Universidade do Minho
Escola de Engenharia

Relatório Nº3

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

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*.

Nas fases anteriores, desenvolveram-se o Modelo de Domínio e o Diagrama de Use Cases, assim como um Diagrama de Componentes, Diagrama de *Packages*, Diagrama de Classes e ainda Diagramas de Sequência.

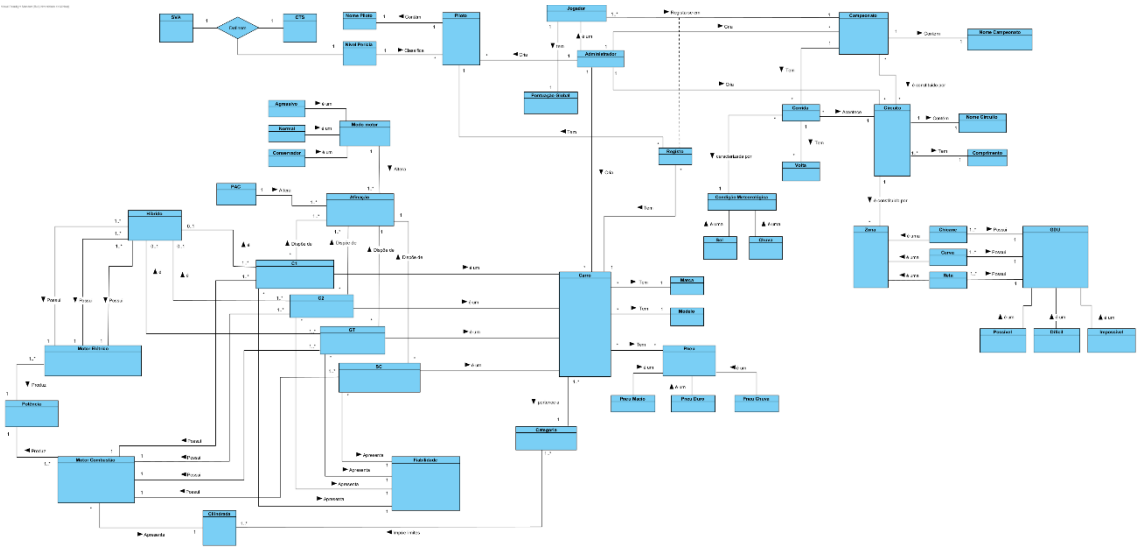
O presente relatório serve de suporte à realização e entrega da terceira e última fase deste trabalho, que diz respeito à apresentação dos modelos necessários à descrição da implementação, assim como a implementação propriamente dita: o desenvolvimento de código em Java, implementando a aplicação que cumpra todos os requisitos especificados. Indicaremos todas as alterações que nos foi necessário fazer, bem como os passos tomados na implementação. No final, faremos uma breve conclusão e análise crítica a todo o processo de realização deste trabalho prático.

Objetivos

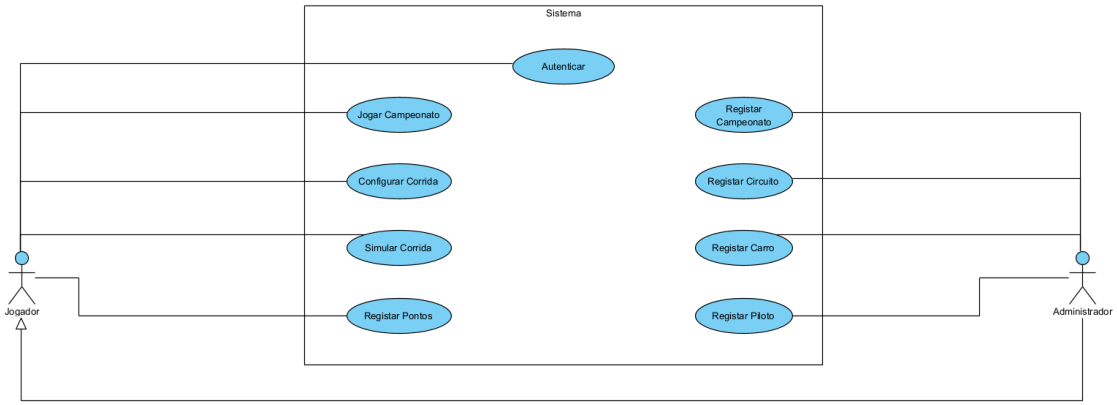
Nesta que é a última etapa do projeto possuímos tarefas bem definidas; primeiramente, reajustar, modificar e/ou terminar a conceção de todos os nossos modelos e proceder à implementação completa do nosso software. É neste ponto que, para além da simplificação do problema, todo o processo de trabalho até ao momento se fundirá para a criação de um caminho para a implementação com sucesso do sistema de software que, visto cumpridos todos os requisitos e regras de boa prática, será funcional, irá satisfazer os seus casos de uso e arquiteturas, e ainda muito importante: será flexível, escalável e utilizável.

Modelação

Modelo de Domínio (Fase 1)



Modelo de Use Case (Fase 1)



Especificação dos Use Cases (Fase 2 atualizados)

Use cases definidos para a realização do cenário 5.

Use Case: Configurar campeonato

Descrição: O Francisco e três amigos resolvem jogar um campeonato

Cenário: Cenário 5 - Configurar Campeonato

Pré-condição: *True*

Pós-condição: Campeonato fica configurado

Fluxo Normal:

1. Sistema apresenta os campeonatos
2. Jogador escolhe um campeonato da lista de campeonatos

3. Sistema apresenta os circuitos que pertencem a esse campeonato
4. Sistema apresenta os carros disponíveis
5. Jogador escolhe um carro da lista de carros
6. Sistema apresenta os pilotos disponíveis
7. Jogador escolhe um piloto da lista de pilotos
8. Sistema configura o campeonato

Use Case: Configurar Corrida

Descrição: Jogador prepara o carro para a corrida

Cenário: Cenário 5 - Configurar Corrida

Pré-condição: Campeonato configurado

Pós-condição: Corrida fica pronta a começar

Fluxo Normal:

1. Sistema apresenta o circuito e a situação meteorológica
2. Sistema verifica que é possível alterar PAC e escolher modo de motor por se tratar de um carro da categoria C1 ou C2
3. Sistema verifica que é possível realizar mais uma afinação
4. Jogador indica um novo valor do PAC para o carro
5. Sistema aceita o novo valor do PAC para o carro
6. Jogador escolhe um novo modo do motor
7. Jogador escolhe os pneus a utilizar
8. Sistema termina o processo de configuração de corrida

Fluxo Alternativo (1): [Não é possível alterar PAC nem escolher modo de motor por não se tratar de um carro da categoria C1 ou C2] (passo 2)

2.1. Sistema verifica que o PAC do carro não pode ser alterado e que não é possível escolher o modo de motor

2.2. Regressa a **6.**

Fluxo Alternativo (2): [Jogador não pretende alterar o PAC do carro] (passo 4)

4.1. Jogador não pretende alterar o valor do PAC do carro

4.2. Regressa a **6.**

Fluxo de Exceção (1): [Valor de PAC não se encontra entre 0 e 1] (passo 5)

5.1. Sistema não aceita valor do PAC do carro, visto que não se encontra entre 0 e 1

5.2. Regressa a **4.**

Use Case: Simular corrida

Descrição: *Racing Manager* simula a corrida

Cenário: Cenário 5 – Jogar (Simular Corrida)

Pré-condição: Corrida configurada

Pós condição: Simulação da corrida terminada

Fluxo Normal:

1. Sistema distribui os carros, ao longo da grelha de partida, agrupados pelas respetivas categorias
2. Sistema inicia a corrida
3. Sistema indica para cada curva, reta e chicane de cada volta, eventuais ultrapassagens, despistes e avarias.
4. Sistema indica as posições dos carros/pilotos/jogadores no final de cada volta
5. Sistema termina a corrida
6. Sistema apresenta os resultados da corrida
7. Sistema apresenta os pontos de cada jogador
8. Sistema regista e apresenta os pontos de cada jogador no respetivo ranking do campeonato, consoante categoria automobilística escolhida.

Alterações na Arquitetura do Sistema

Antes de iniciar a nova fase do projeto, realizamos algumas alterações na arquitetura previamente proposta. Todas as referências a F1Manager foram alteradas para RacingManager.

Quanto à casse Piloto foi adicionado a operação despiste, que se estava a ser utilizada nos diagramas de sequência, e referimos os tipos das variáveis. Na classe Jogador adicionamos um atributo versão do tipo *String*, para que um jogador tenha associado qual a versão do jogo que possui.

Na interface *IRacingManager* a operação categoriaC1ouC2 passou a receber um inteiro como parâmetro, para corresponder à descrição dos diagramas de sequência. A operação setCarroCampeonato em vez de receber o nome do jogador e o id do carro, apenas recebe o id do respetivo registo.

Antes de iniciar a nova fase do projeto, realizamos algumas alterações na arquitetura previamente proposta.

No subsistema campeonato, na classe corrida decidimos ao invés de ter dois *maps* distintos (acontecimentos e desistências) optamos por uma lista de Acontecimentos, super

classe que criamos para registar todos os acontecimentos de uma simulação. As subclasses correspondem aos diferentes acontecimentos que devem ser relatados. Mudamos ainda o tipo do clima passando este de *int* para *boolean*.

Para a classe Registo adicionamos-lhe um novo atributo, a pontuação, alterando deste modo também os *maps* do campeonato referentes aos rankings para uma lista de registos.

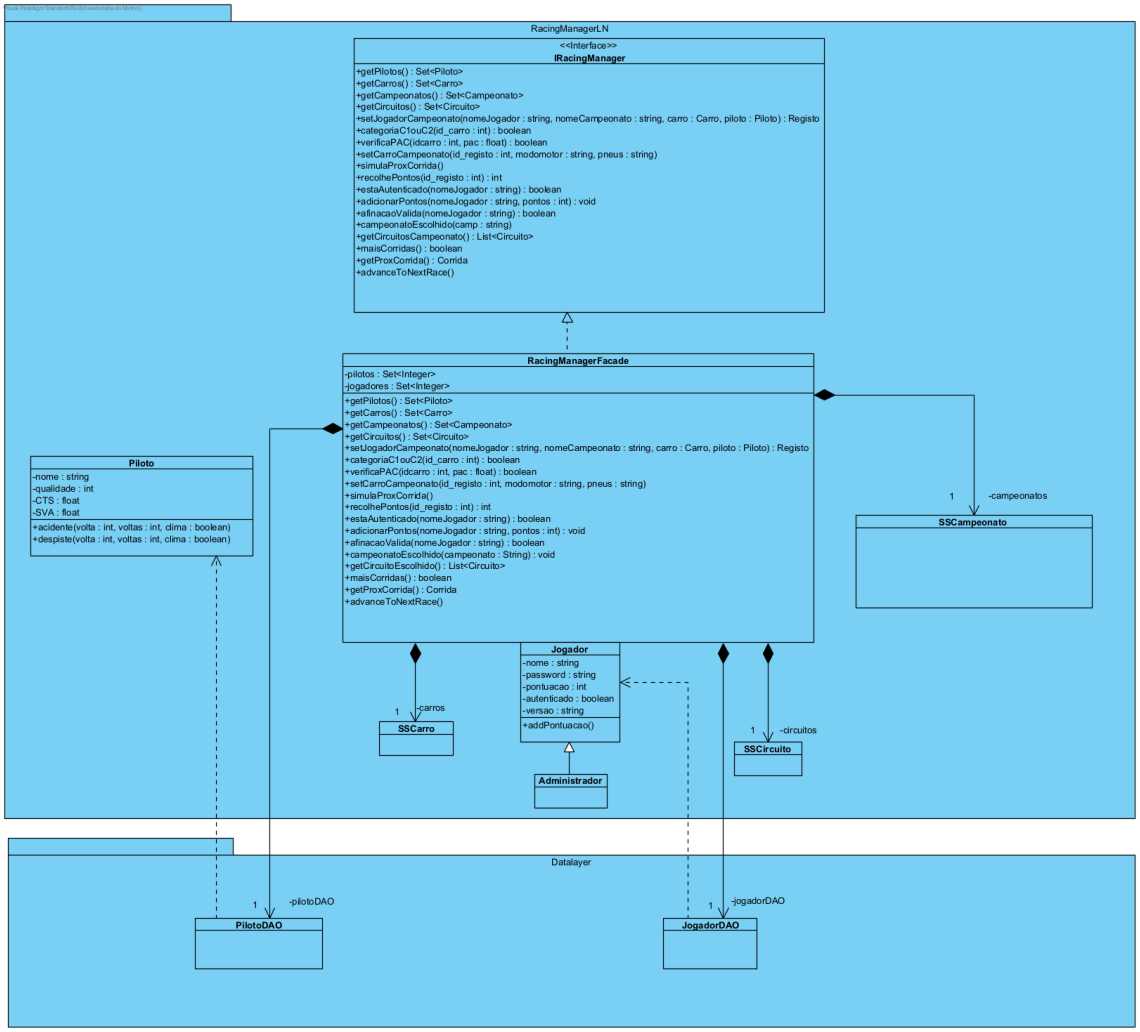
No subsistema circuito na classe Zona adicionamos um novo atributo, o número, para saber, por exemplo se é a quarta ou a quinta curva.

Diagrama de Classes

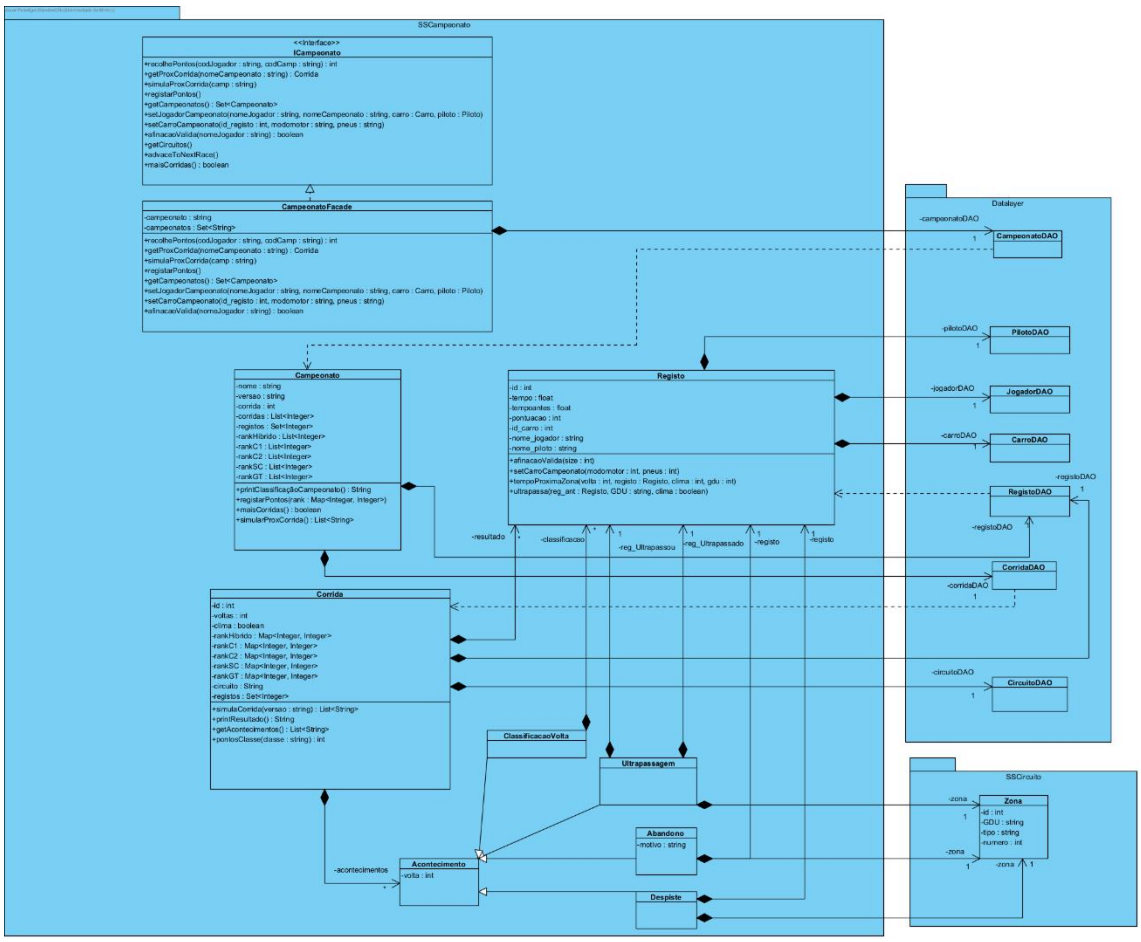
Neste capítulo, vamos discutir as modificações feitas no diagrama de classes definido anteriormente. As alterações eram necessárias para refletir corretamente o design atual do sistema e garantir a estabilidade e funcionalidade do software. Vamos explicar as razões por trás de cada modificação e como ela contribui para a melhoria geral do sistema.

Partindo do diagrama de Classes anterior, respeitando as diretrizes do *Object Relational Mapping – ORM*, identificamos as entidades que têm estado, que têm necessidade de "armazenar" dados e que devem ser persistidas. De seguida, mapeamos os relacionamentos, substituindo todas as coleções e *maps* por acesso à base de dados utilizando DAOs.

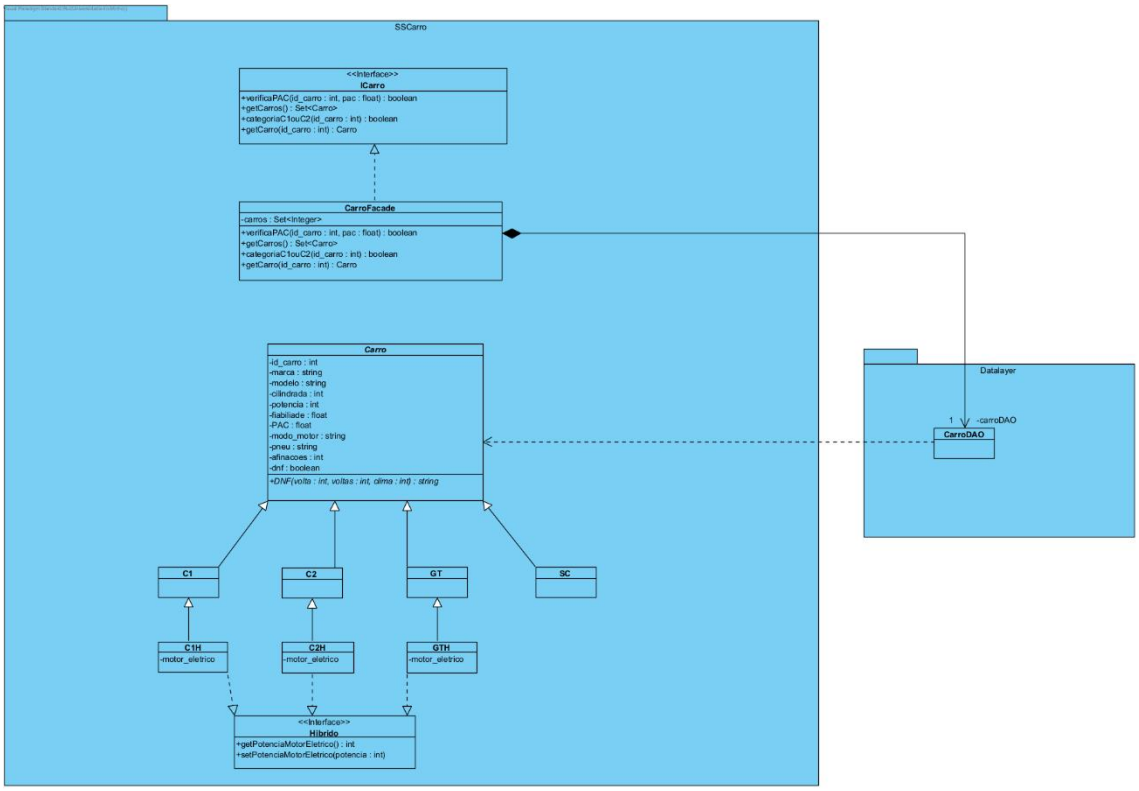
1. Diagrama de classes



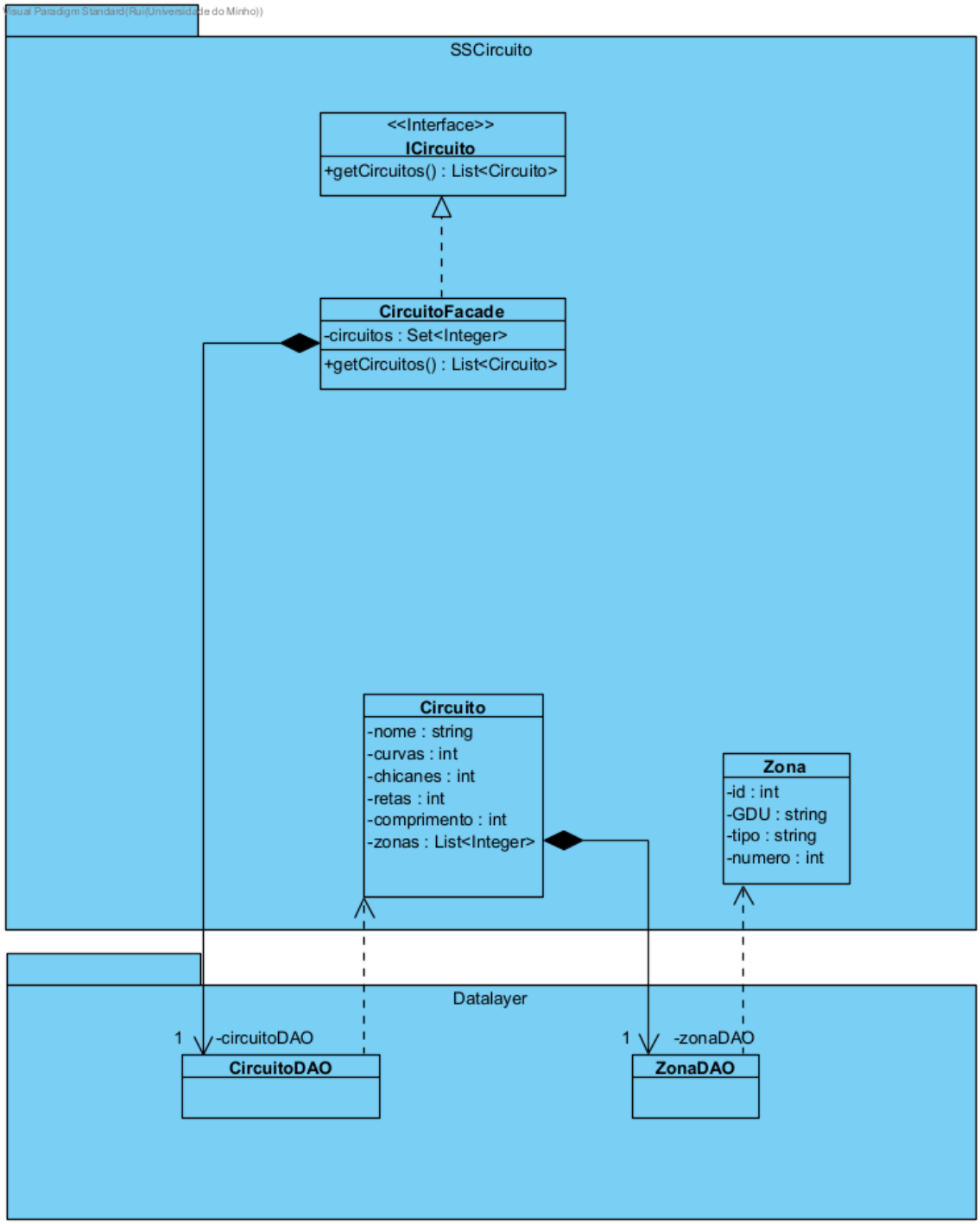
2. Diagrama de classes Subsistema Campeonato



3. Diagrama de classes Subsistema Carro



4. Diagrama de classes Subsistema Circuito

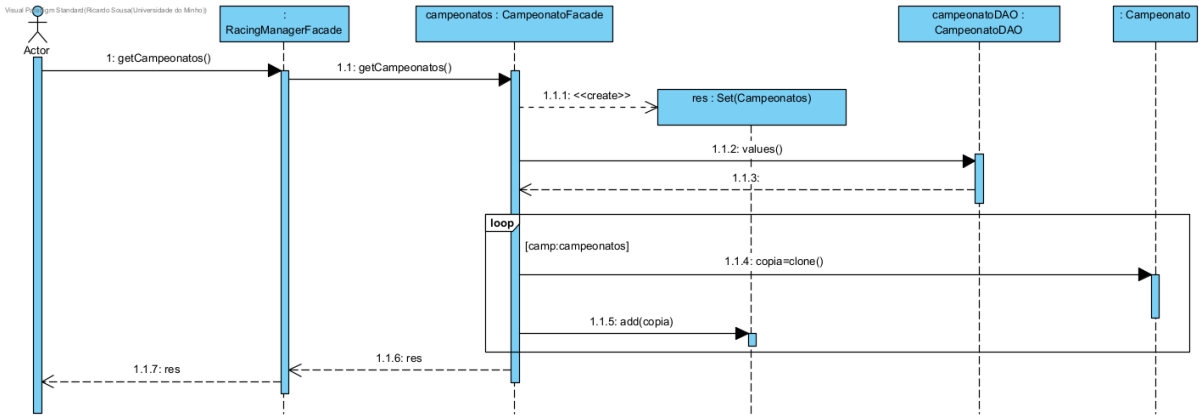

Visual Paradigm Standard(Rui(Universidade do Minho))
SSCircuito
<<Interface>>
ICircuito
+getCircuitos() : List<Circuito>
CircuitoFacade
-circuitos : Set<Integer>
+getCircuitos() : List<Circuito>
Circuito
-nome : string
-curvas : int
-chicanes : int
-retas : int
-comprimento : int
-zonas : List<Integer>
Zona
-id : int
-GDU : string
-tipo : string
-numero : int
Datalayer
1
-circuitoDAO
CircuitoDAO
1
-zonaDAO
ZonaDAO

Diagramas de Sequência

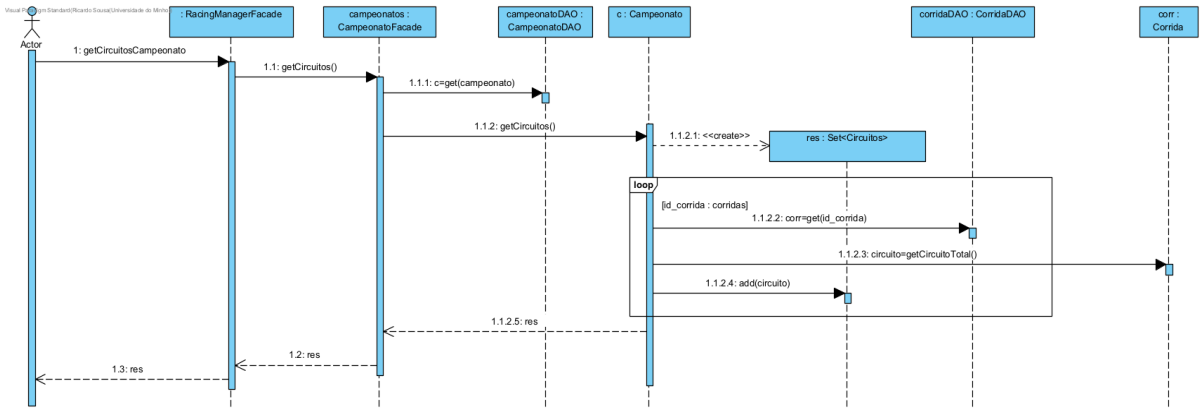
Agora vamos explorar as alterações realizadas nos diagramas de sequência anteriormente definidos, devido à introdução de DAOs na arquitetura do sistema. Estas modificações foram essenciais para garantir que a integração dos DAOs na arquitetura atual do sistema estava corretamente refletida e para entender como eles afetam os fluxos de dados existentes. Vamos explicar as razões por trás de cada mudança e como elas contribuem para a melhoria geral do sistema.

Relativamente ao use case "Configurar Corrida" os seguintes diagramas:

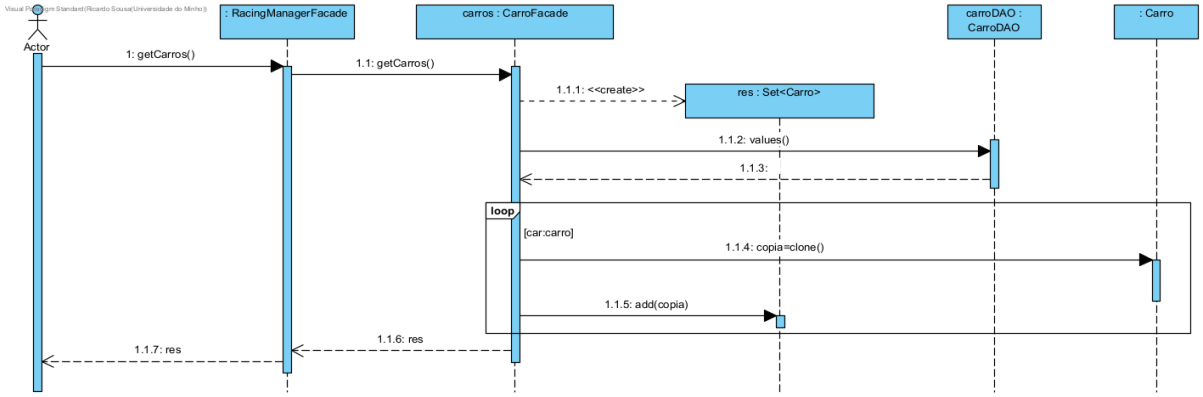
1. getCampeonatos():



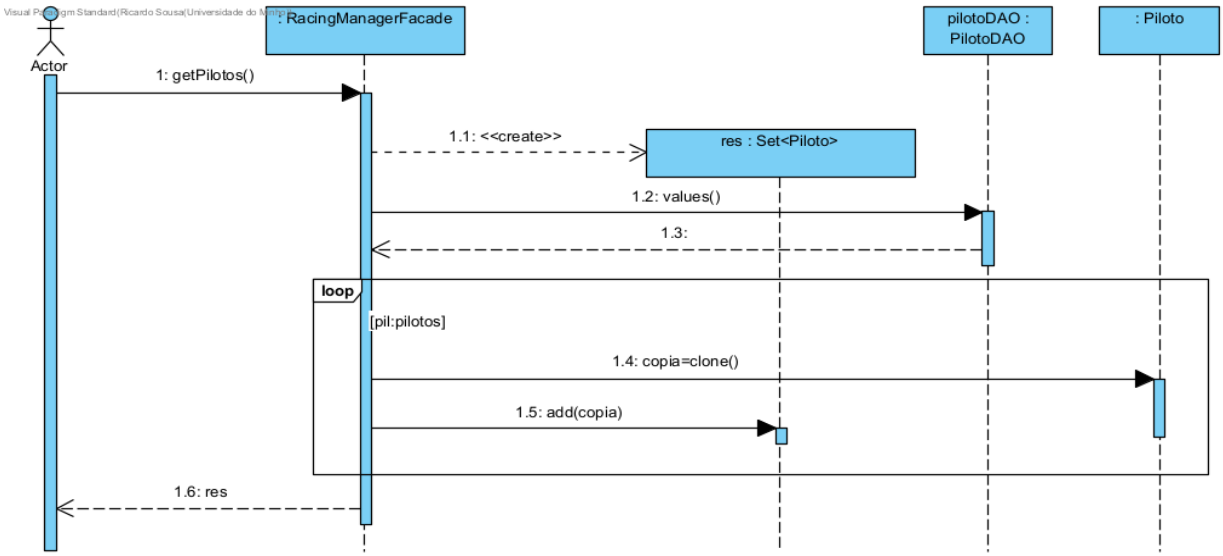
2. getCircuitosCampeonato():



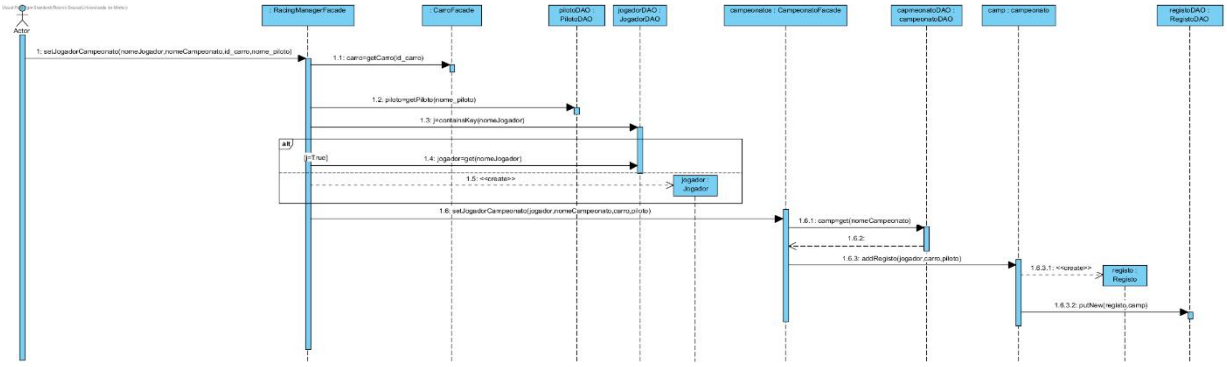
3. getCarros():



4. getPilotos():

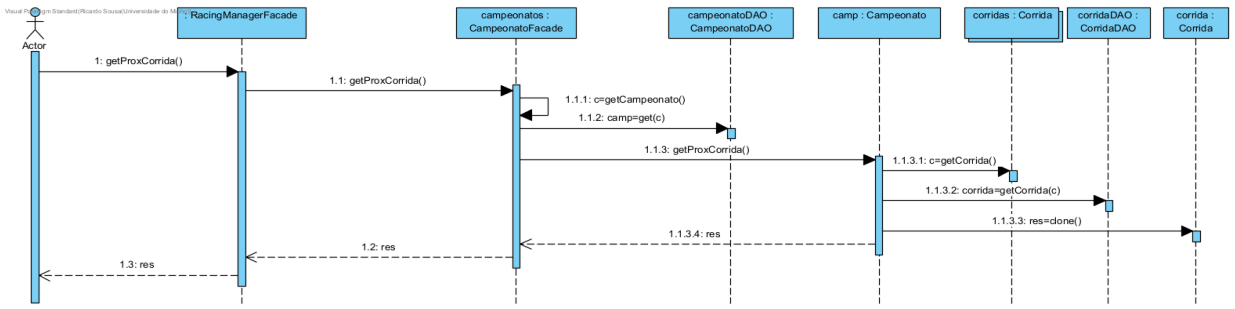


5. setJogadorCampeonato(nomeJogador, nomeCampeonato, id_carro, nome_piloto)

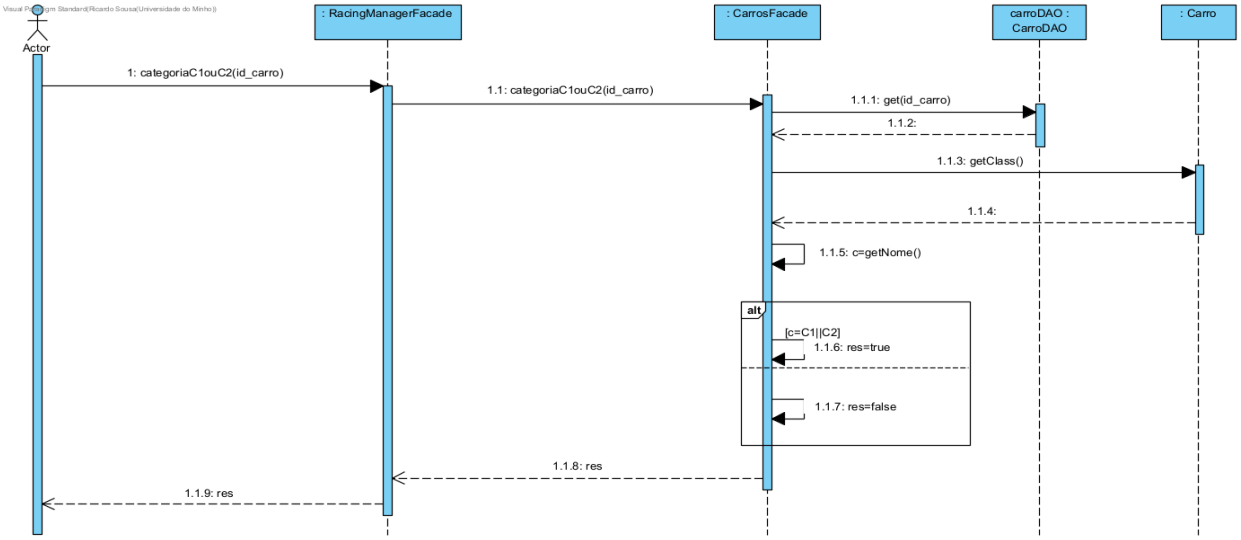


Relativamente ao use case “Configurar Corrida” os seguintes diagramas:

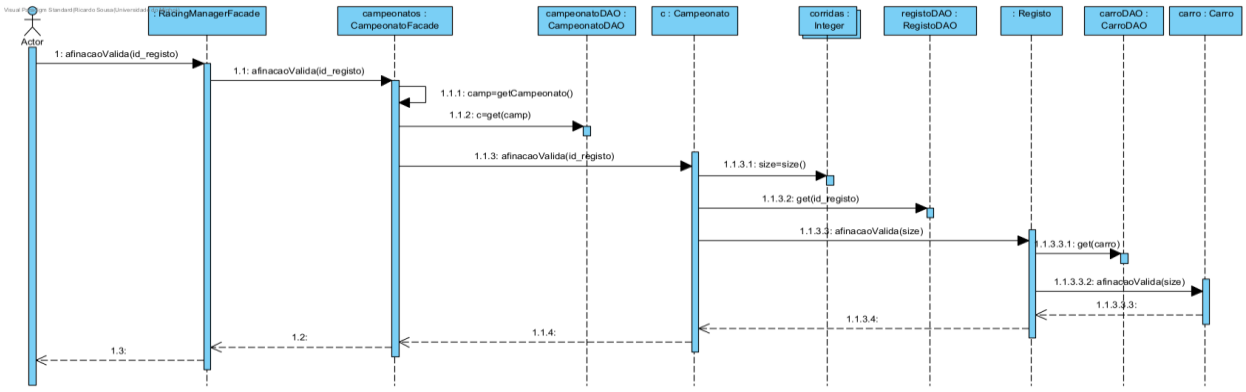
1. getProxCorrida():



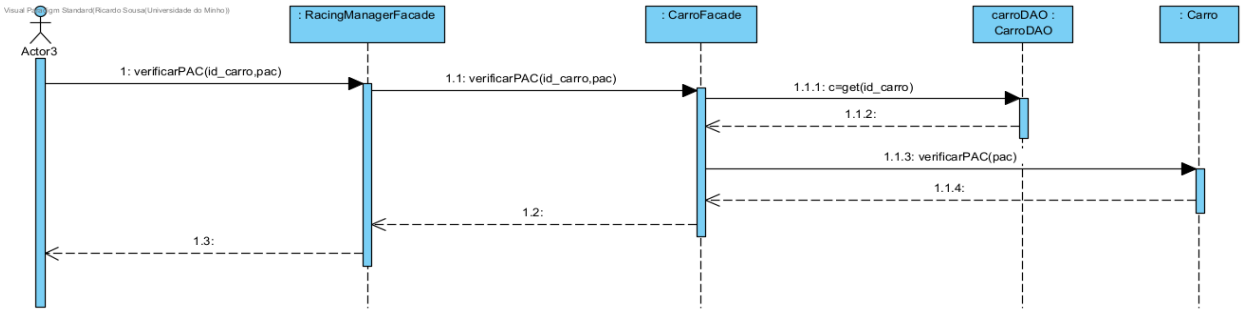
2. categoriaC1ouC2(id_carro):



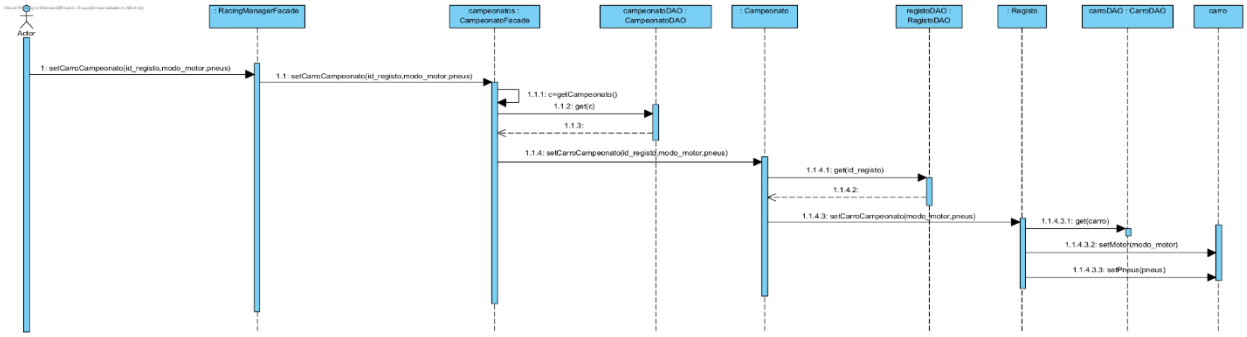
3. afinacaoValida(id_registo):



4. verificarPAC(id_carro,pac):

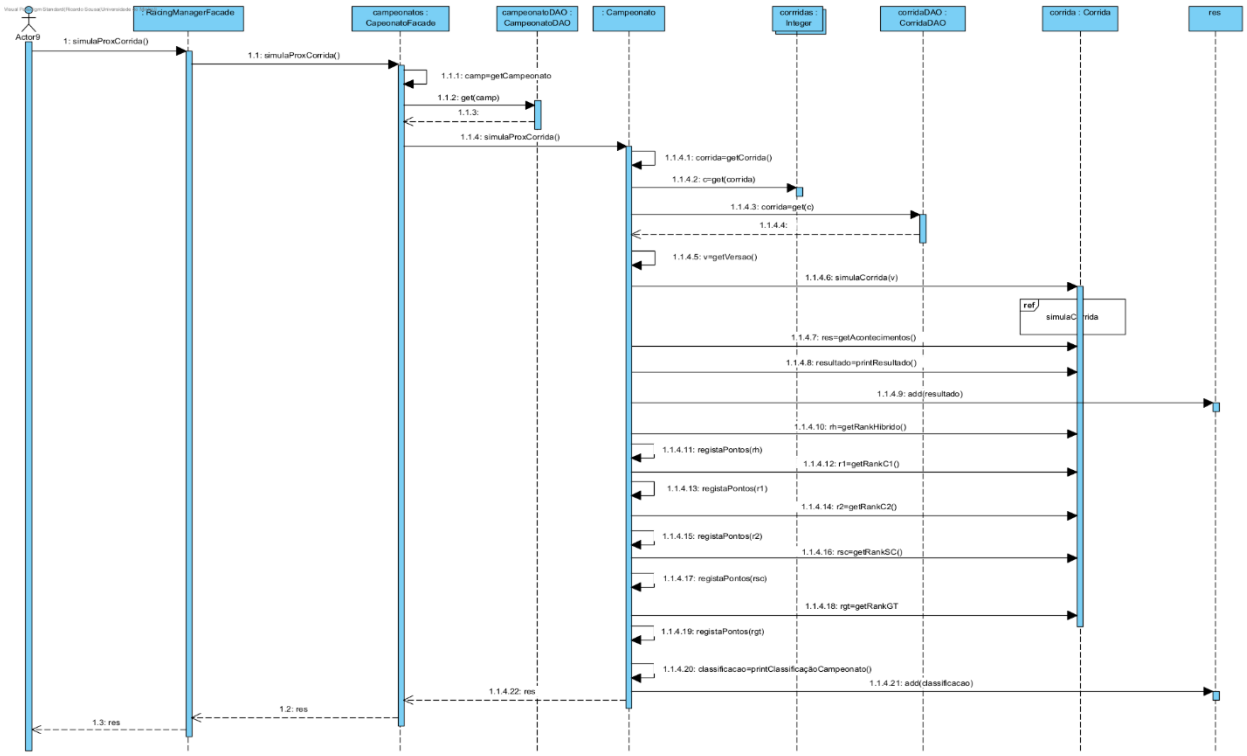


5. setCarroCampeonato(id_registo, modo_motor, pneus):

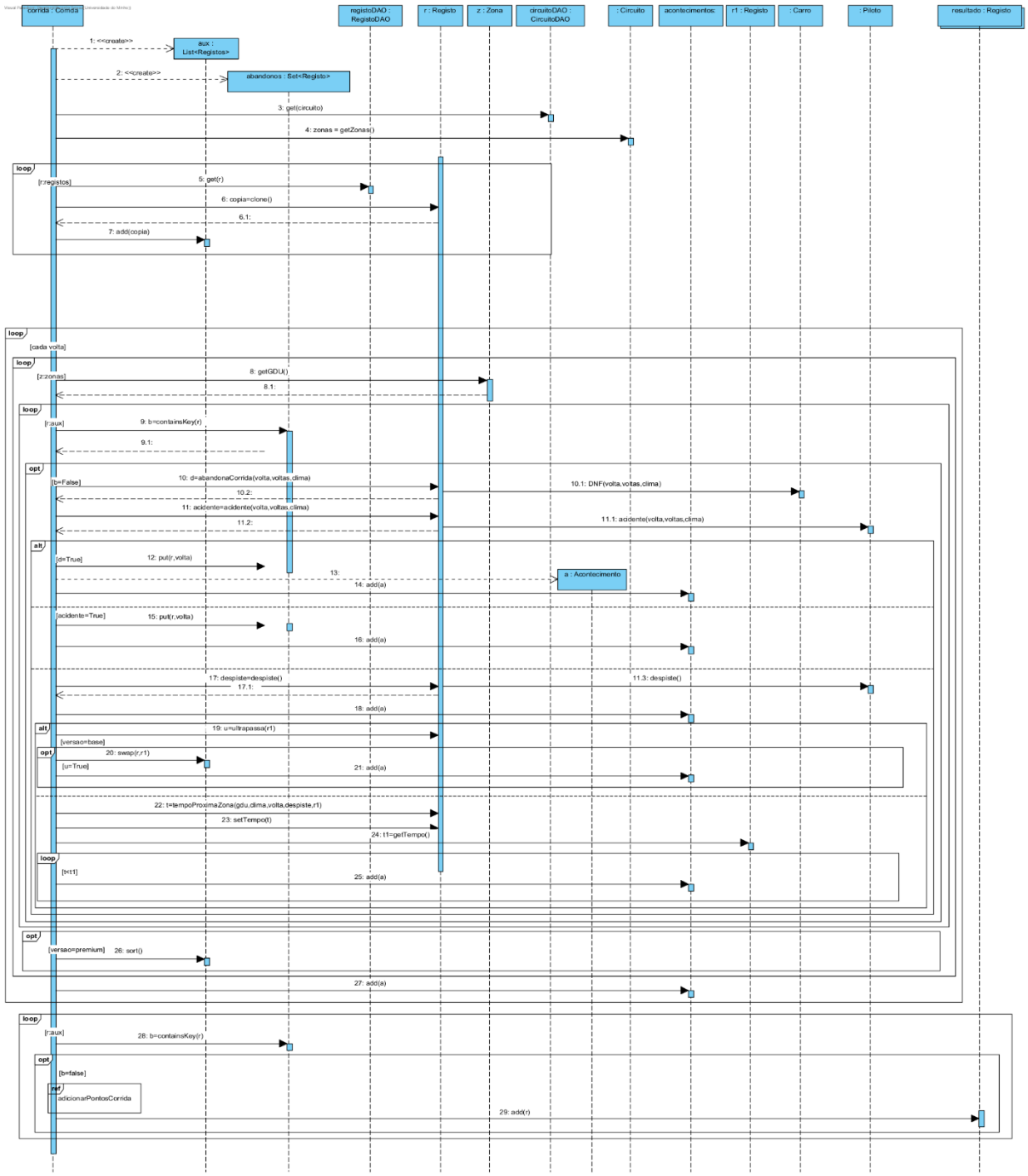


Relativamente ao use case “Simular corrida” os seguintes diagramas:

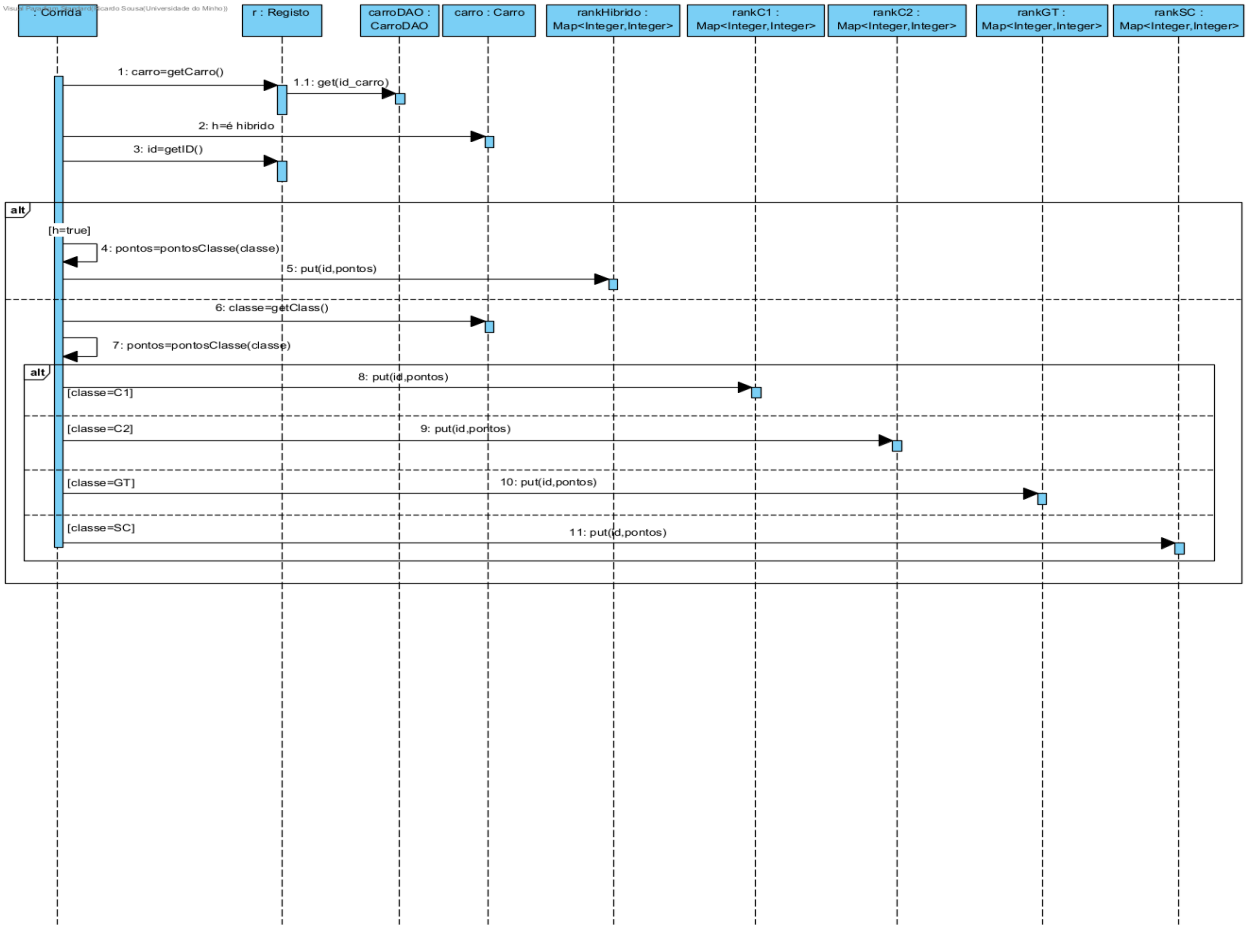
1. simulaProxCorrida():



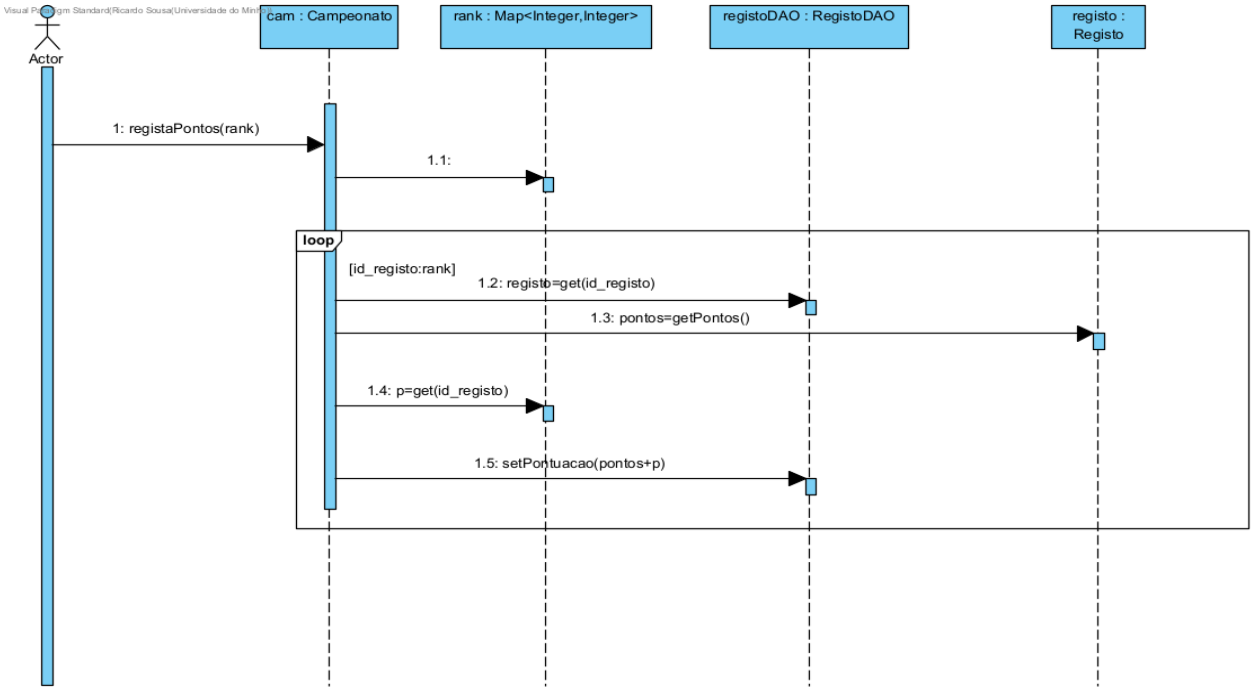
2. simulaCorrida():



3. adicionarPontosCorrida():



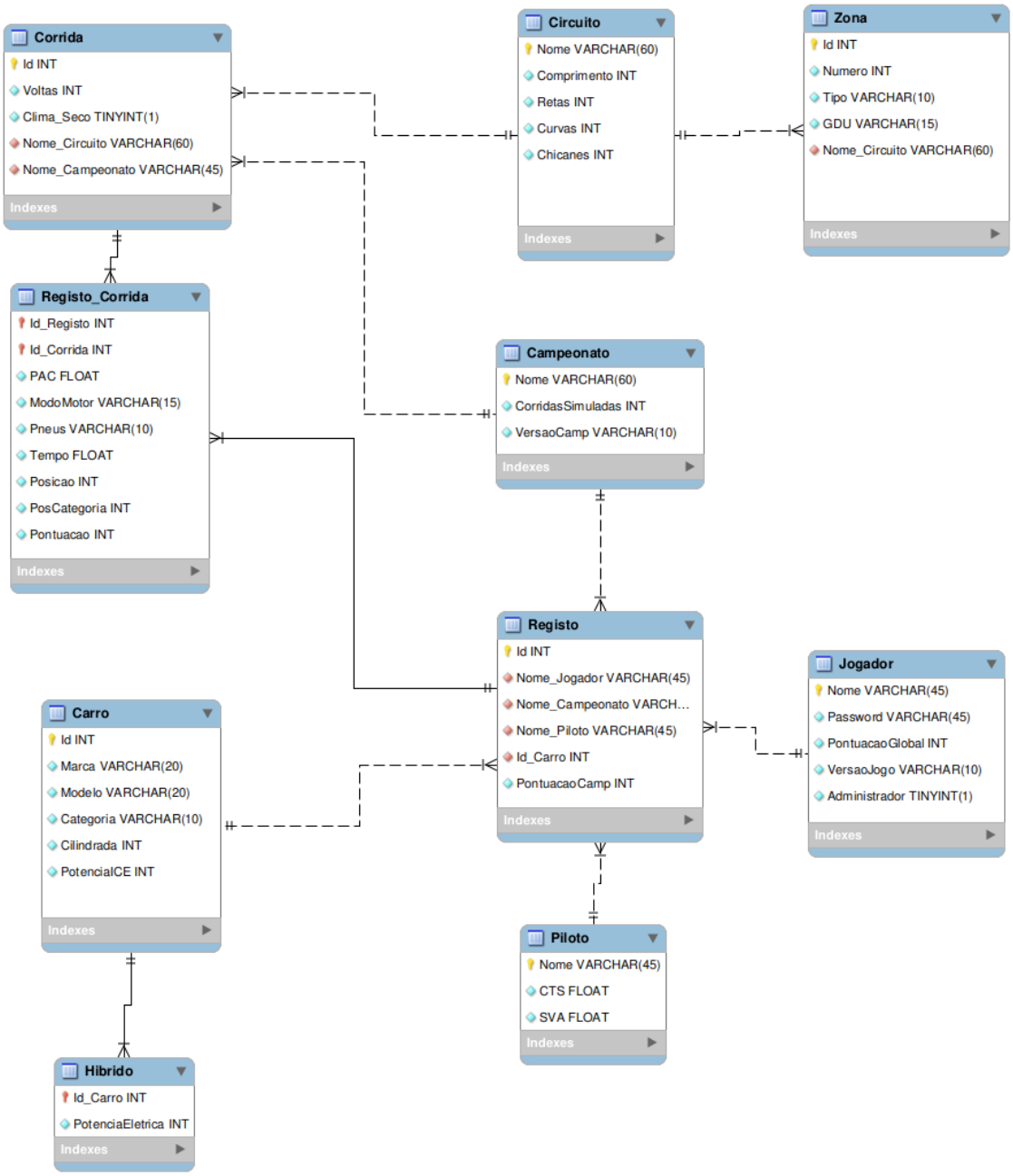
4. registaPontos(rank):



Base de Dados

Como seria de esperar, a nossa implementação necessita de uma base de dados bem concebida, dedicada a armazenar informações pertinentes para o funcionamento do nosso programa. O sistema de gestão de base de dados relacional escolhido para implementar a base de dados foi o MySQL e em anexo encontra-se o script de criação da base de dados e um script de povoamento, meramente exemplificativo, para a aplicação poder ser testada.

Segue-se então o nosso modelo lógico da base de dados.



Na implementação da nossa base de dados, começamos por criar tabelas para o armazenamento de informação sobre alguns dos objetos fundamentais do jogo, dispensando assim ao utilizar a criação dos mesmos sempre que jogasse no nosso *RacingManager. E*stas tabelas são, nomeadamente, as tabelas "Carro" e "Híbrido", onde são armazenadas informações sobre os carros já criados em sessões anteriores no *RacingManager*, as tabelas "Campeonato" e "Corrida", onde são armazenadas informações sobre cada campeonato já criado e a suas respetivas corridas, as tabelas "Circuito" e "Zona", onde são armazenadas informações sobre os

circuitos já criados e respetivas zonas de cada um e, finalmente, a tabela “Piloto”, contendo os pilotos do jogo já criados e a tabela “Jogador”, responsável por armazenar as informações relacionadas com cada utilizar do mesmo, em particular, o seu nome de utilizador e password. Ainda que não fosse estritamente necessário, definimos ainda duas tabelas extras, “Registo” e “Registo_Corrida”, de modo a armazenar as informações sobre a participação de cada jogador em cada campeonato e em cada corrida do mesmo, de forma, a haver possibilidade de, no futuro, implementar uma funcionalidade que permita rever corridas já simuladas anteriormente. Entre estas informações, destacam-se o carro utilizado pelo jogador em cada campeonato e a afinação utilizada para o mesmo em cada corrida, bem como a sua pontuação, no campeonato e em cada corrida do mesmo.

Implementação

Depois de reunidas todas as condições estruturais e comportamentais necessárias, procedemos então à implementação do projeto. Começamos pela geração automática de código através dos vários diagramas que tinham sido desenvolvidos até ao momento; desde já entendemos a utilidade dos mesmos até para aspetos estruturais, pois neste momento estavam já definidos estruturalmente todo os métodos necessários na nossa arquitetura e faltava apenas implementá-los propriamente dizendo.

Seguidamente, e sempre acompanhando de perto, principalmente, os diagramas de sequência, seguimos esta estrutura e procedemos à sua transformação em código Java, este processo necessitou também alguma lógica adicional, um olhar atento e preciso às alterações de arquitetura nomeadamente no que toca a DAOs, e também a tentativa de uma utilização cuidada de manipulação de dados e controlo de fluxo.

Conclusão e Análise Crítica

Em retrospetiva, nesta terceira e última etapa em que terminamos o nosso projeto, as principais funcionalidades e modulações do projeto já se encontravam definidas, assim, as decisões tomadas cingiram-se maioritariamente a detalhes que iriam afetar a implementação do sistema do que a detalhes relacionados com o domínio principal do problema. Posto isto, todo o processo se revelou muito enriquecedor pois deu-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 suportaram a codificação e que se revelaram fundamentais no

desenvolvimento sólido e sustentado da aplicação que garante os requisitos propostos na fase inicial.

Todo o processo deste trabalho prático mostrou-se enriquecedor pois deu-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 em todo o trabalho prático, que demos uma boa resposta na elaboração de casos de uso, na conceção de todos os diagramas e modelos representacionais necessários e no desenvolvimento do nosso código. Tudo isto contribuiu para a nossa consolidação de conceitos lecionados nas aulas práticas, e para que compreendamos, de ponto a ponto, como funciona o desenvolvimento de um sistema de software.