Universidade do Minho
Escola de Engenharia

Relatório Nº2

https://github.com/joaoabreu5/DSS-Grupo40

Desenvolvimento de Sistemas de Software

João Abreu	João Faria	Ricardo Sousa	Rui Silva	Tiago Ferreira
				
A91755	A97652	A96141	A97133	A97141

Grupo 40

2022/23

Índice

Índice de Figuras

Introdução

No âmbito da Unidade Curricular de Desenvolvimento de Sistemas de Software foi-nos proposta a conceção de um simulador de campeonatos de automobilismo. A génese por trás do sistema pedido é similar ao já conhecido jogo *F1 Manager*.

Na primeira fase, desenvolveu-se o Modelo de Domínio e o Diagrama de Use Cases, responsáveis por representar as interações existentes entre as diversas entidades, de modo a haver uma perceção de como o sistema funcionará.

O presente relatório serve de suporte à realização e entrega da segunda fase deste trabalho, e apresenta uma descrição pormenorizada de como abordamos a arquitetura conceptual bem como os modelos comportamentais do nosso sistema. Ao longo deste relatório serão mostrados os diagramas elaborados, juntamente com uma breve explicação sobre a construção dos mesmos. Por fim, apresentamos algumas conclusões e uma análise crítica ao trabalho realizado nesta etapa.

Objetivos

Nesta segunda fase do trabalho possuímos 2 grandes objetivos a elaborar: a arquitetura conceptual do sistema e os modelos comportamentais.

A arquitetura conceptual do sistema, que deverá ser capaz de suportar os requisitos identificados previamente, passa pela conceção de um Diagrama de Componentes, que descreva os componentes do nosso sistema e as dependências entre eles, permitindo identificar, em cada nível, o que é necessário para construir o sistema, um Diagrama de Classes e ainda um Diagrama de *Packages*, que agrupa classes.

Quanto aos modelos comportamentais, competiu-nos desenvolver Diagramas de Sequência, responsáveis por representar as interações entre objetos.

Arquitetura Conceptual do Sistema

Tendo em conta que as definições arquiteturais são as mais importantes, e que, no futuro, alterá-las terá um custo e repercussões em cadeia muito significativas, tomamos muito cuidado e precaução na elaboração dos nossos Diagramas de Componentes, de Classes e de *Packages*.

Diagrama de Componentes

Olhando ao trabalho desenvolvido até ao momento e tendo em conta o nosso Modelo de Domínio e Diagramas de Use Cases, bem como as suas especificações, identificamos as suas responsabilidades. Seguidamente, adotamos o método aprendido nas aulas:

- Dividimos os fluxos em sequências de transações;
- Identificamos responsabilidades da lógica de negócio;
- Identificamos métodos;
- Agrupamos os métodos em subsistemas.

A partir daqui, compreendemos o relacionamento entre os diferentes componentes do sistema, tornando-se bastante útil a fim de conseguirmos agrupar os métodos que melhor se adequam a cada um dos subsistemas. Visto que cada subsistema implementa uma interface com os métodos que lhe estão associados, são implementadas *Facades*, que escondem a complexidade de um sistema maior e fornecem uma interface mais simples ao sistema. Isto possibilita a interação com outros subsistemas através da API de cada *Facade* sendo que, para tal, não é necessário conhecer a organização interna do subsistema em questão. Deste modo, garantimos encapsulamento na nossa arquitetura.

Por fim, tendo sempre em conta também que um componente deve ser um pedaço de software reutilizável, bem encapsulado e “facilmente” substituível, ou seja, uma parte modular do sistema e considerando ainda a nossa arquitetura, elaboramos o seguinte Diagrama de Componentes:

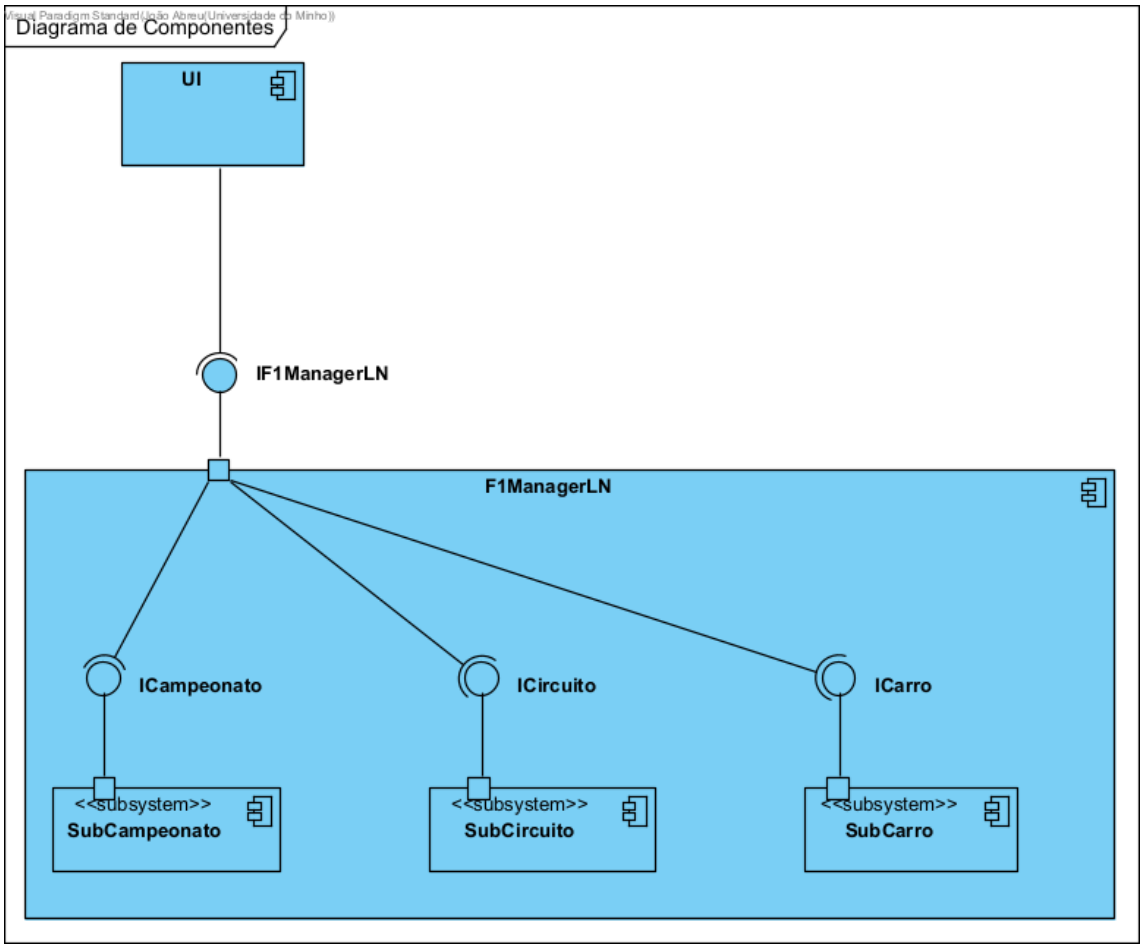

Figura 1 - Diagrama de Componentes

Por conseguinte, foram decididos os seguintes subsistemas:

- SubCampeonato – Trata de toda a gestão dos Campeonatos e Corridas do mesmo;
- SubCircuito – Trata de toda a gestão dos Circuitos do sistema;
- SubCarro – Trata de toda a gestão dos Carros e consequentes categorias.

Diagrama de Classes

De seguida procedemos à organização da nossa arquitetura por classes, que consideramos fundamental, e que mais uma vez facilita a reutilização de código e, também, a sua manutenção. Deste modo, o sistema é pensado de forma a que a alteração de uma classe no futuro tenha o menor impacto possível no resto do sistema.

Sistema principal da lógica de negócio

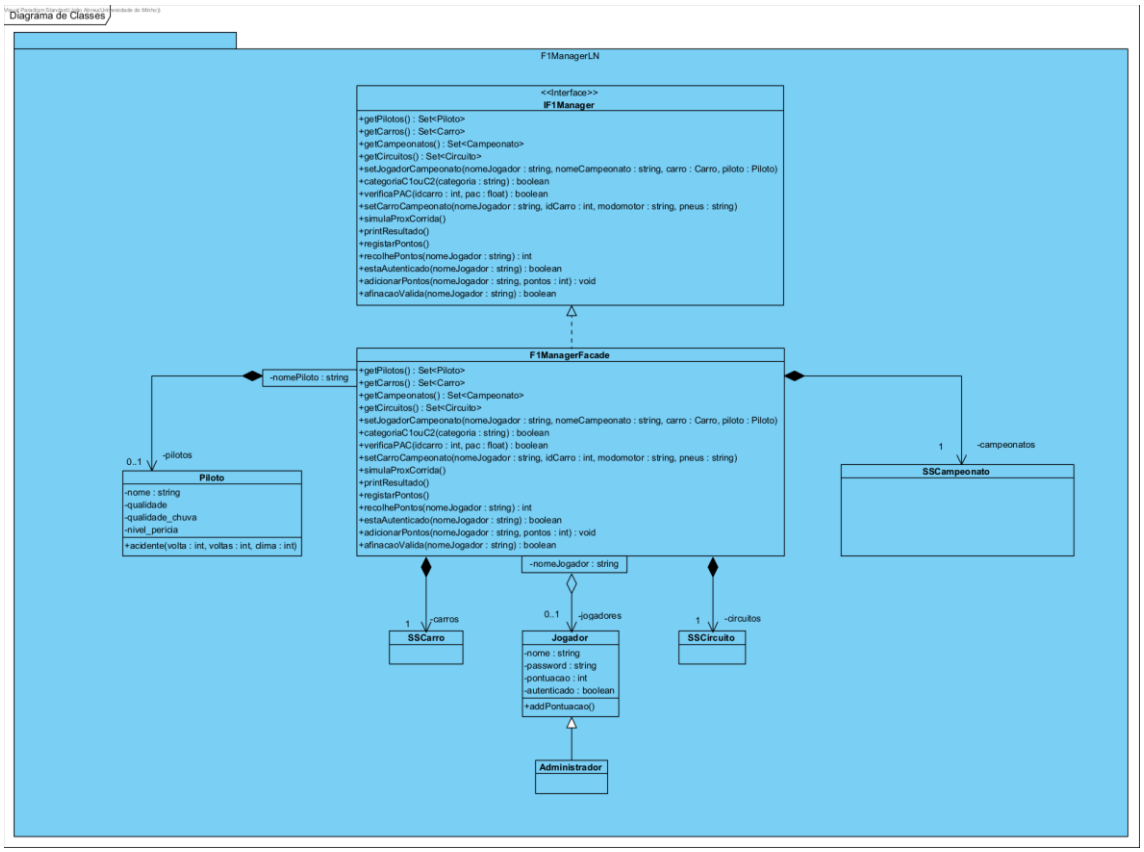

Figura 2 - Diagrama de Classes

Neste diagrama podemos observar algumas das classes fundamentais para o funcionamento do nosso sistema. Temos, então, uma interface responsável pelas funções fundamentais para o desenvolvimento da nossa aplicação. Tal como anteriormente referido, esta apresenta uma *F1ManagerFacade* responsável por tornar a interação entre o utilizador e aplicação mais simples. De seguida, dispomos de dois *Maps* cuja função principal é registar Pilotos e Jogadores na aplicação, respetivamente. O *Map* de Pilotos apresenta como chave uma *String* que representa o nome de um piloto, e valores do tipo *Piloto.* O *Map* de Jogadores apresenta, tal como o de Pilotos, como chave uma *String*, que representa o nome do jogador na aplicação, e valores do tipo *Jogador.* Finalmente, a nossa arquitetura apresenta ainda três subsistemas, definidos anteriormente no modelo de Componentes: Subsistema do Campeonato (*SSCampeonato*), o Subsistema do Circuito (*SSCircuito*) e o Subsistema do Carro (*SSCarro*).

Subsistema Campeonato

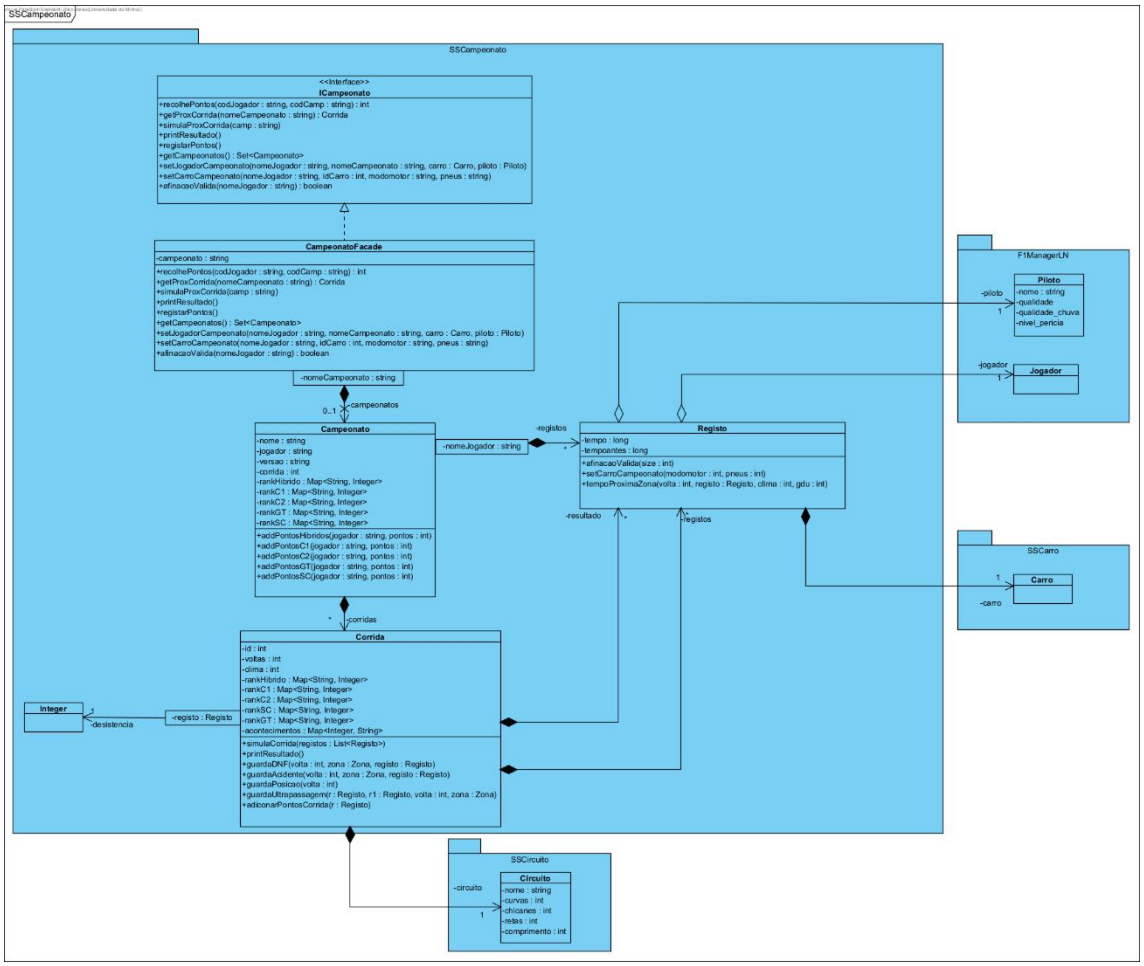

Figura 3 - Diagrama de Classes Subsistema Campeonato

Neste subsistema, para além da *Facade* do Campeonato, utilizada com o mesmo propósito da anteriores, temos a classe *Campeonato*, ressalvando o facto de nesta classe existirem vários *Maps,* responsáveis por guardar os pontos de cada jogador referentes ao campeonato em questão, separados por categoria, respeitando aquilo que nos foi pedido no enunciado deste trabalho. É neste método, também, que guardamos uma lista com as várias corridas presentes no campeonato. A classe *Corrida*, igualmente ao que acontece com a classe anteriormente referida, apresenta essencialmente *Maps* com o propósito de guardar informação sobre a pontuação dos vários jogadores, divididos pelas diferentes categorias. A classe *Corrida* guarda ainda duas listas de objetos do tipo *Registo:* uma lista "registos" responsável por guardar os registos que irão "participar" na corrida, e uma lista "resultados", onde será armazenada a classificação geral da corrida em questão, para todos os carros que terminaram a corrida, e que, portanto, não tem em conta a categoria de cada carro. De notar

que um objeto *Registo* é constituído por um carro, um piloto e pelo jogador que efetuou esse mesmo registo.

Subsistema Circuito

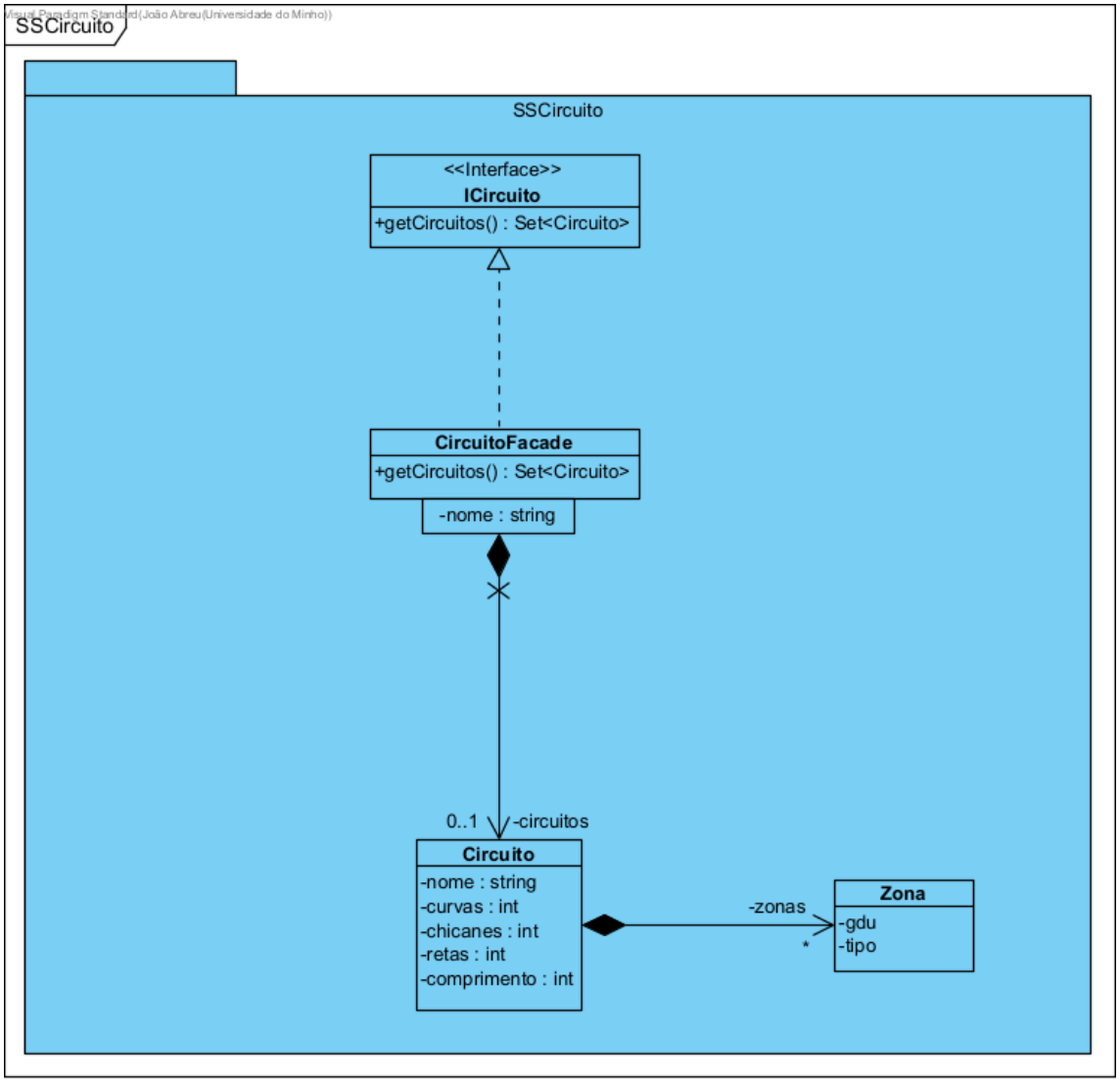

Figura 4 - Diagrama de Classes Subsistema Circuito

No subsistema *SSCircuito*, para além da já habitual *Facade*, com o propósito de facilitar a comunicação entre um elemento exterior e os métodos presentes neste subsistema, apresentamos a classe *Circuito*, que contêm uma lista de zonas. Devido à importância destas para aquilo que será, mais tarde, a simulação das corridas, decidimos isolar o sistema de *Circuito* num subsistema próprio.

Subsistema Carro

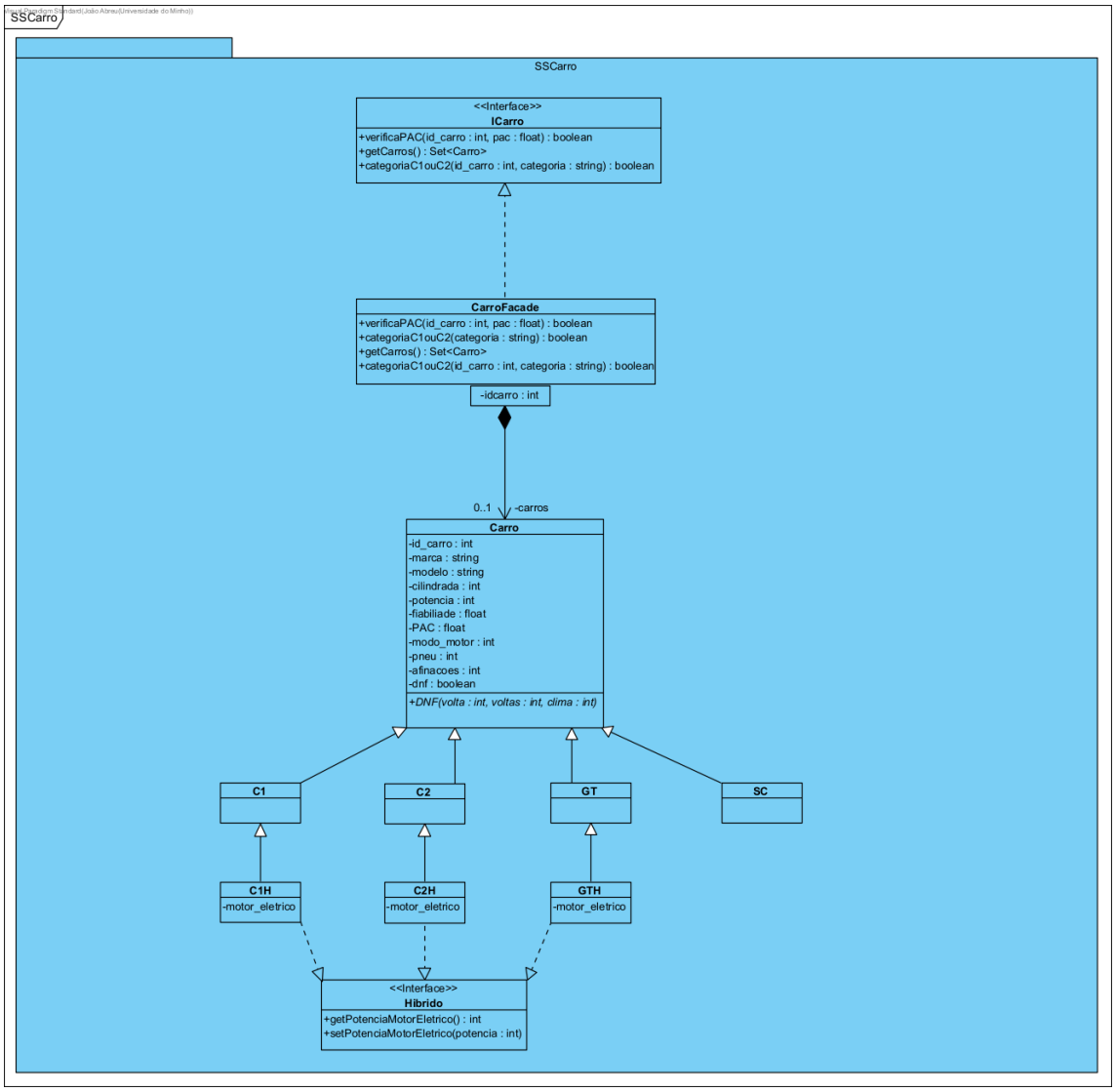

Figura 5 - Diagrama de Classes Subsistema Carro

Finalmente, no subsistema dos Carros, existe uma classe *Facade,* pelos motivos anteriormente referidos, e também uma classe *Carro*. Esta classe será superclasse das classes que representam as diferentes categorias de carros presentes no nossa arquitetura, nomeadamente, *C1*, *C2*, *GT* e *SC*. De referir, ainda, que as subclasses *C1*, *C2* e *GT* são estendidas por subclasses próprias (*C1H*, *C2H* e *GTH*, respetivamente) que implementam uma interface *Hibrido*, a fim de conseguir expressar que os carros das categorias C1, C2 e GT podem ser híbridos (caso apresentem motor elétrico e motor de combustão), ou carros apenas com motor de combustão.

Diagrama de *Packages*

Por fim, elaboramos o nosso diagrama de *Packages*, começando por considerar as dependências existentes entre classes, o que nos proporcionou uma maior facilidade em perceber a arquitetura do nosso sistema.

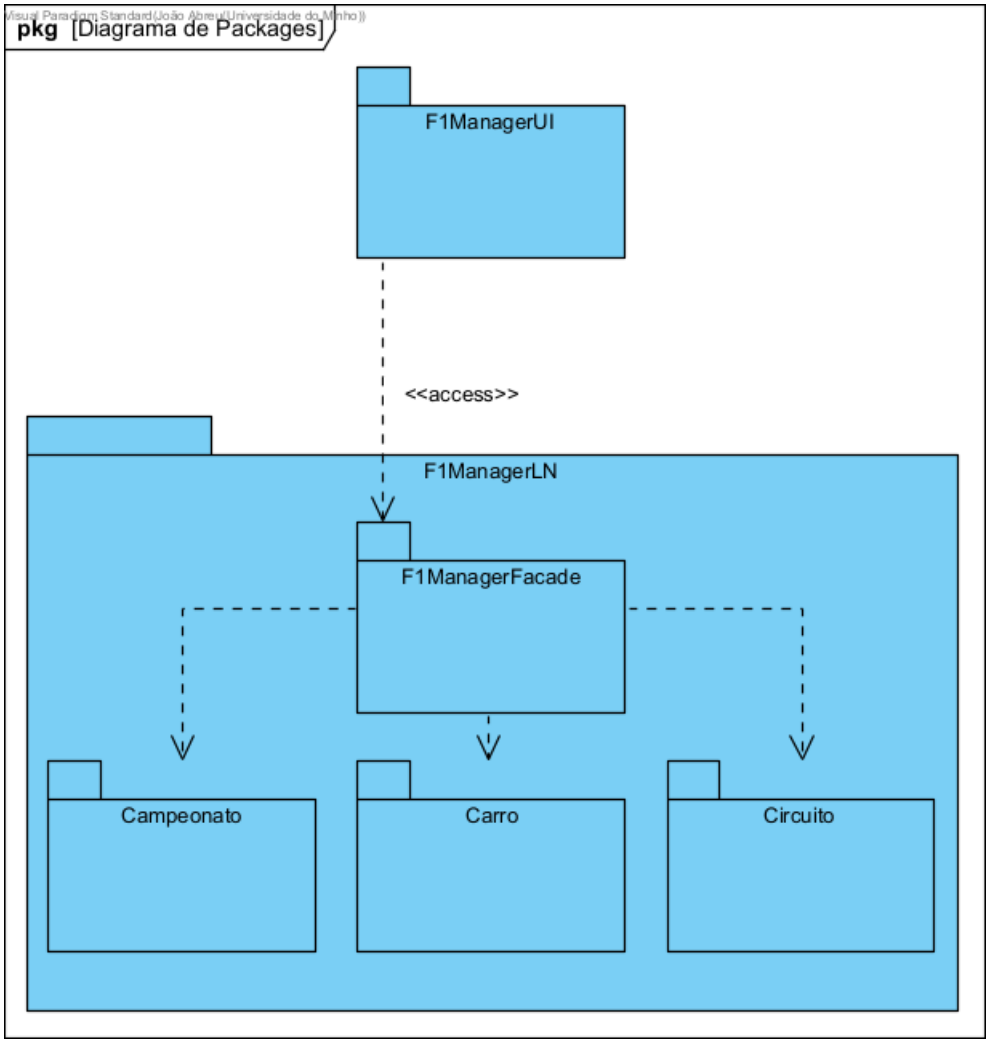

Figura 6 - Diagrama de Packages

Modelos Comportamentais

Diagramas de Sequência

Com base nos *Use Cases* definidos e respetivas especificações dos mesmos, construímos diversos Diagramas de Sequência, ou seja, diagramas comportamentais que descrevem como um conjunto de objetos coopera a fim de realizar um determinado comportamento. Estes diagramas permitem-nos compreender com maior pormenor o fluxo de eventos necessário para o bom funcionamento idealizado do nosso sistema, representando as interações entre objetos através das mensagens que são trocadas entre estes, dando ênfase à ordenação temporal das

mesmas. Além disso, estes diagramas permitem-nos ainda analisar as “responsabilidades” das diferentes entidades, mostrando onde está a ser efetuado o processamento.

1. adicionarPontos

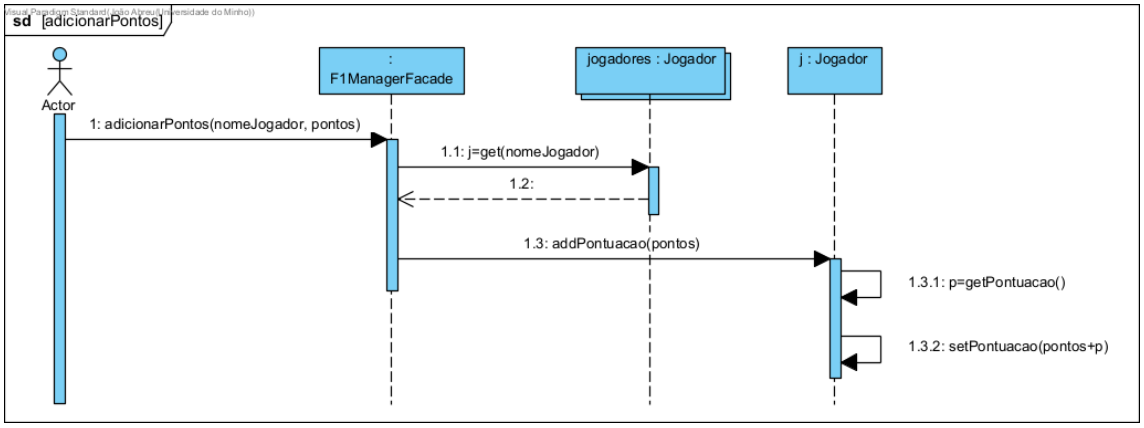

Figura 7 - Diagrama de Sequência adicionarPontos

2. adicionarPontosCorrida

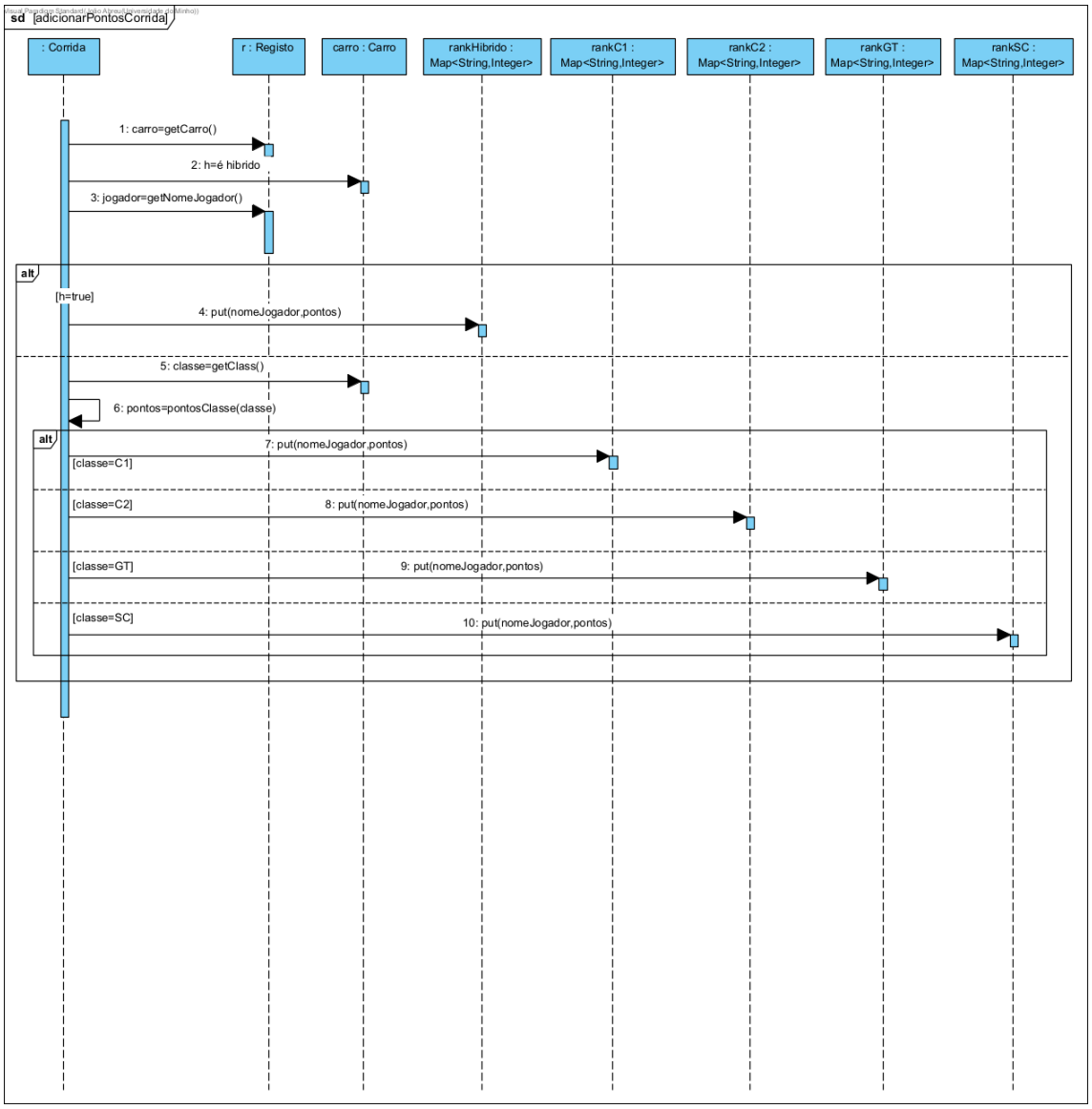

Figura 8 - Diagrama de Sequência adicionarPontosCorrida

3. afinacaoValida

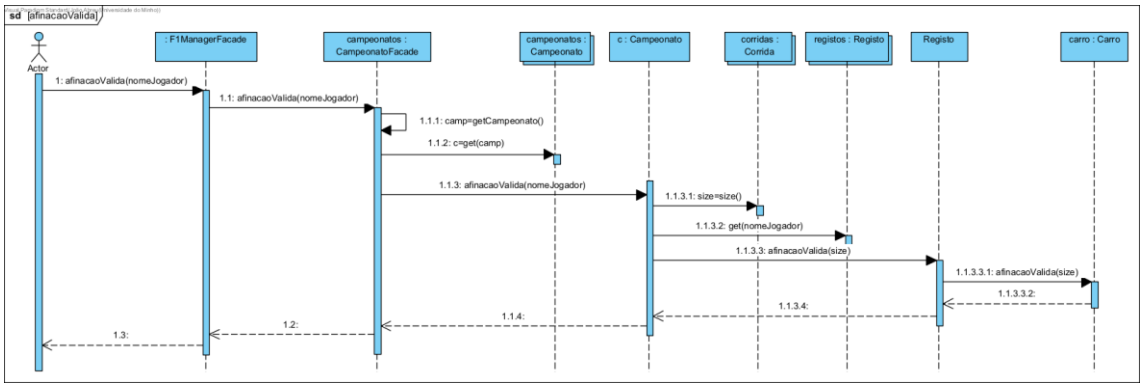

Figura 9 - Diagrama de Sequência afinacaoValida

4. categoriaC1ouC2

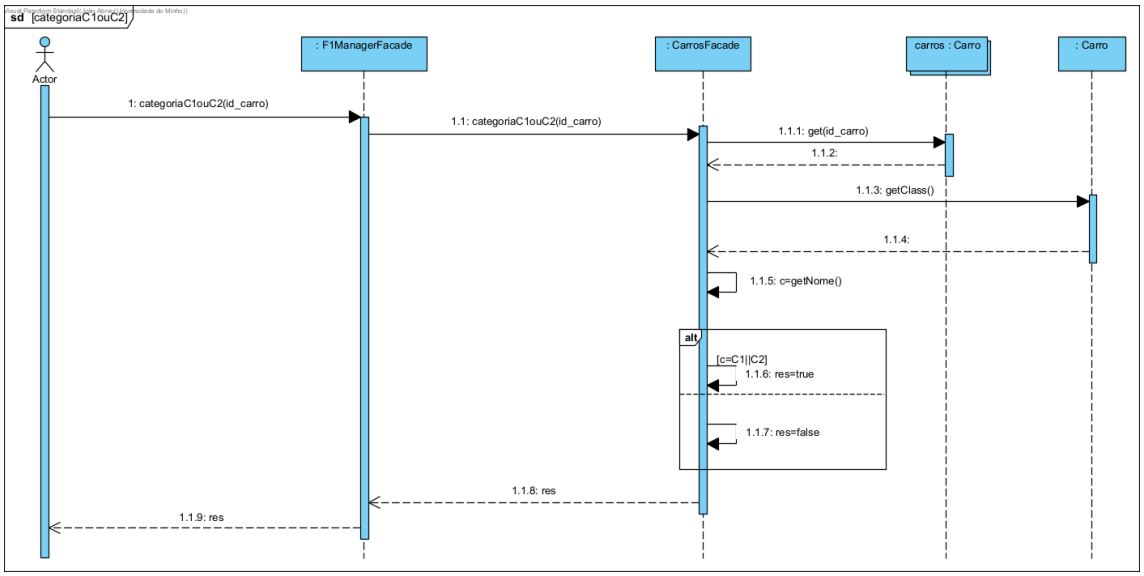

Figura 10 - Diagrama de Sequência categoriaC1ouC2

5. estaAutenticado

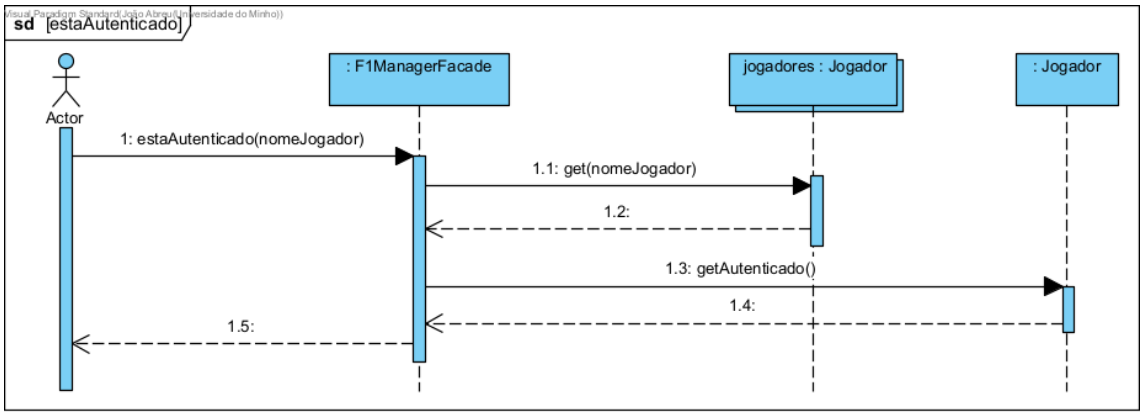

Figura 11 - Diagrama de Sequência estaAutenticado

6. getCampeonatos

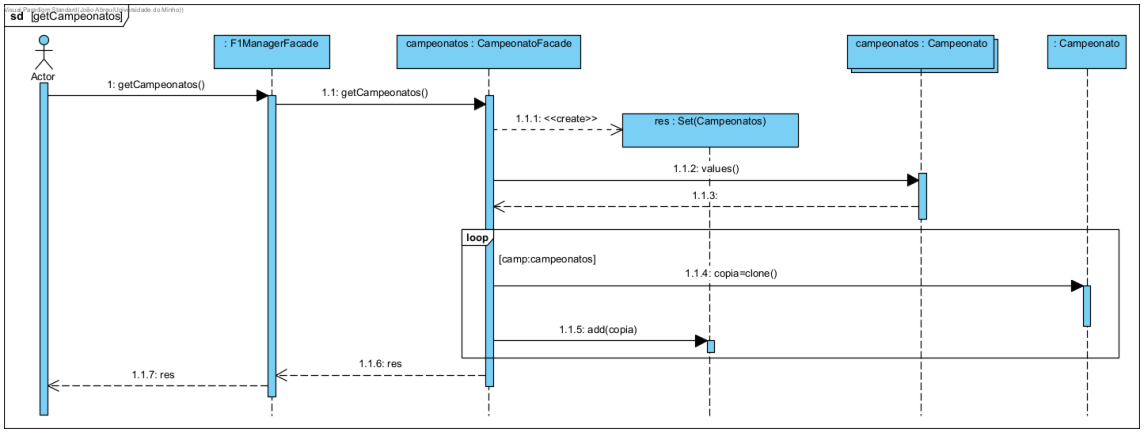

Figura 12 - Diagrama de Sequência getCampeonatos

7. getCarros

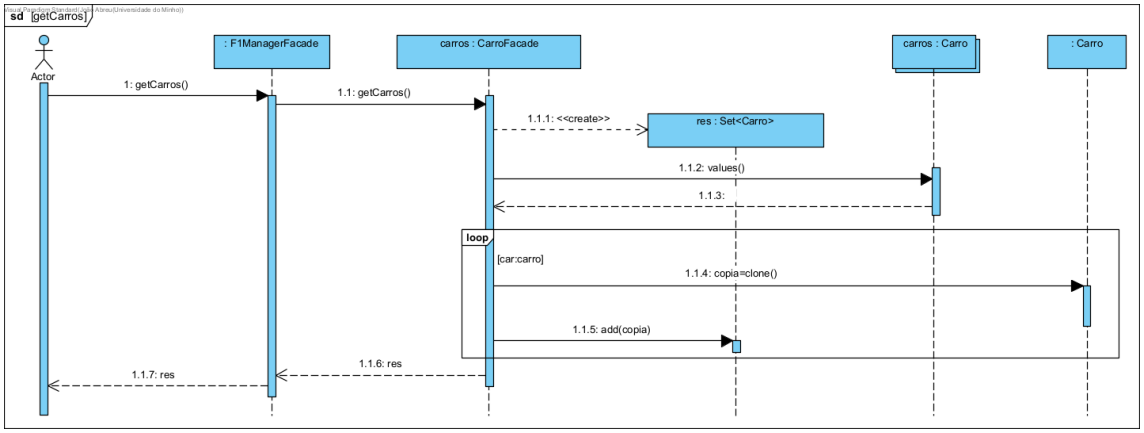

Figura 13 - Diagrama de Sequência getCarros

8. getCircuitos

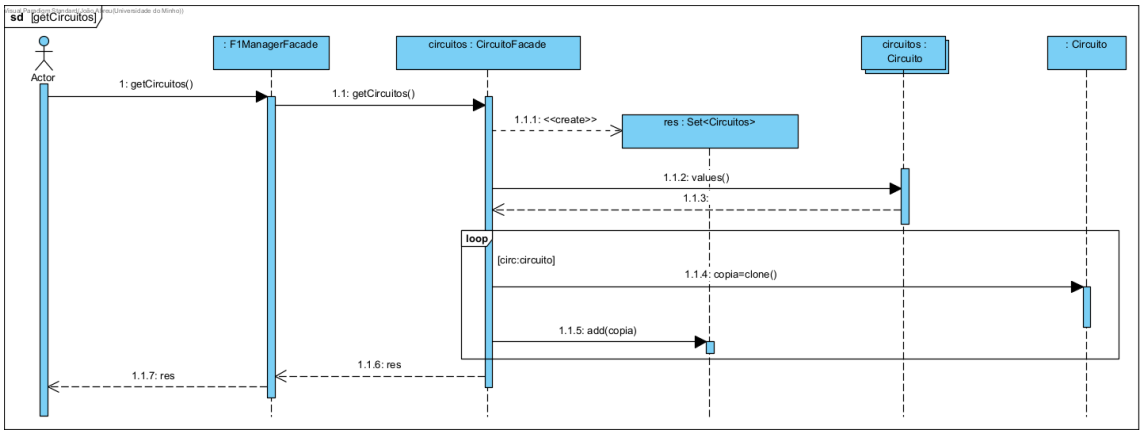

Figura 14 - Diagrama de Sequência getCircuitos

9. getPilotos

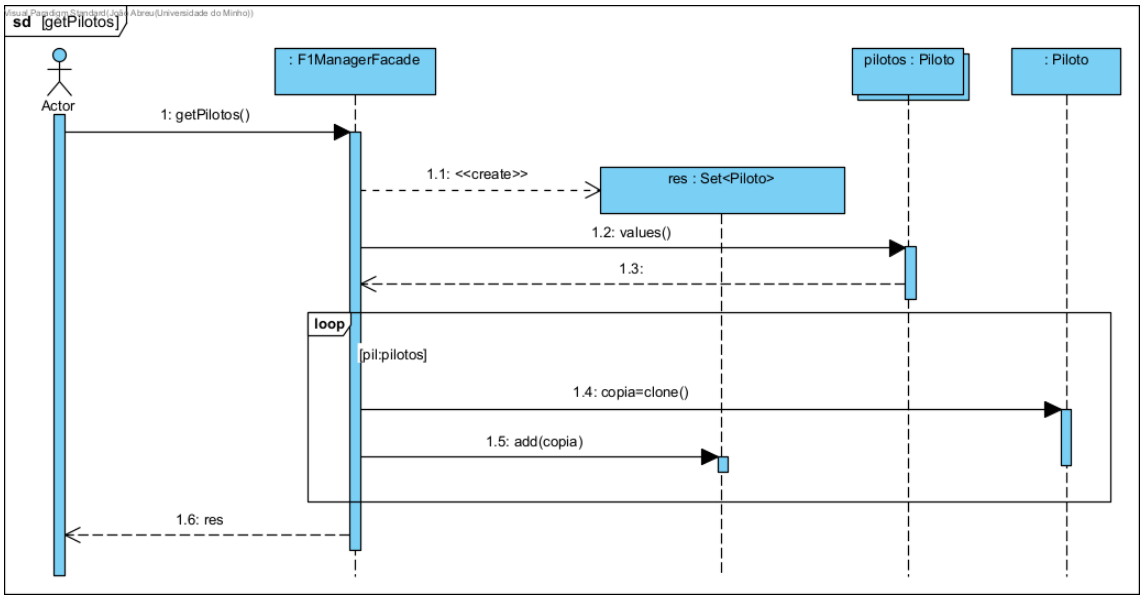

Figura 15 - Diagrama de Sequência getPilotos

10. getProxCorrida

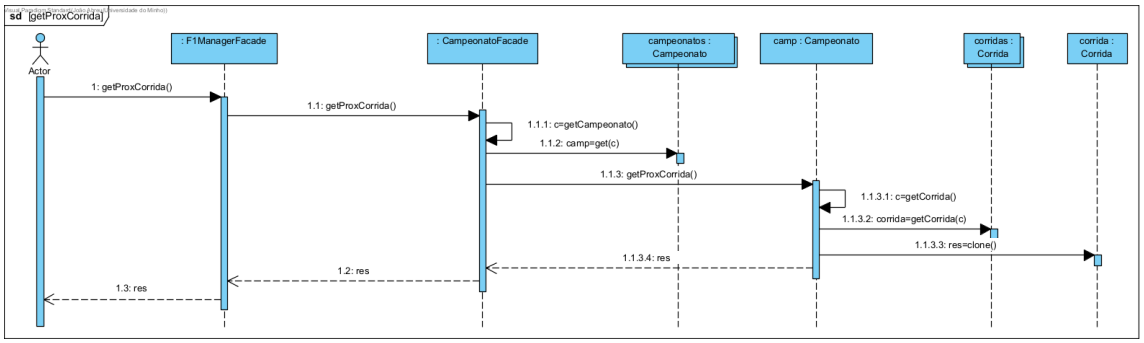

Figura 16 - Diagrama de Sequência getProxCorrida

11. printResultado

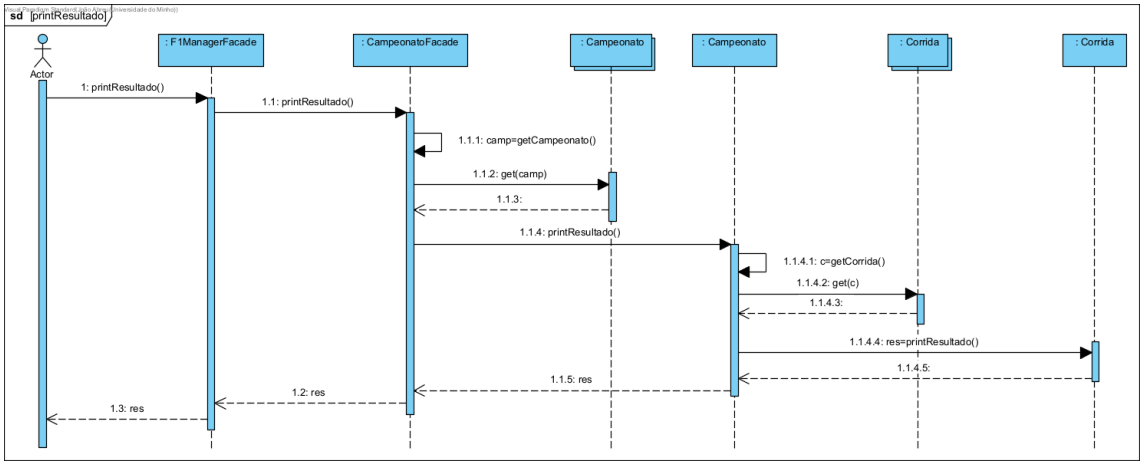

Figura 17 - Diagrama de Sequência printResultado

12. recolhePontos

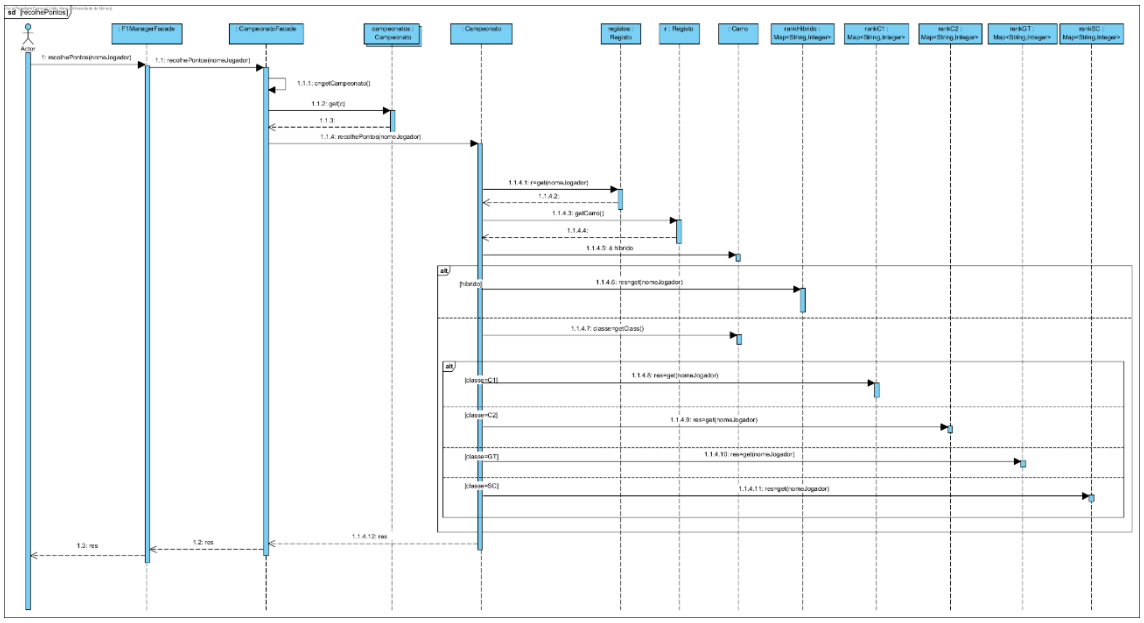

Figura 18 - Diagrama de Sequência recolhePontos

13. registaPontos

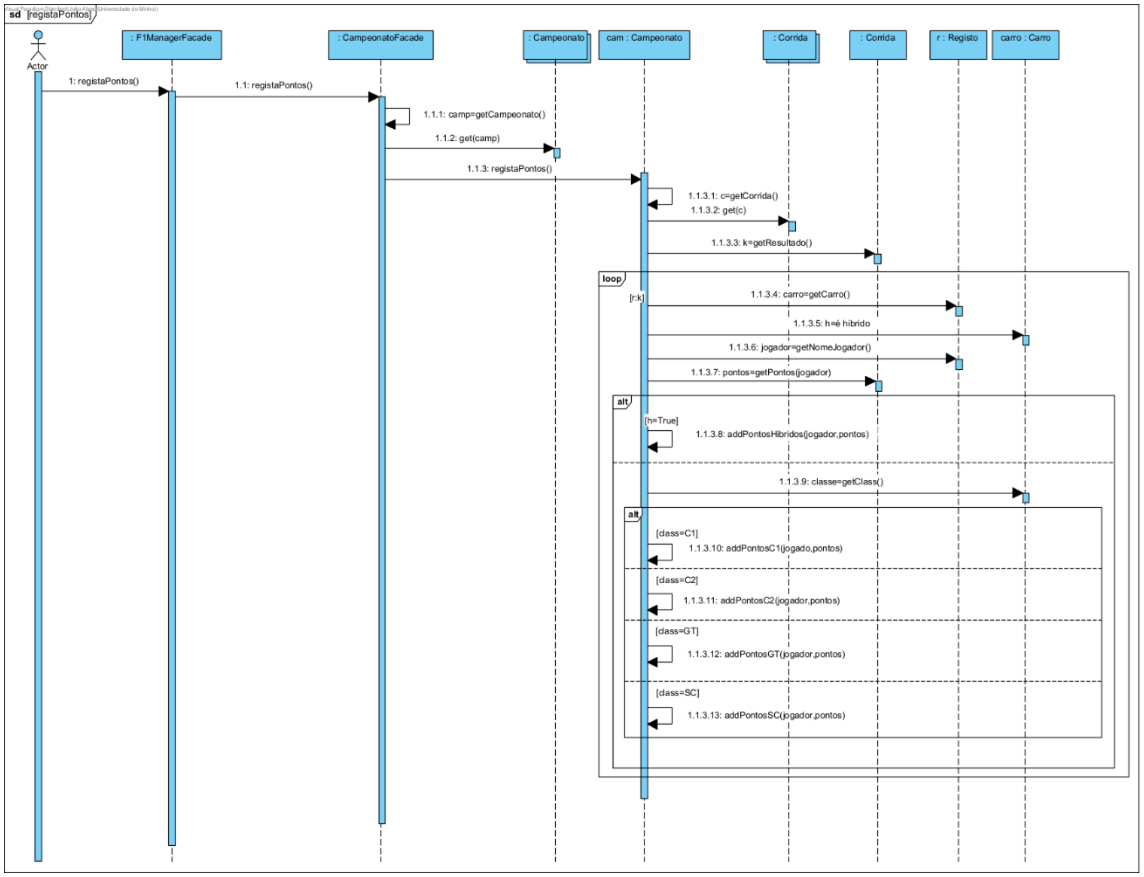

Figura 19 - Diagrama de Sequência registaPontos

14. setCarroCampeonato

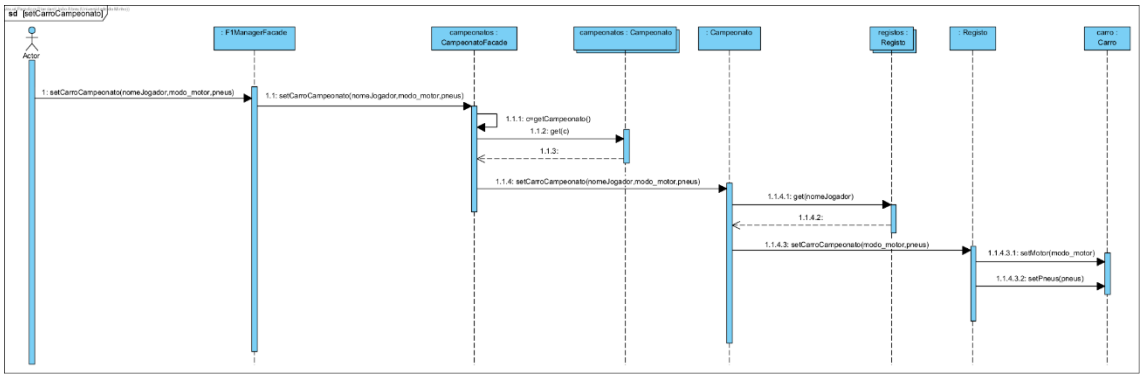

Figura 20 - Diagrama de Sequência setCarroCampeonato

15. setJogadorCampeonato

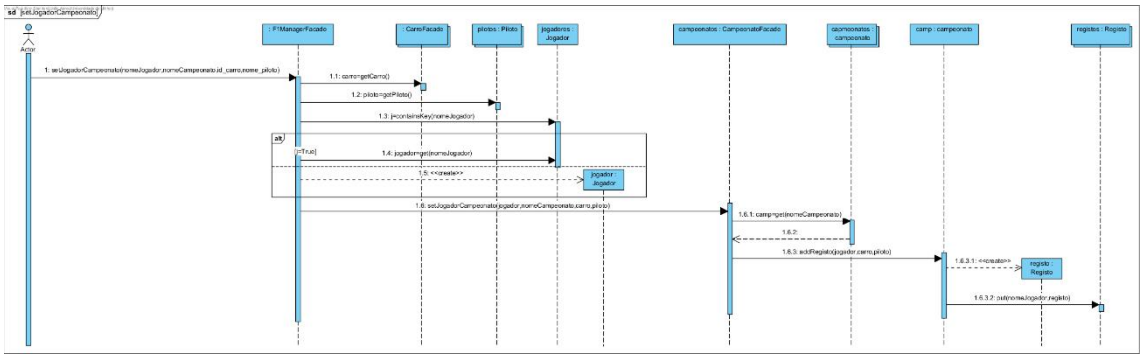

Figura 21 - Diagrama de Sequência setJogadorCampeonato

16. simulaCorrida

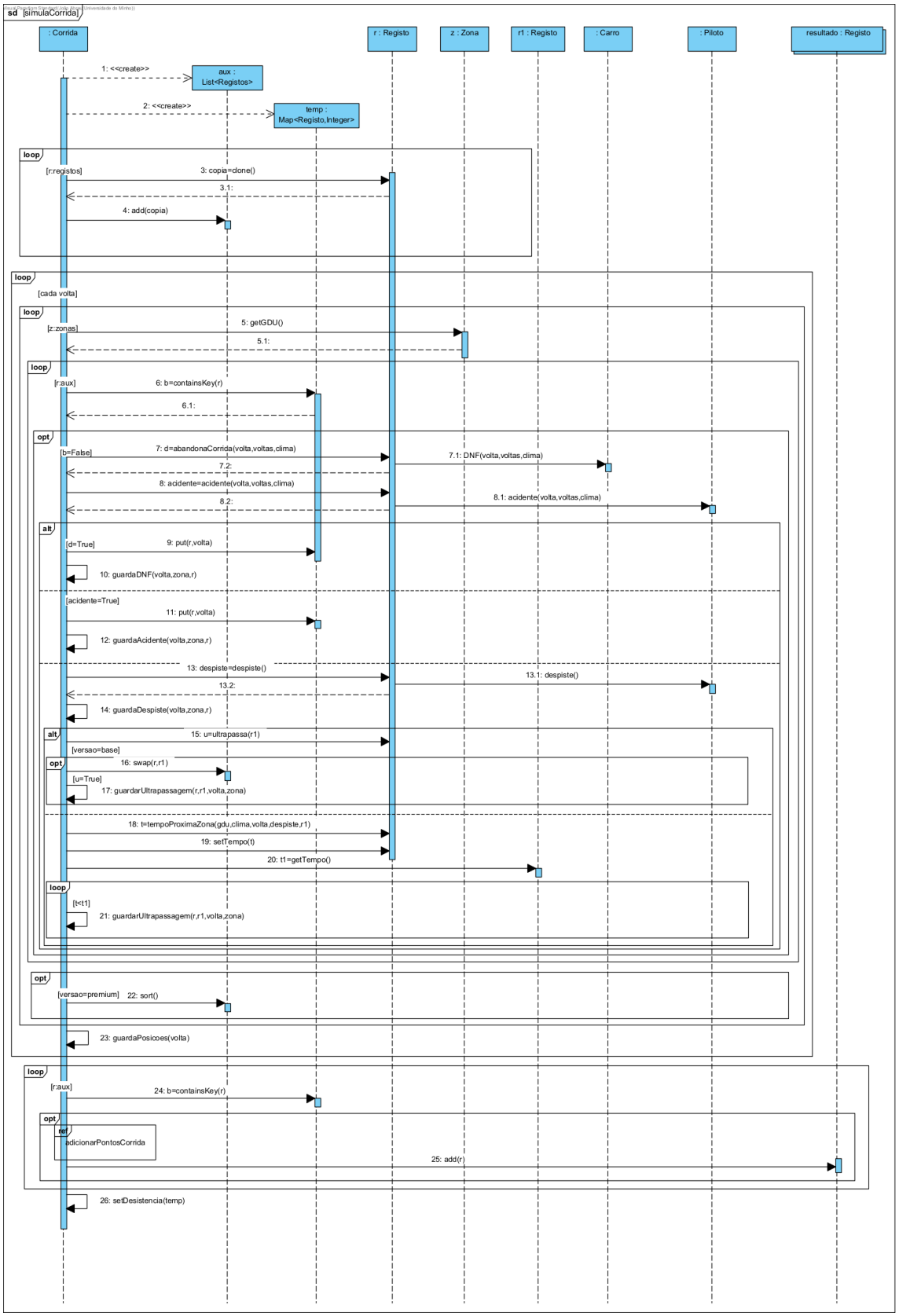

Figura 22 - Diagrama de Sequência simulaCorrida

17. simulaProxCorrida

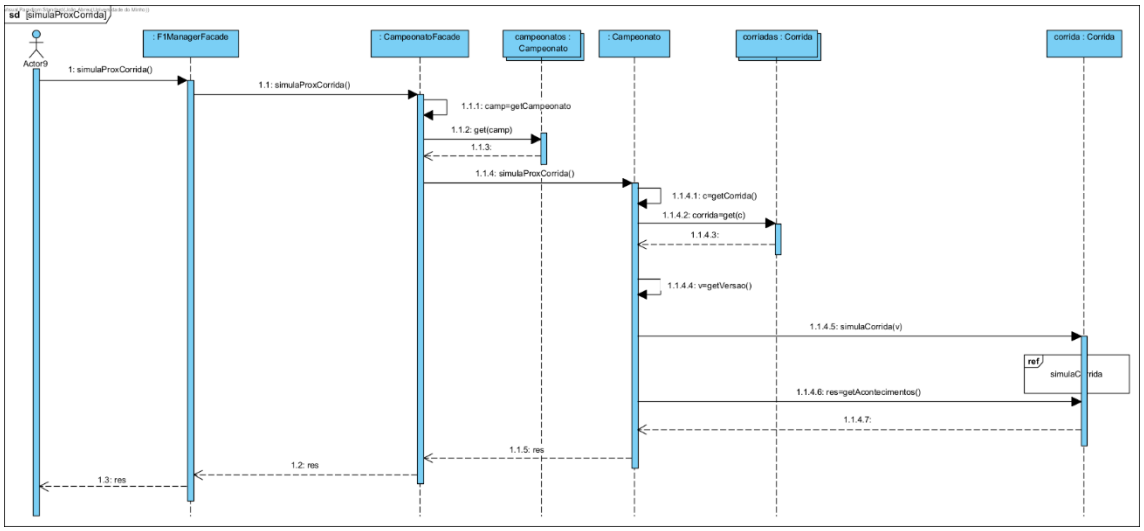

Figura 23 - Diagrama de Sequência simulaProxCorrida

18. verificarPAC

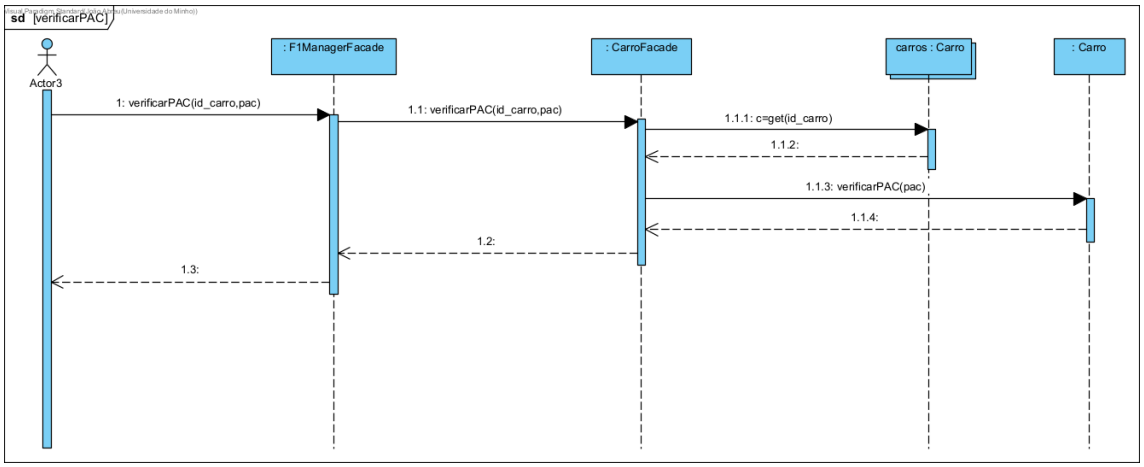

Figura 24 - Diagrama de Sequência verificaPAC

Reutilização do Código Disponibilizado

- As subclasses de *Carro* e a interface *Hibrido* foram reutilizadas;
- Da classe *Carro* foram mantidos essencialmente todos os atributos, exceto os atributos "tempo", uma vez que as corridas são simuladas a partir de objetos *Registo*, sendo o tempo de corrida armazenado nestes, e "equipa", já que na nossa arquitetura não existe uma entidade equipa, uma vez que os pilotos não apresentam equipa;

- Usaremos uma função semelhante à "tempoProximaVolta()" presente na classe *Carro*, no entanto, implementada na classe *Registo*, uma vez que o tempo de corrida é armazenado em objetos da classe *Registo*, tal como referido no ponto anterior;
- Na classe *Campeonato*, está presente uma lista de corridas, semelhante ao apresentado, bem como um atributo que é um número inteiro e que se refere à corrida a ser realizada;
- Em relação à classe *Piloto*, não utilizamos o atributo "palmarés", uma vez que decidimos não armazenar o palmarés de cada piloto na nossa arquitetura.

Conclusão e Análise Crítica

Em retrospetiva, nesta segunda fase continuamos a desenvolver o nosso projeto com uma base sólida e ideias bem definidas. No entanto, sempre que achámos necessário, fomos revendo os modelos construídos anteriormente e discutimos ideias, tentando também conciliar sempre as várias ideias dos elementos do grupo através de reuniões frequentes.

Todo o processo deste trabalho prático tem-se revelado enriquecedor pois dá-nos ferramentas e hábitos importantes para o desenvolvimento de software robusto, com um planeamento cuidado, sem saltar etapas. Neste ponto, a linguagem *UML* foi, sem dúvida, bastante útil na construção dos mais diversos modelos e diagramas que suportam a codificação e que se têm revelado fundamentais no desenvolvimento de um projeto sólido e sustentado, que garanta os requisitos propostos.

Em suma, consideramos que cumprimos de forma adequada os objetivos apresentados até ao momento, e que demos uma boa resposta na conceção de todos os diagramas e modelos representacionais necessários, podendo assim também entender a importância destes processos para o culminar num produto final de qualidade.