FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO


# Design Automation

**Luís Maria Travassos Pinheiro Jorge Machado**

PARA APRECIAÇÃO POR JÚRI

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Mário Jorge Rodrigues de Sousa

Co-orientador: Eng. João Marques Martins - EFACEC

28 de junho de 2021

# Resumo

Ao longo dos últimos anos, foram-se denotando desenvolvimentos significativos no setor ferroviário, principalmente devido ao impacto que os sistemas informáticos possuem no mesmo. Estes sistemas, pela sua complexidade e, principalmente, pela sua criticidade, são regidos pela norma EN (*European Standard*) 50126, a qual é composta por outras duas: uma ligada ao *software* e proteção dos sistemas ferroviários (EN 50128) e outra relacionada com a segurança dos mesmos (EN 50129). A norma EN 50128, que servirá de ponto de partida para o projeto em questão, apresenta um conjunto de recomendações que permitem assegurar a correção do *software* e a confiança neste. Assim, a metodologia de desenvolvimento fica ao critério da entidade que desenvolve o *software*. Dependendo do sistema a projetar, a norma classifica o *software* em 4 níveis de SIL (*Safety Integrity Level*), contudo este tema será abordado posteriormente no presente relatório.

Até aos dias de hoje, sempre que se pretendia desenvolver um projeto para a criação de aplicações gráficas para sistemas de sinalização ferroviária, este era criado, quase na sua totalidade, manualmente, sujeito a erros e falhas. Assim, como solução, a automatização destes projetos permite a configuração sem entropias e mais eficaz, referindo a importância da rapidez na sua conclusão. Para além de evitar erros, elimina desperdícios no processo produtivo, o que se traduz num aumento de produtividade. A presente dissertação surge da necessidade da *EFACEC* em automatizar o processo de *design* dos seus produtos para soluções de sinalização ferroviária, minimizando assim a criação manual do mesmo.

O trabalho foi desenvolvido na sua totalidade em Java, no IDE (*Integrated Development Environment*) Eclipse. Esta linguagem está relacionada com a outra aplicação utilizada, o SCADE (*Safety Critical Application Development Environment*) Display (plataforma dedicada à modelação de interfaces homem-máquina) através das Java API's (*Application Programming Interface*), que permitiram a especificação do sistema. O SCADE Display propriamente dito permitiu visualizar o sistema especificado, isto é, permitiu interpretar o ficheiro sgfx criado a partir do RailML fornecido.

Este documento, para além de apresentar uma metodologia (automatização da criação de aplicações gráficas) aplicável no desenvolvimento de sistemas ferroviários, evidencia a aplicabilidade do método a desenvolver. O caso de estudo automatizado foi um sistema de encravamento. [1]

[1] Como nota adicional, os estrangeirismos utilizados, como por exemplo *software*, *design*, *default*, *flag*, entre outros, são termos bem aceites na comunidade para qual a presente dissertação se dirige. Assim, por este motivo, não serão alterados ao longo do mesmo.

# Abstract

Over the past few years, significant developments have taken place in the railway sector, mainly because of the impact that IT (Information Technology) systems have on it. These systems are governed by the EN 50126 standards, which is composed by two standards: one related to software and protection of railway systems (EN 50128) and another related to their safety (EN 50129). The EN 50128 standard, which will be the starting point of the project, presents a set of recommendations that ensure the correctness and reliability of the software. Thus, the development methodology is left to the discretion of the entity that are developing the software. Depending on the system to be designed, the standard classifies the software into 4 SIL levels, although this topic will be discussed later in this report.

Until these days, whenever a new project starts, it is almost entirely created manually, subject to errors and failures. Therefore, as a solution, the automation of these projects allows more effective configurations, highlighting the importance of the speed in their conclusion. Besides avoiding errors, it eliminates waste in the production process, which translates into an increase of productivity. This project arises from *EFACEC*'s need to automate the design process of its products for railway signaling solutions.

The work was developed entirely in Java, in Eclipse (IDE). This language is related to the other application used, SCADE Display (platform dedicated to modelling human-machine interfaces) through Java API's, which allowed the system specification. SCADE Display itself allowed the specified system to be visualised (interpreted the sgfx file created from the provided RailML).

This document, besides presenting a methodology (automation of the creation of graphic applications) applicable in the development of railway systems, shows the applicability of the method to be developed. The case study to be automated is an interlocking system.

# Agradecimentos

Em primeiro lugar, agradeço ao Professor Mário Sousa, meu orientador, pelos ensinamentos, incentivos e ajuda na elaboração da dissertação, sem a sua ajuda não seria possível obter um resultado tão positivo, obrigado!

Em segundo lugar, agradeço à EFACEC, que mesmo não estando nas instalações presencialmente, disponibilizou todos os meios necessários para a elaboração do trabalho, neste ano atípico.

Um agradecimento ao Engenheiro João Marques Martins, pelo auxílio prestado aquando do início da dissertação e pelas reuniões semanais, tanto para mostrar o trabalho desenvolvido como para esclarecimento de qualquer dúvida existente.

Um agradecimento especial ao engenheiro Daniel Martins, pelo apoio incansável dia após dia, que sempre que algum problema surgiu estava disponível para me ajudar.

A todos os meus colegas de faculdade, que, ao longo destes anos, contribuíram e ajudaram neste percurso. A eles, um muito obrigado e boa sorte para a conclusão da dissertação e para o futuro que se avizinha!

À minha namorada, pela paciência e compreensão prestada ao longo do meu percurso, principalmente na fase final. Sem ela não teria sido possível!

Por fim, e não menos importante, agradeço à minha família. Aos meus pais, que sempre me apoiaram e decerto apoiarão no futuro e que sem eles não teria chegado onde cheguei; à minha irmã, pela paciência demonstrada; aos meus tios pelas palavras de incentivo; aos meus primos que sempre me acompanharam, nem que fosse apenas para desabafar; e, por último, aos meus avós, que sempre se mostraram presentes e me ajudaram com um sorriso ou uma chamada.

A todos, o meu sincero obrigado, sem vocês nada disto seria possível.

Luís Machado

*"This is how you do it: you sit down at the keyboard and you put one word after another until its done. It's that easy, and that hard"*

Neil Gaiman

# Conteúdo

# Lista de Figuras

# Lista de Códigos

# Lista de Tabelas

# Abreviaturas e Símbolos

| | |
|---|---|
| API | Application Programming Interface |
| COTS | Commercial-Of-the-Shelf hardware platforms |
| EN | European Standard |
| HMI | Human Machine Interface |
| IDE | Integrated Development Environment |
| IEC | International Electrotechnical Commission |
| IP | Internet Protocol |
| IT | Information Technology |
| JAXB | Java Architecture for XML Binding |
| MBSE | Model Based System Engineering |
| RAMS | Reliability, Availability, Maintainability, and Safety |
| RAS | Railway Automation Suite |
| SIL | Safety Integrity Level |
| SCADE | Safety Critical Application Development Environment |
| XML | Extensible Markup Language |

# Capítulo 1

# Introdução

O presente relatório foi desenvolvido no âmbito da Dissertação, o qual apresentará uma revisão bibliográfica sobre os sistemas de encravamento, os sistemas ferroviários e as suas normas e sobre a aplicação e os métodos/ficheiros utilizados. Este documento visa, também, caracterizar o problema, apresentar o plano de trabalho das tarefas realizadas, demonstrar a forma de realização das mesmas e, por último, caracterizar a utilização do programa desenvolvido e respetivos testes efetuados.

Este capítulo estará dividido em três secções, a secção [1.1] contextualiza o tema da presente tese, a secção [1.2] apresenta a motivação que levou à escolha deste tema e, por último, a secção [1.3] enuncia os objetivos para o trabalho elaborado.

## 1.1 Contexto

A *EFACEC*, com origem há mais de 100 anos, iniciou a sua atividade com a denominação social de "A Moderna - Sociedade de Serração Mecânica". Atualmente, o seu capital é detido pela *EPS EFACEC Power Solutions S.A.*. Esta sociedade, fundada em 2014, englobava um grupo de empresas com todos os meios de produção, tecnologias e competências para atividades inseridas nos domínios das soluções de Energia, Engenharia, Ambiente, Transportes e Mobilidade Elétrica. Aposta no desenvolvimento de produtos e sistemas com elevado valor acrescentado, baseada numa tecnologia própria, com forte capacidade na inovação, e, acima de tudo, com uma referência internacional proeminente no domínio da energia, ambiente, indústria, mobilidade e transportes [9].

Posto isto, a *EFACEC* desenvolve soluções *turnkey* para os mais variados segmentos, nos quais se inserem o segmento ferroviário, o material circulante e a sinalização. A segurança ferroviária, o desenvolvimento de sistemas de segurança de baixo custo, e os sistemas de *hardware* e *software* para aplicações gerais de automação (COTS (*Commercial-Of-the-Shelf hardware platforms*)) permitiram à *EFACEC* uma nova abordagem ao nível das novas gerações de soluções baseadas em [10]:

- Arquiteturas abertas;

- Certificados de segurança;
- Redes de comunicações, baseadas em tecnologias IP (*Internet Protocol*).

No seu vasto portfólio, a *EFACEC* oferece uma variedade de produtos destinados a sistemas ferroviários, cobrindo comboios e metro ligeiro. Sempre que se inicia um novo projeto, há a necessidade de configurar produtos genéricos de acordo com o *design* requerido. A mais valia aqui aportada será a geração automática destes projetos para os sistemas ferroviários, tornando a componente manual mais residual, eliminando entropias, tais como atrasos e possíveis falhas. Além disso, a configuração manual envolve verificações constantes e demoradas de erros que ocorreram durante o processo de configuração. A fim de diminuir o trabalho manual desta configuração, a *EFACEC* criou uma nova ferramenta, a RAS (Railway Automation Suite). Esta ferramenta tem como objetivo automatizar a criação das configurações necessárias para todos os produtos utilizados no âmbito do projeto e agrega várias operações de automatização. Este projeto, que será inserido nesta ferramenta, focou-se numa dessas operações, apresentada em capítulos posteriores.

Para finalizar, a *EFACEC* desenvolve e comercializa sistemas de segurança, realçando que o sistema a estudar (o encravamento) é certificado pelo SIL 4 (*Safety Integrity Level*), de acordo com as normas ferroviárias.

## 1.2 Motivação

O projeto desenvolvido permitirá a redução de erros e falhas na gestão dos sistemas ferroviários, isto é, como a configuração manual acarreta bastantes problemas e elevados tempos de verificação dos mesmos, este processo será automatizado, eliminando assim os problemas e reduzindo os tempos de testes e possíveis erros. Dentro destes sistemas ferroviários, os que se destacam são os sistemas de sinalização para linhas ferroviárias, bem como os sistemas de proteção de passagens de níveis. A segurança é o cerne do novo portfólio das soluções agora desenvolvidas pela empresa, o que será um desafio. Com o projeto desenvolvido inserido na ferramenta acima mencionada (RAS), pretendeu-se eliminar erros criados durante a configuração manual, agilizando este processo.

Em resumo, a principal motivação por detrás deste projeto foi o desafio que a automatização das configurações dos produtos *EFACEC*, de acordo com um determinado *design*, apresentou, a par da compreensão e aplicação dos princípios dos sistemas de automação em sistemas de sinalização, baseados em elevados padrões de segurança, com estrito cumprimento dos requisitos.

## 1.3 Objetivos

O principal objetivo da Dissertação foi a configuração automática do sistema de encravamento para projetos específicos, críticos em termos de segurança de acordo com o nível SIL4 (detalhado

na subsecção [2.2]). Assim, as ferramentas desenvolvidas para automatizar a criação destas configurações, para estes produtos, poderão ter um forte impacto no seu comportamento e, como tal, devem também ser certificados de acordo com as normas ferroviárias, nomeadamente a EN (*European Standard*) 50128 (norma padrão para sistemas de *software*).

O presente projeto desenvolveu-se em 4 fases:

- Compreensão dos princípios do *Design Automation*;
- Compreensão dos princípios das normas ferroviárias, nomeadamente da EN 50128;
- Compreensão dos princípios dos sistemas de sinalização;
- Implementação das ferramentas para automatizar a criação de configurações de produtos críticos para a segurança.

Concluindo, e a fim de entender melhor o processo envolvido no presente projeto, é apresentado o seguinte diagrama:

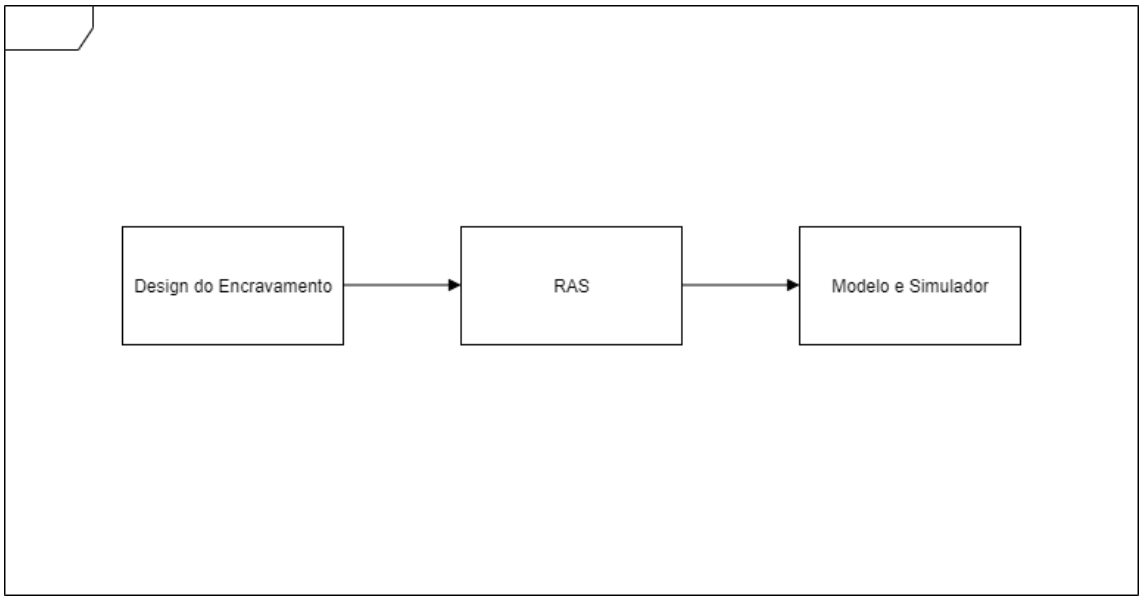


Figura 1.1: Diagrama geral do projeto

# Capítulo 2

# Revisão Bibliográfica

Este capítulo é constituído por seis partes. A primeira parte trata de definir, ao pormenor, o sistema de encravamento [2.1], que foi o sistema de sinalização utilizado para esta dissertação. A segunda é composta pelas diferentes normas que existem [2.2]. Na terceira parte será introduzido o conceito de sistemas Ansys [2.3], destacando a explicação da aplicação utilizada, o SCADE Display. Por último, na quarta parte será explicado o RailML [2.4], o JAXB (método de leitura do ficheiro RailML) [2.5] e serão realizadas comparações com outras possíveis tecnologias [2.6].

## 2.1 Sistema de Encravamento

O projeto proposto visa modelar um sistema de sinalização que permita a gestão do tráfego de uma linha ferroviária. A esta gestão chama-se encravamento e envolve a prevenção da existência de mais do que um comboio num único troço de via. O encravamento é a função central para garantir que os comboios circulem em segurança em termos técnicos e é realizado com o apoio de equipamentos de via tais como sinais ferroviários, agulhas, blocos, circuitos de via e/ou contadores de eixos, balizas, entre outros [2]. Um exemplo real de um sistema de encravamento encontra-se retratado na imagem seguinte:


Figura 2.1: Retirado de [1], Sistema (real) de um encravamento

De uma forma mais detalhada e atendendo às denominações mais importantes para a realização do projeto (sendo que este sistema é composto por vários elementos), as agulhas são o conjunto de peças movíveis e paralelas entre si, integrantes do sistema de mudança de via, e cujo deslocamento leva o comboio a passar de uma via para outra; as balizas são os equipamentos pertencentes ao sistema de controlo de velocidade (que garante o cumprimento da sinalização e das velocidades máximas) e ao sistema de localização dos veículos, junto a pontos chaves (sinais, limites de velocidade) que permitem transmitir informações dos mesmos aos comboios; os circuitos de via são equipamentos que detetam a presença de uma composição (conjunto de veículos ferroviários) num determinado troço de via; os blocos de sinalização são sistemas que salvaguardam o distanciamento entre dois comboios a circular na mesma via e impedem que dois comboios em sentidos opostos se encontrem na mesma secção em simultâneo; o sinal é um aparelho na qual é dada indicação convencional com respeito à circulação dos veículos ferroviários; a secção é um troço de via localizado entre dois pontos singulares; e, por último, os elementos de rede que são os elementos constituintes do troço ferroviário [11] [12]. Para o conseguir (interligar todas as funcionalidades acima descritas), o encravamento obtém informações sobre a ocupação da via e a posição dos elementos movíveis da mesma.

Existem dois princípios básicos deste sistema que são aplicados, tecnicamente, através de funções de encravamento, das quais [2]:

- Um sinal só pode permitir o movimento de um comboio se todos os elementos movíveis da via estiverem na posição correta e trancados;
- Com espaçamento de comboios por blocos fixos, um comboio só pode ser autorizado a entrar num troço que esteja livre de outro material circulante, ou seja, nenhum outro comboio pode ser autorizado a entrar nesse troço.

Todo o controlo destes equipamentos era feito mecânica e/ou eletromecanicamente, mas, atualmente, o funcionamento dos caminhos-de-ferro é cada vez mais suportado pela eletrónica e por computadores. No que diz respeito à sinalização ferroviária, existem aplicações programáveis que controlam estes dispositivos.

### 2.1.1 Requisitos Básicos

Ao contrário de outras formas de transporte, as linhas ferroviárias têm duas características decisivas que determinam os requisitos de segurança [2]:

- A fricção é baixa e, por conseguinte, as distâncias de travagem são longas. Isto diminui a capacidade do maquinista de evitar colisões com outros veículos ferroviários, unidades movíveis de travessia de vias ou quaisquer obstáculos externos;
- Os veículos ferroviários são guiados pelos carris, e a manutenção dos mesmos é fundamental para a segurança da linha.

Para tal, pode-se concluir que as seguintes funções de proteção para os caminhos-de-ferro têm de ser incorporadas, como mostra a imagem seguinte:

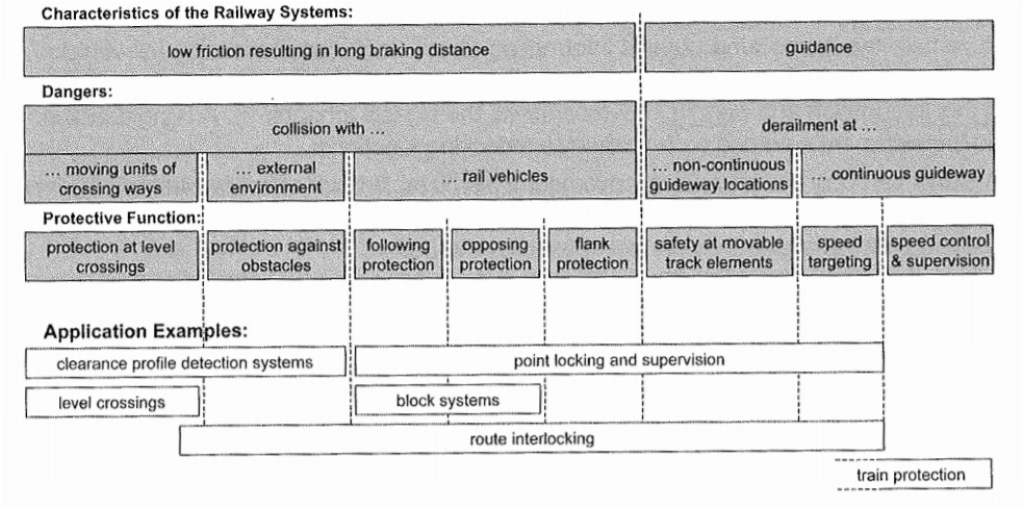


Figura 2.2: Retirado de [2], Perigos e medidas de segurança nas operações ferroviárias

### 2.1.2 Percurso

O percurso é, de uma forma resumida, uma zona delimitada de um itinerário. Tem início num sinal que comanda este itinerário, e vai até um dado ponto a jusante do final [11]. É possível afirmar que o percurso é composto por carris ferroviários.

Os carris ferroviários são uma forma de salvaguardar um percurso completo de um comboio em toda a extensão sobre a qual o movimento é permitido. Logicamente, apenas é permitida a presença de comboios, na mesma linha, no mesmo sentido. Quando todos os elementos movíveis da via estão na posição correta para que um comboio possa tomar uma determinada direção, este percurso é chamado "caminho". As partes de um percurso que precisam de ser salvaguardadas são [2]:

- O ***running path***, o que significa que os carris serão regularmente ocupados pelo comboio durante a sua travessia do itinerário;
- Uma ***overlap***, que se pode estender por um pequeno troço para proteger o comboio contra uma pequena ultrapassagem de outro (no fim da secção protegida);
- As ***flank areas***, que protegem o comboio contra movimentos não autorizados a partir do flanco (exterior);
- A ***end of the protection section*** para proteger os comboios de movimentos opostos;

- A ***start section***, que está na retaguarda do sinal de entrada do itinerário, mas ocupado pelo comboio de partida ou situado entre a cabeça do comboio de partida e o sinal de entrada do itinerário.

A imagem demonstra a informação acima descrita:

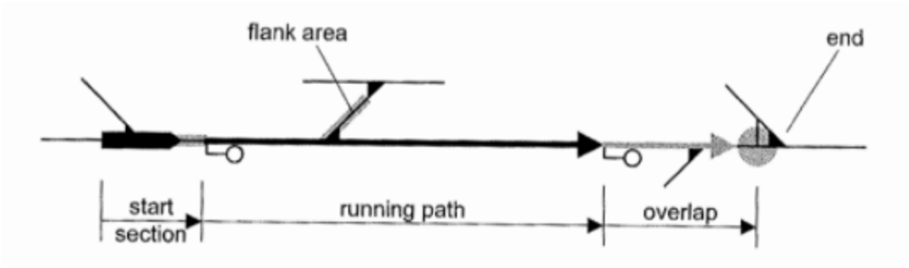


Figura 2.3: Retirado de [2], Partes lógicas de um percurso

De realçar que na entrada de cada secção existem, tanto sensores que detetam a presença de um comboio no troço de via, como também a sinalização (equipamento físico) inerente. É possível verificar na figura acima apresentada dois sinais (um que dita o início da secção e um que dita o fim do ***running path***).

## 2.2 Normas Ferroviárias

No domínio ferroviário da União Europeia, o desenvolvimento e a conceção dos sistemas ferroviários devem responder a certas normas, a fim de garantir a segurança e a integridade dos mesmos [13]. Atualmente, existem três normas que foram desenvolvidas [14] [15]:

- EN 50126, que está relacionada com o sistema como um todo, mais precisamente à especificação e demonstração de fiabilidade, disponibilidade, manutenibilidade e segurança (RAMS) do mesmo;

- EN 50129, relacionada com a segurança funcional do sistema, mais especificamente de sistemas eletrónicos de segurança para a sinalização;

- EN 50128, que está relacionada com a conceção e desenvolvimento de *software* integrado nos sistemas, isto é, *software* para sistemas de proteção e comando ferroviário.

Embora existam mais normas, as três normas acima mencionadas são as que se devem focar e são chamadas normas complementares da norma IEC (*International Electrotechnical Commission*) 61508.

A IEC 61508 é a norma que estabelece os requisitos para o desenvolvimento e implementação da segurança em sistemas eletrónicos e/ou programáveis industriais em geral. Esta norma está dividida em sete partes e entre elas há princípios gerais e exemplos de métodos que determinam o nível de integridade e segurança, que se designa por SIL (*Safety Integrity Level*). Na IEC 61508 existem quatro níveis, variando entre 1 e 4, e baseia-se na quantidade de redução de risco necessária para manter uma taxa aceitável de falhas. Cada um dos quatro níveis de SIL representa uma ordem de magnitude de redução do risco - quanto maior for o nível, maior será o impacto de uma falha e, por conseguinte, menos desejável se torna esse risco. Portanto, quanto maior for o nível de SIL, menor será a taxa de insucesso aceitável. O SIL 4 tem o nível de segurança mais elevado, enquanto que o SIL 1 tem o mais baixo. Não obstante o exposto, existem várias maneiras de definir estes níveis SIL [3].

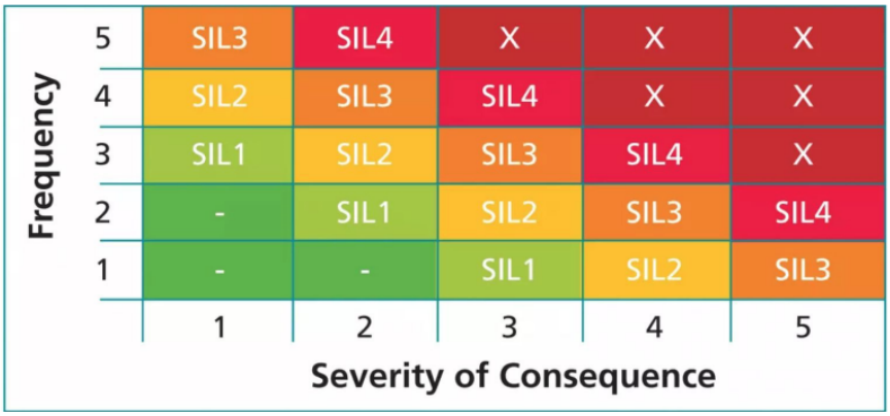


Figura 2.4: Retirado de [3], Níveis SIL

Na prática, os sistemas de sinalização, enquanto sistemas críticos, têm de ser certificados de acordo com as normas ferroviárias (EN 50126, EN 50128 e EN50129) para o nível de integridade pretendido (SIL).

### 2.2.1 Classes de Ferramentas de acordo com a Norma EN 50128

Dentro da norma EN 50128 existem ferramentas de *software* que influenciam direta ou indiretamente um sistema de segurança, neste caso o sistema de encravamento. Por outras palavras, o objetivo é mitigar o impacto de potenciais falhas das ferramentas, o qual pode não ser detetado por medidas técnicas ou organizacionais fora das mesmas. Para este fim, as ferramentas de *software* são enquadradas em três classes, T1, T2 e T3 [16].

Esta subsecção explicará brevemente T1 e T3, e a subsecção seguinte explicará a ferramenta que será utilizada para o projeto (T2).

- Ferramenta classe T1, que não gera quaisquer resultados que possam contribuir direta ou indiretamente para o código executável (incluindo dados) do *software*. Alguns exemplos da

classe T1 incluem um editor de texto, um requisito ou uma ferramenta de apoio à conceção dos sistemas sem capacidades de geração automática de código;

- Ferramenta classe T3, que gera resultados que podem contribuir para o código executável (incluindo dados) do sistema relacionado com a segurança. Exemplos de uma classe T3 incluem um compilador de código fonte, um compilador de dados/algoritmos, uma ferramenta para alterar pontos de ajuste durante as operações do sistema, um compilador de otimização [16], entre outros.

Quando as ferramentas estão a ser utilizadas como um substituto para operações manuais, a prova da integridade das ferramentas tem as mesmas etapas do processo como se a saída fosse feita em operações manuais. Estas etapas de processo podem ser substituídas se for apresentada uma argumentação sobre a integridade dos resultados das ferramentas e o nível de integridade do *software* não for diminuído por essa substituição [16].

#### 2.2.1.1 Ferramenta classe T2

Esta classe, essencial para a norma EN 50128, suporta o teste ou verificação do *design* ou código executável, mas não cria diretamente erros no mesmo. Exemplos desta classe poderão ser [16]:

- Um gerador de testes;
- Uma ferramenta de medição de cobertura de teste;
- Uma ferramenta de análise estática.

Esta classe tem requisitos necessários que devem ser cumpridos, tais como [16]:

- As ferramentas de *software* devem ser selecionadas como parte coerente das atividades de desenvolvimento do *software*;
- A selecção das ferramentas na classe T2 deve ser justificada (a justificação deve incluir a identificação de potenciais falhas que podem ser injetadas na saída das ferramentas e medidas para evitar tais falhas), isto é, a especificação da arquitetura de *software* deve justificar que as técnicas, medidas e ferramentas escolhidas satisfazem a segurança requerida no SIL;
- Todas as ferramentas da classe T2 devem ter um manual que defina claramente o comportamento da ferramenta e quaisquer restrições à sua utilização;
- A gestão da configuração deve assegurar que, para as ferramentas desta classe, apenas sejam utilizadas versões justificadas;
- Cada nova versão de uma ferramenta que seja utilizada deve também ser justificada. Esta justificação pode basear-se em provas fornecidas anteriormente, se forem fornecidas provas suficientes que as diferenças funcionais (se existirem) não afetarão a compatibilidade das

mesmas com o resto do conjunto de ferramentas e que a nova versão não contenha falhas significativas e/ou desconhecidas.

Esta ferramenta vai servir de apoio para verificar se o resultado final do projeto (o encravamento e a sinalização) cumpriu ou não os requisitos mencionados anteriormente, isto através de testes internos efetuados pela empresa. Não sendo a principal ferramenta para o teste do encravamento, entra na definição de T2. É possível assumir que, como o encravamento necessita de cumprir o SIL 4, esta classe (T2) estará enquadrada neste nível de segurança.

## 2.3 A família de produtos ANSYS Systems/Embedded Software

O ANSYS SCADE® é um ambiente de desenvolvimento para soluções *Safety Critical*, suportando todo o fluxo de trabalho, desde a conceção de requisitos até à sua implementação e verificação [17]. É composto por 5 tipos, o SCADE LifeCycle, o SCADE Display, o SCADE System, o SCADE Suite, e por fim, o SCADE Test, como se pode verificar na imagem abaixo apresentada:


Figura 2.5: Retirado de [4], Família Ansys

Infra será sucintamente explicada a família SCADE, contudo, a que foi utilizada foi o SCADE Display.

### 2.3.1 SCADE System

O SCADE System® é uma plataforma da família Ansys. A sua utilização permite cumprir elevados requisitos de fiabilidade, fornecendo apoio a processos de engenharia de sistemas industriais, tais como a EN 50126. Qualquer arquitetura desenvolvida neste ambiente é realizada através de diagrama de blocos [18].

### 2.3.2 SCADE Suite

O SCADE Suite® é um ambiente de desenvolvimento de *software* crítico. É o único ambiente de conceção integrada que abrange a gestão de requisitos, a conceção baseada em modelos, a simulação, a verificação, a interoperabilidade com outras ferramentas e plataformas, entre outros. Dos exemplos desenvolvidos nesta aplicação destacam-se sistemas de controlo de motores, pilotos automáticos, proteção contra excessos de velocidade, sistemas de travagem e sinalização ferroviária.[17].

### 2.3.3 SCADE Test

O SCADE Test® é um ambiente de testes. Em virtude da morosidade que ocorre, quer na sua realização, quer na sua manutenção, os engenheiros utilizam o SCADE Test para tarefas de verificação e validação da arquitetura desenvolvida [17]. Este automatismo apresenta bastantes vantagens, como por exemplo, uma maior fiabilidade nos resultados e redução significativa do seu tempo de execução, permitindo dessa forma aumentar o número de testes efetuado.

### 2.3.4 SCADE Display


Figura 2.6: Retirado de [5], SCADE Display

O SCADE Display® é uma ferramenta de desenvolvimento e de prototipagem de interfaces HMI (*Human Machine Interface*). É utilizada por empresas de amplas áreas de atividade, como aerospaciais, transportes ferroviários, industriais, entre outros [17]. É vital na conceção de sistemas de visualização integrados (painéis de controlo, instrumentação digital, velocímetros, troços ferroviários), mas também para criar simulações. Esta é a aplicação com base na qual se desenvolveu esta dissertação.

O SCADE Display foi, então, utilizado para a visualização de uma infraestrutura interligada a um sistema ferroviário [19]. Para o projeto em questão, a plataforma interpretou o ficheiro sgfx gerado pelo programa desenvolvido. Este sgfx é do tipo xml com características específicas para o SCADE Display.

### 2.3.5 SCADE LifeCycle

O SCADE LifeCycle® é um programa que permite, ao seu utilizador, a gestão do ciclo de vida da arquitetura por si desenvolvida. O SCADE LifeCycle melhora as funcionalidades do SCADE System, do SCADE Suite, do SCADE Display e do SCADE Test [17], proporcionando um conjunto de recursos de rastreabilidade e documentação, que estão disponíveis desde a fase inicial até à fase de conclusão e de testes [20].

## 2.4 RailML

O RailML é um esquema de dados *open-source* que foi desenvolvido para simplificar a transferência dos mesmos entre vários programas informáticos de simulação e operações ferroviárias [21]. Este tipo de linguagem é baseada em XML (*Extensible Markup Language*), o que significa que este ficheiro inclui tanto os dados como as suas descrições [6], como por exemplo as posições absolutas dos elementos ferroviários. O elemento raiz do documento RailML chama-se <railml> e os subesquemas contêm os dados relevantes sobre os caminhos-de-ferro, tais como infraestruturas, material circulante, entre outros, como é demonstrativo na seguinte figura:

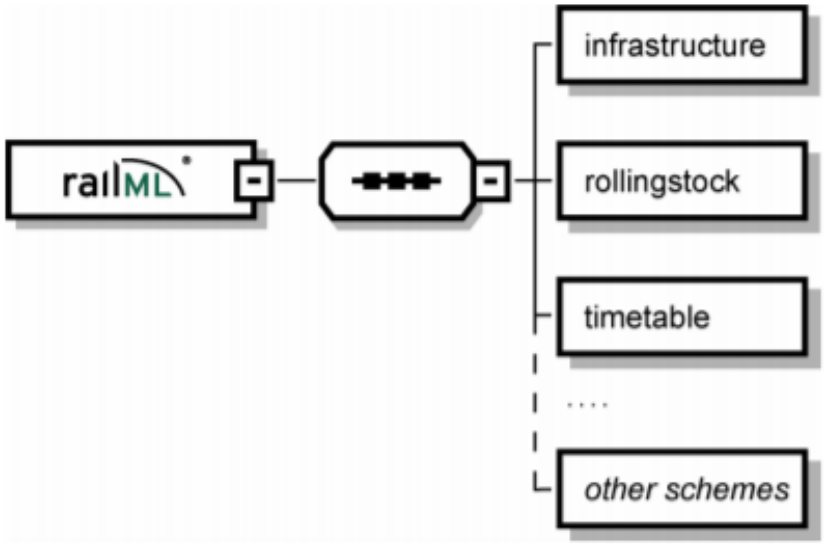


Figura 2.7: Retirado de [6], Esquemas do RailML

Utilizando esta abordagem flexível, cada aplicação individual compatível com o RailML determina que tipos particulares de dados devem ser utilizados.

Outra especificidade são os dados dos documentos RailML, que são organizados por elementos e atributos. Um ficheiro RailML começa com uma etiqueta inicial <railml>, e termina com uma etiqueta final </railml> [6]. Um elemento pode ter atributos para descrição mais detalhada e pode também conter elementos adicionais [6]. Um exemplo de um excerto de um ficheiro RailML é:

```
<signalsIS>
    <signalIS id="sig19" isSwitchable="false">
        <name name="S3" language="en"/>
        <spotLocation id="sig19_sloc01" netElementRef="ne5" applicationDirection="reverse" intrinsicCoord="0.7097"/>
        <designator register="_Hjallese_1_0" entry="SIGNAL S3"/>
    </signalIS>
    <signalIS id="sig23" isSwitchable="false">
        <name name="S1" language="en"/>
        <spotLocation id="sig23_sloc01" netElementRef="ne1" applicationDirection="reverse" intrinsicCoord="0.6386"/>
        <designator register="_Hjallese_1_0" entry="SIGNAL S1"/>
    </signalIS>
    <signalIS id="sig24" isSwitchable="false">
        <name name="S2" language="en"/>
        <spotLocation id="sig24_sloc01" netElementRef="ne2" applicationDirection="reverse" intrinsicCoord="0.3977"/>
        <designator register="_Hjallese_1_0" entry="SIGNAL S2"/>
    </signalIS>
    <signalIS id="VirtSig48" isSwitchable="false">
        <name name="E11" language="en"/>
        <spotLocation id="VirtSig48_sloc01" netElementRef="ne1" applicationDirection="normal" intrinsicCoord="0.8759"/>
        <designator register="_Hjallese_1_0" entry="SIGNAL E11"/>
    </signalIS>
    <signalIS id="VirtSig49" isSwitchable="false">
        <name name="E20" language="en"/>
        <spotLocation id="VirtSig49_sloc01" netElementRef="ne2" applicationDirection="normal" intrinsicCoord="0.7725"/>
        <designator register="_Hjallese_1_0" entry="SIGNAL E20"/>
    </signalIS>
    <signalIS id="VirtSig50" isSwitchable="false">
        <name name="E21" language="en"/>
        <spotLocation id="VirtSig50_sloc01" netElementRef="ne4" applicationDirection="normal" intrinsicCoord="0.7234"/>
        <designator register="_Hjallese_1_0" entry="SIGNAL E21"/>
    </signalIS>
</signalsIS>
<switchesIS>
    <switchIS id="swi5" continueCourse="right" branchCourse="left" type="ordinarySwitch">
        <name name="P4" language="en"/>
        <spotLocation id="swi5_sloc01" netElementRef="ne4" applicationDirection="reverse" intrinsicCoord="0.0000"/>
        <designator register="_Hjallese_1_0" entry="SWITCH P4"/>
        <leftBranch netRelationRef="nr_ne4ne8_swi5" branchingSpeed="0" joiningSpeed="0" radius="0"/>
        <rightBranch netRelationRef="nr_ne3ne4_swi5" branchingSpeed="0" joiningSpeed="0" radius="500"/>
    </switchIS>
    <switchIS id="swi6" continueCourse="right" branchCourse="left" type="ordinarySwitch">
        <name name="P1" language="en"/>
        <spotLocation id="swi6_sloc01" netElementRef="ne7" applicationDirection="normal" intrinsicCoord="1.0000"/>
        <designator register="_Hjallese_1_0" entry="SWITCH P1"/>
        <leftBranch netRelationRef="nr_ne5ne7_swi6" branchingSpeed="0" joiningSpeed="0" radius="0"/>
        <rightBranch netRelationRef="nr_ne3ne7_swi6" branchingSpeed="0" joiningSpeed="0" radius="-500"/>
    </switchIS>
    <switchIS id="swi7" continueCourse="right" branchCourse="left" type="ordinarySwitch">
        <name name="P3" language="en"/>
        <spotLocation id="swi7_sloc01" netElementRef="ne7" applicationDirection="reverse" intrinsicCoord="0.0000"/>
        <designator register="_Hjallese_1_0" entry="SWITCH P3"/>
        <leftBranch netRelationRef="nr_ne6ne7_swi7" branchingSpeed="0" joiningSpeed="0" radius="-500"/>
        <rightBranch netRelationRef="nr_ne1ne7_swi7" branchingSpeed="0" joiningSpeed="0" radius="0"/>
    </switchIS>
    <switchIS id="swi8" continueCourse="right" branchCourse="left" type="ordinarySwitch">
        <name name="P2" language="en"/>
        <spotLocation id="swi8_sloc01" netElementRef="ne2" applicationDirection="normal" intrinsicCoord="0.0000"/>
        <designator register="_Hjallese_1_0" entry="SWITCH P2"/>
        <leftBranch netRelationRef="nr_ne2ne6_swi8" branchingSpeed="0" joiningSpeed="0" radius="-500"/>
        <rightBranch netRelationRef="nr_ne2ne8_swi8" branchingSpeed="0" joiningSpeed="0" radius="0"/>
    </switchIS>
```

Figura 2.8: Exemplo de um excerto de um ficheiro RailML

A infraestrutura presente no RailML poderá ser composta por vários elementos ferroviários, tais como sinais, agulhas, secções, elementos de rede, entre outros. No exemplo acima representado é possível verificar a existência de seis sinais (três sinais normais (sig) e três virtuais (VirtSig)) e de três agulhas, bem como dos seus atributos (nome, elemento de rede associado, Id). Como nota adicional, os sinais normais são sinais físicos (à vista dos maquinistas) e os sinais virtuais são apenas elementos lógicos necessários para suportar algumas funcionalidades, como por exemplo o estabelecimento de itinerários para zonas não sinalizadas fisicamente. Outra parte integrante deste tipo de ficheiro são as coordenadas dos componentes ferroviários e dos respetivos elementos de rede, que servirão para os dispor em qualquer *design*.

## 2.5 JAXB

O JAXB (*Java Architecture for XML Binding*) é um dos métodos relativos à leitura e utilização da informação presente em ficheiros XML, seja ela raiz, elemento ou atributo [22] [23]. Como, neste caso, o ficheiro RailML é do tipo XML, aplica-se na perfeição. De forma figurativa, a seguinte imagem demonstra a utilização do método:

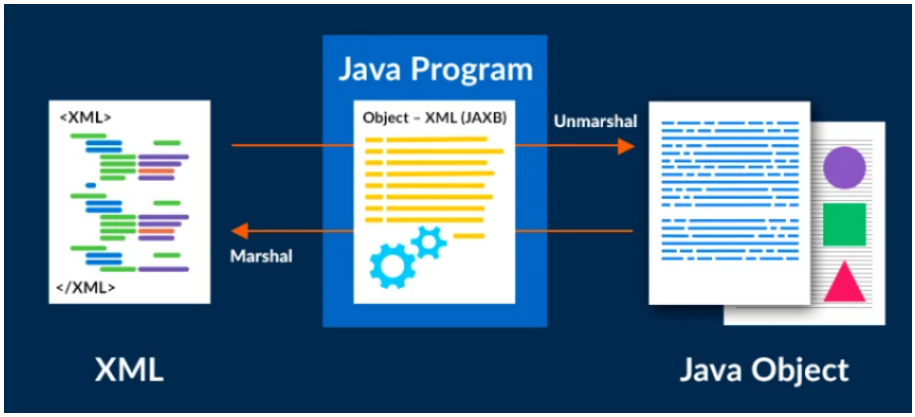


Figura 2.9: Retirado de [7], Método JAXB

Como se pode verificar pela figura acima apresentada, este método, não só permite converter a informação presente num ficheiro XML em objetos Java (*Unmarshal*), como também converter objetos em informação para o XML (*Marshal*). Alguns dos métodos presentes no JAXB são o @XmlRootElement, que permite ler o elemento raiz do ficheiro, o @XMLElement, que permite a leitura de um qualquer elemento do XML, o @XMLAttribute que acede ao atributo do elemento, entre outros.

Um exemplo de utilização deste método é o seguinte:

```
public class name {
    |
    public String getName() {
        return name;
    }

    public name() {
        super();
    }

    @XmlAttribute(name = "name")
    private String name;
}
```

Figura 2.10: Exemplo do método JAXB

É possível verificar, através da imagem, que o que se escreve no parênteses do @XMLAttribute deve ser igual ao que consta no ficheiro a ler. Já o *getter* utilizado, o (*getName()*), permite aceder ao valor (em formato *string*) desse mesmo atributo.

A explicação, mais detalhada, da forma de utilização estará presente na subsecção [4.2].

## 2.6 Comparações

Em relação ao RailML, não existe outro tipo de linguagem universal, baseada em XML, associada aos sistemas ferroviários, pelo que foi obrigatório e exclusivo o seu uso.

Em complemento, uma vez que o objetivo do próprio projeto é a geração de ficheiros SCADE Display e que a empresa já fazia uso desta ferramenta anteriormente, para efeitos deste projeto, esta ferramenta torna-se na única ferramenta passível de ser utilizada.

Para a realização desta dissertação, a linguagem de programação adotada foi Java, mas poderia ter sido elaborada noutro tipo de linguagem, como por exemplo Python, que também apresenta bibliotecas funcionais adequadas ao tipo de trabalho desenvolvido.

Para finalizar, o método escolhido para a leitura do RailML foi o JAXB, contudo poderia ter sido o Dom Parser, que tem a mesma finalidade. Optou-se pelo JAXB pelas vantagens que traria, isto é, pela simplicidade na escrita do código e pelo facto de permitir o processamento não sequencial da informação, ao contrário do Dom (que obriga a navegar seguindo uma estrutura em árvore) [24].

Até à data, a conceção destas infraestruturas foi sempre realizada manualmente, configurações estas, que serviram de comparação ao longo do desenvolvimento do projeto.

# Capítulo 3

# Levantamento dos Requisitos e Contextualização

Este capítulo é constituído por duas partes. Na primeira parte serão explicados os requisitos que foram definidos [3.1]. Na segunda, será demonstrado como foram aplicados os métodos abordados na secção anterior [3.2].

## 3.1 Requisitos

Esta subsecção está dividida em duas secções, a primeira irá referir os requisitos funcionais e a segunda os não funcionais, mais pormenorizadamente os de desempenho. Os outros requisitos não funcionais, isto é, os de *design* e os operacionais, não serão mencionados, uma vez que não foram relevantes para o projeto em questão.

### 3.1.1 Requisitos Funcionais

Os requisitos funcionais consistem em todas as funcionalidades de que o sistema necessitará para atingir os objetivos estabelecidos.

Tabela 3.1: Requisitos Funcionais

| **Número** | **Descrição** | **Prioridade** |
|---|---|---|
| 1 | Ler ficheiro RailML e dados de importação<br>• Importar ficheiro RailML de acordo com a versão 3.1, para o efeito, importação de elementos ferroviários, como os sinais, as agulhas e as secções;<br>• Extrair informação sobre os elementos de acordo com o modelo de rede (Estrutura de dados);<br>• Extrair informação do encravamento de acordo com o modelo de rede (Estrutura de dados); | P1 |
| 2 | Criar modelo de rede (Estrutura de Dados)<br>• Processar a informação extraída dos elementos presentes no RailML;<br>• Adicionar os dados processados de acordo com o modelo de rede (Estrutura de dados); | P1 |
| 3 | Criar modelo de especificação (modelo na qual se insere o *design* requerido) do SCADE Display usando Java API's (*Application Programming Interface*)<br>• Definir uma representação interna de um modelo de especificação;<br>• Guardar modelo de especificação num ficheiro sgfx de destino; | P1 |

### 3.1.2 Requisitos de desempenho

Os requisitos de desempenho referem-se à quantificação do desempenho de cada uma das funcionalidades, definindo a qualidade do sistema.

Tabela 3.2: Requisitos de desempenho

| Número | Descrição | Prioridade |
|---|---|---|
| N1 | A ferramenta deve ser escalável, na medida em que deverá ler o número de elementos ferroviários (sinais, agulhas e secções) que existirem no ficheiro RailML, sem limite; | P1 |
| N2 | O tempo de geração do ficheiro sgfx, a partir do programa desenvolvido, não deve exceder 1 minuto; | P2 |

## 3.2 Solução para o Problema

Nesta parte, serão abordados dois subtemas, o primeiro explicará, em detalhe, a funcionalidade das Java API's [3.2.1] e o segundo apresentará a forma como todo o projeto foi desenvolvido, de acordo com os passos tomados e a estrutura adotada [3.2.2]. De realçar que este processo que será apresentado se encontra enquadrado na ferramenta RAS, visto que as operações adotadas pertencem a esta ferramenta.

### 3.2.1 SCADE Java API

O SCADE Java API permite o acesso programático de leitura/escrita e a conectividade a ambientes de *design* baseados em modelos SCADE, a partir da plataforma de desenvolvimento Eclipse. Foi concebido para criar e manipular informação de dados de modelos como o SCADE Display, quer através de programação direta em Java, quer através de procedimentos genéricos de manipulação de modelos. Permite, ao utilizador, aceder a metamodelos representados por objetos Java que descrevem um modelo do modelo existente do SCADE Display, daí o nome metamodelo. [25].

### 3.2.2 Descrição da solução e Arquitetura de Alto Nível

Após a instalação e a configuração do Eclipse, foi possível iniciar a fase de desenvolvimento do projeto. Este passou por desenvolver um programa em Java, constituído por vários passos. O primeiro passo foi a definição da estrutura de dados, onde seriam inseridos os valores provenientes dos RailML. O segundo, foi a leitura dos elementos ferroviários necessários para o trabalho fornecidos pelo RailML, mais precisamente, dos elementos relativos ao encravamento e à sinalização. De notar que, este ficheiro, contém as posições absolutas (x, y) que permitiram orientar os elementos no SCADE Display, de acordo com o necessitado. Depois de criado o modelo de rede (estrutura de dados) composto pelos elementos ferroviários, foram utilizadas estas Java API's, mencionadas anteriormente, com o intuito de despoletar estes elementos no SCADE Display, através da "transformação"deste RailML na estrutura de dados já definida, salvaguardada num ficheiro

do tipo .sgfx, muito semelhante a XML. Isto é, foram carregadas, no programa, as bibliotecas associadas e foi criado este ficheiro final. Esta aplicação (SCADE Display) abriu e interpretou este ficheiro e apresentou a interface gráfica final. Posto isto, o programa será, posteriormente, testado pela empresa através do simulador presente no SCADE Display, para verificar e validar os resultados finais. Toda a explicação, detalhada, deste processo estará presente no capítulo [4].

A fim de exemplificar melhor o processo, a seguinte arquitetura demonstra, de uma forma geral, o mesmo:

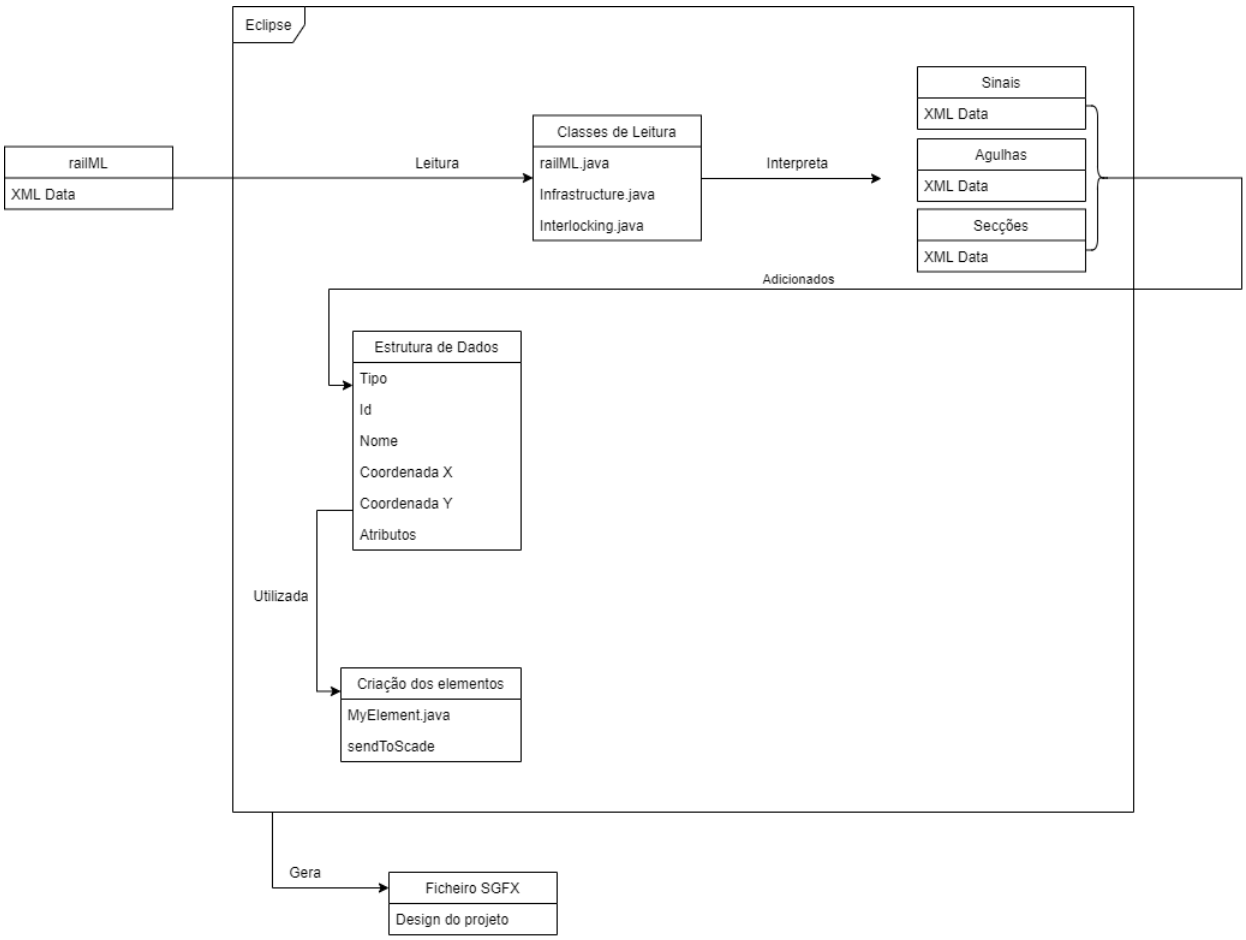


Figura 3.1: Diagrama (em detalhe) do projeto

# Capítulo 4

# Realização do Projeto

Neste capítulo serão abordados os passos e as respetivas descrições base para a realização do trabalho. O presente projeto é constítuido por três grandes temas, que serão desenvolvidos nos respetivos capítulos, nomeadamente, a definição da estrutura de dados [4.1], a leitura e passagem dos objetos do RailML para a estrutura de dados [4.2] e, por último, a criação do ficheiro sgfx, com os elementos ferroviários, através dos atributos guardados na estrutura de dados [4.3].

## 4.1 Estrutura de Dados

Para o projeto em questão, a definição da estrutura de dados resultou da necessidade de guardar a informação que seria necessária para a conclusão da aplicação. Foi uma opção lógica, uma vez que permitiu uma mais eficiente utilização da informação aí armazenada. Esta estrutura foi inserida numa lista de estruturas (visto ser constituida por mais que um elemento), em que cada elemento da mesma é composto por um elemento do RailML e seus atributos. Desta forma foi possível utilizar a informação armazenada, aquando da criação e definição dos atributos relativos a cada elemento. Esta estrutura de dados é constituida por vários parâmetros, parâmetros estes que foram alterados em função da sua aplicabilidade neste projeto. A cada parâmetro do elemento, com base na leitura do RailML, foi atribuido um valor. Com a atribuição de todos os valores a cada parâmetro foi possível assumir que o elemento estaria criado, sucedendo situação similar para todos.

Neste sentido, a estrutura de dados foi definida da seguinte maneira:

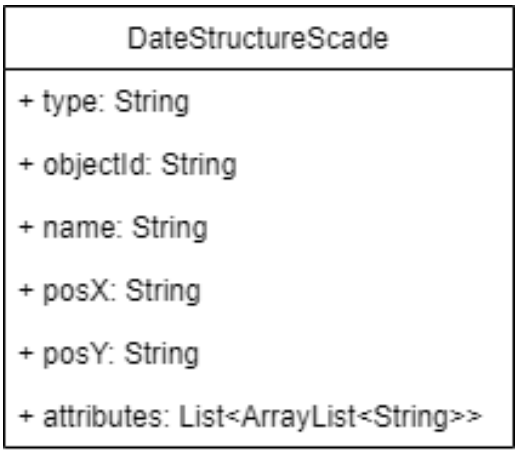


Figura 4.1: Classe Estrutura de Dados

Esta estrutura é composta por 6 parâmetros (em formato *string*, visto que toda a informação proveniente do RailML era desse formato), o tipo do elemento, o Id, o nome, a coordenada X, a coordenada Y e uma lista de listas de atributos (devido ao facto de poderem existir mais do que um atributo por elemento), nomeadamente nome e valor. Um ponto a destacar na estrutura são as coordenadas, as quais estão presentes no RailML em sentido inverso do que seria de esperar no SCADE Display, assim foi necessário criar uma função cujo objetivo fosse inverter estes valores antes de serem inseridos na estrutura. A função recebe como argumento a *string* coordenada, que foi transformada num *float* de forma a ser possivel multiplicá-la por (-1), posteriormente retornou o valor em formato *string* e foi inserida na estrutura.

Definiram-se os *setter's* de forma a adicionar os valores do RailML a cada parâmetro e os *getter's* para aceder à informação presente na estrutura. A forma como a informação ficou armazenada foi a seguinte (os valores presentes na estrutura já são os do RailML, que se encontra explicado na secção [4.2]):

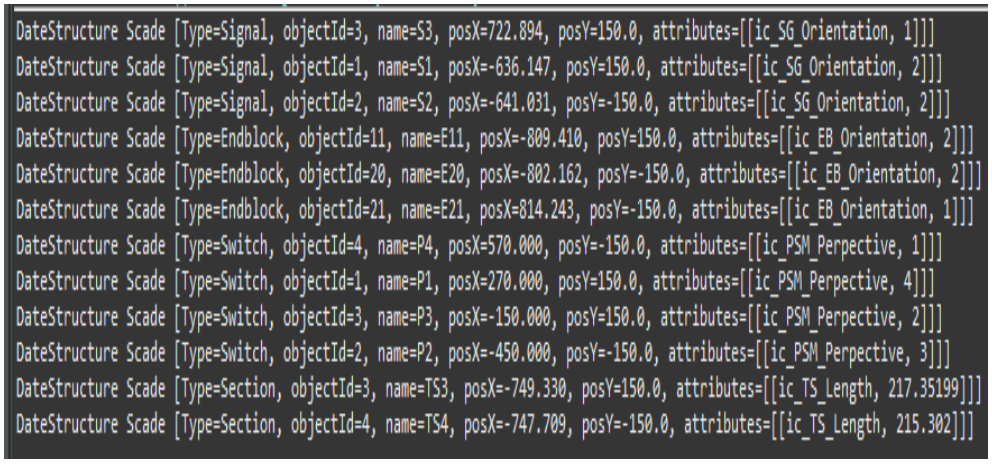


Figura 4.2: Estrutura de Dados

Analisando a estrutura em si, o tipo do sinal foi definido manualmente; o *objectId* refere-se ao *parsing* do name, isto é, se o nome do sinal for S1, o objectId era 1; o nome foi retirado diretamente do RailML; o posX e posY também; e os atributos, mesmo após a realização dos vários passos (funções), também foram de lá extraidos.

Assim que definida a estrutura de dados, foi possível prosseguir para a extração dos valores e atributos dos elementos ferroviários, presentes no RailML, a inserir nesta estrutura.

## 4.2 RailML

Depois de estabelecida a estrutura de dados, o passo seguinte focou-se no RailML, mais precisamente na sua leitura (de forma a poder inserir a informação na estrutura previamente abordada). Para a mesma, optou-se pelo método JAXB, um dos métodos relativos à leitura deste tipo de ficheiros, como abordado na secção [2.5].

- **Leitura da informação, relativa aos elementos, presente no RailML**:

Começou-se por criar o *package* (utilizado para agrupar as classes relacionadas) relativo ao ficheiro RailML. Seguidamente identificaram-se, neste tipo de ficheiros, as classes, os elementos e os atributos presentes nas mesmas. Após a identificação, seguiu-se a criação das classes, em ambiente Java, com os mesmos nomes presentes no RailML. Verificou-se que estas teriam que ser públicas, com o intuito de poder utilizar os objetos agora lidos em diferentes classes, permitindo assim aceder à informação pertinente presente nas mesmas. Cada classe é composta por um construtor, e por *getter's* de forma a utilizar o pretendido nas classes dedicadas ao SCADE Display. De notar que foram desenvolvidas várias, ao que foi necessário incluí-las dentro de outras, e definiu-se uma classe principal com duas sub-classes, "interlocking"e "infrastructure", onde estão presentes os elementos relativos ao projeto, tais como:

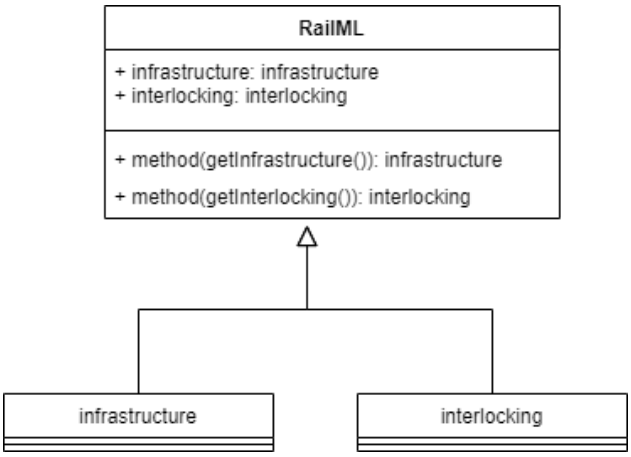


Figura 4.3: Classe RailML

Nesta fase foram utilizados o @XMLElement e o @XMLAttribute para cada caso (método JAXB). Isto é, para uma classe com vários elementos utilizou-se o @XMLElement, seguido do nome do elemento e para um atributo de um elemento o @XMLAttribute, seguido do nome do atributo. Para ambos os casos, o que se colocou dentro de cada um foi o nome (do atributo ou do elemento) que aparece no RailML, de forma a reconhecê-los. Apresenta-se, de seguida, um exemplo de uma agulha, mais precisamente do formato da mesma no RailML:

```
<switchIS id="swi38" continueCourse="right" branchCourse="left" type="ordinarySwitch">
    <name name="Sw01" language="en"/>
    <spotLocation id="swi38_sloc01" netElementRef="ne29" applicationDirection="reverse" intrinsicCoord="0.0000"/>
    <designator register="_Parkering_1_0" entry="SWITCH Sw01"/>
    <leftBranch netRelationRef="nr_ne28ne29_swi38" branchingSpeed="0" joiningSpeed="0" radius="-500"/>
    <rightBranch netRelationRef="nr_ne29ne37_swi38" branchingSpeed="0" joiningSpeed="0" radius="0"/>
</switchIS>
```

Figura 4.4: RailML - Agulha

Como nota adicional, e não focando apenas neste exemplo, toda a informação proveniente do RailML é do formato *string*.

Numa análise mais pormenorizada do código supra apresentado, é possível verificar a existência dos elementos *name*, *spotLocation*, *designator*, *leftBranch* e *rigthBranch*. Para o projeto em questão, todos os elementos presentes na classe *switchIS* foram necessários, bem como os atributos *name*, *applicationDirection* e *radius*. Por exemplo, foi através do *spotLocation* e do *leftBranch* e *rightBranch* que se calculou a orientação das agulhas (explicado no ponto seguinte). Com base nos mesmos definiu-se um código relativo à leitura destes componentes, cujo exemplo é:

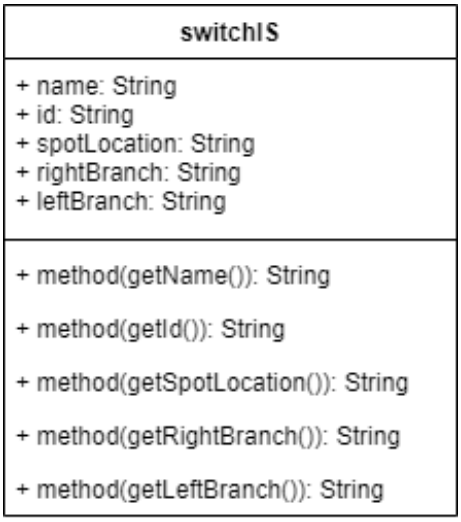


Figura 4.5: Classe *switchIS*

Para aceder ao atributo, utilizando-se por exemplo o radius presente no leftBranch, apresenta-se o código que infra se descrimina, em que o *getter* permite retornar a *string* com o valor, definindo-se o "XMLAttribute - name"tal e qual aparece no RailML:

Listing 4.1: Classe leftBranch

```
public class leftBranch {

                public String getRadius() {
                return radius;
        }
                public leftBranch() {
                super();
            }

            @XmlAttribute(name = "radius")
            private String radius;

}
```

Analisando a figura [4.5] a par da estrutura de dados, é possível compreender que o nome, o Id e o *spotLocation* presentes no RailML foram inseridos aos argumentos nome, *ObjectId*, *posX* e *posY* da estrutura (esta é a informação retirada diretamente do RailML). O tipo foi definido manualmente, isto é, sempre que a leitura fosse daquele elemento, o tipo seria esse elemento, por exemplo, quando foi retirada a informação relativa aos sinais a variável *type* ficou atribuída com "Sinal". A informação adicionada à lista de atributos será explicada de seguida.

- **Cálculo dos valores a adicionar a cada atributo (da lista de atributos) da estrutura de dados**:

Alguma informação relativa aos atributos não era direta, pelo que foram necessários vários procedimentos para calcular os valores de cada elemento. Para este caso específico, tanto para os sinais virtuais como para os sinais normais apenas se teve que alterar o atributo direção (1 - esquerda, 2 - direita, como exemplificado no anexo [A]), já para as agulhas houve a necessidade de se alterar para além da direção (8 opções de direção que se encontram detalhadas no anexo [B], das quais, 1 - *ReverseRightUp*, 2 - *ReverseLeftDown*, 3 - *ReverseLeftUp*, 4 - *ReverseRightDown*, 5 - *FacingNegativeDownReverseLeft*, 6 - *FacingNegativeUpReverseLeft*, 7 - *FacingPositiveUpRightReverse*, 8 - *FacingPositiveUpLeftReverse*), o tamanho de cada *branch* e, por fim, para as secções, apenas se teve que alterar o atributo tamanho.

1. Sinais:

   O primeiro passo definido para a resolução deste problema foi o de verificar em cada sinal que coincidências ou ligações existiam no RailML que resultasse nalgum valor, o encontrado foi que, a cada sinal corresponde um elemento de rede (ne1 a ne8), assim verificou-se quais os que estariam associados a cada um e consequentemente as suas coordenadas

(cada elemento de rede possui dois valores de coordenadas, quer x quer y). Seguidamente compararam-se as devidas coordenadas, isto é, se a primeira coordenada a aparecer fosse maior que a segunda então a direção seria da esquerda para a direita, caso contrário, seria da direita para a esquerda. Um exemplo das coordenadas de um elemento de rede é a seguinte:

```
<linearElementProjection refersToElement="ne1" id="vis01_lep01">
    <name name="netElement_ne1" language="en"/>
    <coordinate x="-150.000" y="-150.000"/>
    <coordinate x="-900.000" y="-150.000"/>
```

Figura 4.6: Elemento de rede

Posto isto, e como cada sinal contém um *applicationDirection* (*normal* ou *reverse*), caso fosse da direita para a esquerda e o *applicationDirection* fosse *normal* a direção seria direita (1), caso fosse da direita para a esquerda e o *applicationDirection* fosse *reverse* a direção seria esquerda (2), e assim sucessivamente. Com este procedimento foi possível calcular as direções tanto dos sinais normais como dos virtuais. A figura [4.7] representa o procedimento descrito.

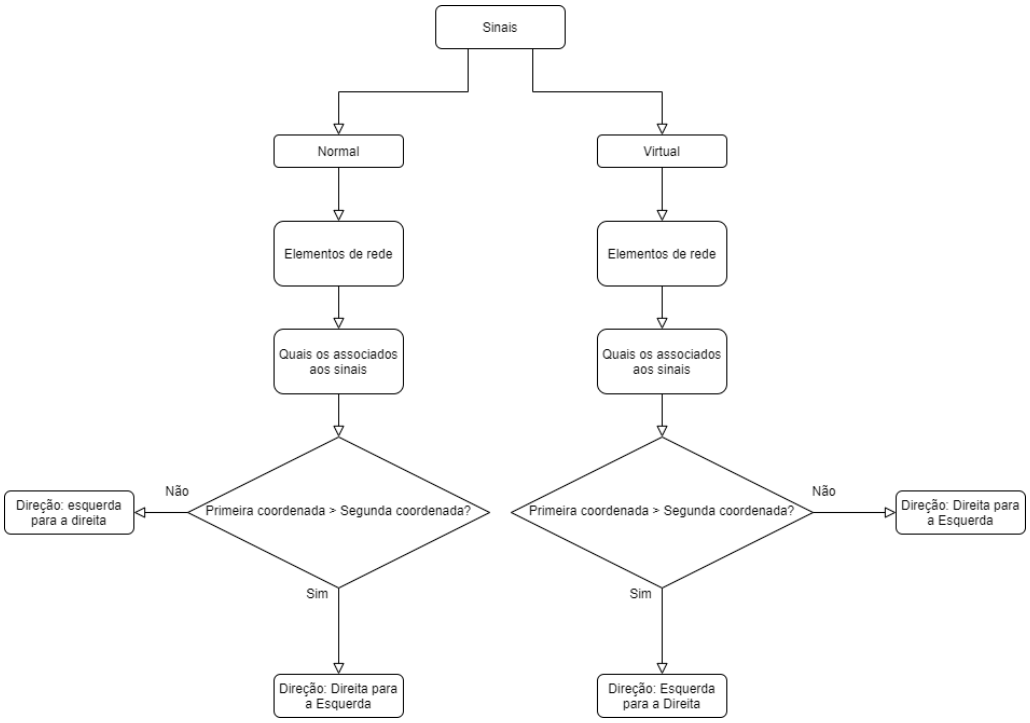


Figura 4.7: Procedimento para o cálculo das direções dos sinais

2. Agulhas:

Para as agulhas, o método inicial foi semelhante, o primeiro passo foi verificar quais os elementos de rede associados a cada agulha. O segundo, uma vez que as agulhas poderiam alterar consoante a sua ordenada, foi, através das coordenadas destes elementos de rede associados, verificar se as coordenadas Y destes elementos variavam, se sim então seria disposta na vertical, se não variasse seria na horizontal. De seguida, foi necessário verificar o *radius* de cada *branch* da agulha. Do RailML, é possível retirar o valor do *radius* tanto do *leftBranch* como do *rightBranch*. Se, por exemplo, o *radius* do *rigthBranch* não variar e o *radius* do *leftBranch* variar, então este braço esquerdo da agulha é o que desvia e o direito é o que se mantém normal (braço horizontal). A imagem seguinte retrata um exemplo do supra mencionado, em que se pode verificar que o *branch* da agulha que desvia, estando a olhar de frente para a mesma, é o esquerdo e o que se mantém normal o direito. Para este caso, a orientação seria, de entre as opções, *reverseLeftUp*.

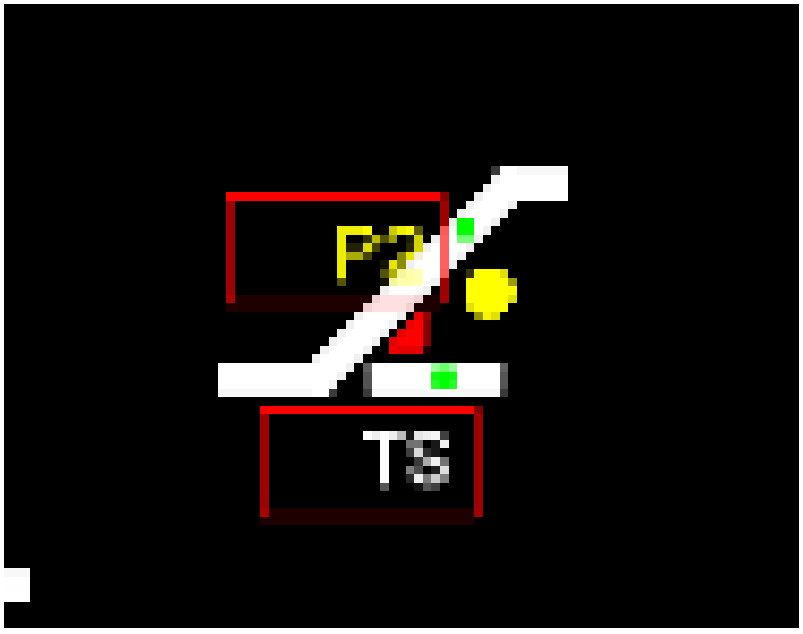

Figura 4.8: Agulha

Para além disto, o *applicationDirection* de cada agulha também tem impacto, isto é, se o *radius* fosse positivo e a *applicationDirection* fosse *normal* então seria desviado para o lado positivo do gráfico (para cima), se o *radius* fosse negativo os *applicationDirection* funcionariam contrariamente ao referido infra. Juntando então o vertical/horizontal com o resultado do *branch* que desvia foi possível descodificar a orientação das agulhas. Detalhadamente, o procedimento é o seguinte:

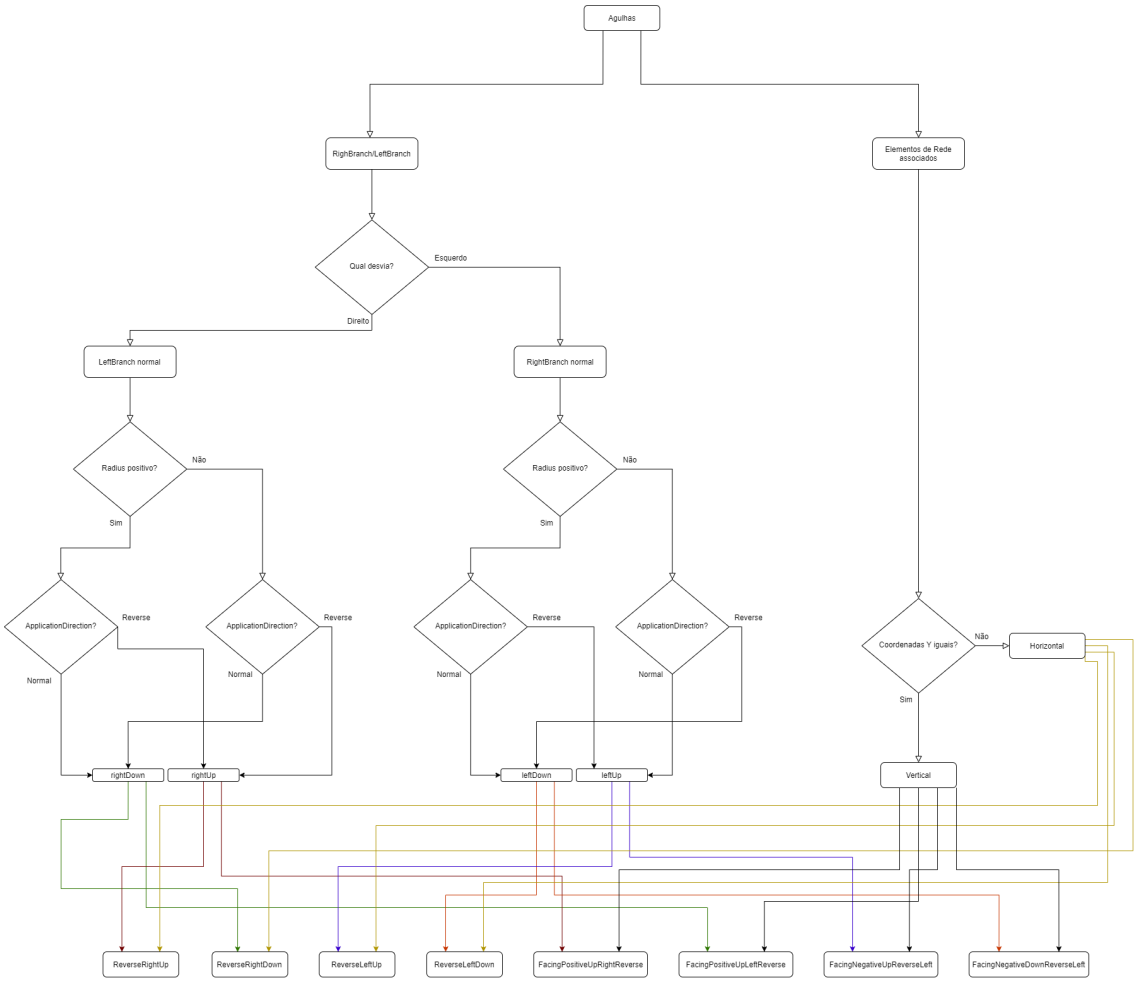


Figura 4.9: Cálculo da direção das agulhas

3. Secções:

Para as secções, o primeiro passo foi verificar quais as que possuiam tanto o *bufferstop* como o *traindetector* (requisitos base, presentes em todos os RailML, para uma secção ser considerada independente), porque se assim não fosse estas secções fariam parte das agulhas e não seria possível calcular o valor dos tamanhos das mesmas. Encontradas as secções com estes dois atributos de fim de secção, verificou-se quais seriam. Seguidamente, encontrou-se as coordenadas de cada um destes elementos, e para o cálculo dos tamanhos fez-se a diferença entre as coordenadas X de cada elemento presente nas secções. O método adotado encontra-se representado na seguinte figura:

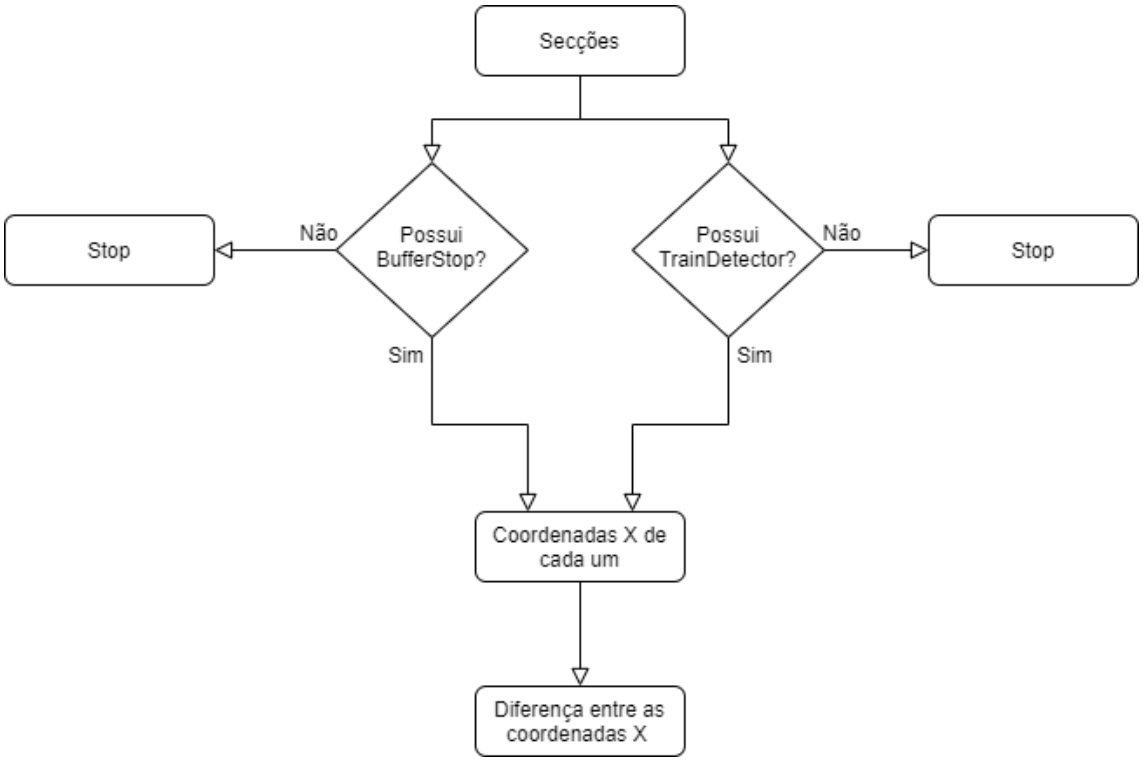


Figura 4.10: Cálculo do tamanho das secções

Posteriormente à leitura de todos os elementos e atributos necessários para o *design* do sistema ferroviário a inserir na estrutura de dados, passou-se à última fase da aplicação, isto é, à geração do ficheiro sgfx, interpretado pelo SCADE Display.

## 4.3 SCADE Display

O primeiro procedimento para a realização do *design* foi a compreensão do projeto fornecido pela empresa, denominado Hjallese, presente na figura [5.1]. Este seria o projeto a automatizar, uma vez que até à data teria sido realizado manualmente. Esta análise passou pelo interpretação de cada elemento ferroviário (sinal, agulha, secção), bem como a interpretação dos atributos a eles associados. Cada um é constituido pelos seus próprios atributos e valores.

Posto isto, deu-se seguimento à análise das API's [26], definindo quais a utilizar para a inserção dos elementos no ficheiro sgfx, a partir do RailML.

- **API's utilizadas**:

1. Definição do tamanho da janela do SCADE Display ao ficheiro sgfx:

   Na primeira abordagem foi imprescindível definir o *layer* do SCADE Display, isto é, o tamanho da página principal, visto existirem várias possibilidades e dependendo dos elementos era necessário que os tamanhos fossem definidos de acordo com os mesmos. O código associado a esta definição é o seguinte:

   Listing 4.2: Atribuição do tamanho da janela do SCADE Display

```
Specification lSpecification =
theScadeFactory.createSpecification();

        // Add model to the resource
        modelResource.getContents().add(lSpecification);

        // Set Specification Properties
        // ---------------------------

        //Definicao do tamanho da janela
        lSpecification.setWidth(1910);
        lSpecification.setHeight(980);
```

   De forma a esclarecer o leitor, o *modelResource* é o modelo de especificação de todo o projeto, onde foram guardadas as especificações e tudo o que era necessário para o *design*. A *lSpecification* é o tamanho da janela.

2. Realização da função que coloca os elementos, ainda que por criar, no ficheiro sgfx (ficheiro composto por características específicas do SCADE Display), como o *layer* (janela da aplicação onde se encontra o *design*), o *reference container* (explicado posteriormente), entre outros). De notar que apenas será mostrada a utilização da função no *main*:

   Esta função *sendToScade* dispõe os elementos no SCADE Display, já com os parâmetros de cada elemento incluidos. A cada elemento a criar foi necessário chamar a função de forma a que fosse interpretado pela aplicação, como por exemplo:

   Listing 4.3: Aplicação da função *sendToScade*

```
sig.sendToScade(lLayer);
```

3. Criação do *reference container*, onde foram inseridos os elementos:

   Como cada elemento na aplicação é considerado um *reference object* (objeto que pode ser apontado para qualquer modelo ou utilizado noutro projeto e que fica guardado num ficheiro OGFX) foi preciso utilizar uma API que criasse um *reference container* (onde se inserem os *reference objects*) por cada elemento, que se encontra na função "send to scade".

4. Definição dos atributos do *layer*:

   Posto isto, foi criado o *layer* com todas as definições *default*, isto é, o ponto de origem da aplicação, os Id's gerados para cada elemento (visto que cada um necessita de um Id diferente), o nome do *layer*, entre outros, como representado no excerto seguinte:

   Listing 4.4: Definição do *layer* da aplicação

```
// Append a layer to the Specification
        Layer lLayer = theScadeFactory.createLayer();
        lSpecification.getLayers().add(lLayer);

        // Set Layer Properties
        // --------------------
        {
        lLayer.setHorizontal_ratio(1.0f);
        lLayer.setVertical_ratio(1.0f);
        lLayer.setName("symbology_layer");

        lLayer.setOid(createNewOid(theScadeFactory));
        // Visible Prop
        BooleanProp lVisibility =
           theScadeFactory.createBooleanProp();
```

```
lVisibility.setInit(true);
lLayer.setVisible(lVisibility);

// Origin
CoordinatePoint lOrigin =
   theScadeFactory.createCoordinatePoint();
RealProp lX = theScadeFactory.createRealProp();
lX.setInit(0.0f);
RealProp lY = theScadeFactory.createRealProp();
lY.setInit(0.0f);
lOrigin.setX(lX);
lOrigin.setY(lY);
lLayer.setOrigin(lOrigin);

// Id
IntegerProp lId = theScadeFactory.createIntegerProp();
lId.setInit(1);
lLayer.setId(lId);

}
```

- **Criação de classes relativas aos elementos a inserir**:

1. Classe de criação dos elementos:

   Definido o *layer*, foi necessário entender como seriam criados os elementos a inserir. Como cada elemento tem a sua biblioteca (fornecida pela empresa), foi preciso atribuir a cada um a mesma. A classe respetiva é a seguinte:

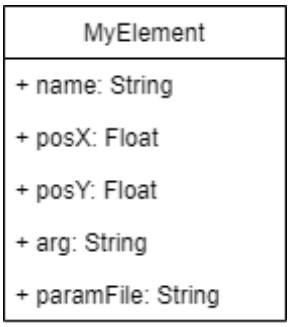


Figura 4.11: Classe Elemento

Como parâmetros iniciais, foram definidos o nome, a coordenada X, a coordenada Y, o *arg* (biblioteca com o formato de cada elemento) e o *paramFile* (biblioteca com os atributos de cada elemento).

Depois de criado o elemento, ao chamar a função *sendToScade*, como referido anteriormente, o elemento surgiu no ficheiro sgfx, tal como:

Listing 4.5: Instanciação da classe MyElement a par da chamada da função sendToScade

```
MyElement sig = new MyElement(list.get(i).getName(),
                Float.parseFloat(list.get(i).getPosX()),
                                 Float.parseFloat(list.get(i).getPosY()),
                                 files.getModelFile(),
                                 files.getParamFile());


sig.sendToScade(lLayer);
```

2. Atribuição das bibliotecas aos elementos:

Para a atribuição das bibliotecas ao *arg* e ao *paramFile*, foi criada uma classe *singleton* (classe que apenas necessita de ser instanciada uma única vez durante toda a aplicação para que possa ser utilizada a sua informação, onde poderão ser guardadas, por exemplo, variáveis e os seus valores, e que permite também manter os mesmos dados acessíveis de uma forma transversal a todas as classes onde é utilizado [27]), que continha os ficheiros de cada elemento. A mesma era composta pelos *setter's* (a pensar no futuro, caso outra pessoa pretendesse utilizar a aplicação pudesse alterar o valor das duas variáveis para o diretório onde se encontram as suas bibliotecas), por uma função que atribuíu a uma *string* as biblioteca por tipo de elemento e pelos *getter's*, que lhes permitiram poder aceder às mesmas, como se pode ver seguidamente:

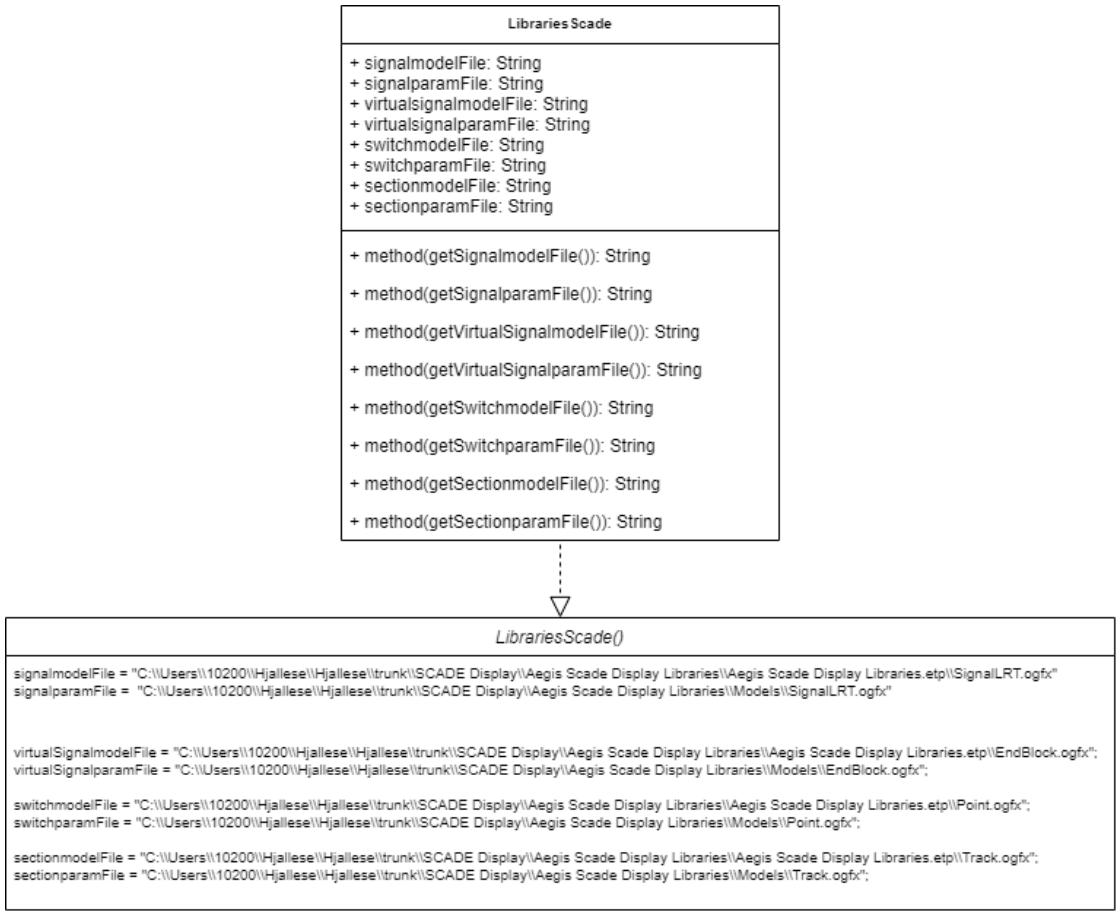


Figura 4.12: Classe Singleton das Bibliotecas do SCADE Display

Exemplificando a utilização dos *getter's* na classe *main*, instanciou-se a classe e acedeu-se às bibliotecas de cada elemento, conforme se detalha:

Listing 4.6: Instanciação da classe singleton e utilização dos *getter's*

```
LibrariesScade files = LibrariesScade.getInstance();

                    (...)

 (...) files.getSignalmodelFile(),
    files.getSignalparamFile());
 (...) files.getVirtualSignalmodelFile(),
         files.getVirtualSignalparamFile());

                    (...)
```

Como nota adicional, o *modelFile* é o modelo do elemento criado na aplicação manual e o *paramFile* é o atributo de cada um, como por exemplo, o nome, a direção, o tamanho, entre outros.

3. Atribuição dos valores dos atributos calculados anteriormente aos elementos criados:

Depois de colocados os elementos, presentes no RailML, na aplicação e de surgirem com as posições, nomes e tamanhos *default*, foi imprescindível corrigir os mesmos, mais precisamente alterar os valores dos parâmetros associados a cada um. Esta alteração implicou a realização de numerosos passos intermédios. O primeiro dos quais foi verificar qual o tipo de cada parâmetro, de forma a que, através de uma função que o receba, esta verificasse se existia, qual o seu tipo e o alterasse para o valor desejado. Os valores usados para alterar cada atributo podem ser inteiros, boleanos, conjunto de caracteres ou *float's*. Para este efeito, permitindo assim alterar os parâmetros depois de receber o valor a corrigir, foi necessário "descascar"a biblioteca onde se encontravam estes atributos. Desta forma, foi possível entender qual o seu tipo, se, nesse caso específico, seria *arrayExpr*, *literalExpr* ou *unaryOpExpr* (três e únicos tipos dos parâmetros de cada elemento para todos os projetos do SCADE Display). Dado que os valores a alterar teriam que ser do formato *string*, (visto que os objetos que vêm do RailML são *strings*) realizaram-se as conversões de cada valor, quer de *string* para inteiros, quer de *string* para caracteres, quer de *string* para *float's*. A função descrita é a seguinte:

Listing 4.7: Classe para alterar parâmetros

```
        public void ChangeParam(String param, String value) {

                SdyFactory theScadeFactory = SdyPackage
                                    .eINSTANCE.getSdyFactory();

                List<Expr> list = inits;

for (int i = 0; i < inits.size(); i++) {
        if (list.get(i) instanceof ArrayExprImpl) {
                ArrayExpr aex = (ArrayExpr) inits.get(i);
                InputParamPropImpl nameText =
                (InputParamPropImpl) aex.eContainer();
                String name = nameText.getName();
                List<Expr> lista = aex.getElems();
                if (name.equals(param)) {
                                lista.clear();
                        for (int j = 0; j < value.length();
                           j++) {

                                LiteralExpr literal;

                                literal theScadeFactory
                                            .createLiteralExpr();


                        char character = value.charAt(j);
                        int ascii = (int) character;
                        literal.setValue(Integer.
                                     toString(ascii));

                        lista.add(literal);

                        }
                        LiteralExpr literal1 =
                            theScadeFactory.createLiteralExpr();
                        literal1.setValue("0");
                        lista.add(literal1);
```

```
                }

            }

                if (list.get(i) instanceof LiteralExprImpl) {
                        LiteralExpr aex = (LiteralExpr)
                           inits.get(i);
                        InputParamPropImpl aux =
                            (InputParamPropImpl)
                               aex.eContainer();
                        String aux1 = aux.getName();

                        if (aux1.equals(param)) {
                                aex.setValue(String.valueOf(value));
                        }
                }
            if (list.get(i) instanceof UnaryOpExprImpl) {
                        UnaryOpExpr aex1 = (UnaryOpExpr)
                           list.get(i);
                        InputParamPropImpl aux2 =
                        (InputParamPropImpl) aex1.eContainer();
                        String aux3 = aux2.getName();

                        LiteralExpr aex = (LiteralExpr)
                           aex1.getArg();
                        if (aux3.equals(param)) {
                        aex.setValue(String.valueOf(value));
                        }
                }
            }

    }
```

A forma como esta função foi utilizada no main é apresentada da forma seguinte:

Listing 4.8: Aplicabilidade da classe alterar parâmetros

```
sig.ChangeParam("ic_SG_NameText",
   list.get(i).getName());
sig.ChangeParam("p_Object_ID", list.get(i).getObjectId());
sig.ChangeParam("ic_SG_Orientation",
```

```
                    list.get(i).getAttributes().get(k).get(1));
                (...)
```

De notar que o *list* se refere à lista da estrutura de dados com os dados provenientes do RailML e os *get's* ao valor em si.

Para alterar os parâmetros com os valores do *array* de atributos, o procedimento realizado ocorreu da forma que seguidamente se demonstra (dependendo do número de *array's* de atributos que existirem):

Listing 4.9: *Getter's* para aceder aos atributos calculados

```
.getAttributes().get(k).get(1));
.getAttributes().get(k+1).get(1));
.getAttributes().get(k+2).get(1));

        (...)
```

Assim que os elementos foram criados (no anexo [C] encontra-se um exemplo de um sinal criado, presente no ficheiro sgfx gerado) e os parâmetros dos mesmos alterados para os valores vindos do RailML, prosseguiu-se para a criação do xml com os parâmetros não alterados.

- **Criação e leitura do XML para os parâmetros *default***:

1. Criação do XML com os parâmetros (nomes e valores) que não foram alterados com os valores do RailML:

Visto que alguns elementos presentes na aplicação possuíam valores de alguns parâmetros incorretos (em relação à aplicação configurada pela empresa), foi necessário agrupar todos os parâmetros não utilizados num XML, em suma, agrupar todos os nomes e valores *default* associados a cada um. Desta forma, caso se pretendesse alterar algum deles, apenas seria necessário ler o XML e fazer *set* ao valor a alterar, ou, em alternativa, alterar de forma manual, o que permitiria utilizar, na aplicação a desenvolver, o valor *default* do parâmetro que fosse pretendido, para qualquer elemento.

O primeiro procedimento foi a criação do XML. O método selecionado para a criação do ficheiro foi o DOM parser [28] [29], pelas razões enunciadas no subcapítulo [2.6]. No âmbito deste método, primeiro efetuou-se a instanciação da *Factory* (considerada um "padrão de criação", na medida em que retorna algo que pode ser usado sem que o programador se

preocupe com a implementação, isto é, se, por acaso, o mesmo decidir substituir o código presente numa das bibliotecas usadas por outro de outras bibliotecas, o programa funcionará na mesma [30]), e a criação do documentBuilder:

Listing 4.10: Criação do *DocumentBuilder*

```
DocumentBuilderFactory dbFactory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
```

De seguida, criou-se o documento respetivo do document builder acima apresentado:

```
Document doc = dBuilder.newDocument();
```

Seguidamente definiram-se os elementos a ser utilizados no XML e o método *appendChild* para os atribuir ao mesmo, neste caso, teremos na raiz o conjunto de parâmetros e dentro do elemento raiz teremos os vários parâmetros, com os seus atributos nome, valor e tipo, como se pode verificar no código seguinte (como os vários parâmetros são de três tipos, foi necessário utilizar para cada um *if* distinto):

Listing 4.11: Atribuição do elemento raiz e dos atributos ao XML

```
// root element
        root = doc.createElement("configurations");
        doc.appendChild(root);

        (...)
if (inits.get(i) instanceof LiteralExprImpl) {
        LiteralExpr aex = (LiteralExpr) inits.get(i);
        InputParamPropImpl aux = (InputParamPropImpl)
           aex.eContainer();
        String aux1 = aux.getName();

        // parameter element
        Element parameter = doc.createElement("parameter");
        parameter.setAttribute("name", aux1);
        parameter.setAttribute("value", aex.getValue());
        parameter.setAttribute("type", type);
        root.appendChild(parameter);
```

Após a atribuição dos elementos, definiram-se, então, a diretoria e o nome do ficheiro, através do método *transformer*:

Listing 4.12: Criação do ficheiro XML

```
DOMSource source = new DOMSource(doc);
TransformerFactory transformerFactory =
    TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
StreamResult result = new StreamResult(new
    File("C:\\parametros.xml"));
transformer.transform(source, result);
```

O ficheiro ficaria então com o formato representado no anexo [D]. De realçar que, neste anexo, apenas se encontram alguns exemplos dos parâmetros de cada elemento.

2. Criação das *flag's*:

Dado que a cada *run* do programa o ficheiro XML reinicializava com os valores *default* de cada elemento, foi fulcral criar uma variável (que funcionou como *flag*) para que, se algum parâmetro do XML fosse alterado manualmente, ao desativá-la, o mesmo não fosse reinicializado durante todo o uso da aplicação e o valor do parâmetro se mantivesse inalterado. Ativar a *flag* permitiu então regressar aos valores iniciais.

Outro aspeto a realçar foi o facto de se pretender criar este XML por tipo, e como este XML é criado no *main* (onde são criados os elementos) foi necessário definir mais quatro variáveis, fazendo com que, assim que entrasse no ciclo de um tipo, apenas adicionasse os parâmetros na primeira passagem e não os voltasse a adicionar consoante o número de elementos que existiam. O excerto de código que retrata este exemplo é o seguinte:

Listing 4.13: *Flag's*

```
xmlParameters xmlFile = new xmlParameters();

//A flag seria true ou false consoante uso da aplicacao
final boolean flag = true;

boolean firstTimeSignals = true;

                (...)
if(flag) {
        if(firstTimeSignals) {
        test.addToXml(list.get(i).getType());
```

```
                firstTimeSignals = false;
                }
```

3. Leitura do XML criado:

Definido o XML, procedeu-se à sua leitura, de forma a utilizar os valores dos parâmetros na aplicação. O método utilizado para a leitura deste XML foi semelhante ao método utilizado para a leitura do RailML (JAXB), visto que, como referido anteriormente, o ficheiro RailML possui um formato deste tipo. O código que retrata este método de leitura espelha-se da seguinte forma:

Listing 4.14: Classe leitura do XML

```
public readXmlParamaters() {
                super();

        // Read the xml File
        File xmlFile = new File("C:\\parametros.xml");

        JAXBContext jaxbContext;

                try {

        jaxbContext =
           JAXBContext.newInstance(configurations.class);

        Unmarshaller jaxbUnmarshaller =
           jaxbContext.createUnmarshaller();

        configurations myxml = (configurations)
                                    jaxbUnmarshaller.unmarshal(xmlFile);
```

A classe *configurations* é a classe que representa o elemento raiz do XML, a qual permite a sua leitura.

Para a utilização dos valores, foi necessário, dado o nome do parâmetro, verificar a sua existência, e, em caso afirmativo, atribuir o seu valor a uma variável para posterior utilização na aplicação. O seguinte excerto de código simula o seu uso, após o procedimento de verificação na classe definida para esse efeito:

Listing 4.15: Utilização dos valores dos atributos do xmll

```
ResultXmlValues value = new ResultXmlValues();

 String defaultValue = xmlValue("ic_SG_VisibleDirection",
    xmlParameters.getNames().get(i),
    xmlParameters.getValues().get(i) ,
        xmlParameters.getTypes().get(i));

                if(defaultValue != null) {
                parameterXmlValue = defaultValue;

                }

                (...)


sig.ChangeParam("ic_SG_VisibleDirection",
   value.getParameterXmlValue());
```

De referir que, para a alteração do valor do parâmetro, foi utilizado o método *ChangeParam* definido e explicado anteriormente.

# Capítulo 5

# Utilização e Validação da Aplicação

Neste capítulo serão abordados dois temas, dos quais, a utilização da aplicação desenvolvida no decorrer da elaboração deste projeto [5.1] e a fase de testes [5.2].

## 5.1 Caso de Uso

Como focado anteriormente, a funcionalidade da aplicação desenvolvida era a de transferir o RailML recebido, ou seja o RailML com o *design* dos elementos ferroviários, para o SCADE Display. Dado que toda a configuração do *design* era realizada de forma manual, o desenvolvimento desta ferramenta iria traduzir-se numa melhoria a nível de tempo e produtividade por parte da empresa, aumentando assim a sua eficiência.

A aplicação aqui desenvolvida permite a leitura de qualquer RailML, mais precisamente, quaisquer elementos ferroviários presentes no mesmo (sinais, agulhas e secções) um dos requisitos mais abordados nas reuniões e dos mais importantes definidos pela empresa. Foi utilizado, como aplicação de apoio, um ficheiro fornecido pela mesma, denominado Hjallese. A aplicação Hjallese caracteriza-se pelo seguinte *design*:

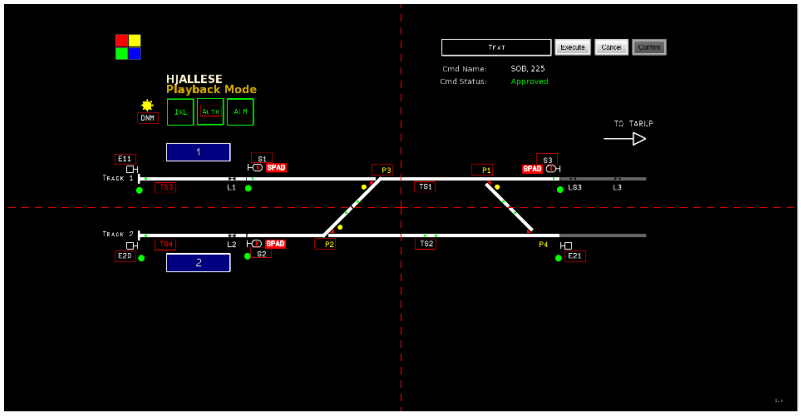


Figura 5.1: Hjallese - SCADE Display

Como é visível na imagem anteriormente apresentada, o *design* é composto por seis sinais (E11, E20, E21, S1, S2 e S3), quatro agulhas (P1, P2, P3 e P4) e quatro secções (TS1, TS2, TS3 e TS4). O objetivo da aplicação foi então tentar replicar este exemplo, pelo que o trabalho se focou na inserção destes elementos. Por este motivo, as plataformas, os nomes, a barra de texto, entre outros, foram ignorados, por não existir informação relativa aos mesmos no RailML. Este modelo, que foi usado como exemplo para o desenvolvimento do programa, foi realizado manualmente, sendo que todos os elementos e os seus atributos foram desenvolvidos pela área técnica da empresa. Esta aplicação gerou então as bibliotecas já referidas, bibliotecas essas que foram usadas para o projeto em questão.

O trabalho desenvolvido permitiu alcançar um resultado bastante positivo, dado que alcançou o objetivo fundamental nele definido. Não obstante este facto, e por decisão da *EFACEC* ficaram apenas as agulhas, isto é, os seus tamanhos, como trabalho a desenvolver numa fase posterior à conclusão deste projeto. Esta decisão prendeu-se com o facto deste ponto já se encontrar em fase de conceção interna pelas respetivas áreas. O resultado obtido está retratado na seguinte imagem:

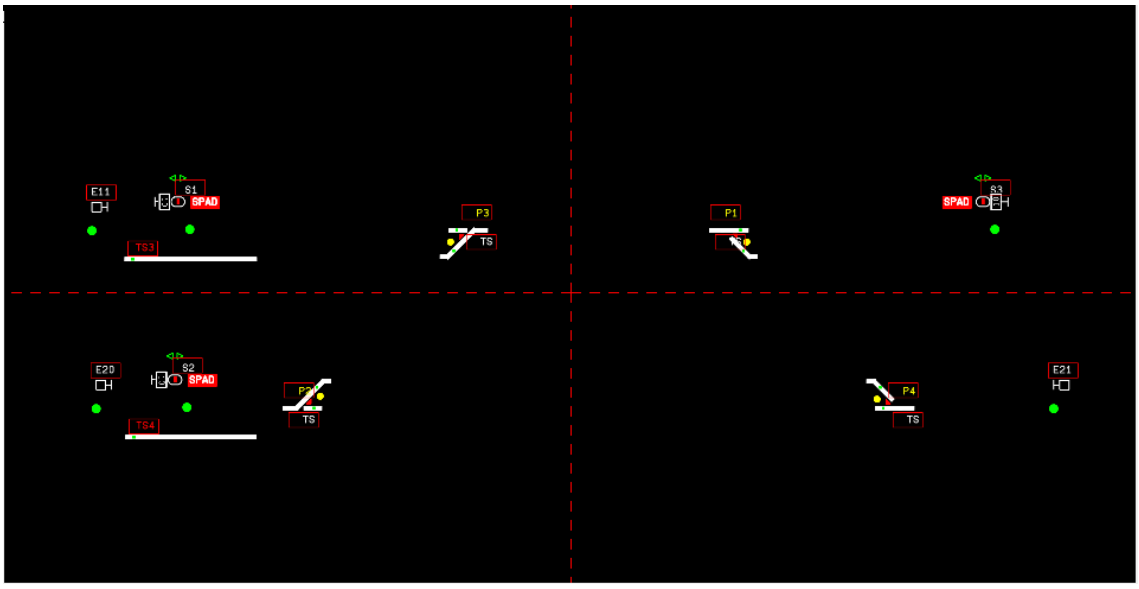


Figura 5.2: Aplicação Desenvolvida - SCADE Display

Comparando os dois *design's*, foi possível verificar que os sinais, as secções e as agulhas de encravamento possuem a mesma disposição na aplicação, o mesmo nome, o mesmo tamanho e a mesma direção.

Para além da alteração que terá que ser realizada nas agulhas (como já mencionado), os elementos terão que sofrer algumas alterações nas bibliotecas, principalmente nas coordenadas, visto que a origem das coordenadas da aplicação no projeto desenvolvido pela empresa poderá ter sido diferente do realizado aquando do estágio, na qual foi considerado (0,0). É possível verificar na figura

[5.2] que os sinais, as secções e as agulhas não se encontram perfeitamente alinhados. Sendo este um dos principais requisitos, necessitam de estar dessa forma para dar seguimento e continuidade à infraestrutura (formar a infraestrutura). O resultado esperado, ainda que as alterações às agulhas tivessem sido realizadas manualmente (após geração do *design*), foi o seguinte:

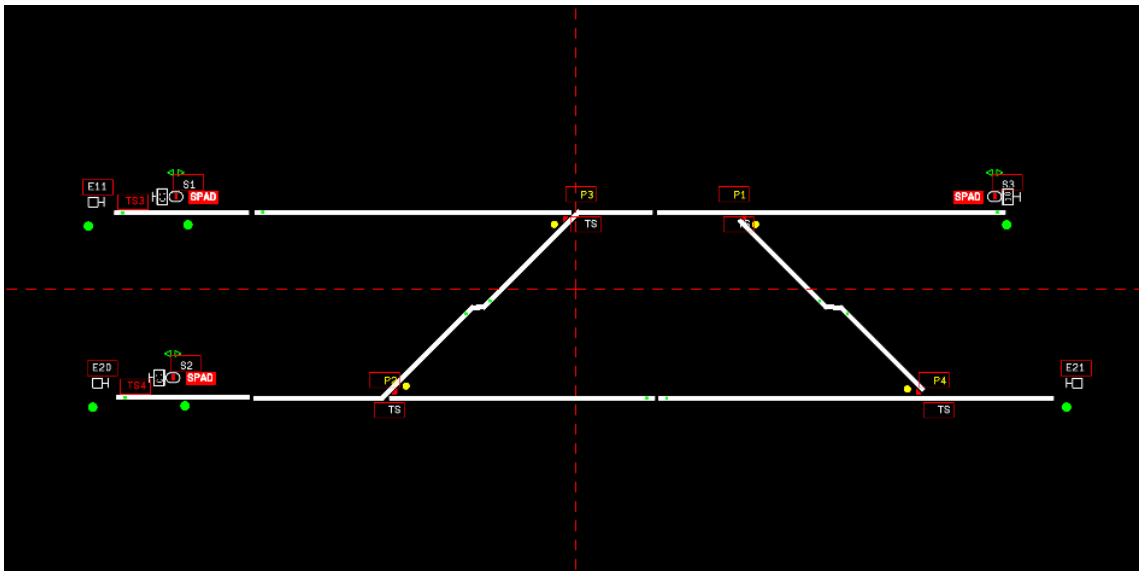


Figura 5.3: *Design* final - SCADE Display

Em suma, a aplicação poderá ser utilizada pela empresa num futuro próximo ou aplicada a um *design* que precisem de simular.

## 5.2 Testes

A última parte do projeto focou-se na fase de testes. Fase esta, que foi constituida por três tipologias, a primeira foi o teste com outros RailML's, a segunda foi a verificação do tempo de execução do programa, quer com o primeiro RailML (Hjallese), quer com o segundo (Parkering) e a terceira, ainda que numa fase posterior, seria composta pela utilização de testes incorporados do *Java Unit Test*.

A fase de testes com outros RailML's permitiu verificar se a aplicação funcionava para todos os casos (escalabilidade), isto é, se dado um RailML, o programa desenvolvido gerasse o ficheiro sgfx relativo a esse RailML. Para esta verificação, foi necessário substituir a seguinte parcela de código (com o presente RailML) por outro que foi disponibilizado pela empresa.

Listing 5.1: Código a alterar para qualquer RailML

```
File xmlFile = new File("src/Main/Hjallese_1_0.railml");
```

No âmbito desta verificação, o *design* a simular (Parkering) foi o seguinte:

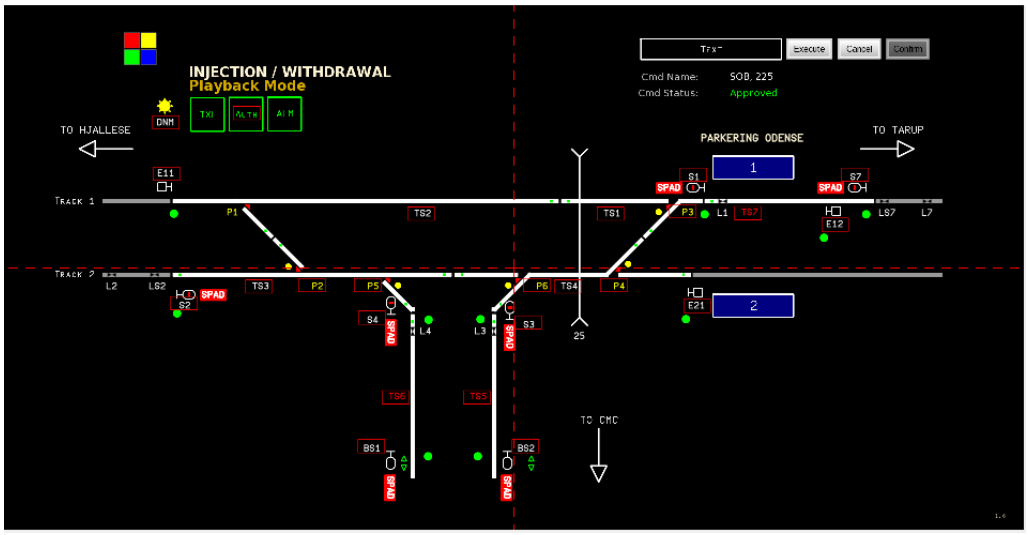


Figura 5.4: Parkering RailML - SCADE Display

É possível verificar que este requisito foi cumprido, gerando todos os sinais, agulhas e secções e posicionando-os na aplicação, sendo a imagem infra apresentada prova disso mesmo:

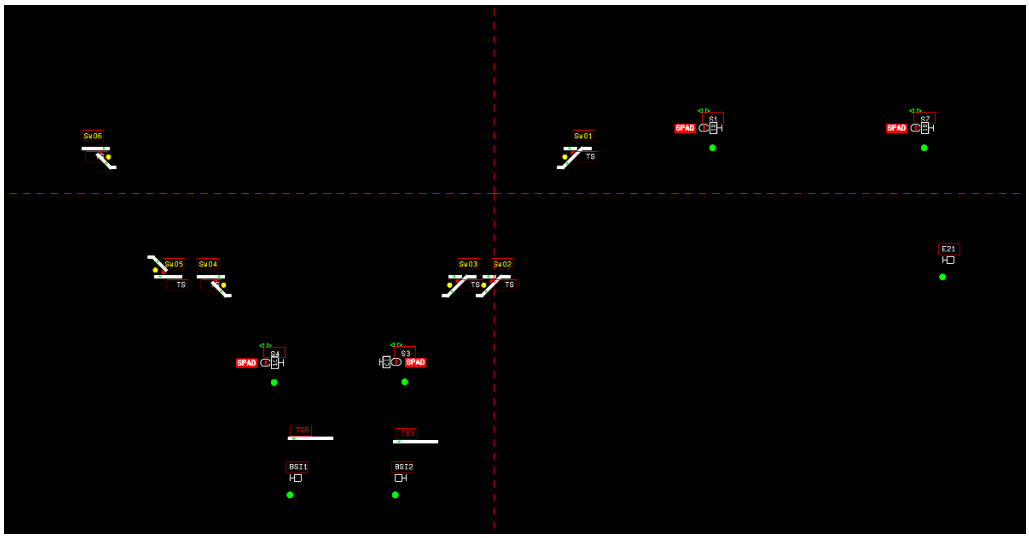

Figura 5.5: Teste do Parkering RailML - SCADE Display

A segunda fase foi a verificação do tempo de execução e geração do ficheiro sgfx, através dos RailML fornecidos pela empresa. Foi realizada através de uma função Java já existente, que se encontra no excerto seguinte:

Listing 5.2: Retirado de [8], Teste ao tempo de execução do programa (geração do ficheiro sgfx)

```
long startTime = System.nanoTime();

// code

long endTime = System.nanoTime();
System.out.println("Demorou: "+(endTime - startTime)/1000000 + "
    ms");
```

De acordo com o requisito definido na subsecção [3.1.2], o programa não poderia exceder 1 minuto a gerar o ficheiro sgfx. De acordo com as imagens infra apresentadas, é possível verificar que este requisito foi cumprido, sendo que o tempo de execução do programa desenvolvido é extremamente inferior ao previamente estabelecido (1.292 segundos para o Hjallese e 1.436 segundos para o Parkering):

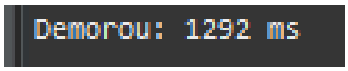


Figura 5.6: Tempo de geração do ficheiro através do primeiro RailML (Hjallese)

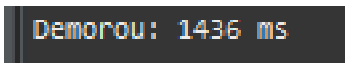


Figura 5.7: Tempo de geração do ficheiro através do segundo RailML (Parkering)

De realçar que o tempo relativo ao segundo RailML (Parkering) é mais elevado, devido ao facto de ser composto por mais elementos ferroviários.

A terceira fase passaria pelo *Java Unit Test*, sendo que este só seria passível de realização caso existisse tempo para o mesmo. Este *software* é constituido por mecanismos que fornecem suporte para a criação e execução de testes em Java, sendo que um exemplo de um dos testes a realizar seriam os testes unitários (testes dedicados à verificação dos métodos implementados, isto é, o teste à lógica destes métodos [31]). Permitiria, de uma forma geral, executar testes às funcionalidades das classes e dos seus métodos [32].

# Capítulo 6

# Conclusões e Trabalho Futuro

Este capítulo é composto por três subcapítulos. No subcapítulo [6.1], será realizada uma análise crítica ao projeto desenvolvido. Na secção [6.2], serão exploradas propostas de trabalho para realização futura. Por último, no subcapítulo [6.3] serão descritas as conclusões relativas ao trabalho desenvolvido.

## 6.1 Análise Crítica

Como apresentado ao longo do documento, o projeto foi realizado em duas aplicações distintas, no Eclipse e no SCADE Display, contudo é possível afirmar que o mesmo não representa todas as funcionalidades destas aplicações.

Quanto ao Eclipse, ferramenta já utilizada em várias unidades curriculares ao longo do curso, não existiram grandes dúvidas, as funcionalidades do mesmo eram sobejamente conhecidas, como por exemplo, a forma de criar os *package's*, a forma de criar as classes dentro dos *package's*, como organizar as pastas, entre outros. O facto do trabalho ter sido realizado neste programa, pelos motivos anunciados anteriormente, facilitou o desenvolvimento do código.

Quanto à linguagem Java, esta foi uma linguagem secundária ao longo do curso. É possível referir que, o facto do presente trabalho ter sido realizado na sua totalidade em Java, permitiu o desenvolvimento de novos conhecimentos da mesma e compreendidas novas regras. Esta linguagem orientada a objetos foi a mais indicada para o projeto em questão, na medida em que poderia fazer interface com a ferramenta RAS já existente.

A novidade focou-se na utilização do SCADE Display, aplicação desconhecida até então. Considerou-se, por este motivo, uma mais valia, a nível de conhecimento, e quem sabe, a nível futuro, possibilitando a sua aplicabilidade em novos projetos.

Esta é uma aplicação escalável, adaptada a qualquer *design*, o que significará que a empresa a poderá utilizar, como pretendido, evitando assim qualquer configuração manual. Considera-se,

por este motivo, um ponto positivo, visto que, este tipo de configuração poderá ser demorada e sujeita a erros e falhas. Para o efeito, apenas é necessário alterar o ficheiro RailML lido pelo pretendido.

Para finalizar e comparando o metódo apresentado com métodos já existentes, encontrou-se um similar, o denominado XSLT, que transforma um ficheiro XML num ficheiro de texto XSLT (ficheiro do tipo XML) [33]. Este método apresenta parecenças face ao método apresentado na dissertação, contudo, pela fiabilidade e pelas características típicas dos ficheiros sgfx, não seria possível a sua utilização.

## 6.2 Trabalho Futuro

Como foi abordado no capítulo [5], a parte central do *design*, mais precisamente o tamanho das agulhas, não foi implementado, uma vez que a empresa já possui informação relativa a este processo. Será integrado posteriormente pela mesma.

Outro aspeto a relevar será a possível necessidade de alteração dos métodos utilizados, mais precisamente as classes e as funções em Java. Toda a elaboração do trabalho foi realizada de forma a não repetir código e a ser o mais sucinto e percetível, contudo, poderão ter que ser alterados.

Um dos pontos a relevar seria a certificação da ferramenta T2 (ferramenta enquadrada na norma EN 50128), abordada no início do presente documento. Esta validação será algo a ser realizado pela empresa.

Para finalizar e em virtude de não terem sido realizados os testes do *Java Unit Test*, os mesmos serão efetuados posteriormente pela *EFACEC*. Pela relevância do mesmo, será aconselhável que esta proceda à sua realização para verificação e confirmação do código, visto que poderá ser detetado alguma anomalia que seja necessário corrigir.

## 6.3 Conclusões

De uma forma geral, os resultados obtidos são bastante satisfatórios, na medida em que os requisitos estabelecidos aquando do início da dissertação foram cumpridos, como demonstrado ao longo do presente documento.

Não obstante, para o projeto em questão, não terem sido utilizadas todas as funcionalidades que o SCADE Display possui, permitiu um conhecimento aprofundado do programa, programa este que era, até então, totalmente desconhecido. Outro ponto a realçar foi a utilização das Java API's,

que permitiram entender, de forma mais otimizada, as mesmas. Para além disto, foi apresentado pela empresa, um novo formato de um ficheiro xml, o RailML, que foi utilizado no decurso da dissertação.

Ao longo do desenvolvimento da aplicação, foram-se denotando alguns imprevistos e erros no código, contudo todos os prazos das tarefas semanais assignadas pela empresa foram cumpridas. Este trabalho permitiu não só conhecer a aplicação, como também aprofundar os parcos conhecimentos na linguagem Java. Considera-se, por esta razão, um trabalho bastante enriquecedor e com aplicabilidade em futuros projetos.

Um aspeto bastante positivo a realçar foi o estilo de desenvolvimento do código efetuado. Neste caso, todo o programa permite a reutilização do mesmo e evita ter que ser alterado no futuro, aquando da utilização das funcionalidades inerentes.

Por outro lado, todo o trabalho realizado inicialmente, de leitura e de compreensão do problema permitiu desenvolver um pouco mais os conhecimento sobre os sistemas ferroviários, tanto na vertente de *software*, como na vertente física, visto que o programa efetuado visa representar todos os equipamentos físicos presentes no mundo real.

Aprofundando o abordado no estágio realizado na *EFACEC*, este foi dividido em duas fases, a primeira de compreensão e análise de sistemas ferroviários, com foco no conhecimento dos vários elementos que o compôem (compreensão do formato dos ficheiros RailML) e a segunda de execução do conhecimento adquirido.

Concluindo, considera-se, então, que o programa detalhado nesta dissertação cumpre com os requisitos e expetativas propostas pela empresa.

# Anexo A

# Direções dos Sinais

Neste anexo, é possível observar as diferentes direções dos sinais, tanto dos normais como dos virtuais.

## A.1 Sinal normal - *Left*

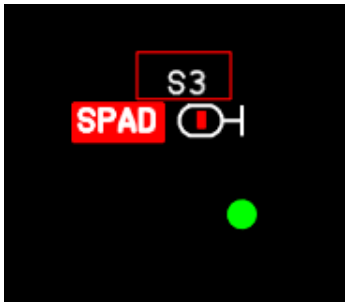


Figura A.1: Sinal normal - *Left*

## A.2 Sinal normal - *Right*

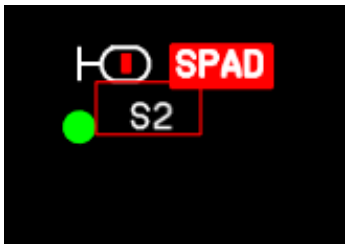


Figura A.2: Sinal normal - *Rigth*

## A.3 Sinal virtual - *Left*

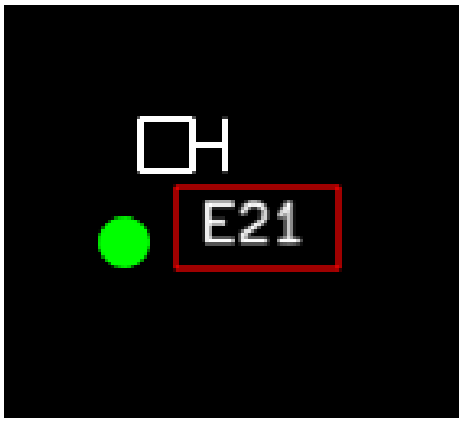


Figura A.3: Sinal virtual - *Left*

## A.4 Sinal virtual - *Rigth*

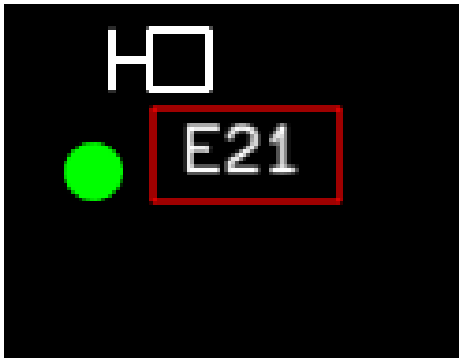


Figura A.4: Sinal virtual - *Rigth*

# Anexo B

# Direções das Agulhas

Neste anexo, é possível verificar as diferentes direções das agulhas.

## B.1 1 - *ReverseRightUp*

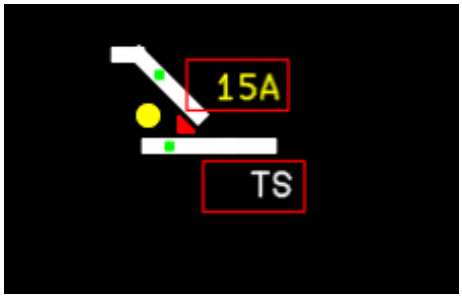


Figura B.1: Agulha - *ReverseRightUp*

## B.2 2 - *ReverseLeftDown*

:

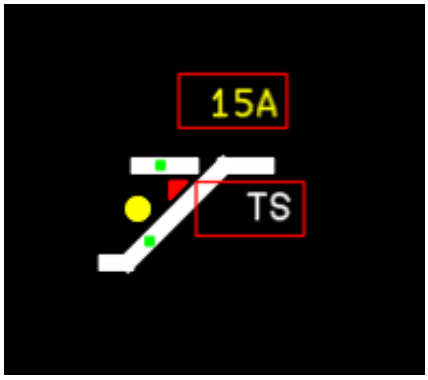


Figura B.2: Agulha - *ReverseLeftDown*

## B.3 3 - *ReverseLeftUp*

:

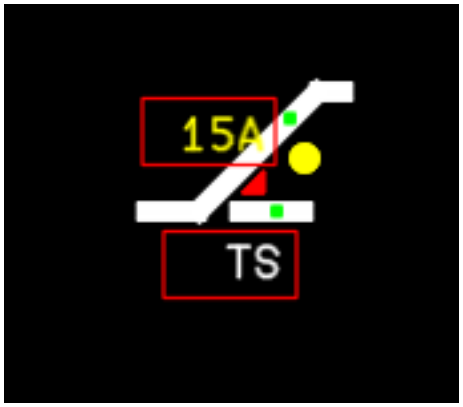


Figura B.3: Agulha - *ReverseLeftUp*

## B.4 4 - *ReverseRightDown*

:

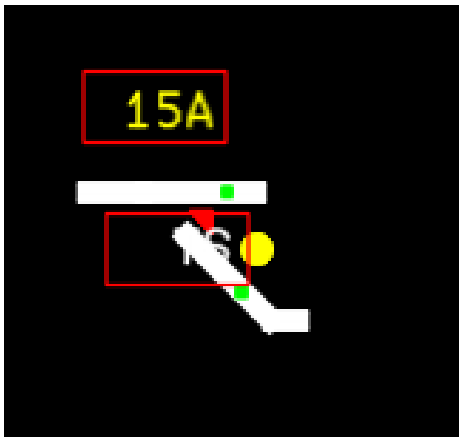


Figura B.4: Agulha - *ReverseRightDown*

## B.5 5 - *FacingNegativeDownReverseLeft*

:

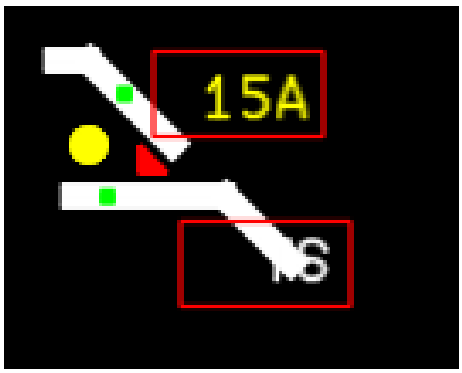


Figura B.5: Agulha - *FacingNegativeDownReverseLeft*

## B.6 6 - *FacingNegativeUpReverseLeft*

:

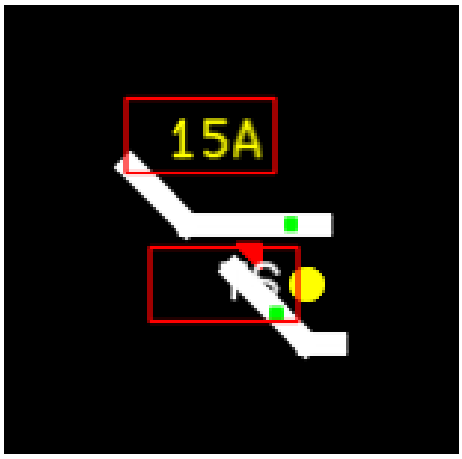


Figura B.6: Agulha - *FacingNegativeUpReverseLeft*

## B.7 7 - *FacingPositiveUpRightReverse*

:

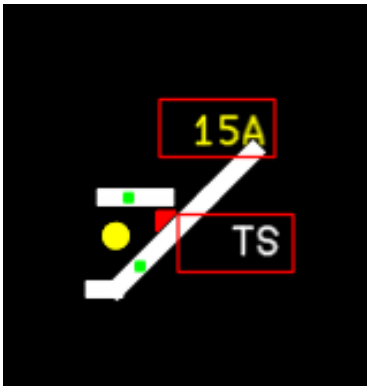


Figura B.7: Agulha - *FacingPositiveUpRightReverse*

## B.8 8 - *FacingPositiveUpLeftReverse*

:

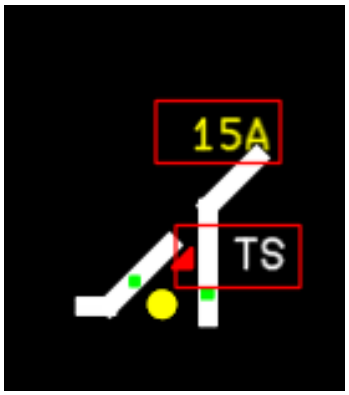


Figura B.8: Agulha - *FacingPositiveUpLeftReverse*

# Anexo C

# Representação de um sinal no ficheiro sgfx

:

Neste anexo, é possível observar um excerto de código presente no ficheiro sgfx gerado. Este caso apresenta um sinal normal, o "S1".

```
<referenceContainer oid="d8e76811-d0cc-4b9a-8c40-a06a043d82e7">
    <properties modular="false" name="S1">
        <file name="SignalLRT.ogfx" path="..\..\..\Hjallese\Hjallese\trunk\SCADE Display\Aegis Scade Display Libraries\Aegis Scade Display Libraries.etp"/>
        <visible>
            <init>true</init>
        </visible>
        <origin>
            <x>
                <init>-636.147</init>
            </x>
            <y>
                <init>150.0</init>
            </y>
        </origin>
        <rotate>
            <angle>
                <init>0.0</init>
            </angle>
        </rotate>
        <orientation clockwise="false"/>
        <scale>
            <x>
                <init>1.0</init>
            </x>
            <y>
                <init>1.0</init>
            </y>
        </scale>
        <inputParameters>
            <parameter name="i_LastTimeCycle" oid="a8ddc936-b310-4d5d-b019-f3b6e3fa6d14" representation="None" type="int32">
                <init>50</init>
            </parameter>
            <parameter name="i_BlinkOn" oid="72fb9def-267d-4e3e-bb4e-076071f7aa71" representation="None" type="bool">
                <init>false</init>
            </parameter>
            <parameter name="i_SG_EndReleasingType" oid="42d9c379-4c5e-46ee-88aa-f3fc63ff5aec" representation="None" type="int32">
                <init>1</init>
            </parameter>
            <parameter name="i_SG_BlockingStatus" oid="07ec3b9b-2c7e-4e5a-9322-c8c24e27a31a" representation="None" type="int32">
                <init>1</init>
            </parameter>
            <parameter name="i_SG_LockingStatus" oid="8ff0276c-1b8c-437e-b67c-3ea0e60b6bb2" representation="None" type="int32">
                <init>1</init>
            </parameter>
            <parameter name="i_SG_AspectStatus" oid="36de3b30-1cbc-48e2-93bd-209690dca646" representation="None" type="int32">
                <init>8</init>
            </parameter>
            <parameter name="i_SG_DirectionStatus" oid="c4a89136-1ebd-43bc-be80-29bc0fb2573d" representation="None" type="int32">
                <init>5</init>
            </parameter>
```

Figura C.1: Sinal "S1"

# Anexo D

# Ficheiro XML com os parâmetros dos elementos

Neste anexo, é possível observar um excerto do ficheiro XML criado, constituído por alguns parâmetros (nome e valor) de cada tipo de elemento ferroviário (sinal normal, sinal virtual, agulha e secção).

Listing D.1: Ficheiro XML com os parâmetros dos elementos

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <configurations>
    <parameter name="i_LastTimeCycle" type="Signal" value="50"/>
    <parameter name="i_BlinkOn" type="Signal" value="false"/>
    <parameter name="i_SG_EndReleasingType" type="Signal" value="1"/>
    <parameter name="i_SG_BlockingStatus" type="Signal" value="1"/>
    <parameter name="i_SG_LockingStatus" type="Signal" value="1"/>
    <parameter name="i_SG_AspectStatus" type="Signal" value="8"/>
    <parameter name="i_SG_DirectionStatus" type="Signal" value="5"/>
    ...

    <parameter name="i_LastTimeCycle" type="Endblock" value="0"/>
    <parameter name="i_Playback" type="Endblock" value="false"/>
    <parameter name="i_BlinkOn" type="Endblock" value="false"/>
    <parameter name="p_Object_ID" type="Endblock" value="0"/>
    <parameter name="p_Object_IXLAreaID" type="Endblock" value="0"/>
    ...

    <parameter name="i_LastTimeCycle" type="Switch" value="50"/>
    <parameter name="i_BlinkOn" type="Switch" value="false"/>
    <parameter name="i_PSM_BlockingStatus" type="Switch" value="1"/>
    <parameter name="i_PSM_LockingStatus" type="Switch" value="1"/>
    ...

```

```
25      <parameter name="i_LastTimeCycle" type="Section" value="0"/>
26      <parameter name="i_TS_LockingStatus" type="Section" value="0"/>
27      <parameter name="i_TS_BlockingStatus" type="Section" value="0"/>
28      <parameter name="i_Object_SafetyCode" type="Section" value="0"/>
29      <parameter name="i_BlinkOn" type="Section" value="false"/>
30      <parameter name="i_Playback" type="Section" value="false"/>
31      <parameter name="p_Object_ID" type="Section" value="0"/>
32      <parameter name="p_Object_IXLAreaID" type="Section" value="0"/>
33      <parameter name="ic_TS_Exists" type="Section" value="true"/>
34      <parameter name="ic_TS_Name" type="Section"/>
35      ...
36
37  </configurations>
```

# Referências


[1] Funkwerk to install electronic interlocking at Hamburg depot | International Railway Journal. URL: https://www.railjournal.com/signalling/funkwerk-to-install-electronic-interlocking-at-hamburg-depot/.

[2] D. Fenner. Railway signalling, 2011. doi:10.1049/ic.2011.0183.

[3] Beginners Guide to SIL Levels. URL: https://blog.acdist.com/beginners-guide-to-sil-levels.

[4] ANSYS SCADE Model-Based Development Solutions for RAIL TRANSPORTATION. Critical Systems Software Development Solutions - PDF Free Download. URL: https://docplayer.net/6887279-Ansys-scade-model-based-development-solutions-for-rail-transportati
html.

[5] Gérard Berry. The Evolution of the Synchronous Model. Relatório técnico, 2008. URL: www.esterel-technologies.com.

[6] A. Nash, D. Huerlimann, J. Schuette, e V. P. Krauss. RailML - A standard data interface for railroad applications. *Advances in Transport*, 15:233–240, 2004. doi:10.2495/978-1-84564-500-7/01.

[7] Course Java Collections - Lecture: JAXB. URL: https://codegym.cc/quests/lectures/questcollections.level03.lecture07.

[8] eclipse - How to find time taken to run a Java program? - Stack Overflow. URL: https://stackoverflow.com/questions/6646467/how-to-find-time-taken-to-run-a-java-program.

[9] Quem Somos | Efacec. Disponível em https://www.efacec.pt/quem-somos/. URL: https://www.efacec.pt/quem-somos/.

[10] Transportes - Ferrovias, Rodovias e Sistemas Industriais | Efacec. URL: https://www.efacec.pt/transportes-rodoviarios-ferroviarios/.

[11] Léxico | Infraestruturas de Portugal. URL: https://www.infraestruturasdeportugal.pt/pt-pt/negocios-e-servicos/lexico/b.

[12] Glossário Ferroviário. URL: https://www.trainlogistic.com/pt/Comboios/Gabinete/fich_glossario.htm.

[13] EN 50128 Railway applications → Testing and Anaylsis. Disponível em https://www.verifysoft.com/en_EN_50128_Software_for_Railway_Control_and_Protection_Systems.html.

[14] Comunicação da Comissão no âmbito da execução da Directiva 96/48/CE do Conselho. Relatório técnico. URL: http://www.etsi.org.

[15] Functional Safety in the Rail Industry| TÜV SÜD. URL: https://www.tuvsud.com/en/industries/infrastructure-and-rail/rail/functional-safety-for-rail.

[16] European Standard. En 50128. 2001.

[17] CQ Press. Rail Transportation. *Washington Information Directory 2009–2010*, páginas 684–685, 2014. doi:10.4135/9781604265927.n107.

[18] Ansys e Inc. ANSYS SCADE Display Technical Data Sheet. Relatório técnico.

[19] SCADE Display: Embedded Display Graphics Design Environment | Ansys. URL: https://www.ansys.com/products/embedded-software/ansys-scade-display.

[20] ANSYS SCADE Lifecycle - ferramenta de gerenciamento do ciclo de vida do aplicativo. URL: https://www.mistralsolutions.com/defence_products/scade-lifecycle/.

[21] Introduction - railML.org (EN). URL: https://www.railml.org/en/introduction.html.

[22] Course Java Collections - Lecture: JAXB. URL: https://codegym.cc/quests/lectures/questcollections.level03.lecture07.

[23] XML – Trabalhando com JAXB – MBallem | Programando com Java. URL: https://www.mballem.com/post/xml-trabalhando-com-jaxb/.

[24] O que é JAXB em Java | mySoftKey. URL: https://www.mysoftkey.com/java/jee/what-is-jaxb/.

[25] Products Family. SCADE Java/Tcl API Guide.

[26] Display User Manual. (August), 2009.

[27] Singleton Design Pattern - O que é, onde usar e quais as vantagens? URL: https://www.opus-software.com.br/singleton-design-pattern/.

[28] How to Read XML File in Java - Javatpoint. URL: https://www.javatpoint.com/how-to-read-xml-file-in-java.

[29] How-To: Read an XML file using DOM parser in JAVA? | Technofriends. URL: https://technofriends.wordpress.com/2007/12/26/how-to-read-an-xml-file-using-dom-parser-in-java/.

[30] java - JDK DOM Parser: Why Factories? - Stack Overflow. URL: https://stackoverflow.com/questions/39211196/jdk-dom-parser-why-factories.

[31] Implementando testes unitários em Java O que são teste unitários? Relatório técnico.

[32] Unit Testing with JUnit 5 - Tutorial. URL: https://www.vogella.com/tutorials/JUnit/article.html.

[33] XML e XSLT. URL: http://www.macoratti.net/vbn_xlst.htm.