ARQSI 2023-2024

Síntese de representação visual da arquitetura

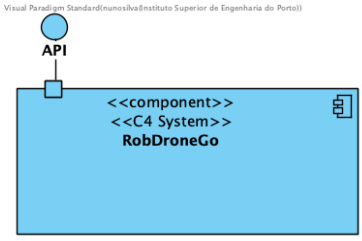
Esta representação visual do sistema em desenvolvimento:

- é um resumo do processo e diagramas desenvolvidos e debatidos nas aulas teóricas;
- é o resultado de processo iterativo e incremental não só do ponto de vista técnico-científico, mas também do pondo de vista didático/académico, ao longo do sprint A e B;
- não é completa (e.g. não representa vistas físicas, de processos ou de granularidades relevantes);
- existem outras alternativas de design arquitetural, tendo sido enfatizado ao longo das aulas a necessidade de eliciar requisitos com o cliente do projeto/product owner.

1 Sprint A

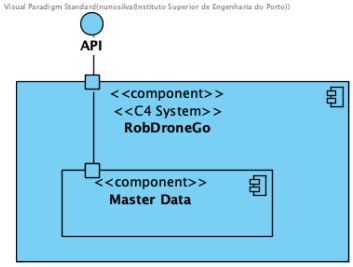
1.1 Nível 1

1.1.1 Vista Lógica

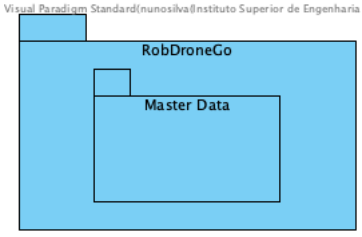


1.2 Nível 2

1.2.1 Vista Lógica

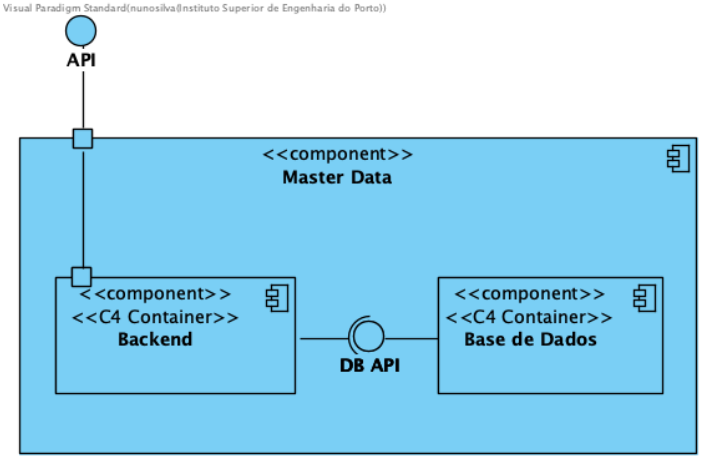


1.2.2 Vista de Implementação



1.3 Nível 3

1.3.1 Vista Lógica



1.4 Nível 4 - Backend

1.4.1 Vista Lógica

Diagrama genérico de adoção do padrão Onion.

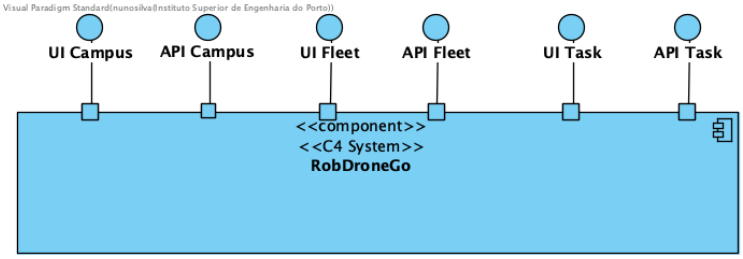
2 Sprint B

O que foi feito de Módulo de Gestão de Campus/Frota/Tarefas/Utilizadores, mencionados no enunciado?

2.1 Nível 1

2.1.1 Vista Lógica

NB: existe motivo evidente para repensar esta vista depois da revisão da vista lógica de nível 2, pelo que o diagrama seguinte é muito mais facilmente compreensível e justificado depois de desenhado a vista lógica de nível 2.

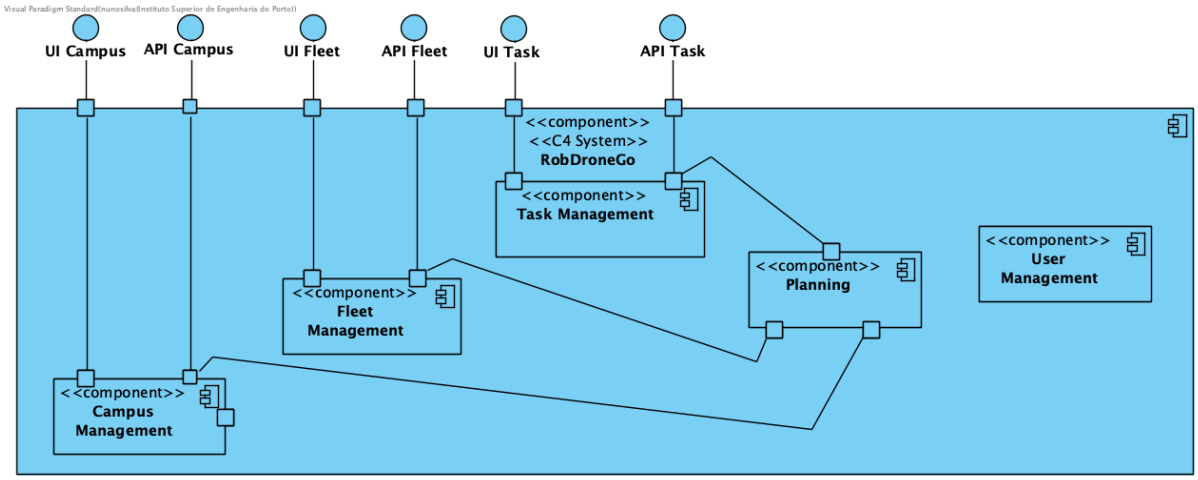


2.2 Nível 2

2.2.1 Vista Lógica

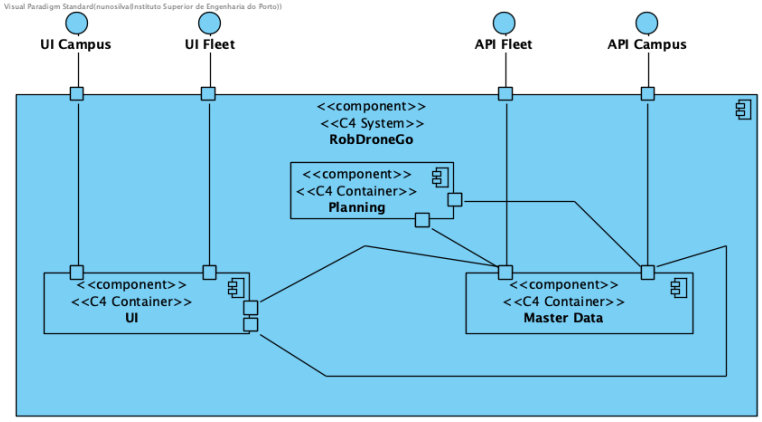
Representa os módulos lógicos (que o cliente menciona), mas não na perspetiva de que são (ou que têm de ser) módulos de software. Analogia: são como conceitos de negócio (e não como “pure fabrications”).

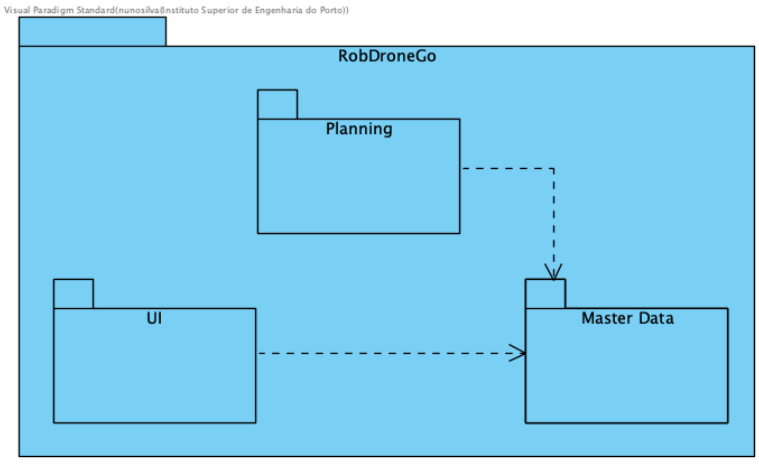
Este raciocínio podia ter sido adotado no sprint A, mas não o foi por motivos didáticos/académicos.



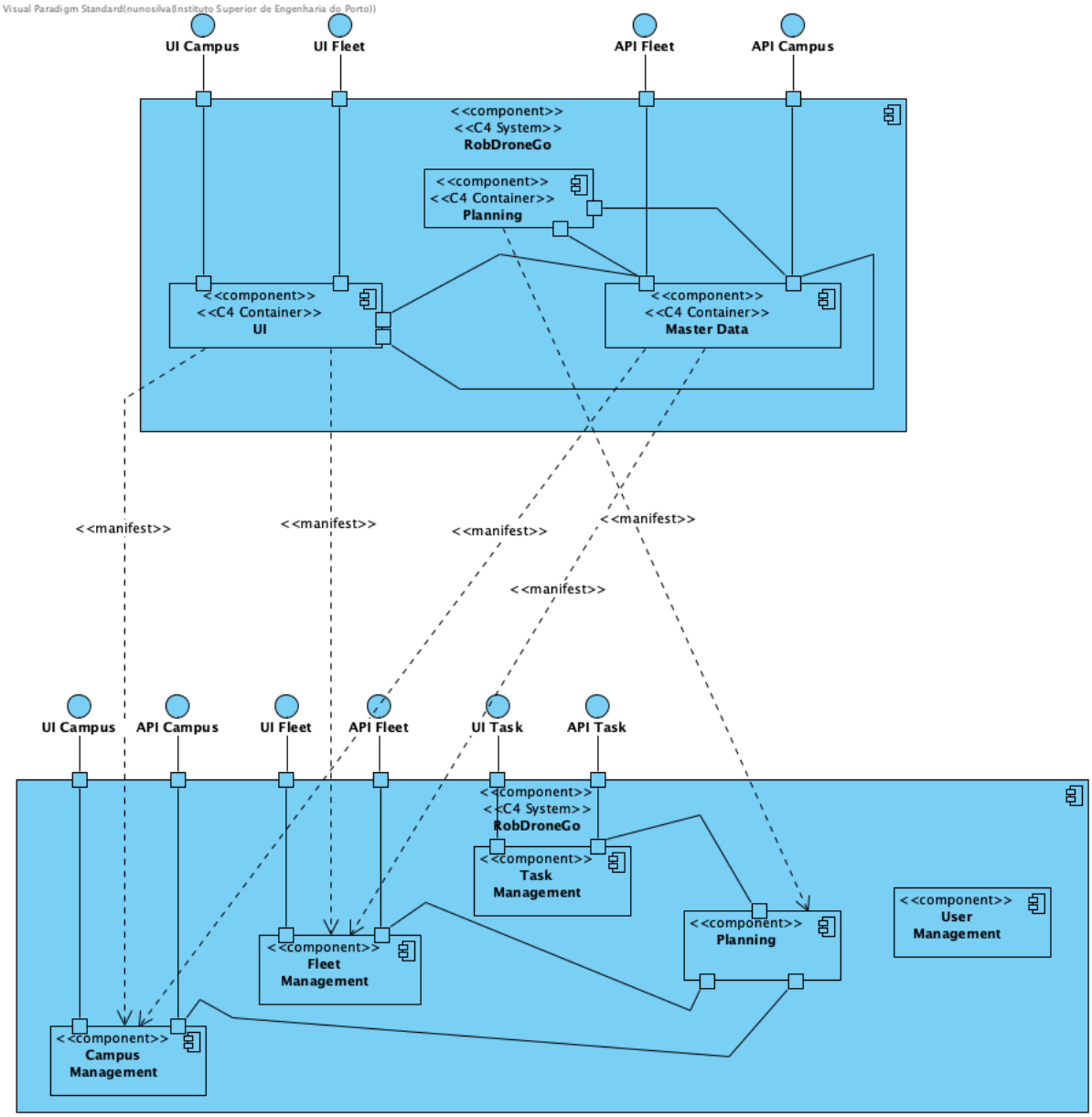
A partir desta vista deste nível, regressa-se ao nível 1 para representar as várias API e UI, que não teríamos conseguido identificar (ou teria sido muito mais difícil e especulativo) sem esta granularidade.

2.2.2 Vista de Implementação





2.2.3 Mapeamento entre Vista Lógica e Vista de Implementação



2.3 Nível 3 - Master Data

Não há alterações à arquitetura de Master Data.

2.4 Nível 4 - Backend

Não há alterações à arquitetura de Master Data.

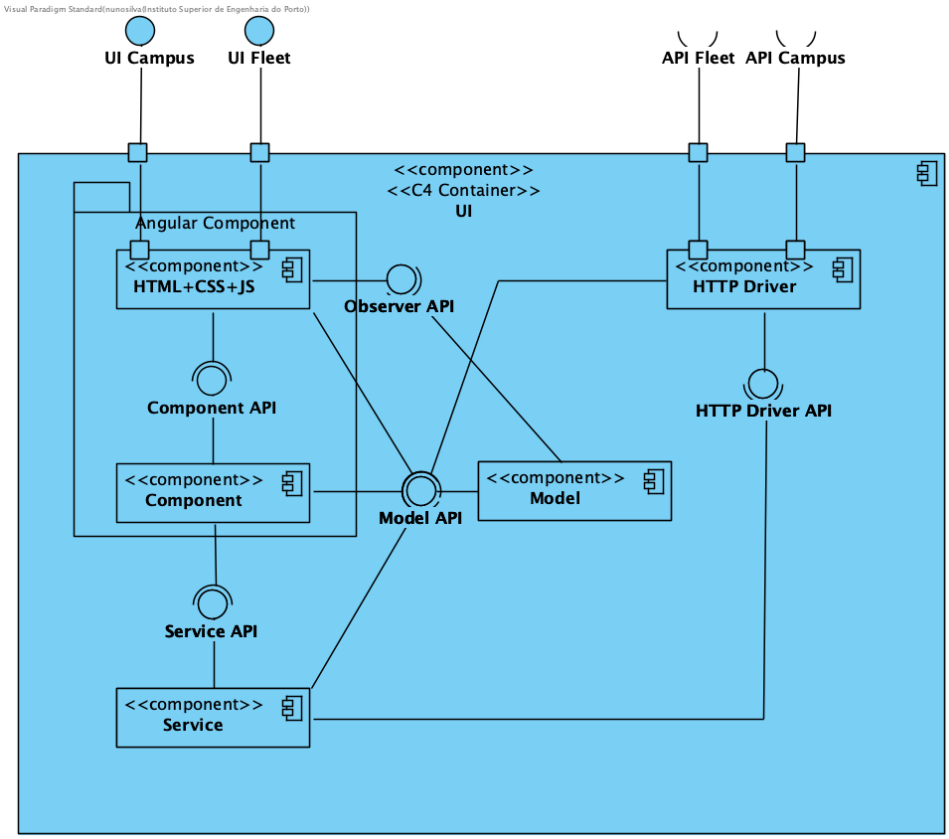
2.5 Nível 3 – Planeamento

Todas as cláusulas têm as mesmas responsabilidades? Se não, quais as diferentes responsabilidades?

2.6 Nível 3 - UI

2.6.1 Vista de Implementação

Descreve-se nesta vista a arquitetura duma aplicação simples desenvolvida em Angular, semelhante àquela resultante do tutorial “Tour of Heroes”.



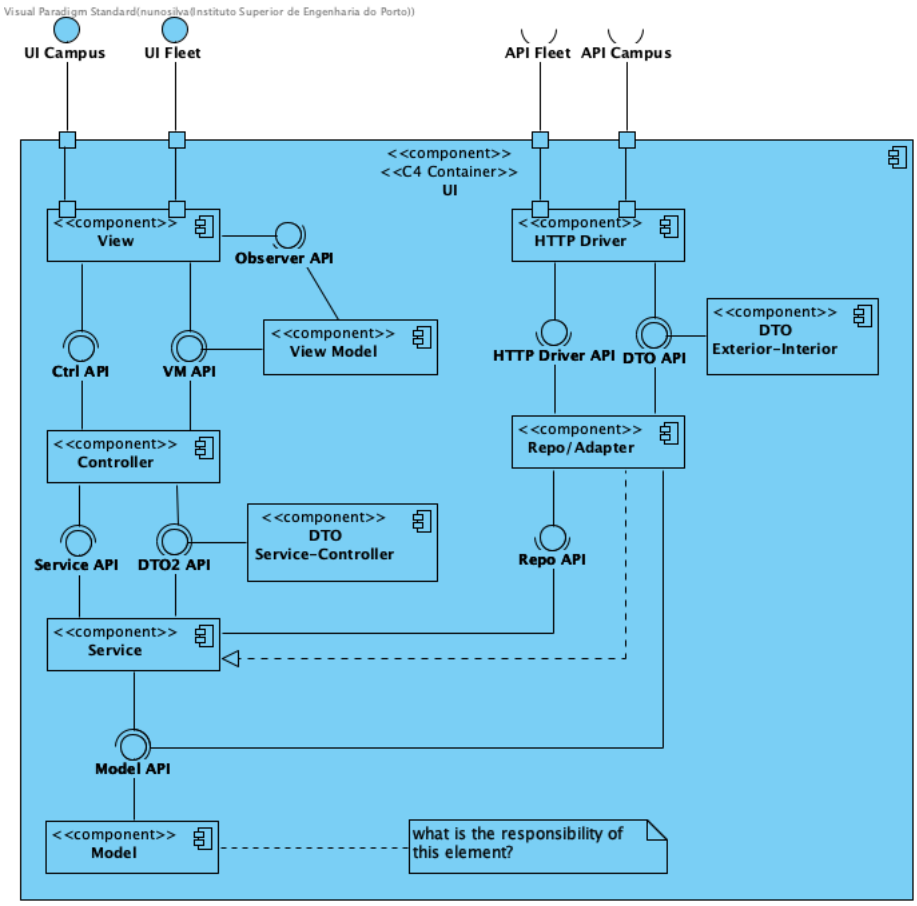
Notas:

- Esta arquitetura adota algum padrão? Sim, o MVC (cf. documentação do Framework).
- O componente Model tem alto acoplamento. De facto, está acoplado as todas as restantes partes/componentes do sistema, o que sugere que tem demasiadas responsabilidades (baixa coesão). Porque não segregar as suas responsabilidades?
- Muitas vezes, o componente (controller) tem muitas responsabilidades, ao invés de ser algo muito orientado ao processamento de entradas e geração de saídas (como deve acontecer e está espelhado no exemplo de projeto backend com a adoção de Onion). Algumas dessas responsabilidades deviam ser atribuídas ao serviço.
- No mesmo sentido, muitas das vezes, os serviços funcionam como adaptadores para acesso ao exterior, ou conjunto de funções úteis a vários componentes (“utils”), e não como representação de processo de negócio. Têm, portanto, responsabilidade mais consentâneas com adaptadores e biblioteca de funções comuns, do que com serviços.

2.6.2 Vista Lógica

Alguns princípios adotados:

- Adoção (explícita) de MVC
- Adoção de SRP: Model(o) é segregado em vários modelos parciais e especializados)
- Aplicação de responsabilidades em conformidade com a semântica habitual: Service como serviço e não como adaptador.
- Embora o objetivo não seja adotar o padrão arquitetural Onion, de facto a arquitetura descrita no diagrama seguinte é muito semelhante a uma arquitetura com tal padrão.


Visual Paradigm Standard(nunosilva(Instituto Superior de Engenharia do Porto))
UI Campus
UI Fleet
API Fleet
API Campus
<<component>>
<<C4 Container>>
UI
<<component>>
View
Observer API
<<component>>
HTTP Driver
<<component>>
View Model
Ctrl API
VM API
HTTP Driver API
DTO API
<<component>>
DTO
Exterior-Interior
<<component>>
Controller
<<component>>
Repo/Adapter
<<component>>
DTO
Service-Controller
Service API
DTO2 API
Repo API
<<component>>
Service
Model API
<<component>>
Model
what is the responsibility of this element?