FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Representação digital e gestão de equipamentos numa plataforma IIoT

João António Gamboa Correia

Mestrado em Engenharia Eletrotécnica e de Computadores

Orientador: António Manuel Lucas Soares

23 de julho de 2022

Resumo

O crescimento da automação nos sistemas de produção requer a digitalização em massa dos processos e equipamentos industriais. Criando uma cópia virtual (*Digital Twin* (DT)) dos componentes físicos de um determinado sistema de produção, é possível emular digitalmente o seu comportamento de forma a prever, simular e melhorar vários fatores ao longo dos processos de produção.

De modo a implementar estes DTs, é necessário primeiro definir métodos de modelação, para representar digitalmente os equipamentos físicos. Depois de definido um modelo, é também importante criar interfaces capazes de registar e gerir estes equipamentos numa plataforma digital.

A implementação de um destes modelos para representar digitalmente equipamentos, bem como o desenvolvimento de uma REST API que os permita registar e gerir, são os dois objetivos principais desta dissertação.

O documento começa por fornecer uma revisão da literatura sobre a *Industrial Internet of Things* (IIoT), DTs, representação digital de ativos, e APIs. Depois, é analisado o trabalho realizado ao longo do período da dissertação, que inclui os detalhes sobre o modelo utilizado e a API desenvolvida. Por fim, são expostas as conclusões da dissertação, e é feita uma pequena reflexão sobre potencial trabalho futuro.

Abstract

The growth of automation in production systems requires the mass digitization of processes and industrial equipment. By creating a virtual copy (Digital Twin (DT)) of the physical components in a production system, it's possible to digitally emulate it's behaviour to predict, simulate and improve several factors along the production process.

To implements these DTs, it's necessary to start by defining modelling methods to digitally represent physical assets. Once that is done, it's also important to create interfaces capable of registering and manage these assets on a digital platform.

The implementation of one of these models to digitally represent assets, as well as the development of a REST API to register and manage said assets, are the two main objectives of this dissertation.

This document starts with a literature review on the Industrial Internet of Things (IIoT), DTs, digital representation of physical assets, and APIs. It continues by providing details on the work developed during the dissertation period, which includes insights on the model used and the developed API. Lastly, some final conclusions are shared and a reflection is made about potential future work.

Agradecimentos

Um especial agradecimento ao Bruno Cunha e ao Luís Lima, que tanto me ensinaram ao longo desta dissertação, pelo apoio, orientação, paciência e disponibilidade, que tornaram esta minha primeira experiência numa equipa de projetos, tão recompensadora.

Ao Paulo Soares, por me dar a oportunidade de aceitar esta proposta de dissertação que me abriu as portas para novas experiências e para o mundo de trabalho fora da faculdade.

Ao meu orientador António Lucas Soares, pela ajuda e orientação ao longo do processo mais difícil de toda a dissertação, a escrita deste documento!

A todos os membros do INESC TEC pela simpatia e disponibilidade que sempre mostraram para me ajudar com qualquer dificuldade.

Aos meus amigos, por tornarem estes 5 anos na FEUP melhores do que eu poderia alguma vez imaginar, e por sempre estarem ao meu lado para me apoiar. Todos os momentos e experiências que partilhámos ao longo dos anos e continuaremos a partilhar, foram e são a maior motivação para enfrentar todos os desafios da vida.

Por fim, aos meus pais, irmão e restante família, por sempre acreditarem em mim, por todo o carinho e apoio, e por sempre me terem proporcionado todas as condições e liberdade para alcançar os meus objetivos.

João Correia

"Dream or nightmare, we have to live our experience as it is, and we have to live it awake. We live in world which is penetrated through and through by science and which is both whole and real. We cannot turn it into a game simply by taking sides"

Jacob Bronowski

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

AAS	Asset Administration Shell
API	Application Programming Interface
APS	Advanced Planning Systems
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CNC	Computer Numerical Control
CPS	Cyber-Physical System
CRUD	Create-Read-Update-Delete
DAL	Data Access Layer
DM	Digital Model
DTO	Data Transfer Object
ERP	Enterprise Resource Planning Systems
I4.0	Industry 4.0
ID	Identification
IED	Intelligent Electronic Devices
IIoT	Industrial Internet of Things
IoT	Internet of Things
JSON	JavaScript Object Notation
MES	Manufacturing Execution System
OPC UA	OPC Unified Architecture
PLM	Product Lifecycle Management Systems
RAMI 4.0	Reference Architectural Model Industrie 4.0
REST	Representational State Transfer
SADCoPQ	Sistema de Apoio à Decisão no Controlo Preditivo da Qualidade
TD	Web of Things Thing Description
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WoT	Web of Things

Capítulo 1

Introdução

Esta dissertação foi realizada no âmbito de final do ciclo de estudos do Mestrado em Engenharia Eletrotécnica e de Computadores, da Faculdade de Engenharia da Universidade do Porto. O trabalho em questão decorreu no período entre Março e Junho de 2022 num ambiente empresarial, ao acolhimento do INESC TEC e da Vanguarda, empresa de software que fornece soluções de gestão e organização empresarial.

1.1 Enquadramento e motivação

Desde o início da quarta revolução industrial, denominada de Indústria 4.0 (I4.0), os sistemas de produção têm evoluído de modo a usufruir da Internet e da conectividade que esta fornece, bem como a enorme capacidade de armazenamento da nuvem. O que antes eram processos puramente físicos, são agora processos cyber-físicos (*Cyber-Physical Systems* (CPS)). Criando uma cópia virtual (*Digital Twin* (DT)) dos componentes físicos num determinado sistema, é possível criar uma versão inteligente do mesmo, capaz de transmitir e armazenar quantidades avultadas de informação através da *Internet of Things* (IoT), o que ajuda as empresas a otimizar as suas linhas de produção, tornando os processos mais eficientes, flexíveis e previsíveis [1].

Atualmente, no mercado extremamente competitivo em que o mundo se encontra, existe uma motivação crescente em desenvolver DTs que apoiem os sistemas de produção. Entre 2017 e 2019, o conceito de DT foi considerado uma das cinco tendências tecnológicas mais importantes pela Gartner, uma das maiores empresas de investigação e consultoria nos Estados Unidos. Em 2020, *hyperautomation*, o conceito de automatizar ao máximo os processos dentro de uma organização, foi considerada a tendência tecnológica mais importante do ano, estando diretamente relacionada com a indústria 4.0 e DTs, tendo em conta que a automatização a esta escala geralmente resulta na criação de DTs para toda a organização [2].

No contexto do projeto abordado durante esta dissertação, uma empresa na indústria metalomecânica pretende implementar soluções de gestão e apoio à decisão para o controlo da qualidade

com capacidades de análise de dados preditiva, de modo a melhorar a qualidade dos produtos, diminuindo significativamente o número de defeitos, ou não-conformidades, nas peças manufaturadas. Uma das soluções passa por, no contexto da *Industrial Internet of Things* (IIoT), desenvolver uma plataforma responsável pela recolha e gestão da informação proveniente dos equipamentos industriais do chão de fábrica. Será assim necessário desenvolver DTs para estes dispositivos, de modo a representá-los digitalmente dentro da plataforma.

1.2 Caracterização do problema

De forma a representar os equipamentos digitalmente, é necessário criar modelos que os descrevam de forma correta e detalhada. Atualmente, embora exista uma série de padrões, recomendados por organizações conceituadas, com este objetivo [3, 4, 5], não existe ainda consenso internacional num padrão genérico para modelar e implementar DTs [6, 1].

Um desafio adicional com a modelação baseada nestes padrões, é a complexidade inerente que os acompanha. Visto que a interoperabilidade é, indubitavelmente, uma das características mais importantes da indústria 4.0 [7, 8], estes padrões são forçados a seguir uma série de requisitos e normas para garantirem a sua interoperabilidade e responderem a todas as necessidades levantadas por sistemas I4.0. Consequentemente, isto gera uma série de dificuldades para pequenas empresas com pequenas equipas de desenvolvimento, visto que frequentemente é necessário o trabalho conjunto de muitas empresas, engenheiros, diretores, operadores e fornecedores de modo a entregar uma solução final unificada por um único padrão [9]. Para além disso, estes padrões regularmente impõem a utilização de tecnologias específicas, como OPC UA e AutomationML [10], que frequentemente estão fora do leque de tecnologias com as quais uma pequena equipa de desenvolvimento está habituada a trabalhar.

Devido a estes desafios, no presente projeto optou-se pelo desenvolvimento de uma solução proprietária, não disponível no mercado, para representar digitalmente equipamentos industriais.

1.3 Objetivos

Com esta dissertação pretende-se implementar um modelo para representar digitalmente equipamentos industriais. Depois de caracterizado o modelo, será também necessário criar interfaces e serviços que facilitem as interações entre os utilizadores da plataforma e os equipamentos registados na mesma. Para isso, foram definidos os seguintes objetivos específicos:

1. A definição de um modelo que represente detalhadamente os equipamentos industriais que serão registados na plataforma IIoT;

2. O desenvolvimento, testagem e implementação de uma API (*Application Programming Interface*), que disponibilizará um conjunto de interfaces e serviços, baseados em métodos

CRUD (*Create-Read-Update-Delete*) que irão permitir o registo e gestão dos equipamentos.

1.4 Estrutura do documento

Este documento está dividido em 4 capítulos:

- **Capítulo 1:**
  O primeiro capítulo destina-se à introdução da dissertação, onde é feito um enquadramento do tema, a caracterização do problema, e são definidos os objetivos esperados ao longo do período de trabalho.

- **Capítulo 2:**
  No segundo capítulo é feita uma revisão bibliográfica sobre IIoT, representação digital de ativos, e APIs. Estes temas são essenciais para formar uma base teórica no contexto do trabalho a desenvolver.

- **Capítulo 3:**
  O terceiro capítulo destina-se à descrição do trabalho realizado durante o período da dissertação. Serão inicialmente analisados alguns detalhes sobre o projeto SADCoPQ e depois será feita uma apresentação sobre o módulo da plataforma IIoT que será desenvolvido.

  De seguida, é feita uma análise ao modelo de informação utilizado para descrever os ativos e representá-los digitalmente na plataforma IIoT. É detalhada inicialmente a arquitetura do modelo e depois é feita uma análise a um dos equipamentos do chão de fábrica, bem como os dados provenientes do mesmo, seguido de uma demonstração de aplicação do modelo para o respetivo equipamento.

  Por fim, é feita uma descrição da arquitetura e implementação da REST API.

- **Capítulo 4:**
  O quarto capítulo destina-se a expor as conclusões do projeto realizado, seguido de algumas considerações relevantes sobre potencial trabalho futuro.

Capítulo 2

Estado da arte

O presente capítulo pretende apresentar em mais detalhe conceitos diretamente relacionados com a proposta de dissertação que irão proporcionar um fundamento teórico essencial para o desenvolvimento do trabalho e alcance dos objetivos esperados. Mais concretamente, esta análise pretende expor algumas das ideias exploradas pela comunidade académica no âmbito de representação digital de ativos e desenvolvimento de APIs, fornecendo, ao longo do caminho, informação relevante que contextualiza o tema na atual indústria.

2.1 Industrial Internet of Things

A indústria 4.0 é caracterizada por um conjunto de tecnologias e conceitos que formam o paradigma atual dos ecossistemas industriais [11, 12]. Fábricas inteligentes e modulares constituem a indústria 4.0, onde CPSs monitorizam e influenciam processos físicos em função da informação recolhida por sensores em tempo real [13], através da IIoT.

IIoT é um subconjunto de IoT, aplicado a ambientes industriais. Hugh Boyes et al. [13] definem IIoT como um sistema composto por equipamentos inteligentes, CPSs, tecnologias de informação e plataformas de computação em nuvem e/ou de borda, ligados entre si, que permitem a gestão e análise de processos, produtos e serviços, dentro do ecossistema industrial, de modo a otimizar o valor de produção. Este valor pode incluir, entre outros, a melhoria do produto e da entrega de serviços, aumento de produtividade, e redução do consumo de energia, custos de mão de obra e custos de produção.

Esta definição de IIoT está alinhada com a indústria 4.0, e é um pilar fundamental na sua caracterização.

2.1.1 Arquitetura de uma plataforma IIoT

Depois de uma definição de IIoT estar estabelecida, uma análise pode ser feita à arquitetura típica de uma plataforma IIoT.

A figura 2.1 representa as diferentes camadas que constituem uma plataforma IIoT, bem como os seus componentes e funções.

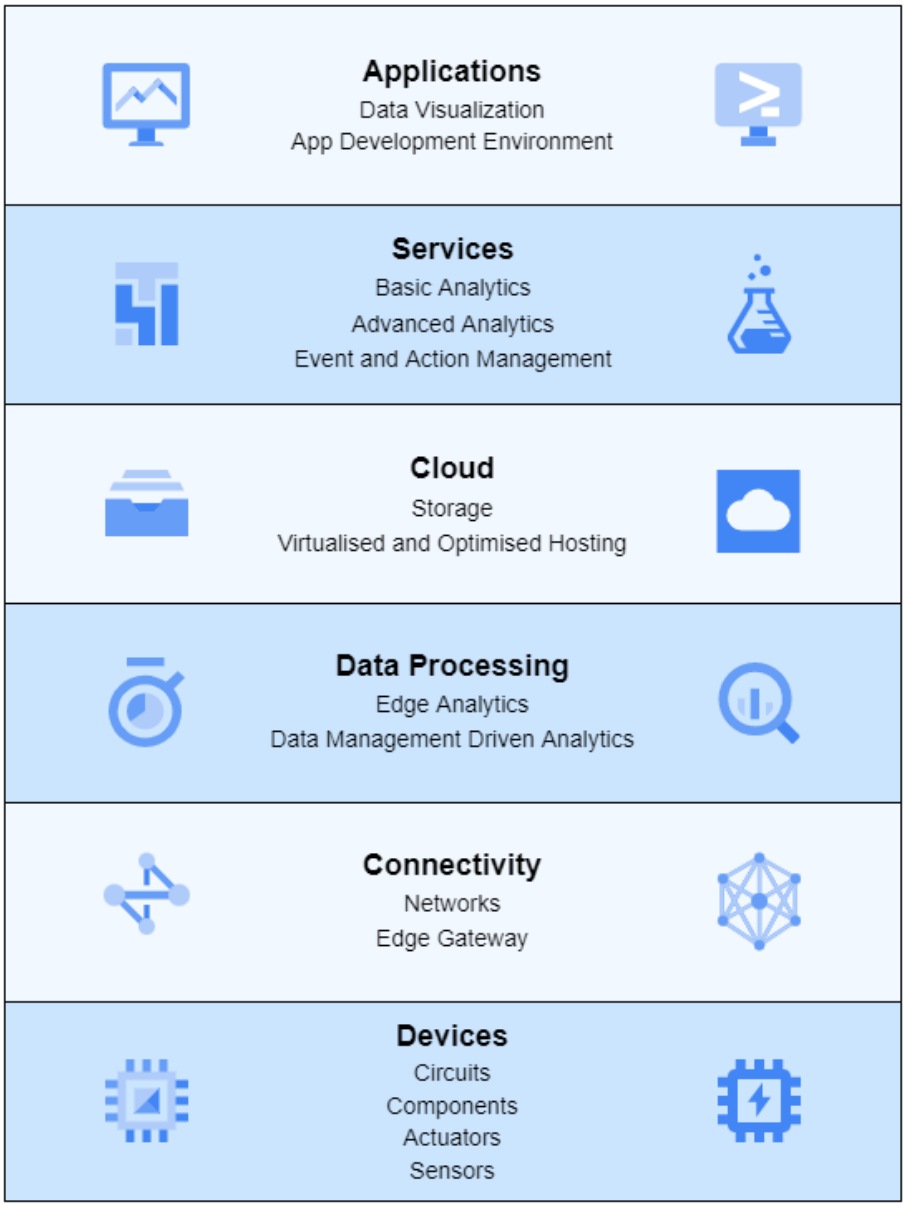

Figura 2.1: Arquitetura de uma plataforma IIoT. Inspirado por [14]

1. Equipamentos (*Devices*): Esta camada representa todas as *"things"* (equipamentos) que constituem uma fábrica inteligente. Desde máquinas industriais de grande volume, a pequenos sensores.

2. Conectividade (*Connectivity*): Toda a rede de comunicação de uma fábrica pertence a esta camada. Os equipamentos transmitem constantemente informação para o *edge gateway*, que

serve como um ponto central onde toda a informação é agrupada e enviada posteriormente para processamento.

3. Processamento de informação (*Data Processing*): Nesta camada são usados métodos analíticos de modo a gerir, organizar e até transformar informação antes de a guardar numa base de dados. Esta camada corre em tempo real, logo, não são feitas aqui análises com um peso computacional muito elevado. Este tipo de processamento consiste em tarefas ligeiras como a etiquetagem de dados. Se transformações forem feitas, estas devem ser simples, como conversão de unidades. Métodos analíticos mais avançados são normalmente realizados posteriormente, e não em tempo real, com a informação disponível nas bases de dados.

4. Nuvem (*Cloud*): As tecnologias de nuvem são maioritariamente utilizadas para o armazenamento dos dados, no entanto, são frequentemente utilizadas para outras funções, como por exemplo, hospedarem a plataforma numa máquina virtual.

5. Serviços (*Services*): A camada de serviços é onde a maioria da computação é realizada. Todos os eventos de baixo nível detetados por sensores e previamente colecionados e guardados, são aqui mapeados para eventos e ações de alto nível. Métodos analíticos complexos são usados para extrair o máximo de informação útil, que pode ser mais tarde interpretada por humanos ou tecnologias de inteligência artificial, de modo a aumentar o valor ao longo do processo de produção.

6. Aplicações (*Applications*): Esta última camada é constituída por todas as aplicações e interfaces que ajudam os utilizadores da plataforma a interagir com o sistema e a visualizar os dados provenientes das outras camadas.

2.1.2 Digital Twin

O conceito de DT foi inicialmente introduzido por Michael Grieves no seu curso sobre gestão de ciclo de vida de produtos, na Universidade de Michigan, em 2002, embora o termo DT só tenha sido efetivamente ligado a este conceito em 2011 [15]. A figura 2.2, utilizada na apresentação de Grieves, expõe os elementos principais que constituem um DT: o espaço real e o espaço virtual, a transferência de informação entre os dois, e também o conceito de vários sub-espaços virtuais.

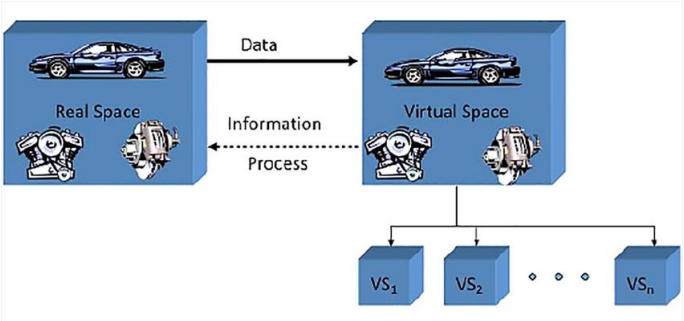

Figura 2.2: Conceito de DT. Fonte: [15]

Desde então, embora a utilização de DTs tenha sido significativamente alargada e seja agora um pilar importante na atual indústria 4.0, o conceito manteve-se essencialmente o mesmo. Uma definição mais detalhada e recente de DT é dada por Michael Grieves e John Vickers em [16], que o caracterizam como um conjunto de informação virtual que descreve um produto físico na sua totalidade. Idealmente, qualquer informação que possa ser obtida inspecionando o produto físico pode ser obtida a partir do seu DT.

Esta definição, no entanto, é bastante genérica, podendo ser aplicada a qualquer ramo da engenharia. Em [17], é dada uma definição de DT virada para sistemas industriais de produção, onde um DT é definido como uma representação virtual de um componente físico e caracterizado pela sincronização entre o sistema virtual e o sistema real, através da informação sensorial recolhida, equipamentos inteligentes, modelos matemáticos e processamento de dados em tempo real.

Considerando o ciclo de vida de um DT (figura 2.3), a modelação é o foco principal no contexto desta dissertação. Ao modelo de um DT dá-se o nome de *Digital Model* (DM). Um DM é uma representação digital de objeto físico e não efetua nenhuma troca de informação em tempo real entre o objeto digital e o objeto físico [18], ao contrário de um DT, que na sua totalidade, é caracterizado pela sincronização e troca de informação em tempo real entre o sistema virtual e o sistema real, como referido anteriormente. A representação digital descreve, de uma forma compreensiva, as características, comportamentos e funcionalidades do objeto real.

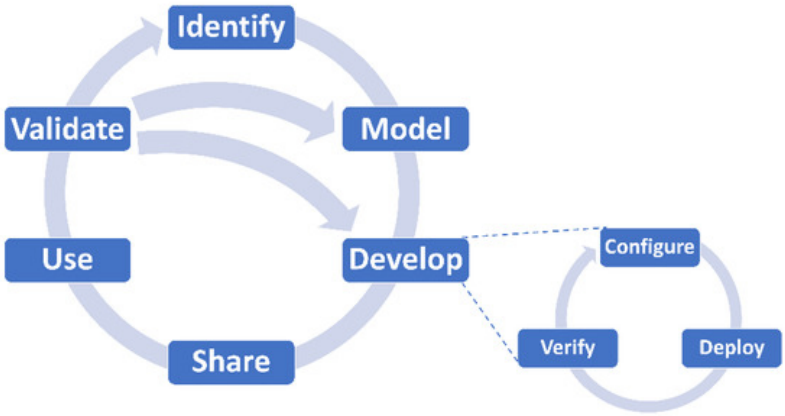

Figura 2.3: Ciclo de vida de um DT. Fonte: [19]

2.2 Modelação e representação digital de ativos

Ao longo das últimas décadas, com o crescimento da indústria 4.0, foi adotada uma miríade de novos termos e conceitos, que, antigamente ou não existiam, ou não eram relevantes. No entanto, não só para a unificação entre conceitos da indústria 4.0, mas também para a escrita desta dissertação, é importante existir uma definição bem clara dos termos que serão utilizados daqui para a frente.

Atualmente, ainda existe alguma falta de estandardização dos termos utilizados cientificamente dentro da indústria 4.0. Constantin Wagner et. al. [20] procuraram melhorar as definições utilizadas dentro do contexto de CPSs, que serão abordados ao longo desta dissertação, e estão no núcleo da indústria 4.0.

É também importante considerar as barreiras linguísticas existentes. A definição de indústria 4.0, bem como muitos dos seus conceitos principais, foram inicialmente formados na Alemanha, depois utilizados em inglês, e agora, neste documento, utilizados em português.

Posto isto, as seguintes definições serão utilizadas ao longo deste documento:

- **Representação digital:**
  - Informação que representa as características, comportamentos e funcionalidades de uma entidade e/ou objeto.
- **Modelo (de informação):**
  - Usado para representar digitalmente um ativo;
  - Constitui as propriedades de um ativo (p. ex. identificador, nome);
  - Referido também frequentemente como "Modelo de Dados".
- **Ativo:**
  - É uma entidade com valor. Esta pode ser tangível (p. ex. um equipamento), ou não tangível (p. ex. um software);
  - Descrito por um modelo de informação.

De seguida, serão apresentados alguns dos padrões mais utilizados na indústria para criar modelos que descrevam equipamentos industriais.

2.2.1 Asset Administration Shell

O *Asset Administration Shell* (AAS) é, atualmente, um dos padrões mais interoperáveis utilizados na indústria, o que o torna uma das opções mais apelativas dentro da indústria 4.0 para descrever ativos. Este padrão foi também desenvolvido pelo mesmo grupo alemão que liderou de forma significativa os avanços feitos na indústria 4.0, desde as suas origens, e que formou a Plattform Industrie 4.0.

O AAS é uma representação digital de um ativo, essencial para obter a interoperabilidade entre aplicações que gerem os sistemas de produção [3]. Um AAS contém os modelos de um componente I4.0 e descreve as suas funcionalidades técnicas, que são expostas posteriormente através de uma API.

A figura 2.4 ilustra um componente que segue as orientações da indústria 4.0. Este é constituído pelo ativo em si, representativo da parte física e tangível do componente, e pelo *Administration Shell*, sendo este a representação lógica do ativo.

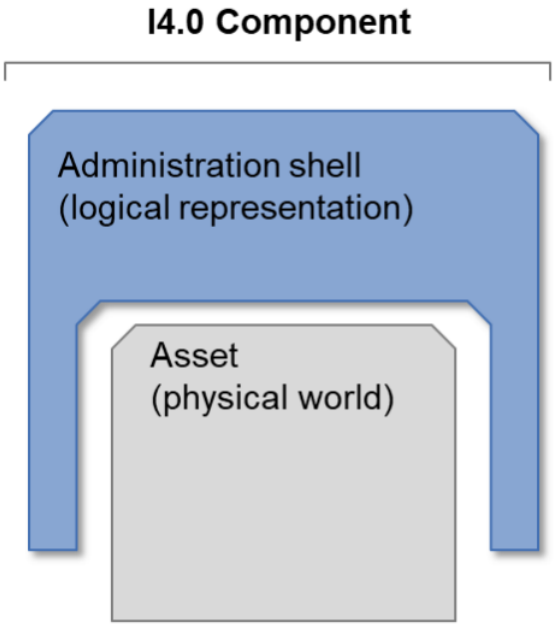

Figura 2.4: Componente I4.0. Fonte: [3]

Com o crescimento da indústria 4.0, sendo o AAS um dos padrões mais recomendados para representar digitalmente componentes i4.0, um número crescente de projetos recentes têm recorrido ao AAS.

Salvatore Cavalieri e Marco Guiseppe Salafia [21] criaram um modelo de manutenção preditiva usando o AAS para integrar toda a informação proveniente de diferentes equipamentos. Devido ao nível de abstração que o AAS consegue proporcionar, equipamentos antes não interoperáveis com tecnologias heterogéneas, foram integrados num sistema conjunto com interfaces comuns que lidam com todas as operações que estes equipamentos implementam, como consulta e agregação de dados, configurações, etc.

Marie Platenius-Mohr et. al. [22] ilustraram uma solução customizável, através do mapeamento de modelos de informação de diversos DTs para AAS, para obter interoperabilidade entre DTs de diferentes empresas, possibilitando assim casos de uso avançados no contexto de IIoT.

Um pequeno resumo sobre AAS será feito nesta secção, visto que revela muitos dos conceitos essenciais para a representação digital de ativos. O manual completo sobre os detalhes e especificações do AAS é extremamente detalhado e contém mais de 500 páginas [3].

2.2.1.1 RAMI 4.0

O modelo RAMI 4.0 (*Reference Architectural Model Industrie 4.0*), representado na figura 2.5, define métodos para tornar tecnologias reutilizáveis e interoperáveis, garantindo que estas utilizam as mesmas normas internacionais, de modo a serem rapidamente desenvolvidas e incorporadas dentro de novas tecnologias da indústria 4.0. Com este modelo, as empresas podem colocar o seu produto (em fase de desenvolvimento ou em produção) dentro do "cubo", de modo a extrapolar

que estruturas e normas devem adotar para estarem de acordo com as diversas tecnologias dentro da indústria 4.0 [23].

Neste modelo de referência, o AAS é recomendado como o pilar da interoperabilidade na indústria 4.0 [24].

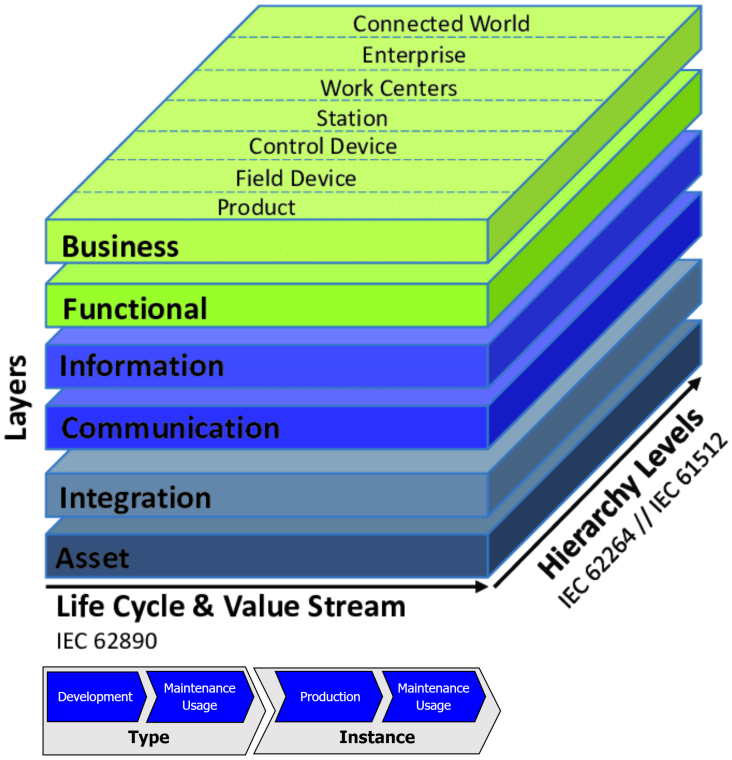

Figura 2.5: Modelo RAMI 4.0. Adaptado de: [25]

O AAS dá especial importância à diferenciação entre tipos (*types*) e instâncias (*instances*). Numa perspetiva de programação orientada a objetos, um tipo é sinónimo de uma classe, e uma instância é sinónimo de um objeto de uma determinada classe. O modelo RAMI4.0 inclui um eixo do ciclo de vida, derivado da norma IEC 62890. O objetivo é distinguir, para todos os ativos dentro da Indústria 4.0, entre possíveis tipos e instância, de modo a caracterizar esta distinção para todos os elementos como materiais, produtos, equipamentos, entre outros. Assim, é possível separar a informação de um determinado ativo em diferentes partes, distribuídas pelo eixo de ciclo de vida. Por exemplo, os fabricantes colocam informação na área *type*, enquanto utilizadores do produto colocam informação na área *instance* (p. ex. para monitorização) [26].

A figura 2.6 exemplifica a utilização de três AASs: um que descreve o tipo de ativo, e dois que descrevem instâncias desse tipo, ou seja, descrevem dois ativos, neste caso, dois sensores de temperatura.

Mantendo estas relações entre tipos e instâncias ao longo de todo o ciclo de vida das instâncias, qualquer atualização feita a um tipo, pode ser reencaminhada para as respetivas instâncias.

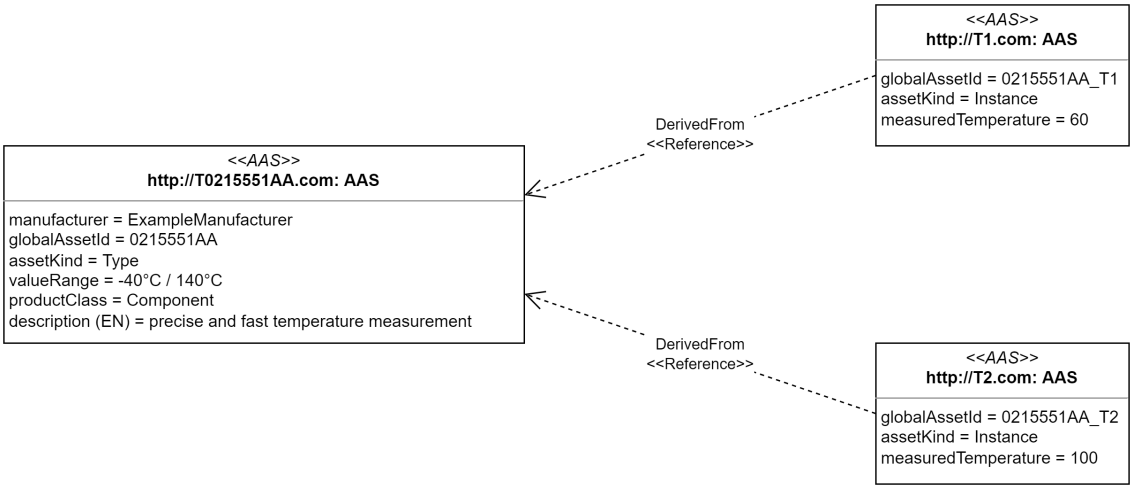

Figura 2.6: Exemplo simplificado de tipos e instâncias. Adaptado de: [3]

2.2.1.2 Estrutura base do AAS

Como ilustrado na figura 2.7, o AAS é composto por uma identificação global e uma série de submodelos. Cada submodelo contém uma quantidade estruturada de propriedades. Estas propriedades todas resultam numa diretoria constantemente disponível, com toda a informação crucial do AAS, ou seja, do componente I4.0.

De acordo com [27], o objetivo é cercar cada ativo com um AAS que consiga fornecer uma descrição mínima mas suficiente do ativo de acordo com os casos de uso da indústria 4.0. Ao mesmo tempo, é importante conseguir mapear normas já existentes, como as IEC e ISO, de acordo com as definições do AAS.

A figura 2.8 exemplifica algumas das possíveis categorias de submodelos do AAS, bem como algumas das normas que devem seguir. O objetivo é que apenas um submodelo corresponda a cada aspeto de um componente I4.0, e cada componente, como explicado anteriormente, pode conter vários submodelos. Por exemplo, para encontrar um equipamento responsável pela pintura, bastaria procurar por um AAS com um submodelo denominado de "Pintura", que conteria as propriedades desse caso de uso. Esse equipamento poderia também, para além desse submodelo, conter outros que fossem responsáveis, por exemplo, pela eficiência energética, segurança, entre outros.

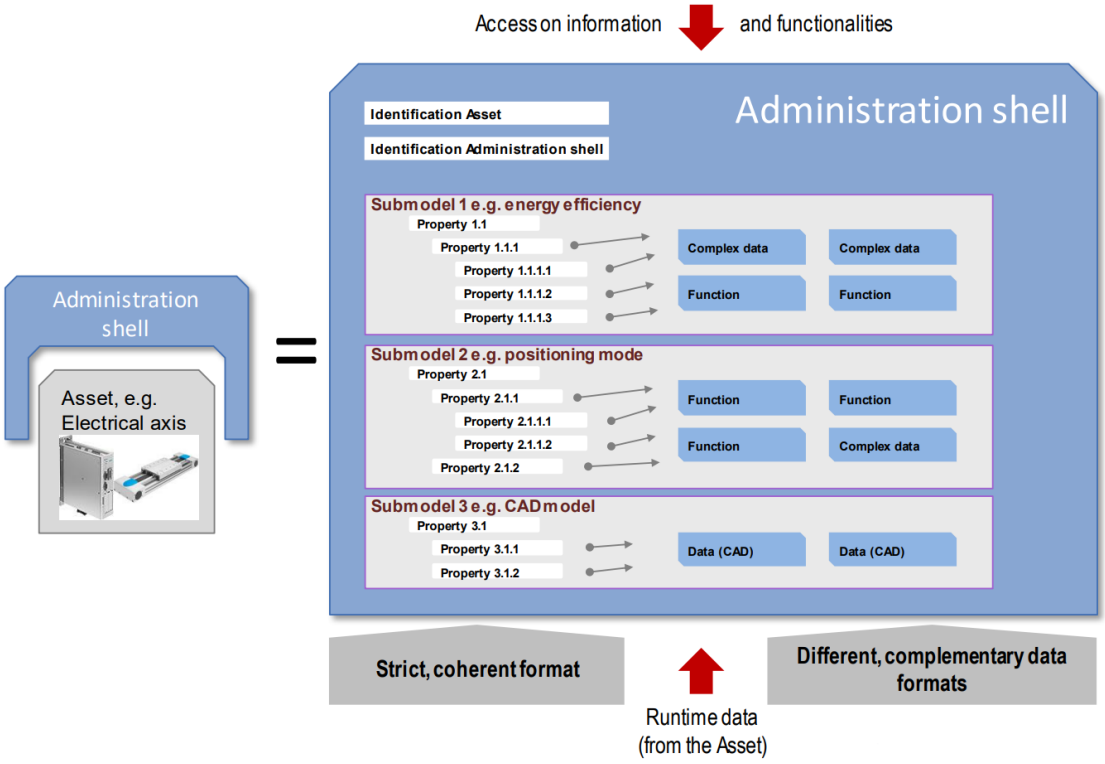

Figura 2.7: Estrutura base do AAS. Fonte: [3]

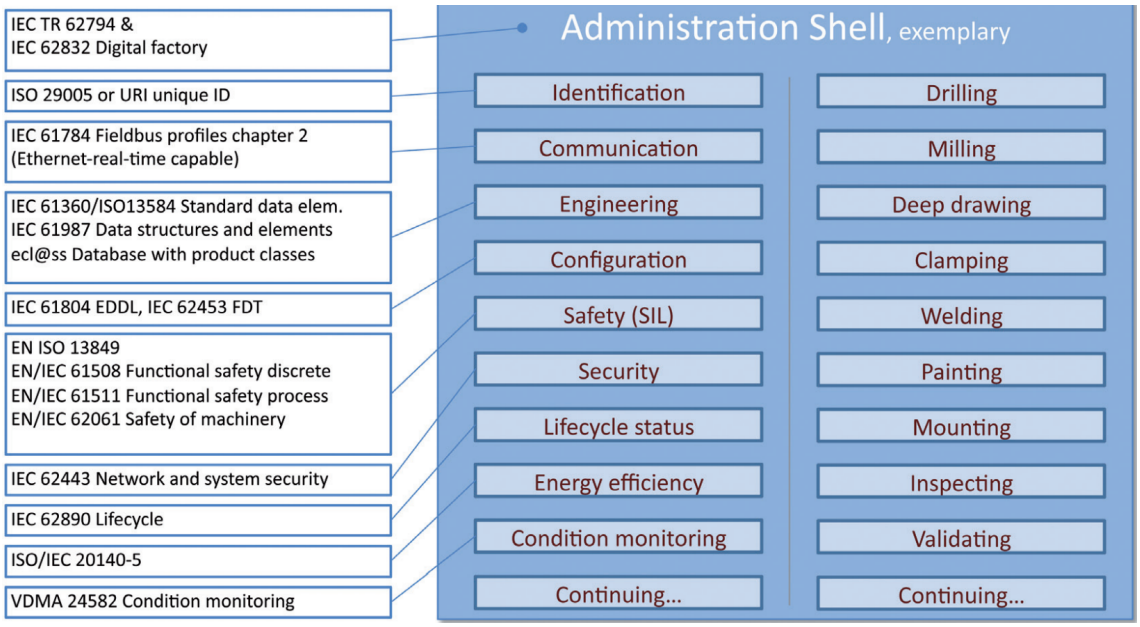

Figura 2.8: Exemplo de categorias de submodelos do AAS. Fonte: [27]

2.2.1.3 Exemplo prático de submodelos

Em [27], é dado um exemplo que ilustra a utilização de diferentes submodelos para descrever equipamentos industriais. Neste exemplo, é dado um cenário com 3 máquinas de perfuração, que competem por novas ordens de produção, fornecidas por um MES (*Manufacturing Execution System*) (figura 2.9).

Como referido anteriormente, cada submodelo contém várias propriedades. A modelação de cada propriedade é dividida em duas secções, sendo estas *Property Definition* (Definição da Propriedade) e *Property Characterisation* (Caracterização da Propriedade). Cada um dos diferentes campos que definem cada propriedade do submodelo encontra-se numa destas duas secções. A figura 2.10 exemplifica o submodelo *"MES connection"* representado na figura 2.9. Os diferentes campos que definem cada propriedade neste exemplo seguem a norma IEC 61360.

É importante notar que os submodelos são flexíveis, podendo ser adicionados novos campos para definir cada propriedade de um respetivo submodelo. Por exemplo, em [28], os autores utili-

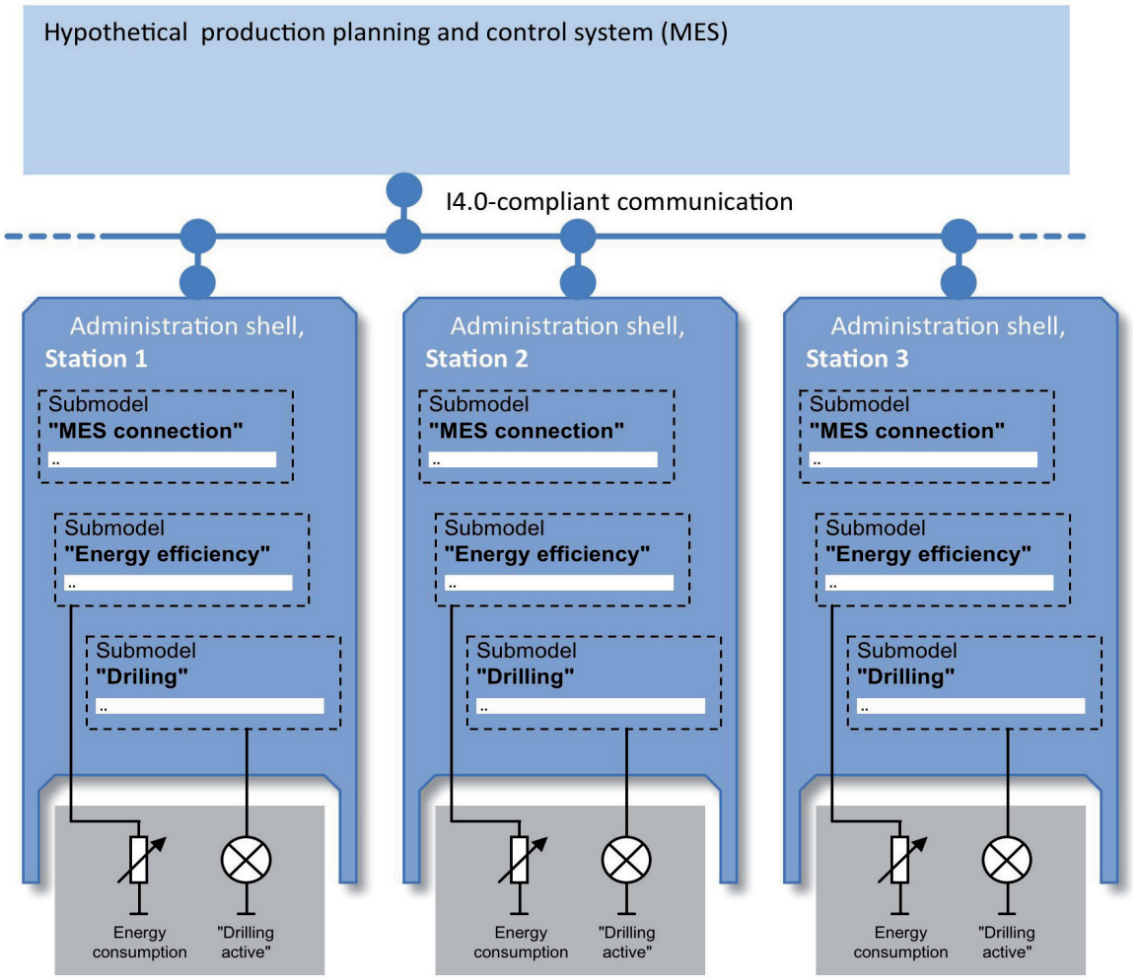

Figura 2.9: Exemplo prático sobre submodelos. Fonte: [27]

			Property definition				Property characterisation					
Hier-archy	ID	(preferred) Name	Definition	Unit of measure	Data type	Value list	Value	Expression semantic	Expression logic	Views	R/D/ F/A/-	Contents
I	AAA020	Asset pro-duction status	This property determines, if the associated asset is able to execute a production task at the time being.	–	ENUM	[Idle, Running, Failure, Restrained, Scheduled down, Unscheduled down]	Running	Measure-ment	Equal	Business	D	-
I	AAA021	Operating hours	This property determines, how long cumulatively the associated asset was switched on ‚mains'.	s	INT64	0..*	153453 s	Measure-ment	Equal	Perfor-mance	D	-

Figura 2.10: Submodelo *"MES connection"*. Fonte: [27]

zam este mesmo submodelo apresentado, da conecção ao MES, no entanto, acharam pertinente incluir uma secção designada de *Property Usage* (Uso da Propriedade), composto por mais 3 campos adicionais na tabela, de forma a obter informação em como melhorar as estratégias de captura de dados, sendo estas: KBITS_PER_SAMPLE, SAMPLING_TIME_MIN e THROUGHPUT_KBMIN, utilizadas para indicar o tamanho da variável em Kilobits, indicar a frequência de amostragem por minuto, e indicar a taxa de mensagens entregues com sucesso em Kilobits por minuto no canal de comunicações, respetivamente.

Depois de desenhados todos os submodelos de um ativo, os únicos elementos que restam são as identificações. São necessárias identificações para o ativo, o respetivo AAS, e para cada submodelo dentro do corpo do AAS. Seguindo o exemplo já utilizado, a figura 2.11 exemplifica possíveis IDs de cada um destes elementos, seguindo as normas recomendadas.

		Asset administration shell		
		Station 1	Station 2	Station 3
DF header	Asset administration shell ID	http://www.zvei.de/demo/11232322	http://www.zvei.de/demo/11232342	http://www.zvei.de/demo/11282322
	ID(s) of asset(s)	[http://pk.festo.com/3S7PLFDRS35]	[http://pk.festo.com/3S7PL9X6K32]	[http://pk.festo.com/3S7PLFNCKDZ]
DF body	ID SM "MES connection"	Type: ADA011 Instance: http://www.zvei.de/demo/2368473473829	ADA011 Instance: http://www.zvei.de/demo/1366423771829	ADA011 Instance: http://www.zvei.de/demo/8364423571326
	ID SM "Energy efficiency"	ADA012 [..]	ADA012 [..]	ADA012 [..]
	ID SM "Drilling"	ADA013 [..]	ADA013 [..]	ADA013 [..]
	ID TM "Documentation"	ADA014 [..]	ADA014 [..]	ADA014 [..]

Figura 2.11: Exemplo de identificações. Fonte: [27]

Após estes passos, ficaria assim terminado o AAS completo do equipamento de perfuração em questão.

2.2.2 Web of Things Thing Description

O *Web of Things* (WoT) *Thing Description* (TD) [4] é um padrão que serve para, tal como o AAS, representar ativos digitalmente, e é recomendado pelo *World Wide Web Consortium* (W3C), que é uma das principais organizações a desenvolver e a oficialmente adotar padrões para a *World Wide Web*.

Uma instância de um ativo tem quatro componentes principais:

1. Metadados textuais sobre o ativo;
2. Um conjunto de informação *(Interaction Affordances)* sobre como é que o ativo pode ser utilizado;
3. Esquemas *(Schemas)* que descrevam os dados trocados com o ativo;
4. *Web links* que expressem qualquer relação, formal ou informal, com outros ativos ou documentos na *Web*.

2.2.2.1 Estrutura base do TD

A designação oficial da estrutura base do TD é *TD Information Model* (Modelo de Informação), e este é constituído por quatro vocabulários independentes:

1. ***Core TD Vocabulary*** que descreve o vocabulário principal do TD, incluindo propriedades, eventos, etc;
2. ***Data Schema Vocabulary*** que descreve o vocabulário dos esquemas;
3. ***WoT Security Vocabulary*** que indentifica os macanismos de segurança e requisitos para a sua configuração;
4. ***Hypermedia Controls Vocabulary*** que representa os conceitos principais de uma comunicação *RESTful*.

Cada um destes vocabulários é essencialmente um conjunto de definições e termos que podem ser usados para estruturar dados, interpretados como um objeto, numa visão de programação orientada a objetos.

As figuras 2.12, 2.13, 2.14 e 2.15, representam os diagramas destes quatro vocabulários independentes, que, em conjunto, formam o modelo de informação.

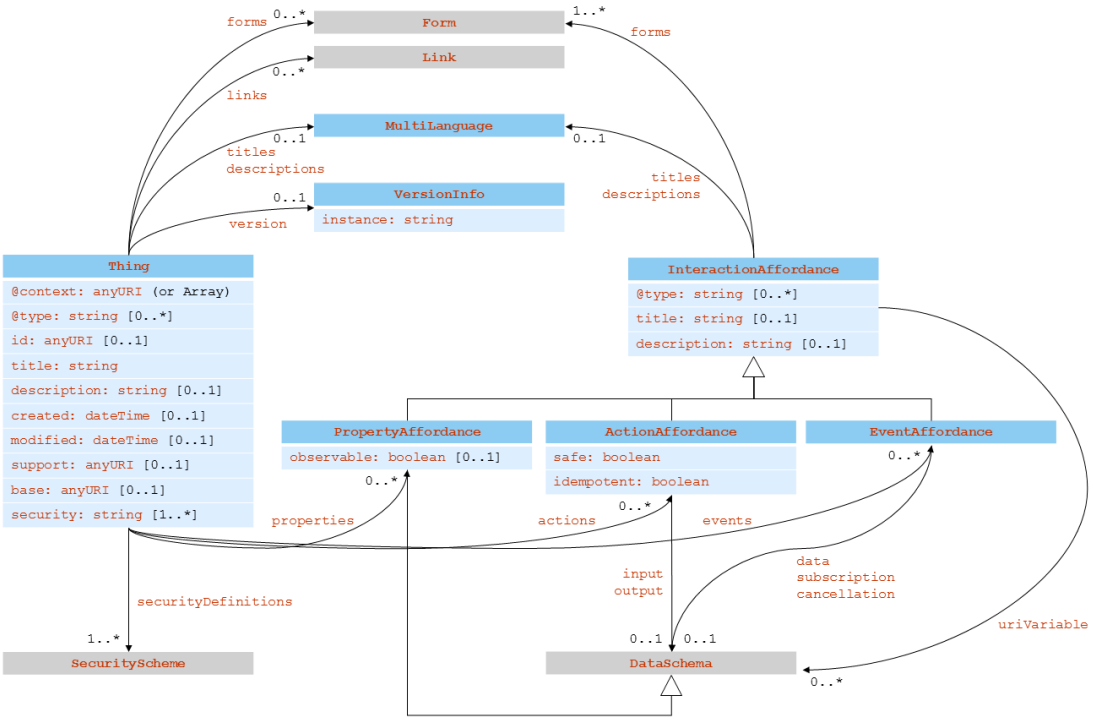

Figura 2.12: TD Core. Fonte: [4]

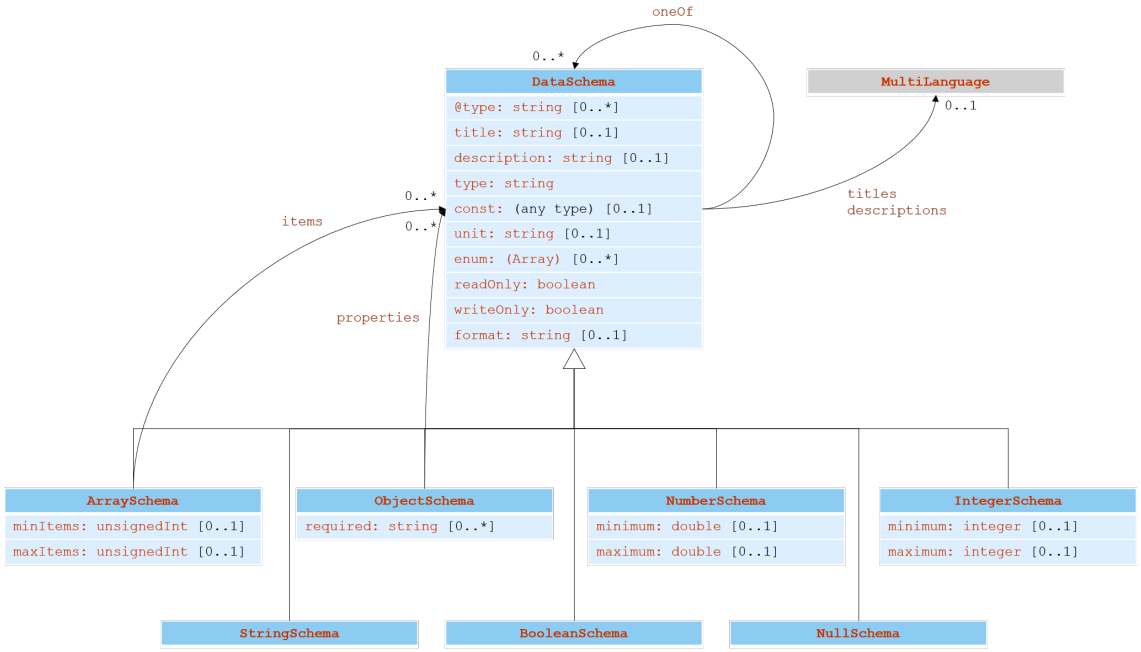

Figura 2.13: Data schema. Fonte: [4]

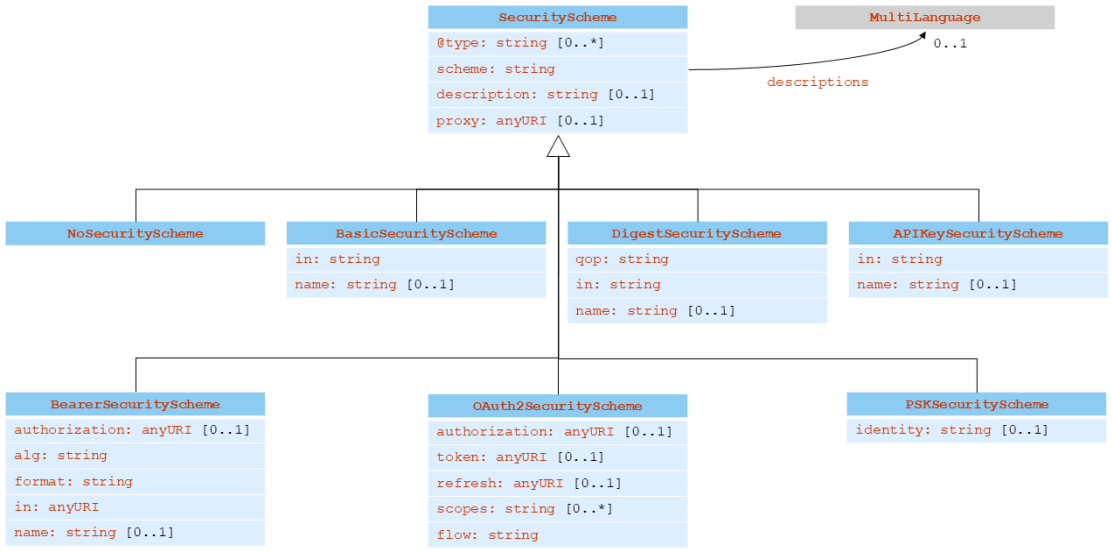

Figura 2.14: TD Security. Fonte: [4]

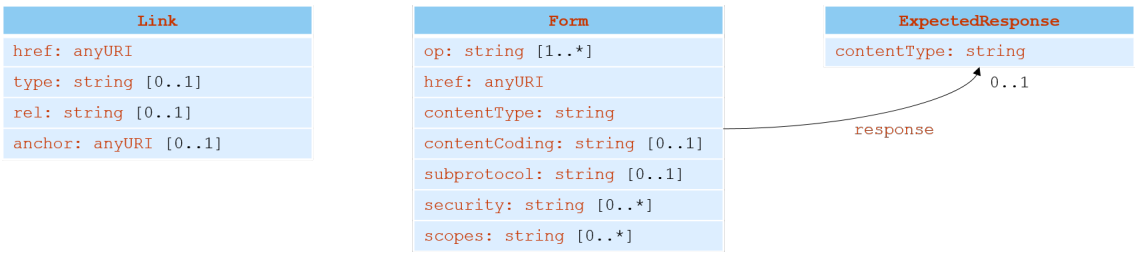

Figura 2.15: Hypermedia Controls. Fonte: [4]

Uma instância de uma *Thing* (p. ex. um equipamento) tem um conjunto de campos que a descrevem. Os únicos campos obrigatórios de serem fornecidos pelo cliente durante o registo de um equipamento são o contexto, título e pelo menos um esquema de segurança, embora este esquema, apesar de não ser recomendado, possa ser vazio, como demonstrado no diagrama da figura 2.14. Os restantes campos são opcionais, no entanto, necessários frequentemente para descrever de forma útil um equipamento, nomeadamente as propriedades, ações e eventos.

O W3C coloca especial importância na segurança e privacidade do TD. Na documentação, uma lista de considerações são levantadas em relação a este aspeto e um conjunto de possíveis mitigações são recomendadas para reduzir riscos de segurança.

De um modo geral, o TD é significativamente mais simples de utilizar do que o AAS, porque não impõe a utilização de muitas tecnologias e normas que requerem fundamentos teóricos avançados.

O TD apresenta também uma característica diferente de outros modelos que é relevante mencionar, porque representa uma perspetiva diferente de interoperabilidade em relação a outros padrões. O TD, ao contrário das outras soluções, não define novas interfaces que são necessárias criar para cada equipamento. Em vez disso, define modelos para representar as interfaces já existentes (p. ex. protocolos utilizados, o formato do *payload*, configurações de segurança, etc). A opinião do W3C é que já existem biliões de equipamentos implementados nas fábricas, muitos deles sem a opção de serem atualizados remotamente, que estar a alterar as suas interfaces não é realista, logo, a favor da interoperabilidade, são necessários modelos que descrevam as APIs já implementadas [2].

2.2.2.2 Serialização do modelo de informação

Serialização é o processo de transformar a informação de uma estrutura (p. ex. um objeto), num formato interoperável que possa ser consumido ou transmitido facilmente pelos vários setores de um sistema digital.

O TD tem documentado um capítulo extenso sobre todas as serializações convenientes e regras a seguir para o modelo de informação, no entanto, aqui, no interesse de manter esta secção relativamente resumida, o mais importante é estipular que o formato utilizado pelo TD para qualquer elemento é obrigatoriamente *JavaScript Object Notation* (JSON), um formato fácil de ler e escrever, quer para humanos, quer para máquinas.

O excerto seguinte exemplifica uma instância / objeto seguindo a estrutura do TD, com todos os campos obrigatórias e opcionais:

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "@type": "Thing",
    "id": "urn:dev:ops:32473-Thing-1234",
    "title": "MyThing",
    "titles": {...},
    "description": "Human readable information.",
    "descriptions": {...},
    "support": "mailto:support@example.com",
    "version" : {...},
    "created" : "2018-11-14T19:10:23.824Z",
    "modified" : "2019-06-01T09:12:43.124Z",
    "securityDefinitions": {...},
    "security": ...,
    "base": "https://servient.example.com/",
    "properties": {...},
    "actions": {...},
    "events": {...},
```

```
    "links": [...],
    "forms": [...]
}
```

A título de exemplo, consideremos o seguinte ativo, neste caso uma lâmpada, com as seguintes características:

- **Title:** MyLampThing;
- **Context Extensions:** nenhum;
- **Affordances:** 1 Property, 1 Action, 1 Event;
- **Security:** PSKSecurityScheme;
- **Protocol Binding:** CoAP over TLS;

A serialização em JSON do ativo apresentado seria algo semelhante ao seguinte exemplo, com eventuais campos adicionais:

```
{
   "@context": [
      "https://www.w3.org/2019/wot/td/v1",
      {
      "cov": "http://www.example.org/coap-binding#"
      }
    ],
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "description" : "MyLampThing uses JSON serialization",
    "securityDefinitions": {"psk_sc":{"scheme": "psk"}},
    "security": ["psk_sc"],
    "properties": {
        "status": {
            "description" : "Shows the current status of the lamp",
            "type": "string",
            "forms": [{
                "op": "readproperty",
                "href": "coaps://mylamp.example.com/status",
                "cov:methodName" : "GET"
            }]
        }
    },
```

```
    "actions": {
        "toggle": {
            "description" : "Turn on or off the lamp",
            "forms": [{
                "href": "coaps://mylamp.example.com/toggle",
                "cov:methodName" : "POST"
            }]
        }
    },
    "events": {
        "overheating": {
            "description" : "Lamp reaches a critical temperature",
            "data": {"type": "string"},
            "forms": [{
                "href": "coaps://mylamp.example.com/oh",
                "cov:methodName" : "GET",
                "subprotocol" : "cov:observe"
            }]
        }
    }
}
```

2.3 Application Programming Interfaces

Uma *Application Programmable Interface* (API) tem como objetivo integrar sistemas, facilitando a troca de informação entre diferentes linguagens de programação e diferentes tipos de plataformas tecnológicas, através de interfaces que expõe funcionalidades já previamente implementadas [29].

No contexto de modelação e descrição de ativos, APIs são frequentemente utilizadas para criar novas instâncias de ativos e consultar, editar e/ou apagar dados das mesmas. Deste modo, é possível ter controlo total sobre as características e propriedades dos ativos em qualquer computador autorizado pela plataforma, através de uma aplicação que consiga efetuar pedidos com o protocolo HTTP.

De seguida, será explicado em mais detalhe o conceito de REST (Representational State Transfer) API, que será o tipo de arquitetura API utilizado no projeto de dissertação.

2.3.1 REST APIs

REST é o tipo de arquitetura API mais utilizado em serviços *Web*. Os princípios fundamentais desta arquitetura são os seguintes [30] [31]:

- **Identificação de recursos:** Cada recurso é identificado com um endereço único denominado de *Uniform Resource Identifier* (URI), de modo a poder ser facilmente acedido.
- **Representação de recursos:** Os clientes dos serviços não sabem diretamente o formato interno e estado dos recursos. Na *Web* os recursos são representados através de padrões como JSON ou XML. Deste modo, é possível manter a interoperabilidade entre diferentes serviços bem como a abstração entre cliente e servidor, o que acarreta benefícios de segurança.
- **Interface uniforme:** Os recursos são manipulados através dos métodos padrão definidos pelo protocolo HTTP (GET, POST, PUT, DELETE, etc). Deste modo, o nível de desacoplamento entre o cliente e o serviço REST é maior. Cada método tem um comportamento expectável e um respetivo *status code* (código de resposta).
- **Interações sem estado:** As interações entre o cliente e a API não têm um estado. Cada pedido que é feito pelo cliente contém toda a informação necessária para ser processado pela API e não é mantido nenhum estado de interação no servidor.
- ***Hypermedia As The Engine of Application State* (HATEOAS)**: O conteúdo de uma página, ou o estado da aplicação como o nome indica, é fornecido dinamicamente através da *hypermedia* (conteúdo com links para outro conteúdo). Deste modo, o cliente não necessita de um conhecimento prévio da aplicação para a navegar, ele é guiado através de links até obter o conteúdo desejado.

Juntos, estes princípios explicam o nome *Representational State Transfer*: o estado da interação entre cliente e servidor não é guardada no lado do servidor. Esta é transferida pelo cliente para o servidor em cada pedido realizado, codificado dentro da representação do recurso a que o pedido se refere.

Atualmente, paradigmas web orientados a serviços, como é o caso de APIs REST, são fortemente recomendados para a comunicação entre as representações digitais dos componentes i4.0 (p. ex. AAS) e outros sistemas industriais interconectados e sistemas de informação, devido à elevada interoperabilidade que oferecem [32, 33].

Capítulo 3

Trabalho realizado

No presente capítulo serão apresentados os detalhes sobre o trabalho realizado ao longo dos 3 meses de acolhimento no INESC TEC.

O capítulo será dividido em 3 partes distintas:

1. **Descrição do projeto:** Nesta secção serão apresentados alguns detalhes gerais sobre o projeto SADCoPQ, para o qual está a ser desenvolvida a plataforma IIoT, e será também descrito em mais pormenor o módulo *Thing Registry* da plataforma, onde o trabalho desenvolvido para esta dissertação se enquadra.

2. **Modelo de informação:** Nesta secção será feita uma análise completa do modelo de informação utilizado para descrever os ativos que serão registados na plataforma IIoT. Inicialmente será apresentada a arquitetura do modelo utilizado e depois será demonstrada a aplicação do modelo num equipamento industrial que será efetivamente registado na plataforma através do *Thing Registry*. Para isso, será feita uma análise das características e dados recolhidos pelo equipamento e será feito o mapeamento desses dados para o modelo utilizado.

3. **REST API:** Nesta secção será feita uma análise completa da REST API desenvolvida para o módulo *Thing Registry* da plataforma. Durante o período da dissertação, foi desenvolvida a parte *backend* da API, responsável por lidar com os pedidos feitos ao servidor e fornecer respostas no formato desejado para serem consumidas pelos utilizadores. Inicialmente será apresentada a arquitetura, demonstrando, a um alto-nível, as diferentes camadas que constituem a API. Depois, será feita uma apresentação mais detalhada sobre as interações entre as diferentes camadas e as suas funcionalidades e interfaces.

3.1 Descrição do projeto

3.1.1 Projeto SADCoPQ

O projeto SADCoPQ - Sistema de Apoio à Decisão no Controlo Preditivo da Qualidade, é um projeto que pretende desenvolver uma solução inovadora, de acordo com os conceitos da Qualidade 4.0, baseada em tecnologias de informação e comunicação suportadas por inteligência artificial e dedicada à gestão de controlo da qualidade em indústrias de manufatura de produtos com elevados padrões de exigência, traduzidos por características quantitativas e qualitativas detalhadas, com tolerâncias muito apertadas como é o caso da indústria metalomecânica de precisão, onde este projeto se enquadra.

A solução será capaz de gerir, de forma holística, os riscos de ocorrência de não conformidades nas etapas do ciclo de vida do produto, incluindo conceção, planeamento e fabrico, e terá a capacidade de descrição, diagnóstico, previsão e prescrição. A solução também irá co-existir com soluções de gestão industrial já implementadas, como ERP *(Enterprise Resource Planning Systems)*, MES *(Manufacturing Execution Systems)*, APS *(Advanced Planning Systems)*, CAD/-CAM *(Computer Aided Design & Computer Aided Manufacturing)* e/ou PLM *(Product Lifecycle Management Systems)*.

O projeto SADCoPQ pretende desenvolver e implementar as seguintes tecnologias:

1. **Implementação de digital twins:** Este objetivo consiste em criar uma cópia virtual do sistema de produção, obtendo assim um modelo digital que represente o comportamento físico do sistema, de forma a apoiar a solução de controlo e qualidade a diagnosticar e prever não-conformidades, bem como prescrever ações que visam minimizar/eliminar o risco e probabilidade de ocorrência das mesmas.

2. **Previsor de não-conformidades:** O principal objetivo deste módulo é calcular o risco de ocorrência de não-conformidades para as combinações significativas dos seus fatores influenciadores, traduzido por medidas de probabilidade, desvio padrão e intervalos de confiança.

3. **Modelador de processos de controlo de qualidade:** Este módulo tem como objetivo auxiliar na seleção de tecnologias de produção mais adequadas ao fabrico de produtos, em função das suas características específicas, de forma a minimizar o risco de ocorrência de não-conformidades. Para além disso, pretende auxiliar na definição de planos de controlo da qualidade para cada combinação específica de produto/tecnologia de produção.

4. **Assistente inteligente do controlo da qualidade:** Os principais objetivos deste módulo são auxiliar na identificação e na explicação das causas de ocorrência de não-conformidades, identificando os fatores que as motivaram, as suas relações e graus de importância relativa; bem como auxiliar no estabelecimento de medidas para redução/eliminação do risco de ocorrência de não-conformidades.

3.1.2 Thing Registry

Um dos componentes essenciais na implementação de digital twins num ambiente industrial, é um módulo capaz de representar digitalmente todos os equipamentos do chão de fábrica. O desenvolvimento de um modelo de descrição destes equipamentos, bem como o desenvolvimento de uma REST API para registar, editar, apagar e consultar estes dados, são os objetivos principais da fase de desenvolvimento da dissertação.

A figura 3.1 representa, a um alto-nível, a arquitetura da plataforma IIoT do projeto SADCoPQ. O módulo *Thing Registry* trabalha em paralelo com o *Thing Replica* que é um módulo que permite ao utilizador visualizar a informação a ser recolhida em tempo real pelos equipamentos da fábrica, estando estes registados no *Thing Registry*.

Para além da comunicação entre estes dois componentes da plataforma, o *Thing Registry* também comunica com outros componentes:

- O *Service Registry*, que gere os serviços e links (explicado em mais detalhe na secção 3.2.1.7) associados a um determinado equipamento;
- O API *Gateway* que serve como intermediário entre o cliente e as APIs disponibilizadas pela plataforma IIoT;
- E o *Message Broker* que é responsável por traduzir as mensagens de um determinado protocolo de um componente da plataforma e enviá-las para outro componente. Este serve também como ponto central para recolher e distribuir informação pelos diferentes componentes da plataforma. Deste modo, cada componente não necessita de vários protocolos

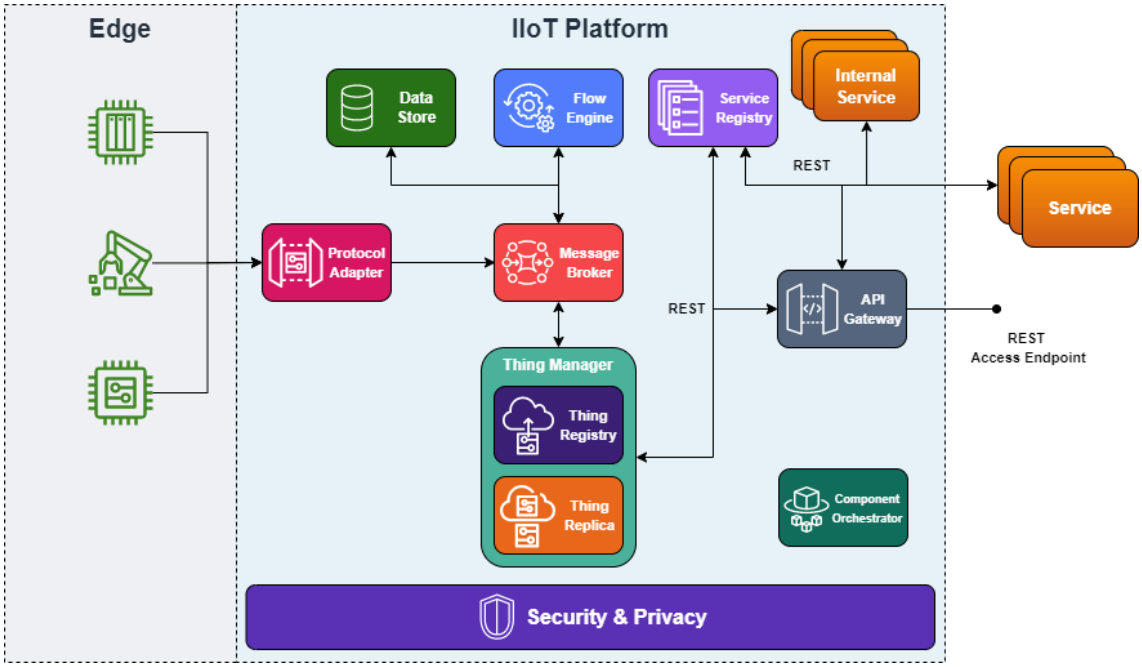

Figura 3.1: Arquitetura da plataforma IIoT do projeto SADCoPQ

específicos de comunicação, reduzindo a sua complexidade. Cada componente apenas se preocupa em enviar dados para o *message broker*, que os reencaminha para o destinatário desejado.

3.2 Modelo de informação

3.2.1 Arquitetura

Embora um modelo já existente pudesse ter sido utilizado no projeto SADCoPQ para representar digitalmente equipamentos, como o AAS ou o TD, a equipa achou adequado fazer um modelo proprietário, adequado às necessidades do projeto, devido à complexidade e restrições que os exemplos referidos impõe, e o tamanho reduzido da equipa de desenvolvimento. Embora os modelos apresentados na secção 2.2 sejam mais interoperáveis que o modelo utilizado, esta característica torna-os também bastante mais complexos e difíceis de aplicar, como explicado na secção 1.2, porque têm de seguir uma série de normas na indústria que, no caso do projeto SADCoPQ, sendo este de tamanho relativamente pequeno, não são necessariamente uma prioridade. Um exemplo que suporta este pensamento é a particular atenção que o AAS dá à interoperabilidade entre componentes i4.0, mesmo entre diferentes empresas, de modo a criar uma enorme rede de valor ao longo de todo o processo de produção. Esta abordagem, embora apresente as suas vantagens, acarreta obstáculos difíceis de ultrapassar para empresas pequenas.

Outra consideração a ter é o contexto do ecossistema em que os equipamentos se encontram. Neste projeto, é apresentado um cenário relativamente conservador, no sentido em que não se espera obter uma fábrica completamente preenchida com componentes i4.0 inteligentes ao longo de todo o ciclo de vida de produção, a comunicarem constantemente entre si. Estamos, em vez de isso, a falar de um controlo mais específico, mais interno, focado num pequeno número de equipamentos, numa pequena fase do ciclo de vida de cada peça, ou seja, para este projeto, não é necessário o nível de interoperabilidade que um padrão como o AAS proporciona.

Depois de colocadas estas considerações, embora o modelo efetivamente utilizado seja diferente dos analisados, este não deixa de seguir os conceitos essenciais no que toca à modelação e descrição de ativos, nomeadamente os apresentados no RAMI 4.0.

Um sistema i4.0 pode ser descrito como uma integração de componentes i4.0, caracterizados da seguinte maneira [34]:

- Cada componente pode ser composto por outros componentes de um nível hierárquico mais baixo e ser parte de um componente de um nível hierárquico mais alto (propriedade de aninhamento lógico (*logical nestability*) de componentes i4.0, discutido também em [35]);
- Componentes do mesmo nível hierárquico podem interagir e cooperar de maneira a atingir objetivos comuns ou individuais;
- Todos os componentes têm fundamentalmente a mesma estrutura interna mas diferentes responsabilidades.

O modelo usado neste projeto utiliza o conceito de nós lógicos (*logical nodes*). De maneira a descrever equipamentos físicos, são utilizados estes nós, que, agrupados, podem formar entidades mais complexas, cumprindo assim a visão de aninhamento lógico mencionada anteriormente.

Cada nó representa um equipamento físico ou parte de um equipamento físico. Tipicamente, cada nó de alto nível representa um equipamento, enquanto cada sub-nó representa uma parte logicamente separável desse componente. Assim, várias partes formam em conjunto um todo, facilitando a representação digital de um ativo complexo, repartindo-a em várias representações (sub-nós) ligadas entre si (p. ex. o nó principal pode ser um robô que contém dois sub-nós, um que representa um braço robótico e outro que representa um gerador de energia).

De seguida, a estrutura de cada nó será apresentada. Estes objetos serão guardados numa base de dados do tipo NoSQL, orientada a documentos, neste caso MongoDB, e devem conter toda a informação necessária para caracterizar e descrever o equipamento. O módulo responsável por gerir esta informação será o *Thing Registry*, que irá fornecer uma API através da qual utilizadores terão acesso aos dados e poderão criar, ler, editar e apagar nós.

É conveniente notar que o modelo não contém apenas as propriedades necessárias para representar os equipamentos que serão modelados neste projeto. O modelo foi feito para ser o mais compatível possível, não só com as máquinas dentro do projeto SADCoPQ, mas também com uma série de equipamentos que possam ser adicionados à plataforma no futuro. Para além disso, espera-se que este projeto sirva de fundamento para projetos futuros dentro do INESC TEC, no contexto de IIoT, que necessitem de um módulo *Thing Registry* para registar e gerir os dispositivos inteligentes conectados à respetiva plataforma, sendo por isso imperativa a implementação de um modelo bem estruturado que responda, da melhor forma possível, a todas as necessidades de representação digital de vários sistemas compostos por uma série de equipamentos.

3.2.1.1 Node (Nó)

Um nó é o objeto principal do modelo e representa um equipamento ou um componente de um equipamento. Cada documento na base de dados MongoDB é um nó diferente e único da plataforma.

Cada nó é representado por uma série de metadados que descrevem o equipamento, como a sua identificação global, protocolo utilizado para publicar dados na plataforma, etc, bem como um conjunto de outros objetos que serão explicados mais à frente que definem propriedades e funcionalidades do equipamento, como configurações e eventos.

A tabela 3.1 representa todos os campos que constituem um nó.

Campos que constituem a vista de um Nó

Nome	Tipo	Descrição
nodeId	string	ID único do nó. Caso não seja escolhido é automaticamente gerado pela plataforma.

Tabela 3.1 continuação da página anterior

name	string	Nome do nó.
description	string	Descrição do nó.
code	string	Código associado ao nó.
manufacturer	string	Nome do fabricante, se aplicável.
model	string	Modelo do nó.
type	string	Tipo do nó.
subType	string	Sub-tipo do nó.
uri	string	URI de acesso do nó.
protocol	string	Protocolo utilizado pelo nó para publicar dados na plataforma.
attributes	array	Lista de atributos do nó (tabela 3.2).
configuration	array	Lista de configurações do nó (tabela 3.3).
nodes	array	Lista de sub-nós do nó.
links	array	Lista de links do nó (tabela 3.12).
services	array	Lista de serviços do nó (tabela 3.8).
variables	array	Lista de variáveis do nó (tabela 3.4).
aggregations	array	Lista de agregações do nó (tabela 3.5).
events	array	Lista de eventos do nó (tabela 3.6).

Tabela 3.1: Modelo de informação de um nó

Dados de um nó em formato JSON - Exemplo:

```
{
    "nodeId": "2450f692-4658-454b-a912-1176488af34e"
    "name": "Joule",
    "description": "Intel Joule",
    "code": "Joule",
    "manufacturer": "Intel",
    "model": "Joule",
    "type": "Device",
    "subType": "Device",
    "uri": "http://localhost:8080/joule",
    "protocol": "kafka",
```

```
    "attributes": [
        {
            "name": "serialnumber",
            "value": "23NC600079L"
        }
    ],
    "configuration": [
        {
            "name": "opcUaNodeId",
            "value": "joule"
        }
    ],
    "nodes": [],
    "links": [],
    "variables": [],
    "aggregations": [],
    "events": [],
    "services": []
}
```

3.2.1.2 Attribute (Atributo)

Um atributo permite adicionar informação extra customizada ao nó para além daquela possível com os outros campos do modelo, como por exemplo, um número de série de um equipamento, como demonstrado no exemplo JSON abaixo.

A tabela 3.2 representa todos os campos que constituem um atributo.

Campos que constituem a vista de um Atributo		
Nome	**Tipo**	**Descrição**
attributeId	string	ID único do atributo. Automaticamente gerado pela plataforma.
name	string	Nome do atributo.
value	string	Valor do atributo.

Tabela 3.2: Modelo de informação de um atributo

Dados de um atributo em formato JSON - Exemplo:

```
{
    "attributeId": "c0bac83a-e3f7-42e3-8a68-01eb4028a27a",
```

```
    "name": "serialnumber",
    "value": "23NC600079L”
}
```

3.2.1.3 Configuration (Configuração)

Uma configuração permite adicionar pares chave-valor a um nó dependendo do contexto de instalação. Por exemplo, no caso de uma instalação onde OPC UA tenha sido usado, uma configuração pode ser o ID de cada nó usado no servidor OPC UA.

A tabela 3.3 representa todos os campos que constituem uma configuração.

Campos que constituem a vista de uma Configuração		
Nome	**Tipo**	**Descrição**
configurationId	string	ID único da configuração. Automaticamente gerado pela plataforma.
name	string	Nome da configuração.
value	string	Valor da configuração.

Tabela 3.3: Modelo de informação de uma configuração

Dados de uma configuração em formato JSON - Exemplo:

```
{
    "configurationId": "c0bac83a-e3f7-42e3-8a68-01eb4028a27a",
    "name": "opcUaNodeId",
    "value": "d65ba5f9-17cd-4462-8f8d-d75d4fef2aec”
}
```

3.2.1.4 Variable (Variável)

Uma variável representa o valor de uma determinada característica, leitura ou quantidade que o nó pretende expor para o mundo exterior. O acesso a cada variável pode ser configurado como sendo *read* (para leitura) ou *write* (para escrita). O exemplo de uma variável seria a leitura de um determinado sensor de um equipamento.

A tabela 3.4 representa todos os campos que constituem uma variável.

Campos que constituem a vista de uma Variável		
Nome	**Tipo**	**Descrição**

Tabela 3.4 continuação da página anterior

variableId	string	ID único da variável. Caso não seja escolhido é automaticamente gerado pela plataforma.
name	string	Nome da variável.
description	string	Descrição da variável.
code	string	Código associado à variável.
unit	string	Unidade de medida da variável.
resolution	decimal	Incremento mínimo possível da variável.
type	string	Define o objeto que a variável representa.
dataType	string	Define o tipo da variável (p. ex. boolean).
origin	string	Origem da variável, usada para fazer a distinção entre diferentes origens da variável em relação ao nó (p. ex. se uma origem for externa, quer dizer que a variável é publicada no node por uma entidade externa).
permissions	array de strings	Permissões de acesso à variável - Read, Write.
configuration	array	Lista de configurações (tabela 3.3).
rangeMax	decimal	Valor máximo da variável.
rangeMin	decimal	Valor mínimo da variável.
admissibleValues	array de strings	Lista de valores admissíveis da variável.
readingsSize	long	Número de leituras da variável a serem mantidas na réplica do nó (p. ex. se este valor for 5, apenas as últimas 5 leituras desta variável serão mantidas na réplica do nó. Se não for escolhido, um valor predefinido será usado).
showInDashboard	boolean	Define se a variável deve ser ou não visível nos painéis de leitura.

Tabela 3.4: Modelo de informação de uma variável

Dados de uma variável em formato JSON - Exemplo:

```
{
    "variableId": "f41cdd96-800a-41e7-b15a-cf602e8f5e4d",
    "name": "Temperature",
    "description": "Temperature Sensor Reading",
    "code": "temp",
    "unit": "ºC",
    "resolution": 0.1,
    "type": "Sensor",
    "dataType": "Decimal",
    "origin": "Internal",
    "permissions": [
        "READ"
    ],
    "configuration": [
        {
            "configurationId":
                "c0bac83a-e3f7-42e3-8a68-01eb4028a27a",
            "name": "opcUaVariableId",
            "value": "temp"
        }
    ],
    "rangeMax": 100,
    "rangeMin": -100,
    "admissibleValues": [],
    "readingsSize": 5,
    "showInDashboard": true
}
```

3.2.1.5 Aggregation (Agregação)

Uma agregação permite agrupar uma série de leituras de múltiplas variáveis numa única mensagem de modo a minimizar não só o número de mensagens enviadas para a plataforma, mas também para reduzir o volume de informação / dados enviados, que, no contexto de equipamentos remotos com um acesso limitado à rede, pode ser muito relevante.

Uma agregação é então definida como uma lista de variáveis existentes cujas leituras serão enviadas para a plataforma numa única mensagem que contém as leituras de cada variável num determinado instante, em vez de serem enviadas mensagens individuais com essa mesma informação.

Esta agregação das leituras pode ser feita diretamente no equipamento ou num gateway de borda (*edge gateway*), que seria responsável por agrupar as mensagens recebidas e enviar a informação numa única mensagem para a plataforma.

A tabela 3.5 representa todos os campos que constituem uma agregação.

Campos que constituem a vista de uma Agregação		
Nome	**Tipo**	**Descrição**
aggregationId	string	ID único da agregação. Caso não seja escolhido é automaticamente gerado pela plataforma.
name	string	Nome da agregação.
description	string	Descrição da agregação.
code	string	Código associado à agregação.
variables	array	Lista de IDs de variáveis (mínimo de 2 variáveis).

Tabela 3.5: Modelo de informação de uma agregação

Dados de uma agregação em formato JSON - Exemplo:

```
{
    "aggregationId": "d65ba5f9-17cd-4462-8f8d-d75d4fef2aec",
    "name": "aggregation",
    "description": "var1_var2_aggregation",
    "code": "AGG12",
    "variables": [
        "variable1",
        "variable2"
    ]
}
```

3.2.1.6 Event (Evento)

Enquanto uma variável reflete um único valor, um evento permite representar uma determinada ocorrência usando objetos mais complexos. Para além dos metadados descritivos do evento, este consiste também numa série de propriedades que o descrevem em mais detalhe e guardam informação relevante adicional.

A tabela 3.6 representa todos os campos que constituem um evento e a tabela 3.7 os campos que constituem uma propriedade de um evento.

Campos que constituem a vista de um Evento		
Nome	**Tipo**	**Descrição**
eventId	string	ID único do evento. Caso não seja escolhido é automaticamente gerado pela plataforma.
name	string	Nome do evento.
description	string	Descrição do evento.
type	string	Texto livre que permite categorizar eventos por tipo. Esta informação é útil quando queremos pesquisar o histórico de um determinado tipo de evento.
readingsSize	long	Número de leituras do evento a serem mantidas na réplica do nó. (p. ex. se este valor for 5, apenas as últimas 5 leituras desta variável serão mantidas na réplica do nó. Se não for escolhido, um valor predefinido será usado).
properties	array	Lista de propriedades (tabela 3.7) (mínimo de 1 propriedade).

Tabela 3.6: Modelo de informação de um evento.

Campos que constituem a vista de uma propriedade		
Nome	**Tipo**	**Descrição**
name	string	Nome da propriedade.
description	string	Descrição da propriedade.
type	string	Tipo da propriedade (p. ex. string, integer, etc).

Tabela 3.7: Modelo de informação de uma propriedade.

Dados de um evento em formato JSON - Exemplo:

```
{
    "eventId": "01286f6a-da52-41e7-89d7-386edb5691ee",
    "name": "Container Arrival",
    "description": "Event describing a container arriving
                    to a specific work center",
    "type": "containerArrival",
```

```
    "readingsSize": 5,
    "properties": [
        {
            "name": "containerId",
            "description": "Identifier of the container",
            "type": "integer"
        },
        {
            "name": "workcenterId",
            "description": "Identifier of the work center",
            "type": "string"
        }
    ]
}
```

3.2.1.7 Links & Services (Links e Serviços)

Os links e os serviços são geridos por um segundo módulo da plataforma chamado *Service Registry*, que trabalha em paralelo com o *Thing Registry*, e tem como objetivo fornecer um nível extra de abstração para estes dois elementos de um nó, bem como fornecer outros serviços à plataforma.

O *Thing Registry* apenas guarda os IDs dos links e serviços, no entanto, é ele que redireciona pedidos feitos pelos utilizadores para o *Service Registry*, funcionando assim como um intermediário, evitando invocações diretas nos dispositivos, o que ajuda a mitigar alguns riscos de segurança, visto que para além do nível extra de abstração, é possível configurar firewalls que bloqueiam o acesso aos serviços e links a qualquer entidade sem autorização.

Service: Este objeto permite descrever os serviços oferecidos por um dispositivo ou aplicação que passarão a ser disponibilizados através da plataforma IIoT.

Visto que o *Thing Registry* guarda apenas o ID dos serviços, a tabela 3.8 representa a vista de um serviço que se obtém ao pedir a informação sobre este ao *Service Registry*.

Campos que constituem a vista de um Serviço

Nome	Tipo	Descrição
serviceId	string	ID único do serviço. Automaticamente gerado pela plataforma.
name	string	Nome do serviço.
description	string	Descrição do serviço

Tabela 3.8 continuação da página anterior

uri	string	URI do serviço.
methods	array	Lista de métodos do serviço (tabela 3.9).
filters	array	Lista de filtros do serviço (tabela 3.10).

Tabela 3.8: Modelo de informação de um serviço.

Campos que constituem a vista de um Método		
Nome	**Tipo**	**Descrição**
methodId	string	ID único do método. Automaticamente gerado pela plataforma.
name	string	Nome do método.
description	string	Descrição do método.
httpPath	string	Esta string será concatenada com o URI do serviço para formar o endereço HTTP de acesso interno ao método, usado pela plataforma, ou seja,{serviceUri}/{httpPath}
httpVerb	string	Verbo usado pelo método: "GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS".
route	RouteDto	Rota registada no API Gateway. Automaticamente gerada pela plataforma.
usageInfo	string	Informação sobre como usar o método.
mapTo	string	Caminho para o qual httpPath será mapeado, permitindo o acesso externo ao serviço, através de {serviceUrl}/{mapTo}.
requestParameters	array	Lista de informação sobre os parâmetros de pedido suportados pelo método (tabela 3.11).
responseParameters	array	Lista de informação sobre os parâmetros de resposta suportados pelo método (tabela 3.11).
requestSample	string	Texto exemplo de como preencher um pedido, se aplicável.
responseSample	string	Texto exemplo de como preencher o conteúdo de resposta, se aplicável.

Tabela 3.9 continuação da página anterior

Tabela 3.9: Modelo de informação de um método.

Campos que constituem a vista de um Filtro		
Nome	**Tipo**	**Descrição**
stripPath	boolean	Indica se o API Gateway deve remover /services/{serviceId}/{method.mapTo} do pedido ao serviço.

Tabela 3.10: Modelo de informação de um filtro.

Campos que constituem a vista de um parâmetro de pedido ou resposta		
Nome	**Tipo**	**Descrição**
name	string	Nome do parâmetro.
type	string	Tipo do parâmetro (p. ex. string, integer, etc).
required	boolean	Indica se é um parâmetro obrigatório.
usageInfo	string	Texto descritivo de como preencher o parâmetro.

Tabela 3.11: Modelo de informação de um parâmetro de pedido ou resposta.

Dados de um serviço em formato JSON - Exemplo:

```
{
    "serviceId": "082f1165-f5ae-4feb-89a2-13f404bbde85",
    "name": "Test Service",
    "description": "Test Service",
    "uri": "http://192.168.8.103",
    "methods": [
        {
            "methodId": "01286f6a-da52-41e7-89d7-386edb5691ee",
            "name": "Get Products",
            "description": "Endpoint to get all products",
            "httpPath": "/products",
            "httpVerb": "GET",
            "route": {},
            "usageInfo": "...",
```

```
            "mapTo": "/products",
            "requestParameters": [],
            "responseParameters": [],
            "requestSample": "{\"my-req\": \"test\"}",
            "responseSample" : "{\"products\": [\"product1\"]}"
        }
    ],
    "filters": [
        {
            stripPath: "true"
        }
    ]
}
```

Link: Um link pode ser usado para expor uma série de relatórios, aplicações ou outro tipo de informação através da plataforma registando um URI para estes.

Visto que o *Thing Registry* guarda apenas o ID dos links, a tabela 3.12 representa a vista de um Link que se obtém ao pedir a informação sobre este ao *Service Registry*.

Campos que constituem a vista de um Link		
Nome	**Tipo**	**Descrição**
linkId	string	ID único do link. Automaticamente gerado pela plataforma.
name	string	Nome do link.
description	string	Descrição do link.
uri	string	URI de acesso do link.
httpPath	string	Esta string será concatenada com o URI para formar o endereço HTTP de acesso interno ao link, usado pela plataforma, ou seja, {uri}/{httpPath}.
mapTo	string	Caminho para o qual httpPath será mapeado, permitindo o acesso externo ao link, através de "{linkUrl}/{mapTo}".

Tabela 3.12: Modelo de informação de um link

Dados de um link em formato JSON - Exemplo:

```
{
    "linkId": "082f1165-f5ae-4feb-89a2-13f404bbde85",
    "name": "Cloud9",
    "description": "Cloud9",
    "uri": "http://192.168.8.103",
    "httpPath": "/c9",
    "mapTo": "/cloud9"
}
```

3.2.2 Aplicação real

A título de exemplo, será feita de seguida uma análise às características e dados recolhidos de um dos equipamentos que será registado no *Thing Registry* da plataforma IIoT do projeto SADCoPQ.

A máquina em questão é uma CNC (*Computer Numerical Control*) que efetua operações de alta precisão de modo a automatizar o fabrico de peças altamente complexas de metal, neste caso, alumínio.

A tabela 3.13 descreve os dados recolhidos pelo equipamento em questão.

Valor lido	**Unidade**
timestamp Fanuc	timestamp
timestamp server	timestamp
machine number	inteiro
spindle speed	decimal
spindle torque %	decimal
C1 program number	inteiro
C1 program line	inteiro
C1 working tool	inteiro
C1 working tool position X	decimal
C1 working tool position Y	decimal
C1 working tool position Z	decimal
C1 working tool X geometry offset	decimal
C1 working tool Y geometry offset	decimal

Tabela 3.13 continuação da página anterior

C1 working tool Z geometry offset	decimal
C1 working tool X wear offset	decimal
C1 working tool Y wear offset	decimal
C1 working tool Z wear offset	decimal
C1 working tool X torque %	decimal
C1 working tool Z torque %	decimal
C2 program number	inteiro
C2 program line	inteiro
C2 working tool	inteiro
C2 working tool position X	decimal
C2 working tool position Z	decimal
C2 working tool X geometry offset	decimal
C2 working tool Z geometry offset	decimal
C2 working tool X wear offset	decimal
C2 working tool Z wear offset	decimal
C2 working tool X torque %	decimal
C2 working tool Z torque %	decimal

Tabela 3.13: Valores lidos pela máquina CNC

De seguida, será demonstrado o mapeamento dos dados provenientes do equipamento, para o modelo apresentado na secção 3.2.1.

Sendo o objetivo representar digitalmente a máquina CNC em questão, é do maior interesse registar no *Thing Registry* a maior quantidade possível de informação que acrescente valor à representação virtual do equipamento.

Os dados recolhidos pela plataforma, representados na tabela 3.13, serão mapeados como variáveis no modelo de informação. Para além disso, no interesse de diminuir a quantidade de mensagens a circular pela plataforma, todas estas variáveis serão agrupadas numa agregação. Deste modo, o *Thing Replica* pode apenas pedir o envio de uma única mensagem que contenha as leituras de todas as 27 variáveis contidas na agregação, não sendo necessário requisitar uma mensagem individual para cada uma das variáveis.

Para além desta informação, serão também registados um evento e um atributo para este equipamento, bem como o conjunto de metadados de descrição do equipamento, como nome, descrição, tipo, etc.

O evento é referente à mudança de peça da máquina CNC, quando esta fica desgastada ou estragada.

O atributo contém informação referente à fábrica em que a máquina se encontra, neste caso o ID da fábrica no contexto da plataforma.

A título de exemplo, apenas será representada uma variável das 27, visto que são todas relativamente parecidas e cada variável tem muitos campos para preencher, o que levaria a uma exemplo JSON extremamente extenso.

Posto isto, o pedido POST HTTP para registar este equipamento na plataforma seria o seguinte:

```
{
    "nodeId": "cnc-57",
    "name": "CNC Machine Number 57",
    "description": "Digital representation of CNC
                Machine Number 57",
    "code": "CNC57",
    "manufacturer": "FANUC",
    "model": "Series 30i",
    "type": "device",
    "subType": "heavy machinery",
    "uri": "http://192.168.8.103",
    "protocol": "kafka",
    "attributes": [
        {
            "name": "Factory ID",
            "value": "2450f692-4658-454b-a912-1176488af34e"
        }
    ],
    "configuration": null,
    "nodes": null,
    "variables": [
        {
            "variableId": "spindle-speed",
            "name": "Spindle Speed",
            "description": "Spindle speed in rotations
                        per minute of CNC machine 57",
            "code": "var1",
            "unit": "rpm",
            "resolution": 0.01,
            "type": "decimal",
            "origin": "internal",
```

```
            "permissions": [
                "read"
            ],
            "configuration": null,
            "rangeMax": 50,
            "rangeMin": 0,
            "admissibleValues": null,
            "readingsSize": null,
            "showInDashboard": true
        },
        (as outras 26 variáveis)
    ],
    "aggregations": [
        {
            "aggregationId": "cnc-57-aggregation",
            "name": "CNC-57 variables bundle",
            "description": "Bundle of all variables
                        of CNC machine 57",
            "code": "cnc57agg",
            "variables":
            {
                "spindle-speed",
                "C1-program-number",
                "C1-program-line",
                "C1-working-tool",
                "C1-working-tool-position-X",
                "C1-working-tool-position-Y",
                "C1-working-tool-position-Z",
                "C1-working-tool-X-geometry-offset",
                "C1-working-tool-Y-geometry-offset",
                "C1-working-tool-Z-geometry-offset",
                "C1-working-tool-X-wear-offset",
                "C1-working-tool-Y-wear-offset",
                "C1-working-tool-Z-wear-offset",
                "C1-working-tool-X-torque",
                "C1-working-tool-Z-torque",
                "C2-program-number",
                "C2-program-line",
                "C2-working-tool",
                "C2-working-tool-position-X",
```

```
                "C2-working-tool-position-Z",
                "C2-working-tool-X-geometry-offset",
                "C2-working-tool-Z-geometry-offset",
                "C2-working-tool-X-wear-offset",
                "C2-working-tool-Z-wear-offset",
                "C2-working-tool-X-torque",
                "C2-working-tool-Z-torque"
            }
        }
    ],
    "events": [
        {
            "eventId": "tool-change",
            "name": "CNC-57 tool change",
            "description": "Used to register when a tool change
                            occurs due to breakage or wear damage",
            "type": "replacement",
            "readingsSize": null,
            "properties": [
                {
                    "name": "new-tool-id",
                    "description": "id of the new tool",
                    "type": "string"
                }
            ]
        }
    ],
    "links": null,
    "services": null
}
```

É relevante notar que a interface gráfica da API não é âmbito da dissertação, logo, apenas será desenvolvida mais tarde. Assim, apenas foi demonstrado acima o texto JSON que seria enviado no pedido HTTP com os campos necessários preenchidos para registar a máquina CNC. Caso a interface gráfica estivesse desenvolvida, a informação seria colocada num formulário e mapeada para o JSON apresentado.

Na próxima secção serão apresentados os detalhes da API desenvolvida, que recebe o pedido descrito acima e regista o equipamento no módulo *Thing Registry* da plataforma IIoT.

3.3 REST API

3.3.1 Arquitetura

A arquitetura da API foi desenvolvida pensando num modelo multicamadas com o objetivo de criar abstração entre o utilizador, os serviços, e a base de dados.

A maior vantagem de uma arquitetura multicamadas, exemplificada na figura 3.2, é o chamado acoplamento solto (*loose coupling*), que oferece um elevado nível de abstração entre os vários elementos de uma aplicação. Para além de proporcionar um maior nível de segurança, a manutenção e processo de desenvolvimento do software tornam-se extremamente mais simples, visto que uma camada pode ser significativamente alterada sem afetar as outras camadas, podendo também ser testada facilmente sem necessitar do funcionamento total da aplicação completa. Esta arquitetura é especialmente apelativa para projetos de média-alta dimensão, porque o nível de complexidade é escondido pelas várias camadas, oferecendo melhor escalabilidade, bem como uma tolerância a erros mais elevada, o que consequentemente facilita o desenvolvimento da aplicação [36].

Porém, este tipo de arquitetura oferece também algumas desvantagens, sendo a principal o efeito no desempenho computacional da aplicação. Este sofre devido ao *overhead* excessivo que é necessário para fazer a informação passar entre as diversas camadas, ao invés de ir buscar a informação requisitada diretamente ao local aonde esta se encontra.

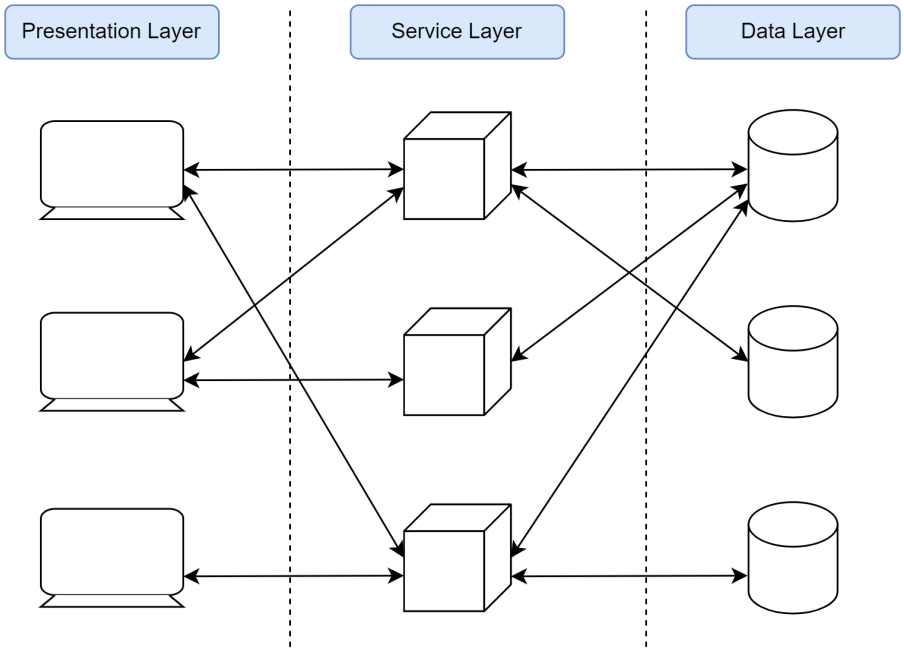

Figura 3.2: Arquitetura multicamadas. Adaptado de: [37]

O diagrama representado na figura 3.3 descreve, a um alto nível, a estrutura interna da API, dividida em 4 partes:

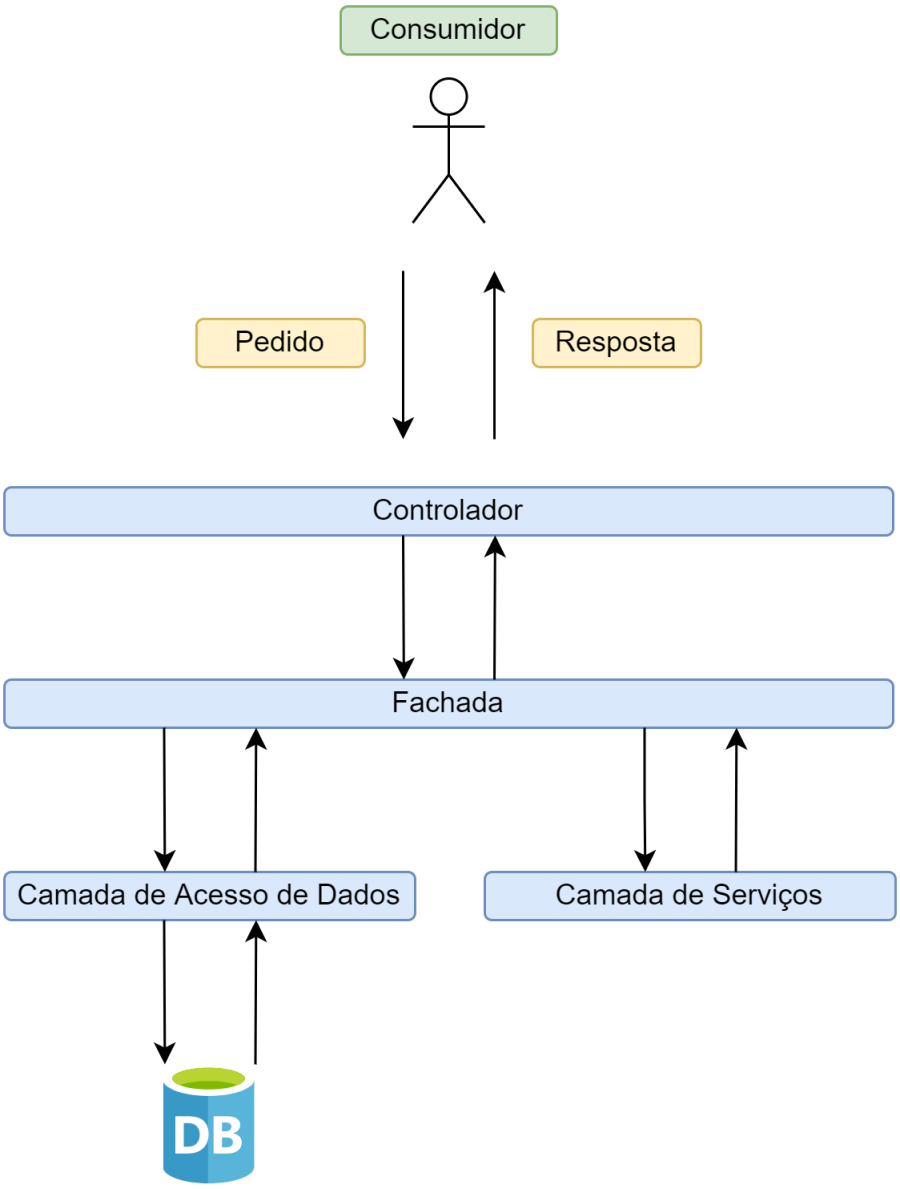

Figura 3.3: Visão de alto nível das camadas backend da API

- **Camada de Acesso de Dados:** Referida como *Data Access Layer* (DAL), esta camada fornece um acesso simplificado à informação contida numa base de dados sem depender da estrutura interna da mesma. Deste modo, é alcançado um acoplamento solto entre a base de dados e o resto do sistema, facilitando o seu desenvolvimento e oferecendo a possibilidade de mais tarde alterar a base de dados sem que isso afete a funcionalidade do sistema, nem que seja requerida a alteração a baixo nível do sistema (p. ex. alteração do código). Resumidamente, o DAL esconde a complexidade do mecanismo de armazenamento ao resto do sistema;

- **Camada de Serviços:** A camada de serviços contem toda a lógica de negócio (*business*

logic) da API. Os serviços recebem um objeto proveniente da base de dados, neste caso um nó, e efetuam as alterações desejadas, enviando de volta este objeto para ser guardado na base de dados e, por vezes, ser enviado (totalmente ou em parte), ao cliente, como resposta a um pedido.

- **Fachada:** Esta camada, embora não seja estritamente obrigatória, proporciona um nível de abstração extra dentro da API, porque evita a comunicação entre o DAL, a camada de serviços e o utilizador. A fachada recebe um pedido do controlador (e por vezes um objeto proveniente do corpo do pedido HTTP), vai buscar o nó necessário à base de dados, e envia este nó, juntamente com o corpo / objeto recebido no pedido, se este existir, à camada de serviços, da qual espera por uma resposta para mandar de volta ao controlador, que a reencaminhará para o cliente.

  A fachada é também responsável por enviar notificações ao *message broker* sobre alterações efetuadas em qualquer nó registado no *Thing Registry*, assim como na criação de novos nós. O *Thing Registry* consome também notificações, enviadas pelo *Service Registry* quando um serviço ou link de um determinado nó é eliminado. Quando uma destas notificações é detetada, a fachada envia o respetivo nó para a camada de serviços, responsável por apagar o serviço ou o link do nó, para este ser atualizado na base de dados.

- **Controlador:** Esta camada é responsável por toda a comunicação entre o utilizador e a API. O controlador consome os pedidos HTTP efetuados pelo utilizador e reencaminha estes dados aos serviços necessários, nas camadas mais abaixo. Depois de recebida uma resposta destes serviços, o controlador envia uma resposta HTTP ao cliente, com toda a informação relevante, como o código de resposta HTTP, cabeçalhos e corpo da resposta, neste caso, um ou vários objetos JSON.

3.3.2 Implementação

A API foi desenvolvida com a *framework* Spring que é uma tecnologia em código aberto baseada em Java. Mais concretamente, foi utilizado Spring Boot, que ajuda a simplificar o desenvolvimento e configuração da aplicação, e Spring Data MongoDB, que lida com a implementação e configuração de um repositório MongoDB.

De seguida, serão apresentados alguns detalhes sobre a implementação de modelos, interfaces e camadas desenvolvidas.

3.3.2.1 Diagrama UML do modelo de dados

Na secção 3.2.1 foi apresentado o modelo usado para representar digitalmente os equipamentos. A figura 3.4 apresenta em forma de diagrama UML, as classes utilizadas para implementar o modelo, bem como as relações entre elas.

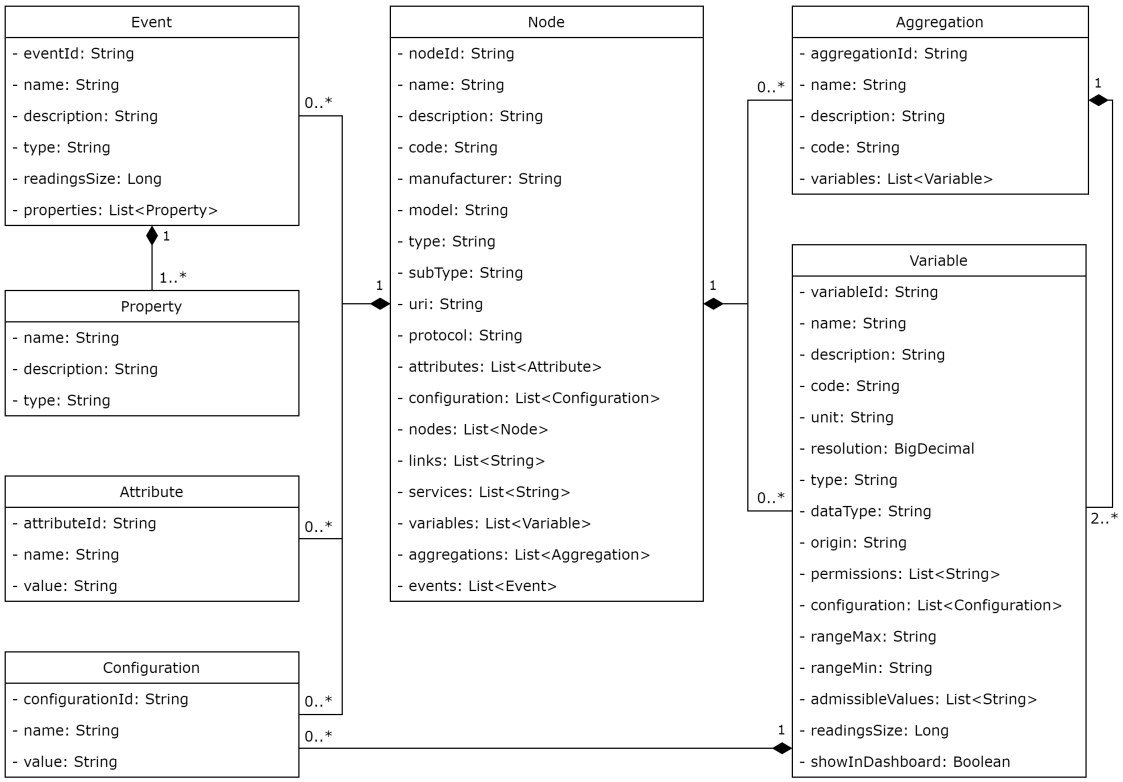

Figura 3.4: Diagrama UML do modelo

Este diagrama demonstra algumas das restrições impostas nos objetos. Uma agregação tem de obrigatoriamente conter pelo menos duas variáveis, e um evento tem de obrigatoriamente conter pelo menos uma propriedade.

É de notar que, por uma questão de simplicidade, neste diagrama UML não estão apresentados os métodos das classes.

3.3.2.2 Camada de acesso de dados

Um repositório é um mecanismo para encapsular o armazenamento de informação numa ou mais bases de dados, bem como os pedidos para consulta dos dados armazenados [38]. Para garantir abstração, os repositórios oferecem uma interface de acesso à informação que esconde a complexidade dos pedidos (*queries*) à base de dados, bem como o mapeamento para os objetos desejados.

Embora o DAL possa ser composto por vários repositórios, no contexto da aplicação desenvolvida, o DAL é composto unicamente pelo repositório configurado através do Spring Data MongoDB. Deste modo, a camada de acesso aos dados é totalmente controlada por este repositório através da interface disponibilizada.

Embora a interface do repositório seja composta por uma série de métodos úteis para enviar

e receber dados armazenados, apenas alguns destes são necessários para obter as funcionalidades pretendidas, logo, foi feita uma interface mais pequena, descrita na figura 3.5, utilizando os métodos fornecidos pela interface principal.

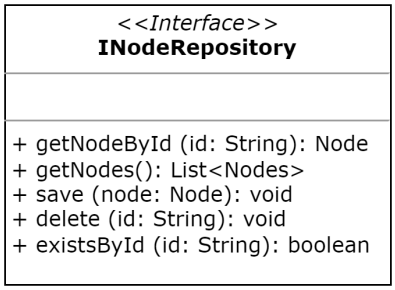

Figura 3.5: Interface do repositório

3.3.2.3 Controlador

O controlador lida com os seguintes pedidos HTTP, representados na tabela 3.14.

Rota	Método	Descrição
/nodes	GET	Retorna todos os nós
	POST	Regista um nó
/nodes/{nodeId}	GET	Retorna o nó com o ID especificado
	DELETE	Apaga o nó
	PUT	Atualiza os dados do nó
/nodes/{nodeId} /variables	GET	Retorna as variáveis de um nó
	POST	Regista uma variável no nó
/nodes/{nodeId} /variables/{variableId}	GET	Retorna uma variável de um nó
	DELETE	Apaga a variável
	PUT	Atualiza os dados da variável
/nodes/{nodeId}/variables /{variableId}/configuration	GET	Retorna as configurações de uma variável
	POST	Regista uma configuração numa variável
/nodes/{nodeId}/variables /{variableId}/configuration /{configurationId}	GET	Retorna uma configuração de uma variável
	DELETE	Apaga a configuração
	PUT	Atualiza os dados da configuração
/nodes/{nodeId} /events	GET	Retorna os eventos de um nó
	POST	Regista um evento num nó
/nodes/{nodeId} /events/{eventId}	GET	Retorna um evento de um nó
	DELETE	Apaga o evento

Tabela 3.14 continuação da página anterior

	PUT	Atualiza os dados do evento
/nodes/{nodeId}/events	GET	Retorna as propriedades de um evento
/{eventId}/properties	POST	Regista uma propriedade num evento
/nodes/{nodeId}	GET	Retorna uma propriedade de um evento
/events/{eventId}	DELETE	Apaga a propriedade
/property/{name}	PUT	Atualiza os dados da propriedade
/nodes/{nodeId}	GET	Retorna as agregações de um nó
/aggregations	POST	Regista uma agregação num nó
/nodes/{nodeId}	GET	Retorna uma agregação de um nó
/aggregations	DELETE	Apaga a agregação
/{aggregationId}	PUT	Atualiza os dados da agregação
/nodes/{nodeId}/aggregations	GET	Retorna as variáveis de uma agregação
/{aggregationId}/variables	POST	Regista uma variável numa agregação
/nodes/{nodeId}	GET	Retorna os atributos de um nó
/attributes	POST	Regista um atributo num nó
/nodes/{nodeId}	GET	Retorna um atributo de um nó
/attributes	DELETE	Apaga o atributo
/{attributeId}	PUT	Atualiza os dados do atributo
/nodes/{nodeId}	GET	Retorna as configurações de um nó
/configuration	POST	Regista uma configuração num nó
/nodes/{nodeId}	GET	Retorna uma configuração de um nó
/configuration	DELETE	Apaga a configuração
/{configurationId}	PUT	Atualiza os dados da configuração
/nodes/{nodeId}	GET	Retorna os links de um nó
/links	POST	Regista um link num nó
/nodes/{nodeId}	GET	Retorna um link de um nó
/links	DELETE	Apaga o link
/{linkId}	PUT	Atualiza os dados do link
/nodes/{nodeId}	GET	Retorna os serviços de um nó
/services	POST	Regista um serviço num nó
/nodes/{nodeId}	GET	Retorna um serviço de um nó
/services	DELETE	Apaga o serviço
/{serviceId}	PUT	Atualiza os dados do serviço
/nodes/{nodeId}	GET	Retorna os sub-nós de um nó
/nodes		

Tabela 3.14 continuação da página anterior

POST	Regista um sub-nó num nó

Tabela 3.14: Pedidos HTTP

Entre os vários pedidos apresentados, o servidor responde com um dos seguintes códigos HTTP de resposta:

- **200 OK**:
  Código de sucesso. Se o cliente fizer um pedido e obter uma resposta, este código é enviado pelo servidor.
- **201 CREATED:**
  Quando o cliente envia um pedido POST com um corpo para criar um objeto, se não ocorrerem erros, este é o código enviado pelo servidor.
- **400 BAD REQUEST:**
  No caso do cliente enviar um pedido inválido, neste caso um POST ou um PUT com um corpo incorreto, por exemplo se um campo obrigatório do objeto não estiver preenchido, este código é enviado pelo servidor e nenhuma alteração é feita na base de dados.
- **404 NOT FOUND:**
  Quando algum ID pedido pelo cliente não existe, este código é enviado pelo servidor a informar que o objeto pedido não existe.
- **409 CONFLICT:**
  Este código indica que houve um conflito entre o pedido e o estado atual do objeto a ser alterado, por exemplo, se dois clientes diferentes tentarem alterar o mesmo objeto, apenas o primeiro consegue fazê-lo com sucesso, o segundo recebe um erro com este código, a informar que a sua versão do objeto já não está atualizada.

3.3.2.4 Fachada

Um *Data Transfer Object* (DTO) é um objeto usado para transportar informação entre processos. Geralmente existem dois motivos para utilizar DTOs:

1. **Poupança de recursos:** A comunicação entre processos é geralmente feita recorrendo a interfaces e serviços remotos (p. ex. a comunicação entre cliente e servidor no caso da API desenvolvida aqui). Cada pedido e resposta tem um custo na rede que estabelece esta comunicação. Posto isto, é boa prática adotar métodos eficientes de comunicação que utilizem a quantidade mínima de recursos necessários. Os DTOs são então regularmente usados neste contexto para agregar a informação toda de resposta a um pedido, sendo esta enviada num único pacote para o cliente.

2. **Segurança:** Responder a um pedido com um DTO garante um nível de desacoplamento entre o formato dos dados dentro da plataforma e o formato dos dados fornecidos ao cliente. Assim, é garantido um nível extra de segurança que esconde informação que poderia de outro modo ser utilizada para efetuar certos ataques à plataforma.

A fachada desenvolvida para esta API envia DTOs à camada de serviços, recebidos pelo controlador em pedidos POST e PUT (p. ex. eventCreateDto e eventUpdateDto), e recebe da camada de serviços DTOs para enviar ao controlador, em resposta a pedidos GET (p. ex. eventReadDto).

Todas as interfaces da fachada seguem aproximadamente o mesmo padrão que a fachada exemplificada na figura 3.6.

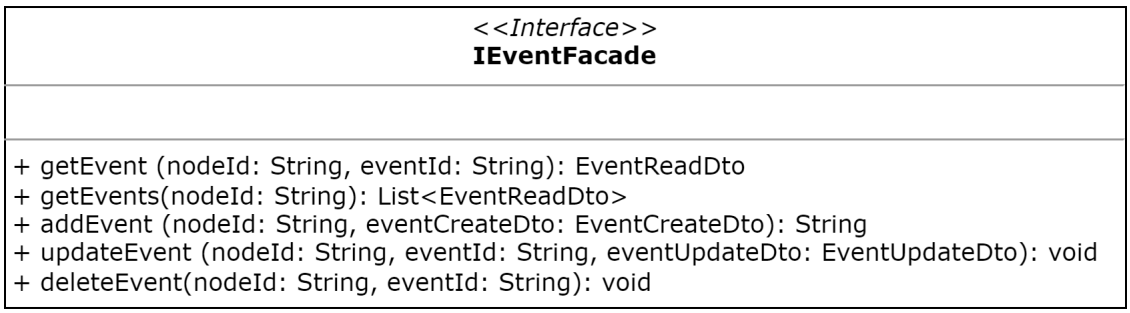

Figura 3.6: Interface da fachada de eventos

3.3.2.5 Camada de serviços

A camada de serviços contém toda a lógica de negócio da aplicação. Aqui são feitas as alterações nos nós recebidos como parâmetro e é também aqui efetuado o mapeamento de DTOs para os modelos de domínio e vice-versa.

Todas as interfaces dos serviços seguem aproximadamente o mesmo padrão que o serviço exemplificado na figura 3.7.

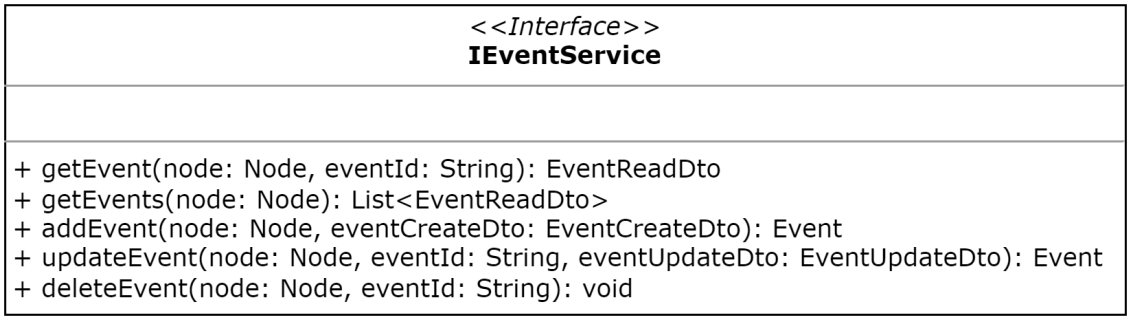

Figura 3.7: Interface do serviço de eventos

3.3.2.6 Diagramas de sequência

A título de exemplo, serão apresentados de seguida dois diagramas de sequência para demonstrar a interação entre as diferentes camadas da API.

A figura 3.8 representa as interações que ocorrem ao adicionar um evento a um nó. Ao receber um pedido POST do cliente, juntamente com um DTO com os dados de um novo evento, o controlador envia esta informação à fachada que por sua vez pede ao repositório o objeto / nó ao qual se pretende adicionar o evento. Nesta fase existem duas possibilidades, ou o nó existe ou não existe.

Se o nó não existir, é gerada uma exceção no programa e esta é apanhada pelo controlador, que informa o cliente que o nó não foi encontrado, através de um código HTTP 404 (NOT FOUND), incluindo também uma mensagem com mais detalhes sobre o erro, neste caso, que o nó correspondente ao ID fornecido pelo cliente não foi encontrado no repositório.

Se o nó existir, este é enviado à camada de serviços juntamente com o DTO, através do método addEvent da interface IEventService, apresentada anteriormente na figura 3.7. Novamente, existem aqui duas situações possíveis, ou o DTO fornecido é válido ou não é.

Se não for, por exemplo quando um campo obrigatório para criar um evento é enviado em branco, ou o ID do evento contem caracteres inválidos ou já existe, é gerada uma exceção no programa que rapidamente é detetada pelo controlador. Este responde ao cliente com um código HTTP 400 (BAD REQUEST), informando-o que o DTO enviado não é válido, não sendo por isso possível criar o evento.

Se o DTO for válido, o evento é mapeado de DTO para o modelo de domínio e é adicionado ao nó, este é enviado de volta para a fachada e de seguida para o controlador que informa o cliente de que o evento foi adicionado com sucesso, através de uma mensagem com o código HTTP 201 (CREATED) e o ID do evento.

O funcionamento da API em resposta a um pedido para editar, neste caso um evento, é muito parecido ao descrito para adicionar um evento. São efetuadas algumas verificações extra, como por exemplo se o ID do evento a atualizar existe ou não.

A figura 3.9 representa as interações que ocorrem entre as diferentes camadas da API ao consultar um evento de um nó. O funcionamento é semelhante aos pedidos para adicionar e editar, no entanto, a aplicação não recebe um DTO do cliente. Após o serviço encontrar o evento pedido dentro do nó que recebe da fachada, é feito o mapeamento para um DTO de leitura, que é enviado para a fachada, para o controlador, e de seguida para o cliente, juntamente com um código HTTP 200 (CREATED).

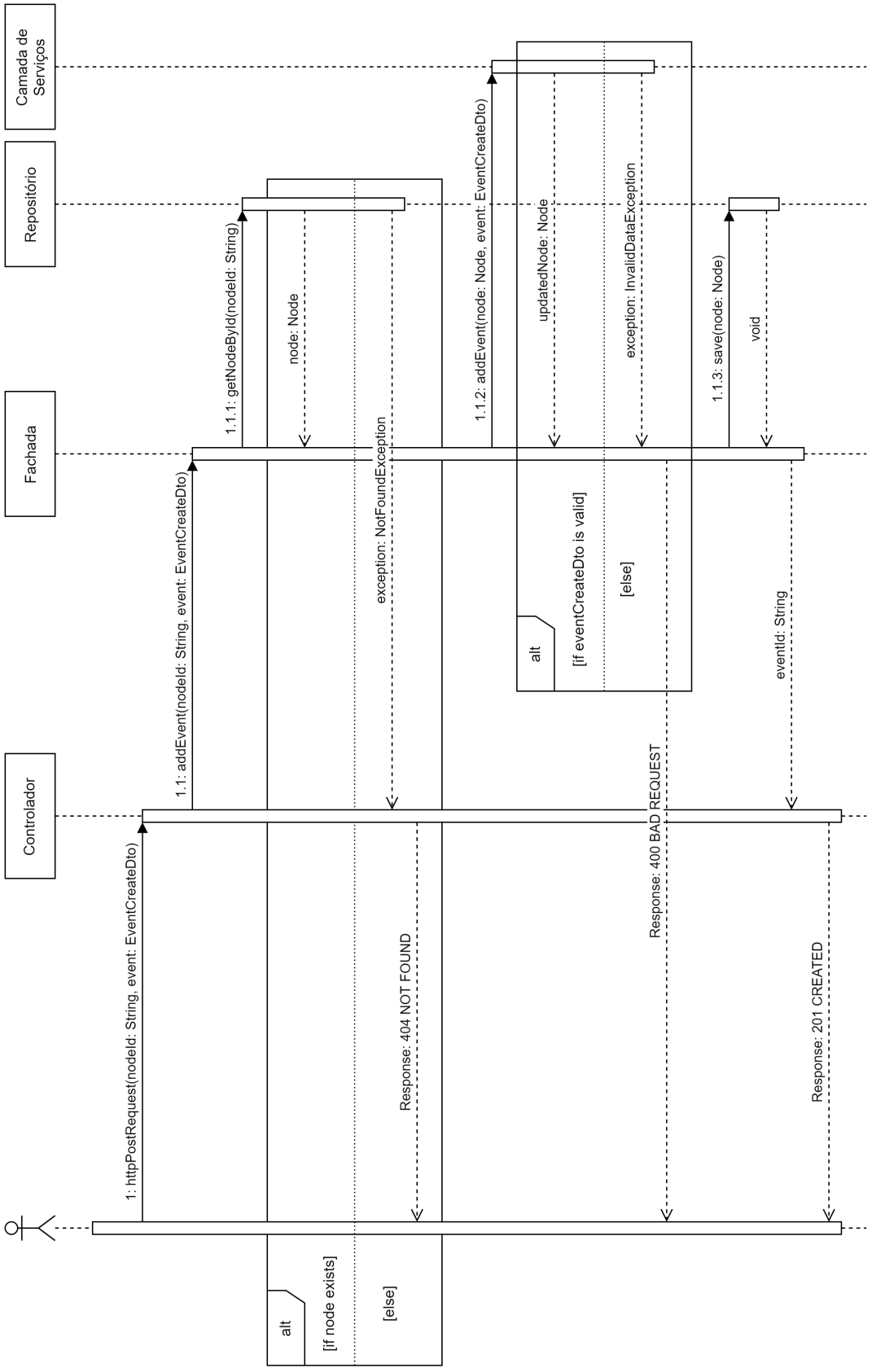

Figura 3.8: Diagrama de sequência do pedido para registar um evento

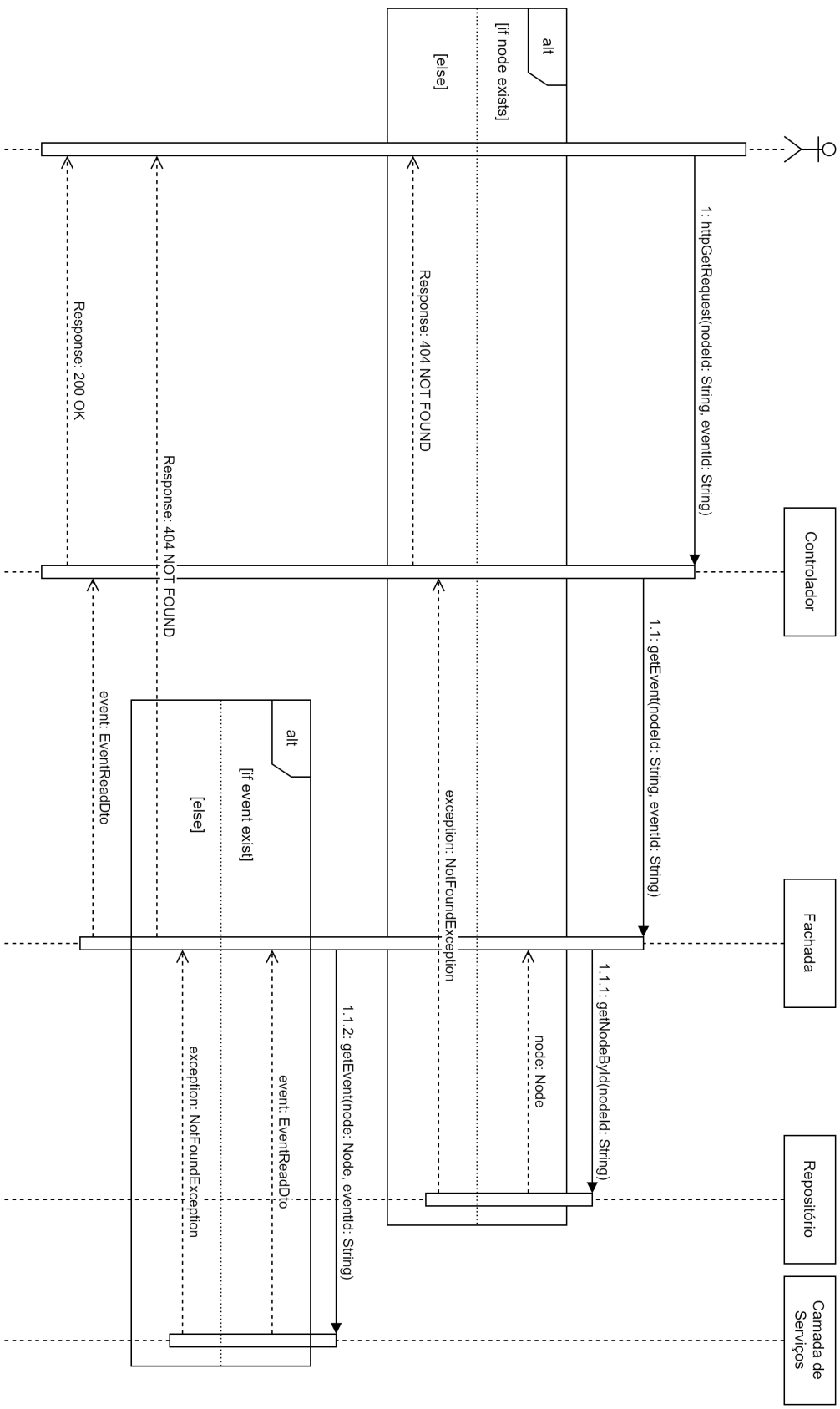

Figura 3.9: Diagrama de sequência do pedido para ler um evento

Capítulo 4

Conclusão

Na secção 1.3, foram colocados dois objetivos específicos para esta dissertação: a implementação de um modelo de informação para representar digitalmente equipamentos industriais, e o desenvolvimento de uma REST API para registar e gerir estes dispositivos.

Inicialmente foi feita a revisão de literatura onde foram explorados os conceitos de IIoT, DTs, Modelação de ativos, e APIs. Foi dada particular atenção ao AAS, visto que este é o modelo mais completo e mais interoperável, recomendado pelas principais organizações da indústria 4.0 para representar digitalmente ativos.

Foi no entanto concluído que, devido à complexidade, tamanho da equipa de desenvolvimento e prazos de entrega do projeto SADCoPQ, estes padrões já existentes não seriam opções viáveis. Foi então descrito o modelo proprietário desenvolvido, baseado em nós lógicos para representar digitalmente os equipamentos. Para terminar, os detalhes da REST API desenvolvida e implementada foram também descritos.

Em suma, foram alcançados todos os objetivos esperados durante o período da dissertação.

Como trabalho futuro no *Thing Registry*, espera-se desenvolver uma interface gráfica com um serviço de autenticação que melhore a segurança do módulo e facilite a sua comunicação com os utilizadores. Com esta implementação, seria possível limitar quem tem autorização para registar novos dispositivos na plataforma e permitir, por exemplo, um número mais alargado de utilizadores consultá-los. Para além disso, através de uma interface gráfica, é mais fácil para utilizadores menos experientes registarem novos dispositivos sem necessitarem de qualquer conhecimento sobre o funcionamento do módulo. Seria simplesmente apresentado um formulário com os campos a preencher, com indicações sobre quais destes são obrigatórios ou opcionais, juntamente com eventuais informações relevantes, como por exemplo, diferentes respostas possíveis para certos campos do objeto.

Num contexto mais geral sobre a modelação de DTs, visto que se trata de um tema relativamente recente, é importante salientar que ainda existe uma margem considerável para melhorias. Idealmente, é desejável que a comunidade académica concorde eventualmente num único padrão uniforme que defina os métodos a seguir para representar digitalmente ativos, e que ferramentas possam ser desenvolvidas para facilitar em grande parte o processo. O AAS continua, neste

momento, a ser intensamente desenvolvido, tendo sido recentemente publicada, em novembro de 2021, a primeira versão da parte 2 do manual de detalhes do AAS, desta vez focada na gestão de informação de AASs através de APIs. Uma terceira parte será também publicada no futuro, que irá detalhar como desenvolver infraestruturas para consultar, registar e interligar AASs, semelhante ao módulo *Thing Registry* desenvolvido durante esta dissertação.

Referências

[1] F. Tao, H. Zhang, A. Liu, e A. Y. C. Nee. "digital twin in industry: state-of-the-art". *IEEE Transactions on Industrial Informatics*, 15(4):2405–2415, Apr. 2019. doi:10.1109/TII.2018.2873186.

[2] M. Jacoby e T. Usländer. "digital twin and internet of things—current standards landscape". *Applied Sciences*, 10(18), Sep. 2020. doi:10.3390/app10186519.

[3] S. Bader *et al. Details of the Asset Administration Shell. Part 1 - The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC01)*. Federal Ministry for Economic Affairs and Climate Action, Berlin, 2020.

[4] World Wide Web Consortium (W3C). Web of Things Thing Description, 2020. (acessed on: June 22, 2022). URL: https://www.w3.org/TR/wot-thing-description/.

[5] Microsoft. Digital Twins Definition Language (DTDL), 2022. (acessed: June 22, 2022). URL: https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v2/dtdlv2.md.

[6] M. Liu, S. Fang, H. Dong, e C. Xu. "review of digital twin about concepts, technologies, and industrial applications". *Journal of Manufacturing Systems*, 58:346–361, Jan. 2021. doi:10.1016/j.jmsy.2020.06.017.

[7] T. Burns, J. Cosgrove, e F. Doyle. "a review of interoperability standards for industry 4.0". *Procedia Manufacturing*, 38:646–653, June 2019. doi:10.1016/j.promfg.2020.01.083.

[8] Y. Lu. "industry 4.0: A survey on technologies, applications and open research issues". *Journal of Industrial Information Integration*, 6:1–10, June 2017. doi:10.1016/j.jii.2017.04.005.

[9] B. Boss, S. Malakuti, S. Lin, T. Usländer, E. Clauer, M. Hoffmeister, e S. Ljiljana. "Digital Twin and Asset Administration Shell Concepts and Application in the Industrial Internet and Industrie 4.0". Report, Industrial Internet Consortium and Plattform Industrie 4.0, 2020. Acessed on June 4, 2022. [Online]. URL: https://www.iiconsortium.org/pdf/Digital-Twin-and-Asset-Administration-Shell-Concepts-and-Application-Joint-Whitepaper.pdf.

[10] L. Sakurada, P. Leitao, e F. De la Prieta. "towards the digitization using asset administration shells". Em *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, páginas 1–6, Oct. 2021. doi:10.1109/IECON48115.2021.9589370.

[11] M. Hermann, T. Pentek, e B. Otto. "Design Principles for Industrie 4.0 Scenarios: A Literature Review", Jan. 2015. doi:10.13140/RG.2.2.29269.22248.

[12] H. Lasi, P. Fettke, H. Kemper, T. Feld, e M. Hoffmann. "industry 4.0". *Business & Information Systems Engineering*, 6(4):239–242, June 2014. doi:10.1007/s12599-014-0334-4.

[13] H. Boyes, B. Hallaq, J. Cunningham, e T. Watson. "the industrial internet of things (iiot): An analysis framework". *Computers in Industry*, 101:1–12, Oct. 2018. doi:10.1016/j.compind.2018.04.015.

[14] A. Gluhak, O. Vermesan, R. Bahr, F. Clari, T. MacchiaMaria, T. Delgado, A. Hoeer, F. Bösenberg, M. Senigalliesi, e V. Barchetti. "Report on IoT platforms activities". Report D03.01, European Commission, Oct. 2016. Acessed on May 28, 2022. [Online]. URL: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5ad603377&appId=PPGMS.

[15] M. Grieves. "Origins of the Digital Twin Concept", Oct. 2016. doi:10.13140/RG.2.2.26367.61609.

[16] M. Grieves e J. Vickers. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, páginas 85–113. Springer, Cham, 2017. doi:10.1007/978-3-319-38756-7_4.

[17] E. Negri, L. Fumagalli, e M. Macchi. "a review of the roles of digital twin in cps-based production systems". *Procedia Manufacturing*, 11:939–948, June 2017. doi:10.1016/j.promfg.2017.07.198.

[18] W. Kritzinger, M. Karner, G. Traar, J. Henjes, e W. Sihn. "digital twin in manufacturing: A categorical literature review and classification". *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. doi:10.1016/j.ifacol.2018.08.474.

[19] L. Stojanovic, T. Usländer, F. Volz, C. Weißenbacher, J. Müller, M. Jacoby, e T. Bischoff. "methodology and tools for digital twin management - the fa3st approach". *IoT*, 2(4):717–740, Oct. 2021. doi:10.3390/iot2040036.

[20] C. Wagner, J. Grothoff, U. Epple, R. Drath, S. Malakuti, S. Grüner, M. Hoffmeister, e P. Zimermann. "the role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant". Em *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, páginas 1–8, Sep. 2017. doi:10.1109/ETFA.2017.8247583.

[21] S. Cavalieri e M. G. Salafia. "a model for predictive maintenance based on asset administration shell". *Sensors*, 20(21), Oct. 2020. doi:10.3390/s20216028.

[22] M. Platenius-Mohr, S. Malakuti, S. Grüner, J. Schmitt, e T. Goldschmidt. "file- and api-based interoperability of digital twins by model transformation: An iiot case study using asset administration shell". *Future Generation Computer Systems*, 113:94–105, Dec. 2020. doi:10.1016/j.future.2020.07.004.

[23] B. Lydon. RAMI 4.0 Reference Architectural Model for Industrie 4.0, 2021. (acessed on: June 6, 2022). URL: https://www.isa.org/intech-home/2019/march-april/features/rami-4-0-reference-architectural-model-for-industr.

[24] S. Cavalieri e M. G. Salafia. "asset administration shell for plc representation based on iec 61131–3". *IEEE Access*, 8:142606–142621, Aug 2020. doi:10.1109/ACCESS.2020.3013890.

[25] P. Weiß, B. Koelmel, e R. Bulander. "digital service innovation and smart technologies: developing digital strategies based on industry 4.0 and product service systems for the renewal energy sector". Em *RESER*, Naples, Italy, Sep. 2016.

[26] F. Pethig, O. Niggemann, e A. Walter. "towards industrie 4.0 compliant configuration of condition monitoring services". Em *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, páginas 271–276, July 2017. doi:10.1109/INDIN.2017.8104783.

[27] H. Bedenbender, M. Billman, U. Epple, T. Hadlich, M. Hankel, R. Heidel, e O. Hillermeier. "Examples of the Asset Administration Shell for Industrie 4.0 Components". Report, ZVEI, 2017.

[28] A. Seif, C. Toro, e H. Akhtar. "implementing industry 4.0 asset administrative shells in mini factories". *Procedia Computer Science*, 159:495–504, 2019. doi:10.1016/j.procs.2019.09.204.

[29] J. Ofoeda, R. Boateng, e J. Effah. "application programming interface (api) research: A review of the past to inform the future". *International Journal of Enterprise Information Systems (IJEIS)*, 15(3):76–95, July 2019. doi:10.4018/IJEIS.2019070105.

[30] C. Rodríguez, M. Baez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, e G. Percannella. "rest apis: A large-scale analysis of compliance with principles and best practices". Em *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9671, páginas 21–39, May 2016. doi:10.1007/978-3-319-38791-8_2.

[31] R. T. Fielding. *"Architectural Styles and the Design of Network-based Software Architectures"*. Doctoral dissertation, University of California, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[32] E. Tantik e R. Anderl. "integrated data model and structure for the asset administration shell in industrie 4.0". *Procedia CIRP*, 60:86–91, 2017. doi:10.1016/j.procir.2017.01.048.

[33] E. Tantik e R. Anderl. "potentials of the asset administration shell of industrie 4.0 for service-oriented business models". *Procedia CIRP*, 64:363–368, 2017. doi:10.1016/j.procir.2017.03.009.

[34] J. Contreras, J. Melo, e J. D. Pastrana. "developing of industry 4.0 applications". *International Journal of Online Engineering (iJOE)*, 13:30, Nov. 2017. doi:10.3991/ijoe.v13i10.7331.

[35] H. Bedenbender *et. al.* "Relationships between I4.0 Components - Composite Components and Smart Production". Report, Plattform Industrie 4.0, Berlin, 2017. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/relationships-i40-components.pdf?__blob=publicationFile&v=5.

[36] Microsoft. Layered Application, 2014. (acessed on: June 17, 2022). URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff650258(v=pandp.10).

[37] P. D. Manuel e J. AlGhamdi. "a data-centric design for n-tier architecture". *Information Sciences*, 150(3):195–206, Apr. 2003. doi:10.1016/S0020-0255(02)00377-8.

[38] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, Aug. 2003.