o objetivo de medir o impacto da migração de Java 8 para 11 na biblioteca Parse & Correlate (PARSEC) foi elaborado um plano de testes com a ajuda da ferramenta JMeter. Foi definido inicialmente que o número de pedidos a ser efetuado seria uma milhão, visto que o objetivo era tanto causar alguma sobrecarga à máquina como obter um número significativo de medições para obter resultados mais precisos.

Posto isto, iniciou-se uma análise sobre como iriam ser distribuídos o milhão de pedidos ao longo do tempo, tendo em conta a utilização de memória e de CPU da máquina e também a taxa de erro dos pedidos.

Primeiro cenário

Para o primeiro cenário definiu-se, através da ferramenta JMeter que iriam ser criadas um total de um milhão de *threads* sendo que cada uma delas apenas iria fazer um pedido, todas elas em simultâneo. Com este teste os resultados obtidos foram os seguintes:

Utilização Memória	100%
Utilização CPU	100%
Percentagem de erro	13%

Tabela 7.1: Resultados do primeiro cenário da configuração dos testes de performance

Analisando os resultados deste cenário observou-se que seria necessário alterar a distribuição dos pedidos ao longo do tempo, uma vez que esta configuração fazia uma utilização de 100% que de memória quer de CPU o que acaba por causar uma percentagem de erro, isto é pedidos que não foram bem sucedidos, de cerca de 13%.

Segundo cenário

Optou-se então por testar um conjunto de dez mil *threads* em que cada uma iria realizar dez pedidos sendo estes sequenciais, apenas seria submetido um pedido após ter sido recebida

a resposta ao pedido anterior. Com esta nova configuração os resultados obtidos foram os seguintes:

Utilização Memória	95%
Utilização CPU	100%
Percentagem de erro	9%

Tabela 7.2: Resultados do segundo cenário da configuração dos testes de performance

Apesar desta configuração apresentar percentagens mais baixas em termos de utilização de memória e de erros ocorridos, optou-se por efetuar alterações no sentido de perceber se seria possível baixar tanto estes valores como o da utilização de CPU que continuava nos 100%, o que não é desejável em qualquer tipo de sistema.

Terceiro cenário

Sendo assim, a terceira configuração utilizada passou por testar um conjunto de cinco mil *threads* estando cada uma encarregue de realizar duzentos pedidos. Como resultados deste teste foram recolhidos os dados que serão apresentados de seguida:

Utilização Memória	90%
Utilização CPU	90%
Percentagem de erro	3%

Tabela 7.3: Resultados do terceiro cenário da configuração dos testes de performance

Com esta nova configuração a melhoria foi mais significativa, conseguindo apresentar resultados sem qualquer tipo de parâmetro nos 100% e com uma percentagem de erro consideravelmente mais baixa. No entanto, durante o processo de monitorização deste cenários ainda continuaram a existir alguns picos de utilização de CPU para os 100% o que acabou por levar à realização de testes com outro tipo de configuração.

Quarto cenário

O cenário de testes que se seguiu tinha como configuração um conjunto de mil *threads* que realizariam mil pedidos cada uma. Este cenário produziu o seguinte conjunto de resultados:

Utilização Memória	75%
Utilização CPU	78%
Percentagem de erro	0%

Tabela 7.4: Resultados do quarto cenário da configuração dos testes de performance

Obtidos estes resultados, esta foi a configuração de do plano de testes escolhida para ser utilizada devido ao facto de apresentar uma percentagem de erro nula e também por apresentar percentagem de utilização de memória e CPU que mesmo em momentos de pico apenas atingiam cerca de 92%.

Posto isto, foi definido então que o plano iria consistir na criação de mil *threads* que iriam realizar mil pedidos sequenciais fazendo com que fossem produzidos um total de um milhão de pedidos para serem processados pela biblioteca. A monitorização destes permitiu que fossem medidos alguns parâmetros tais como o tempo de processamento quer no total de todos os pedidos quer em cada pedido individual e a quantidade de pedidos processados por segundo.

Depois disto, foi ainda definido que este plano de testes iria ser repetido várias vezes durante o processo de migração. As fases escolhida para executar este plano foram as seguintes:

- Sem fazer quaisquer alterações, para percebo o estado inicial do projeto e para obter também um ponto de partida para uma comparação com o resultado final.
- Após a atualização apenas do Java Development Kit (JDK) e dos *plugins* e dependências necessárias, para perceber de que forma é que apenas a mudança de versão do Java impacta o projeto.
- Após terminado o processo de *refactoring*, de modo a verificar se este trouxe melhorias não só a nível de código mas a nível de performance.
- Por fim, no final de terminada a tarefa de atualização de servidores, e consequentemente término do processo de atualização do projeto, para perceber se houve melhoria com a mudança dos servidores e também de forma a medir o ganho total com todo o processo de atualização.

Cada plano de testes foi repetido cinco vezes e foram utilizados os valores médios das cinco medições de forma a mitigar eventuais *outliers*.

7.1.1 Testes de performance

Neste ponto vão ser apresentados os resultados dos testes de performance realizados ao longo das fases identificadas anteriormente. No final será feita uma análise superficial aos mesmos no ponto seguinte será feita uma análise mais aprofundada de modo a validar se os resultados obtidos cumprem os requisitos que foram estabelecidos para o projeto.

Estado Inicial

Sem quaisquer tipo de alterações aplicadas ao projeto, os primeiros resultados obtidos com os testes de performance foram os seguintes:

Operação	**Tempo médio por pedido**	**Tempo máximo num pedido**	**Tempo total**	**Pedidos processados por segundo**
Correlate	298ms	910ms	04:59min	3354
Parse	295ms	1043ms	04:56min	3365
Validar Mac	111ms	845ms	01:52min	8873
Validar Assinatura	107ms	733ms	01:48min	5128
Verificar Campos Para Assinar	109ms	732ms	01:50min	9053

Tabela 7.5: Teste desempenho funcionalidade *Correlate*

Após atualização do JDK e respetivos *plugins*

Estes testes foram repetidos no final da atualização do JDK e dos *plugins* e dependências necessárias para obter uma versão funcional. Os dados recolhidos nesta fase vão ser apresentados de seguida.

Operação	**Tempo médio por pedido**	**Tempo máximo num pedido**	**Tempo total**	**Pedidos processados por segundo**
Correlate	278ms	888ms	04:40min	3573
Parse	273ms	990ms	04:36min	3613
Validar Mac	102ms	814ms	01:41min	9544
Validar Assinatura	708ms	733ms	01:41min	5376
Verificar Campos Para Assinar	100ms	712ms	01:43min	9816

Tabela 7.6: Teste desempenho funcionalidade *Correlate*

Fazendo uma análise superficial é visível, através da comparação de valores, que a simples atualização da versão da linguagem utilizada no projeto trouxe melhorias ao nível de performance, reduzindo os tempos de execução e consequentemente, aumentado o número de pedidos processados por segundo.

Terminada a fase de *refactoring*

Uma vez terminada a fase de *refactoring* voltou a ser feito um ponto de situação do projeto para avaliar se as alterações a nível de código se traduziriam também em alterações a nível da performance.

Operação	**Tempo médio por pedido**	**Tempo máximo num pedido**	**Tempo total**	**Pedidos processados por segundo**
Correlate	270ms	860ms	04:31min	3683
Parse	264ms	954ms	04:27min	3729
Validar Mac	100ms	782ms	01:34min	9804
Validar Assinatura	98ms	670ms	01:34min	5589
Verificar Campos Para Assinar	96ms	674ms	01:35min	10208

Tabela 7.7: Teste desempenho funcionalidade *Correlate*

Comparando os resultados obtidos nesta fase com os anteriores é possível verificar que, também com esta mudança, houve uma melhoria em relação ao estado anterior.

Finalizado o processo de atualização do projeto

A última fase em que foram corridos estes testes foi no final de todo o processo de atualização do projeto. Nesta houve a atualização das dependências que ainda não tinham sido atualizadas e houve também a atualização da versão dos servidores utilizados.

Operação	**Tempo médio por pedido**	**Tempo máximo num pedido**	**Tempo total**	**Pedidos processados por segundo**
Correlate	264ms	848ms	04:29min	3732
Parse	258ms	938ms	04:22min	3816
Validar Mac	97ms	774ms	01:29min	10012
Validar Assinatura	95ms	658ms	01:32min	5735
Verificar Campos Para Assinar	664ms	732ms	01:32min	10464

Tabela 7.8: Teste desempenho funcionalidade *Correlate*

Analisando de uma forma geral, é possível identificar que todas as fases do processo de atualização do projeto trouxeram melhorias ao mesmo. No próximo ponto estas vão ser analisadas mais em detalhe de forma a conseguir extrair métricas desta evolução e perceber se os objetivos delineados inicialmente foram alcançados.

7.2 Validação de Resultados

Depois de realizados todos os testes e recolhidos todos os resultados e métricas necessárias, neste ponto irão ser analisados os mesmos mais detalhadamente de forma a perceber o impacto que cada uma das alterações teve no projeto e validar se essas alterações cumprem os objetivos.

7.2.1 Análise de performance

O primeiro ponto de análise será o impacto que as diversas alterações tiveram na performance da biblioteca PARSEC. Para analisar este impacto escolheram-se alguns parâmetros sendo que o primeiro que será analisado é o tempo médio que cada pedido demorou a ser processado. Para este, os resultados obtidos podem ser vistos no gráfico apresentado nas figuras 7.1 e 7.2.

Analisando o primeiro gráfico, a primeira característica que é bastante evidente é o facto de duas das cinco operações analisadas demorarem significativamente mais tempo a serem processadas, sendo estas *parse* e *correlate*. Isto deve-se ao facto destes serem os pedidos que contém uma maior quantidade de dados para processar.

Outra das características observáveis neste gráfico é que em qualquer que seja a operação realizada existe uma evolução positiva no tempo de processamento médio de um pedido ao longo das diferentes fases de atualização. Todas as operações realizadas têm um impacto positivo, fazendo com que o tempo de processamento baixe, sendo que aparenta ser a atualização do JDK a alteração que produz o impacto mais significativo.

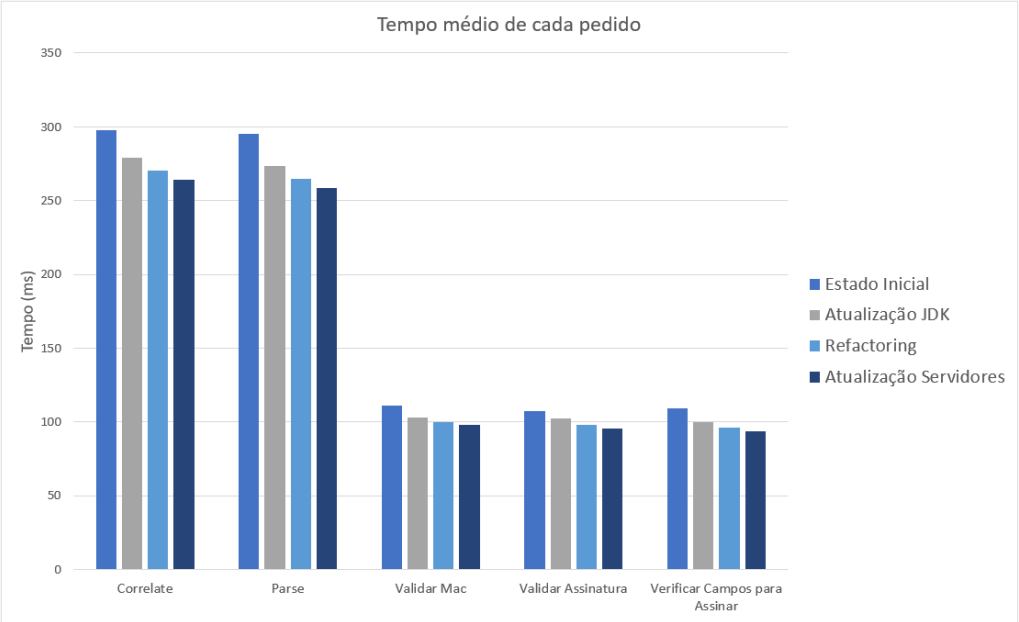

Figura 7.1: Gráfico com o tempo médio de cada pedido ao longo das fases de atualização

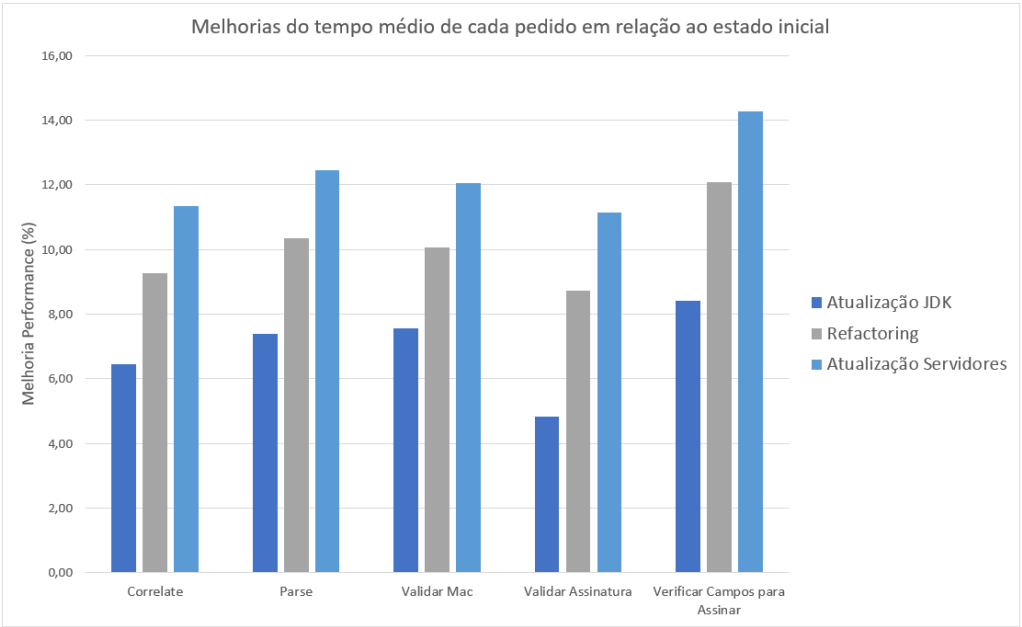

Figura 7.2: Gráfico com as melhorias do tempo médio de cada pedido comparando com o estado inicial

Como era de esperar as alterações de tempo são mais notórias nas operações que demoram mais em relação às outras e por isso foi elaborado o gráfico da figura 7.2 para os impactos poderem ser comparados entre funcionalidades, comparando em percentagem as melhorias introduzidas pelas mudanças.

Analisando este é possível confirmar o que foi referenciado anteriormente, isto é, que a alteração que produz um impacto maior é a alteração da versão do JDK. Isto é justificado pelo facto da alteração JDK alterar propriedades da Java Virtual Machine (JVM) assim como o código o compilado presente no executável, o que faz com o impacto acabe por ser maior comparando com uma operação de *refactoring* que apenas altera o código ou uma atualização de de dependências ou servidores que alteram características do ambiente de execução.

É ainda visível que o impacto das melhorias no projeto em qualquer umas das funcionalidades superam os 10% que era o valor identificado inicialmente como sendo o esperado com esta atualização.

De seguida serão mostrados os dados recolhidos relativos ao parâmetro do tempo total de processamento do milhão de pedidos realizados. Estes vão ser apresentados sob a forma de dois gráficos mantendo a coerência com os que foram apresentados anteriormente.

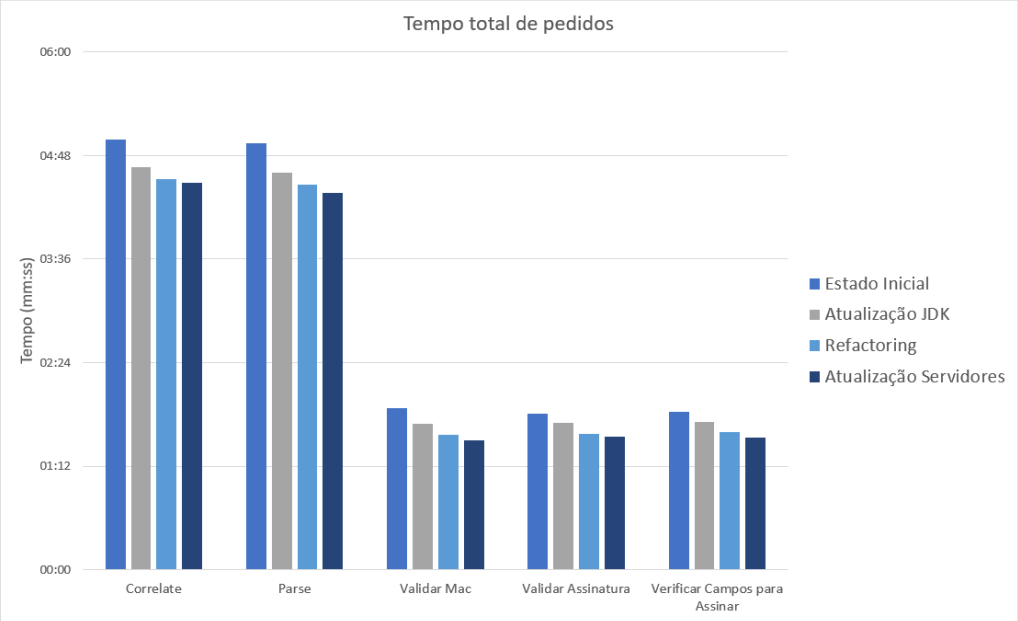

Figura 7.3: Gráfico com o tempo total de todos os pedido ao longo das fases de atualização

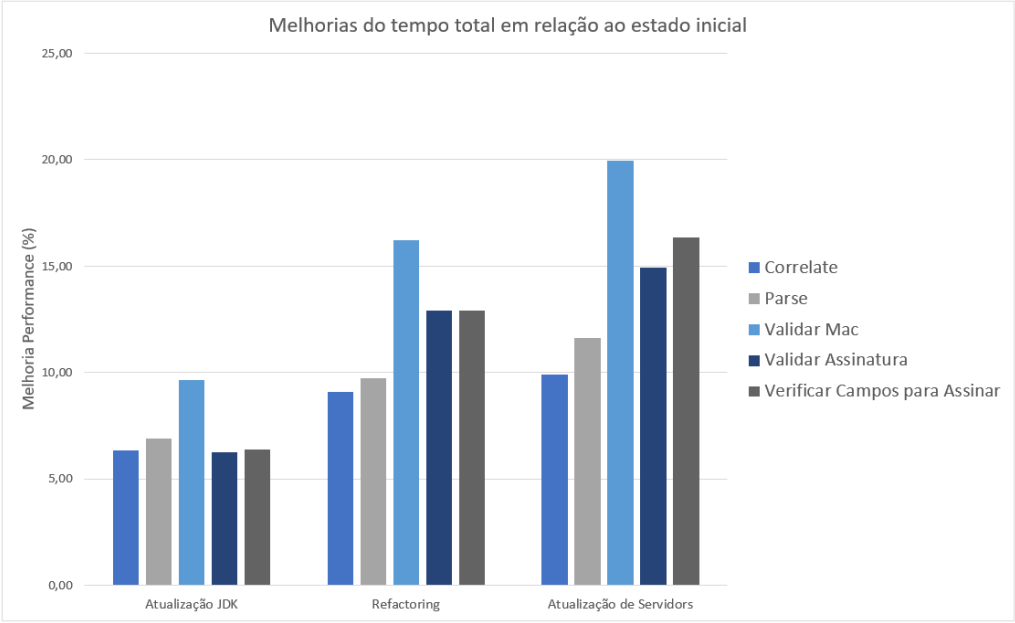

Figura 7.4: Gráfico com as melhorias do tempo do tempo total de todos os pedido ao longo das fases de atualização

Comparando os dados apresentados nestes gráficos com os anteriores é possível constatar que também para esta variável os resultados são em tudo semelhantes. Isto tem haver com o facto do tempo médio de cada pedido estar diretamente relacionado com o tempo de processamento total dos pedidos.

Há a salientar ainda que no caso dos pedidos com maior tempo de processamento a melhoria, em termos de tempo, do estado inicial para o estado final da atualização é cerca de trinta segundos o que é um valor considerável, tendo em conta que os tempos totais são abaixo dos cinco minutos.

Seguidamente serão apresentados os resultados relativos ao tempo de processamento do pior pedido do milhão de pedidos realizados. Esta variável serve para avaliar o impacto das melhorias no pior cenário.

Analisando o gráfico da figura 7.6 é possível observar um pormenor interessante que é o facto dos piores tempos das operações que por norma costumam ser mais rápidas não têm tanta diferença para as outras como acontecia no caso das outra variáveis. Uma possível explicação para este acontecimento seriam os picos de utilização de memória e CPU verificados durante a monitorização dos testes que poderiam provocar um atraso no processamento e, uma vez que as características da máquina não são alteradas entre os tipos de pedidos, este seria independente do tipo de pedido.

Além disto mantém se a mesma nuance de que de alteração para a alteração as melhorias vão aumentando, isto é, o tempo de processamento do pior pedido vai diminuindo.

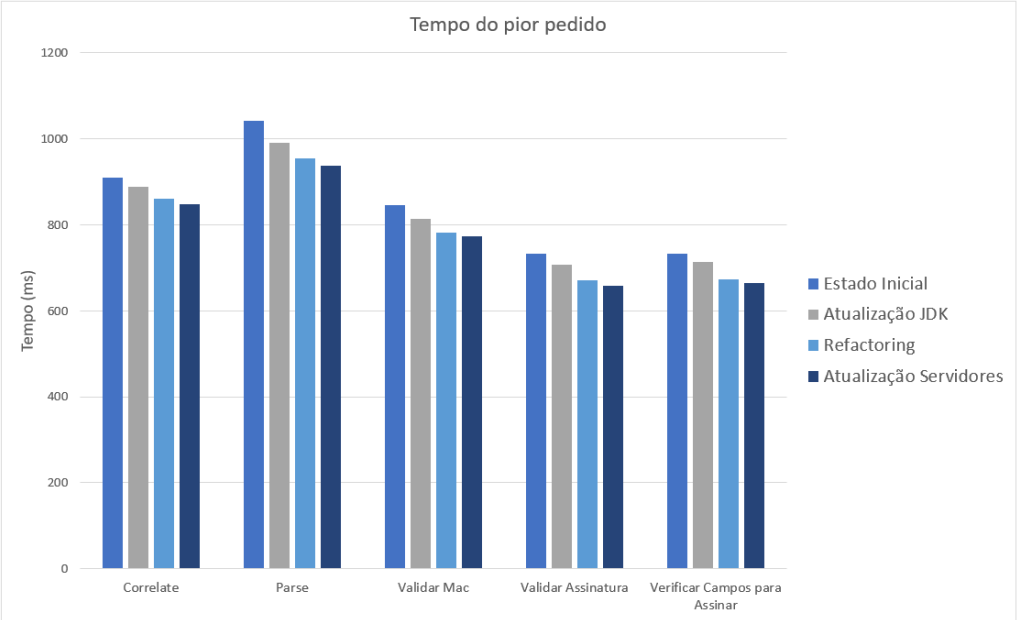

Figura 7.5: Gráfico com o pior pedido ao longo das fases de atualização

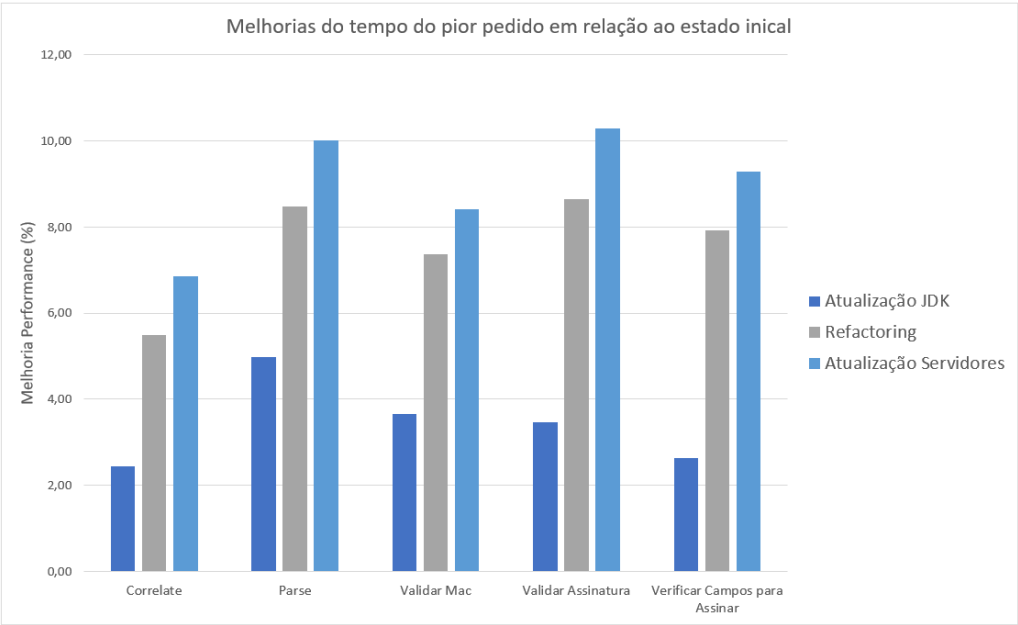

Figura 7.6: Gráfico com as melhorias do tempo do pior pedido ao longo das fases de atualização

Por fim, iremos analisar os dados recolhidos para a variável pedidos processados por segundo.

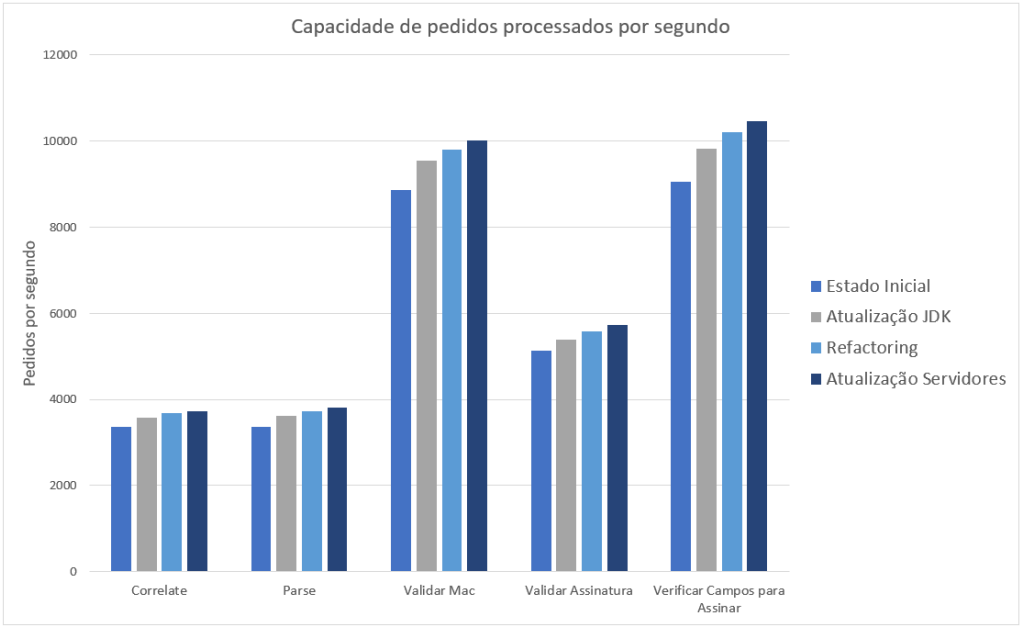

Figura 7.7: Gráfico com o número de pedidos processados por segundo ao longo das fases de atualização

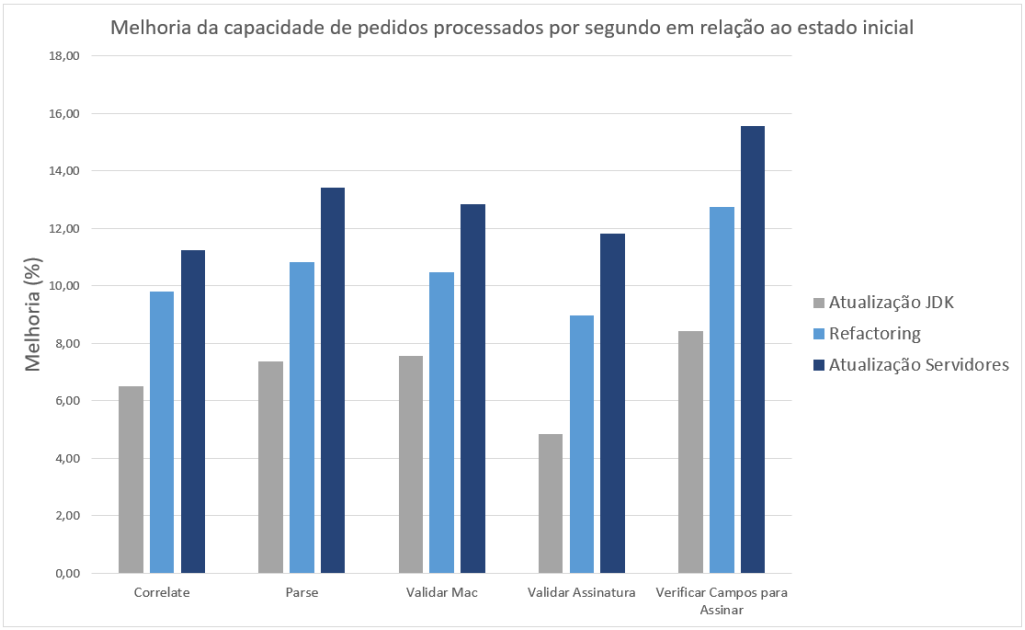

Figura 7.8: Gráfico com as melhorias do número de pedidos processados por segundo ao longo das fases de atualização

Observando o gráfico presente na figura 7.8 constata-se que este não segue a consistência mantida nos resultados anteriores. Neste caso as operações as operações com maior capacidade de processamento de pedidos por segundo são apenas duas e não três, embora sejam ambas pertencentes ao conjunto de operações com menos fluxo de dados. A justificação encontrada para esta nova propriedade prende-se com a complexidade das operações que são executadas durante o processamento do pedido, uma vez que as operações com maior capacidade de processamento de pedidos por segundo são a validação do Mac e Verificação

de campos para assinar, contrastando com as operações de *parse* e *correlate*, que envolvem o processamento de uma maior quantidade de dados, e verificação da assinatura do remetente, que implica operações como a desencriptação de chaves.

Para tentar resumir a avaliação de todas as variáveis cruzadas com todas as operações, foi elaborado um gráfico que contém a média de todas as variáveis em todas as operações em cada uma das fases do projeto para tentar com estes valores validar se o processo de atualização ficou dentro ou fora do esperado. De seguida será mostrado o gráfico que contém esta informação.

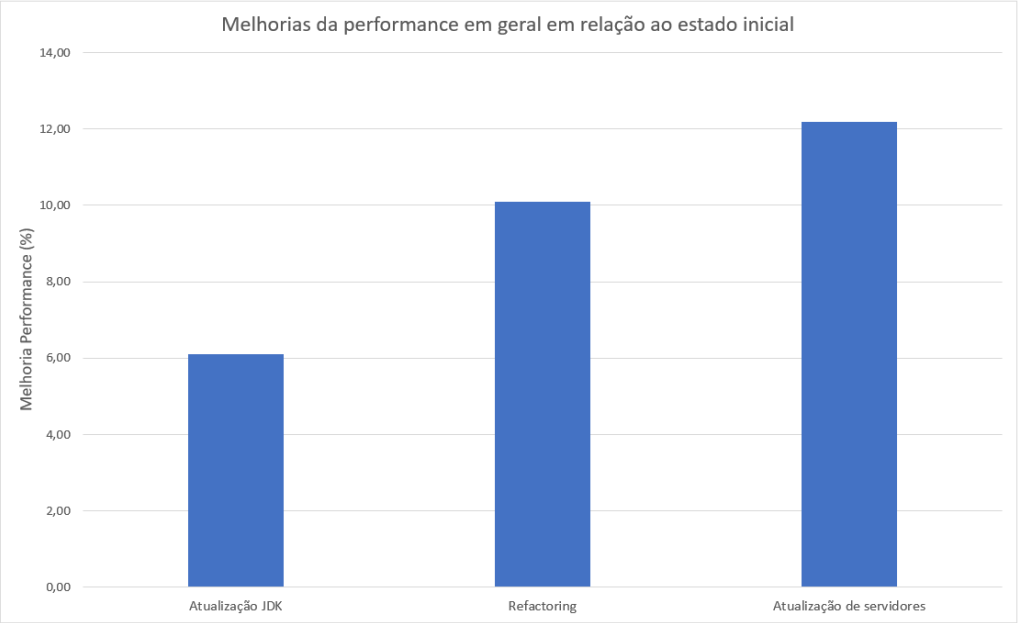

Figura 7.9: Gráfico com as melhorias da performance em geral ao longo das fases de atualização

Analisando o gráfico referente à figura 7.10 com a compilação da informação dos gráficos anteriores, observa-se que a atualização do JDK é, no geral a mudança que tem o maior impacto. Observa-se ainda que, em média, a melhoria de performance do processamento de pedidos introduzida pelo processo de atualização é de cerca de 12%, pelo que se encontra dentro do intervalo considerado como expectável, tendo sido este definido como entre 10 e 15%.

7.2.2 Análise de código

Uma vez concluída a validação dos atributos de performance, neste ponto irão ser validados os atributos respeitantes ao código. Para este processo foi usada a ferramenta SonarQube para extrair métricas de código que permitissem validar os requisitos referentes ao código identificados no capítulo 4. As métricas extraídas no final da atualização foram as seguintes:

Através da métricas extraídas no final do processo de atualização é possível confirmar que os requisitos respeitantes ao código foram cumpridos. Os valores do número de bugs identificados no projeto, vulnerabilidades e *code smells* manteve-se nos zero mesmo após as alterações efetuadas. Também o nível de cobertura dos testes unitários continuou a ultrapassar os 80% situando-se nos 81% no final deste processo. Apesar de não terem sido melhorias ao projeto, uma vez que o estado inicial do mesmo já cumpria com estes requisitos, é de salientar que é importante que estes continuem a ser respeitados de modo

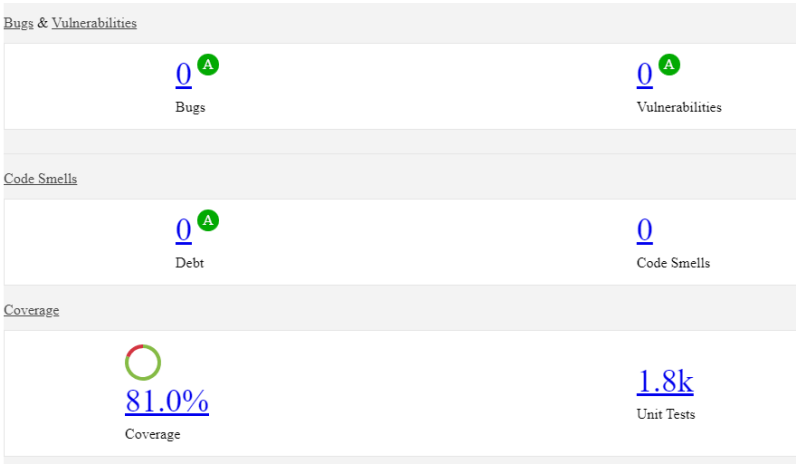

Figura 7.10: Métricas de código extraídas através da ferramenta SonarQube

a ter uma versão da biblioteca PARSEC atualizada que possa ser entregue ao cliente.

7.2.3 Outras métricas interessantes

Durante o processo de monitorização das métricas necessárias para validar os requisitos do projeto foram também observados comportamentos interessantes associados a outras métricas que acrescentam algum valor ao projeto e por isso valem a pena ser discutidos.

Um dos comportamentos observados durante a monitorização dos testes de performance foi a avaliação da utilização de memória e CPU da máquina. Estes parâmetros foram acompanhados na fase de definição da configuração dos testes e, posteriormente, foram também acompanhados durante a última fase de testes. Os resultados observados foram os seguintes:

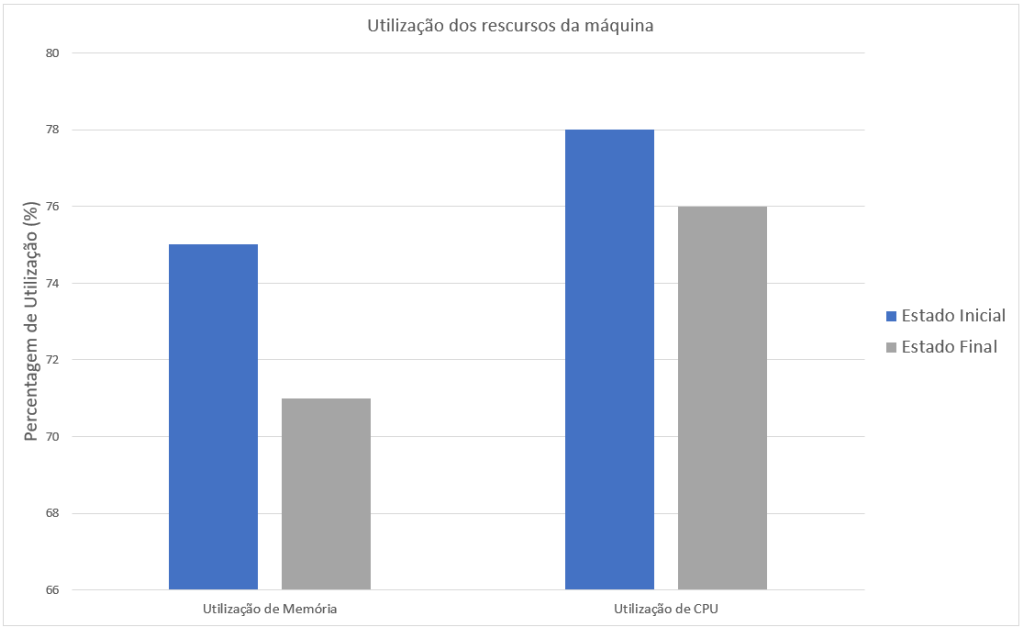

Figura 7.11: Gráfico com a utilização dos recursos da máquina no estado inicial e final

Observando o gráfico da figura 7.11 constata-se que a a atualização teve também impactos no que diz respeito à utilização de recursos da máquina fazendo com que os valores de

ambos os parâmetros monitorizados baixassem. O impacto maior ocorreu na utilização de memória sendo que a explicação para este acontecimento é o facto de terem sido introduzidos novos *garbage collectors* na nova versão do JDK. No caso da utilização de memória foram também observadas melhorias, embora que de forma ligeira.

Na extração das métricas produzidas pela ferramenta SonarQube foram também identificadas algumas métricas consideradas interessantes para o projeto e que vão ser apresentadas de seguida.

Métrica	**Estado Inicial**	**Estado final**
Linhas de código	49348	48956
Complexidade ciclomática	6930	6845
Complexidade cognitiva	4134	3817

Tabela 7.9: Tabela de métricas de código

Com a implementação das novas funcionalidades disponibilizadas na nova versão da linguagem foi possível melhorar o projeto também no que diz respeito ao número de linhas de código e à complexidade do mesmo, tanto ciclomática como cognitiva. Apesar deste parâmetros terem sido melhorados, o valor desta melhoria foi pouco significativo, ficando um pouco aquém das expectativas iniciais. O valor da complexidade cognitiva foi o mais significativo dos três e isto deve-se ao elevado número de expressões condicionais desnecessárias que foram reduzidas.

This page is intentionally left blank.

Capítulo 8

Conclusões

Esta dissertação de mestrado assumiu como objectivo a atualização do projeto Parse & Correlate (PARSEC), desenvolvido pela empresa Critical Software, e avaliar o impacto das alterações efectuadas no mesmo. A atualização principal seria ao nível da linguagem utilizando, passando este da versão 8 para a versão 11 da linguagem de programação Java.

Para conseguir alcançar este objetivo procedeu-se ao estudo do contexto em que este projeto está inserido, prosseguindo-se para o análise funcional do mesmo, com a elaboração de vários diagramas funcionais.

De seguida fez-se o levantamento das alterações produzidas entre as versões da linguagem em questão e também de técnicas e boas práticas a serem adoptadas para o processo de alteração.

Após isto foram identificados os requisitos para o processo de atualização assim como os riscos associados ao mesmo e delineados planos de mitigação para os mesmos.

Seguiu-se a fase de atualização, onde foi investida a maior parte do tempo deste projeto, em que além de alterações ao projeto foram ainda produzidos dados extraídos com recurso a ferramentas como JMeter e SonarQube.

Finalmente surgiu a fase de validação dos resultados obtidos onde foram analisados os resultados e feito o cruzamento destes com os requisitos estabelecidos anteriormente.

Desta análise concluiu-se que a fase da atualização que teve um impacto positivo no projeto, uma vez que conseguiu melhorar o mesmo, quer em termos de performance quer em termos de código e atualização de ferramentas utilizadas.

A fase que teve o maior contributo para a melhoria na performance da biblioteca PARSEC foi a atualização do JDK utilizado.

É importante também salientar a importância da produção de código bem estruturado e respeitando as boas práticas o que, neste caso em específico, acabou por ajudar no processo de *refactoring* principalmente, pois apesar de ser um projeto bastante extenso e com alguma complexidade em termos de contexto de negócio o código referente ao mesmo era fácil de ler e entender o que acabou por ser uma ajuda valiosa.

Por fim realçar ainda que existiram alguns contratempos durante o decorrer deste estágio, dos quais surgiu a necessidade de reajustar o âmbito do mesmo, mas que, apesar de tudo, os objetivos foram atingidos com sucesso pelo que o balanço final acabou por ser positivo.

This page is intentionally left blank.

Referências

[1] Smart Metering Implementation Programme. https://smartenergycodecompany.co.uk/smip/. Online; accessed 29 September 2020.

[2] Smart Metering Implementation Programme Report. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment{_}data/file/245736/smart{_}meters{_}domestic{_}leaflet.pdf. Online; accessed 29 September 2020.

[3] The seven key benefits of SMETS smart meters. https://www.smart-energy.com/industry-sectors/data{_}analytics/the-seven-key-benefits-of-smets-smart-meters/. Online; accessed 29 September 2020.

[4] Smart Metering Implementation Programme Policies - IEA. https://www.iea.org/policies/1376-smart-metering-implementation-programme. Online; accessed 29 September 2020.

[5] The Data Communications Company | Ofgem. https://www.ofgem.gov.uk/gas/retail-market/metering/transition-smart-meters/data-communications-company-ofgem-publications. Online; accessed 30 September 2020.

[6] About DCC. https://www.smartdcc.co.uk/about/. Online; accessed 30 September 2020.

[7] DCC User Gateway Interface Specification. https://www.smartdcc.co.uk/customer-hub/consultations/dcc-user-gateway-interface-specification/. Online; accessed 30 September 2020.

[8] Technical information on Smart Meters: Smart Metering. https://www.smartme.co.uk/technical.html. Online; accessed 30 September 2020.

[9] Message Mapping Catalogue consultation. https://www.smartdcc.co.uk/customer-hub/consultations/message-mapping-catalogue-consultation/. Online; accessed 1 October 2020.

[10] DCC User Interface Specification (DUIS) and Message Mapping Catalogue (MMC). https://www.smartdcc.co.uk/customer-hub/consultations/dcc-user-interface-specification-duis-and-message-mapping-catalogue-mmc/. Online; accessed 1 October 2020.

[11] History of Java | Javatpoint. https://www.javatpoint.com/history-of-java. Online; accessed 12 October 2020.

[12] History of Java programming language. https://www.freejavaguide.com/history.html. Online; accessed 1 October 2020.

[13] What are the major features of Java programming? https://www.tutorialspoint.com/What-are-the-major-features-of-Java-programming. Online; accessed 12 October 2020.

[14] Java features version-wise | BenchResources. https://www.benchresources.net/java-features-version-wise/. Online; accessed 13 October 2020.

[15] Java Latest Versions and Features | HowToDoInJava. https://howtodoinjava.com/java-version-wise-features-history/. Online; accessed 13 October 2020.

[16] Java SE versions history. https://www.codejava.net/java-se/java-se-versions-history. Online; accessed 13 October 2020.

[17] Changes for Java 1.1. https://www.cs.cornell.edu/andru/javaspec/1.1Update.html. Online; accessed 13 October 2020.

[18] The pros and cons of JDK 1.1 | InfoWorld. https://www.infoworld.com/article/2077643/the-pros-and-cons-of-jdk-1-1.html. Online; accessed 14 October 2020.

[19] Java Inner Class| javatpoint. https://www.javatpoint.com/java-inner-class. Online; accessed 14 October 2020.

[20] Advantage of Inner Class in Java. https://studyeasy.org/java/advantage-of-inner-class/. Online; accessed 14 October 2020.

[21] What is JDBC? https://www.educba.com/what-is-jdbc/. Online; accessed 14 October 2020.

[22] What is Java Database Connectivity (JDBC)? https://www.theserverside.com/definition/Java-Database-Connectivity-JDBC. Online; accessed 14 October 2020.

[23] List the advantages of JDBC. https://www.careerride.com/JDBC-Advantages.aspx. Online; accessed 14 October 2020.

[24] JDK 1.2: Immature, but tons of new abilities | InfoWorld. https://www.infoworld.com/article/2076575/jdk-1-2--immature--but-tons-of-new-abilities.html. Online; accessed 14 October 2020.

[25] Java 1.2 New Features. https://web.pa.msu.edu/reference/jdk-1.2.2-docs/relnotes/features.html{#}collections. Online; accessed 14 October 2020.

[26] Collections Framework Overview. https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html. Online; accessed 15 October 2020.

[27] Collections in Java | javatpoint. https://www.javatpoint.com/collections-in-java. Online; accessed 15 October 2020.

[28] Introduction to Collections. https://docs.oracle.com/javase/tutorial/collections/intro/index.html. Online; accessed 15 October 2020.

[29] Advantages and Disadvantages of the Collection Framework. https://www.roseindia.net/java/jdk6/collection-advantages-disadvantages.shtml. Online; accessed 15 October 2020.

[30] New Features and Enhancements. https://docs.oracle.com/cd/E19683-01/806-7930/features/index.html. Online; accessed 22 October 2020.

[31] JDK 1.4 Features. https://way2java.com/java-versions-2/jdk-1-4-j2se-4-version/. Online; accessed 22 October 2020.

[32] Using Java Assertions | Baeldung. https://www.baeldung.com/java-assert. Online; accessed 22 October 2020.

[33] Assertions em Java. https://www.devmedia.com.br/assertions-em-java/28781. Online; accessed 22 October 2020.

[34] Assertions in Java | GeeksforGeeks. https://www.geeksforgeeks.org/assertions-in-java/. Online; accessed 22 October 2020.

[35] Java Logging API. https://www.vogella.com/tutorials/Logging/article.html. Online; accessed 22 October 2020.

[36] The State of Logging in Java | Stackify. https://stackify.com/logging-java/. Online; accessed 22 October 2020.

[37] Experiences with the New Java 5 Language Features. https://www.oracle.com/technical-resources/articles/java/java-5-features.html. Online; accessed 22 October 2020.

[38] JDK 1.5 Features. https://way2java.com/java-versions-2/jdk-1-5-java-se-5-version/. Online; accessed 22 October 2020.

[39] For-each loop in Java | GeeksforGeeks. https://www.geeksforgeeks.org/for-each-loop-in-java/. Online; accessed 22 October 2020.

[40] Java For-each Loop | javatpoint. https://www.javatpoint.com/for-each-loop. Online; accessed 22 October 2020.

[41] For each loop in Java. https://www.learningjournal.guru/article/programming-in-java/for-each-loop-in-java/. Online; accessed 22 October 2020.

[42] Enum Types (The Java™ Tutorials). https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html. Online; accessed 23 October 2020.

[43] Enums In Java - with Examples, Advantages & Use Cases. https://javajee.com/blog/enums-in-java-with-examples-advantages-use-cases. Online; accessed 23 October 2020.

[44] Annotations in Java | GeeksforGeeks. https://www.geeksforgeeks.org/annotations-in-java/. Online; accessed 23 October 2020.

[45] Java Annotations tutorial with examples. https://beginnersbook.com/2014/09/java-annotations/. Online; accessed 23 October 2020.

[46] Java 7 Changes, Features and Enhancements | HowToDoInJava. https://howtodoinjava.com/java7/java-7-changes-features-and-enhancements/. Online; accessed 26 October 2020.

[47] JDK 1.7 Features. https://way2java.com/java-versions-2/jdk-1-7-features/. Online; accessed 26 October 2020.

[48] Java Try with Resources | javatpoint. https://www.javatpoint.com/java-try-with-resources. Online; accessed 26 October 2020.

[49] Java - Try with Resources | Baeldung. https://www.baeldung.com/java-try-with-resources. Online; accessed 26 October 2020.

[50] Java 8 Features | HowToDoInJava. https://howtodoinjava.com/java-8-tutorial/. Online; accessed 26 October 2020.

[51] Java 8 Features with Examples | JournalDev. https://www.journaldev.com/2389/java-8-features-with-examples. Online; accessed 26 October 2020.

[52] Java 8 - Lambda Expressions | Tutorialspoint. https://www.tutorialspoint.com/java8/java8{_}lambda{_}expressions.htm. Online; accessed 27 October 2020.

[53] Lambda Expressions in Java 8 | GeeksforGeeks. https://www.geeksforgeeks.org/lambda-expressions-java-8/. Online; accessed 27 October 2020.

[54] Stream In Java | GeeksforGeeks. https://www.geeksforgeeks.org/stream-in-java/. Online; accessed 27 October 2020.

[55] The Java 8 Stream API Tutorial | Baeldung. https://www.baeldung.com/java-8-streams. Online; accessed 27 October 2020.

[56] What are the advantages of Lambda Expressions in Java? https://www.tutorialspoint.com/what-are-the-advantages-of-lambda-expressions-in-java. Online; accessed 27 October 2020.

[57] Java 9 Features with Examples | JournalDev. https://www.journaldev.com/13121/java-9-features-with-examples{#}process-api. Online; accessed 27 October 2020.

[58] Java 9 Features | javatpoint. https://www.javatpoint.com/java-9-features. Online; accessed 27 October 2020.

[59] 9 New Java 9 Features | pluralsight. https://www.pluralsight.com/blog/software-development/java-9-new-features. Online; accessed 27 October 2020.

[60] Advantages and disadvantages of the Module System. https://www.tutorialspoint.com/what-are-the-advantages-and-disadvantages-of-the-module-system-in-jav
Online; accessed 28 October 2020.

[61] Java 10 New Features | Baeldung. https://www.baeldung.com/java-10-overview. Online; accessed 28 October 2020.

[62] What's New in Java 10 | Stackify. https://stackify.com/whats-new-in-java-10/. Online; accessed 28 October 2020.

[63] Java 10 LocalVariable Type-Inference | Baeldung. https://www.baeldung.com/java-10-local-variable-type-inference. Online; accessed 28 October 2020.

[64] Java 11 Features | JournalDev. https://www.journaldev.com/24601/java-11-features{#}42-java-string-methods. Online; accessed 28 October 2020.

[65] What is new in Java 11. https://mkyong.com/java/what-is-new-in-java-11/.

[66] Java 12 Features | JournalDev. https://www.journaldev.com/28666/java-12-features. Online; accessed 9 November 2020.

[67] Java 12: New Features | Stackify. https://stackify.com/java-12-new-features-and-enhancements-developers-should-know/. Online; accessed 9 November 2020.

[68] Collectors teeing | HowToDoInJava. https://howtodoinjava.com/java12/collectors-teeing-example/. Online; accessed 9 November 2020.

[69] What is new in Java 14. https://mkyong.com/java/what-is-new-in-java-14/. Online; accessed 12 November 2020.

[70] JDK14: Java 14 complete features | TechGeekNext. https://www.techgeeknext.com/java/java14-features{#}rp. Online; accessed 12 November 2020.

[71] Java 14 Record Keyword | Baeldung. https://www.baeldung.com/java-record-keyword. Online; accessed 12 November 2020.

[72] Introducing Java Record | DZone Java. https://dzone.com/articles/introducing-java-record. Online; accessed 12 November 2020.

[73] What's New in Java 15 | Baeldung. https://www.baeldung.com/java-15-new. Online; accessed 13 November 2020.

[74] Java 15 features | TechGeekNext. https://www.techgeeknext.com/java/java15-features. Online; accessed 13 November 2020.

[75] Sealed Classes and Interfaces in Java 15 | Baeldung. https://www.baeldung.com/java-sealed-classes-interfaces. Online; accessed 13 November 2020.

[76] Exploring Sealed Classes in Java | Rebel. https://www.jrebel.com/blog/sealed-classes. Online; accessed 13 November 2020.

[77] What Is Code Migration? https://www.easytechjunkie.com/what-is-code-migration.htm. Online; accessed 21 December 2020.

[78] 5 Reasons You Should Consider Migrating Your Legacy Systems | HowToDoInJava. https://howtodoinjava.com/resources/5-reasons-you-should-consider-migrating-your-legacy-systems/. Online; accessed 21 December 2020.

[79] Nine best practices for a successful migration. https://www.ibm.com/blogs/systems/nine-best-practices-for-a-successful-migration/. Online; accessed 21 December 2020.

[80] What is an Application Migration? Best Practices | Cisco. https://www.cisco.com/c/en/us/solutions/cloud/what-is-application-migration.html{#}{~}best-practices. Online; accessed 21 December 2020.

[81] Refactoring. https://refactoring.com/. Online; accessed 21 December 2020.

[82] Why We Refactor? https://www.groundai.com/project/why-we-refactor-confessions-of-github-contributors/1. Online; accessed 21 December 2020.

[83] What Is Code Refactoring | Perfectial. https://perfectial.com/blog/code-refactoring/. Online; accessed 21 December 2020.

[84] 7 Reasons Why Code Refactoring is Important in Software Development | GeeksforGeeks. https://www.geeksforgeeks.org/7-reasons-why-code-refactoring-is-important-in-software-development/. Online; accessed 22 December 2020.

[85] 7 Code Refactoring Techniques in Software Engineering | GeeksforGeeks. https://www.geeksforgeeks.org/7-code-refactoring-techniques-in-software-engineering/. Online; accessed 22 December 2020.

[86] Code Refactoring Techniques | DZone Agile. https://dzone.com/articles/code-refactoring-techniques. Online; accessed 22 December 2020.

[87] Code Refactoring Best Practices | AltexSoft. https://www.altexsoft.com/blog/engineering/code-refactoring-best-practices-when-and-when-not-to-do-it/. Online; accessed 22 December 2020.