Capítulo

5

Construção de Sistemas de Gêmeos Digitais: Uma Abordagem baseada em Middleware

André Almeida (IFRN), Lucas Pereira (IFRN & UFRN), Thais Batista (UFRN), Everton Cavalcante (UFRN), Flavia C. Delicato (UFF), Rebeca Motta (UFF)

Resumo

Os gêmeos digitais (do inglês, digital twins – DTs) são réplicas digitais dinâmicas de objetos físicos, sistemas ou processos do mundo real que são continuamente atualizadas de forma bidirecional com dados de sensores e dispositivos computacionais. Ao refletir, em tempo quase real, o estado e o comportamento de suas contrapartes físicas, DTs possibilitam monitorar, analisar e controlar sistemas complexos. A construção de sistemas de DTs apresenta desafios significativos decorrentes da heterogeneidade das tecnologias e dos dados envolvidos, bem como da complexidade de manter sincronizadas as interações entre elementos físicos e virtuais. Este capítulo tem por objetivo introduzir os principais conceitos e desafios relacionados à construção de sistemas de DTs e discutir como plataformas de middleware podem ser utilizadas nesse contexto. Para ilustrar esse uso na prática, utilizamos o MidDiTS, um middleware projetado especificamente para dar suporte à construção e à operação de sistemas de DTs por meio de um processo sistemático de desenvolvimento.

Abstract

Digital twins (DTs) are dynamic digital replicas of real-world physical objects, systems, or processes that are continually updated bidirectionally with data from sensors and computing devices. By reflecting the state and behavior of their physical counterparts in near real-time, DTs enable the monitoring, analysis, and control of complex systems. Building DT systems presents significant challenges stemming from the heterogeneity of the technologies and data involved, as well as the complexity of synchronizing interactions between physical and virtual elements. This chapter introduces the main concepts and challenges of building DT systems and discusses how middleware platforms can be used in this context. To illustrate this use in practice, we employ MidDiTS, a middleware system specifically designed to support the construction and operation of DT systems through a systematic development process.

 ©2026 SBC — Soc. Bras. de Computação

5.1. Introdução

A crescente digitalização de sistemas físicos, impulsionada pela proliferação de dispositivos conectados e pela expansão da capacidade computacional, tem dado origem a novas formas de monitorar, compreender e otimizar o comportamento de entidades do mundo real. Nesse cenário, o conceito de *gêmeo digital* (do inglês, *digital twin* – DT) emergiu como uma das tecnologias mais promissoras da Indústria 4.0 e da transformação digital contemporânea, convergindo paradigmas e tecnologias como a Internet das Coisas (IoT), sistemas ciberfísicos, computação em nuvem, simulação computacional e Inteligência Artificial para viabilizar uma integração dinâmica entre os domínios físico e digital.

A Norma Internacional ISO/IEC 30173:2023 formaliza a definição de um DT como a "representação digital de uma entidade-alvo com conexões de dados que permitem a convergência entre estados físicos e digitais a uma taxa apropriada de sincronização" [ISO 2023]. Sob essa ótica, um DT não é apenas um modelo estático, mas uma réplica digital dinâmica, baseada em software, continuamente atualizada com dados provenientes de sensores e dispositivos computacionais ao longo de seu ciclo de vida. Essa integração permite que o modelo virtual reflita, em tempo quase real, o estado e o comportamento de seu correspondente no mundo real. No entanto, um diferencial crítico de um DT em relação aos modelos tradicionais de simulação computacional ou aos modelos digitais construídos apenas a partir dos dados coletados das entidades do mundo real (ao que a literatura denomina sombras digitais) é a sincronização bidirecional automatizada de dados entre os mundos real e virtual [Kritzinger et al. 2018]. Dessa forma, um DT propriamente dito permite que as mudanças na entidade do mundo real sejam refletidas na representação virtual e vice-versa, possibilitando não apenas o monitoramento, mas também o controle e a otimização direta da entidade-alvo.

Estruturalmente, um sistema de DT constitui-se como um ecossistema computacional distribuído que integra modelos digitais, mecanismos de comunicação, serviços de sincronização e componentes de análise e de atuação [Liu et al. 2021]. Essas funcionalidades têm motivado a adoção de sistemas de DTs em diversos domínios de aplicação [Fuller et al. 2020, Iranshahi et al. 2025]. Por exemplo, na indústria de manufatura, sistemas de DTs são utilizados para monitoramento em tempo real e manutenção preditiva de ativos, bem como para otimização de processos produtivos [Tao et al. 2019]. Em cidades inteligentes, sistemas de DTs permitem simular cenários urbanos complexos a partir de dados coletados por sensores de IoT, contribuindo para um planejamento e uma gestão mais eficientes [Shahat et al. 2021]. No setor agrícola, sistemas de DTs viabilizam a criação de réplicas digitais de propriedades rurais inteiras, integrando dados de sensores de solo, condições de plantas, previsões meteorológicas e imagens de satélite para apoiar decisões sobre irrigação, fertilização e manejo de pragas [Pylianidis et al. 2021].

Apesar do seu potencial, o desenvolvimento de sistemas de DTs envolve um processo complexo com múltiplos desafios. Do ponto de vista técnico, destacam-se a necessidade de modelagem precisa das entidades físicas, a integração de fontes de dados heterogêneas, a garantia de comunicação contínua entre múltiplas plataformas e a manutenção da sincronização em tempo real entre os mundos físico e virtual. Esses desafios são ampliados pela diversidade de domínios de aplicação, cada qual com requisitos próprios de latência, escala, segurança e interoperabilidade.

Soluções de *middleware* desempenham um papel importante nesse contexto justamente para endereçar alguns desses desafios. Originalmente concebido para simplificar o desenvolvimento de sistemas distribuídos, um *middleware* atua como uma camada de abstração que fornece interfaces padronizadas e serviços em tempo de execução, abstraindo as heterogeneidades de comunicação e de distribuição [Coulouris et al. 2011]. No contexto de sistemas de DTs, essa função torna-se ainda mais relevante, pois tais sistemas precisam integrar componentes de natureza, origem e protocolo distintos, incluindo sensores físicos, modelos de simulação, serviços de análise e interfaces de atuação. Portanto, um *middleware* promove a interoperabilidade entre esses componentes e reduz significativamente a complexidade de desenvolvimento. No entanto, é crucial que seu uso seja guiado por um processo de desenvolvimento claro e sistemático, que oriente os desenvolvedores ao longo das etapas de concepção e implementação de sistemas de DTs.

Nesse contexto, este capítulo tem por objetivo introduzir os principais conceitos e desafios relacionados a sistemas de DTs e demonstrar o uso de um *middleware* para a construção desses sistemas. Para tanto, utilizaremos o MidDiTS [Pereira et al. 2024], um *middleware* projetado especificamente para o desenvolvimento de sistemas de DT. O MidDiTS visa atender a características essenciais desses sistemas, tais como a modelagem de entidades físicas e seus atributos, a manutenção da sincronização contínua entre estados físicos e digitais, a integração com múltiplas fontes de dados e plataformas heterogêneas, a disponibilização de interfaces padronizadas para monitoramento e atuação e o suporte à análise e à evolução dos DTs ao longo do ciclo de vida da entidade-alvo. Essas características estão alinhadas aos requisitos estabelecidos pelo *Digital Twin Consortium* (DTC)[1], um consórcio internacional que reúne dezenas de centros de pesquisa e empresas de diferentes áreas, voltado à definição de diretrizes e boas práticas para sistemas de gêmeos digitais (DTs).

O restante deste capítulo está organizado da seguinte forma. A Seção 5.2 discute os requisitos arquiteturais e os desafios de implementação de sistemas de DTs. A Seção 5.3 descreve como soluções baseadas em *middleware* podem dar suporte ao desenvolvimento de sistemas de DTs. A Seção 5.4 apresenta uma visão geral da arquitetura do *middleware* MidDiTS e de sua implementação. A Seção 5.5 descreve as etapas do processo de construção de sistemas de DTs com o MidDiTS. Por fim, a Seção 5.6 apresenta algumas considerações finais.

5.2. Desafios e Requisitos de Sistemas de Gêmeos Digitais

Embora o conceito de DTs tenha um potencial transformador, o desenvolvimento dessas soluções ainda é um processo complexo que enfrenta desafios significativos. Parte dessa complexidade está associada ao projeto e à implementação dos elementos que compõem essas soluções [Pereira et al. 2024]. Entre esses elementos, destacam-se as *entidades de DT*, que se referem a ativos físicos, processos ou ambientes no mundo real (por exemplo, um sensor, uma sala ou uma casa inteligente) para os quais é necessária uma representação virtual. Para compor a solução, define-se uma *instância de DT* como a representação digital que se mantém ativamente sincronizada com sua contraparte física. Por fim, um *sistema de DT* é a aplicação ou ecossistema abrangente que agrega e orquestra múlti-

[1] https://www.digitaltwinconsortium.org

plas instâncias de DT interconectadas e funcionalidades voltadas para oferecer suporte a necessidades específicas [van Schalkwyk et al. 2025].

A distinção entre esses elementos é relevante para separar o elemento do mundo real (entidade de DT) de sua contraparte digital operacional (instância de DT). Assim, com a necessidade de manter o sincronismo entre os elementos de forma contínua e precisa, diferentes desafios se apresentam [Almeida et al. 2023]:

- **Modelagem.** A literatura destaca a necessidade de melhorar metodologias e técnicas para o desenvolvimento de sistemas de DTs. No entanto, a falta de uma compreensão mais profunda sobre a conceituação e a modelagem de DTs dificulta a plena realização deste aspecto.
- **Interoperabilidade.** Sistemas de DTs lidam constantemente com dados provenientes de fontes altamente heterogêneas, frequentemente produzidos por diferentes fabricantes e utilizando protocolos diversos. A integração contínua dessas tecnologias e aplicações representa um desafio de interoperabilidade para essas soluções.
- **Sincronização.** Manter interações sincronizadas e preservar a comunicação bidirecional em tempo real entre as entidades do mundo real e as suas respectivas representações virtuais exige o processamento rápido de eventos e garantias de baixa latência nas redes de comunicação.
- **Orquestração.** A comunicação transparente entre diferentes elementos em um sistema de DT, bem como a coordenação dos fluxos de dados e das interações entre múltiplas entidades do mundo real e instâncias virtuais, são desafios que afetam a maturidade e a adoção do paradigma.

Para mitigar esses desafios e guiar o desenvolvimento a partir de necessidades de uso, o DTC organizou os requisitos e capacidades fundamentais de um ecossistema de DTs em seis categorias principais [van Schalkwyk et al. 2025]:

1) **Serviços de Dados.** Requisitos voltados para viabilizar o acesso, a ingestão e o gerenciamento de dados, da borda até a nuvem. O desafio da representação e da modelagem de dados pode ser tratado dentro desta categoria, indicando a necessidade de abstração de modelagem adaptada a diferentes domínios. Do ponto de vista da infraestrutura, esses serviços incluem armazenamento, aquisição e processamento de dados em tempo real ou em lotes. O gerenciamento de dados específicos de domínio, incluindo dados de séries temporais e dados geoespaciais, insere-se também nesta categoria.
2) **Integração.** Em resposta ao desafio da interoperabilidade, esta categoria preocupa-se em habilitar o acesso a dados e a comunicação com sistemas e aplicações pré-existentes, tanto internas quanto externas. Os requisitos desta categoria englobam a integração de sistemas, a interoperabilidade e a integração direta entre diferentes sistemas de DTs.
3) **Inteligência.** Um dos principais usos dos DTs é replicar processos do mundo real por meio de simulações. Técnicas de análise de dados e inteligência computacional podem enriquecer a representação virtual desses processos. Assim, esta

categoria considera o ambiente e as ferramentas para o desenvolvimento e a implantação de soluções de DTs, incluindo orquestração de fluxos, execução de simulações, modelagem de predição, análise de dados e integração com Inteligência Artificial.

4) **Experiência de Usuário.** Esta categoria engloba requisitos que garantem às pessoas usuárias a capacidade de interagir com os DTs e visualizar seus dados de maneira eficiente. Isso inclui a visualização das entidades e o monitoramento em tempo real com *Business Intelligence*, abarcando desde painéis de controle (*dashboards*) até interfaces avançadas, como a realidade aumentada.
5) **Gerenciamento.** Esta categoria compreende as capacidades voltadas para a governança e a manutenção do ecossistema de DTs. A capacidade de monitorar continuamente o ambiente e seus elementos é crucial para garantir o bom funcionamento de um DT, e os requisitos operacionais incluem o gerenciamento contínuo de dispositivos, o monitoramento abrangente do sistema, o registro de eventos e políticas de governança de dados. A orquestração é outra funcionalidade relacionada ao gerenciamento e engloba a coordenação das interações e dos fluxos de dados entre os diversos elementos de um sistema de DT, os quais podem variar de ativos físicos a aplicações corporativas.
6) **Confiabilidade.** Capacidades de segurança, privacidade e criptografia de dados são aspectos essenciais para DTs, principalmente nos domínios de aplicação em que são utilizados. A confiabilidade abrange requisitos críticos que asseguram a adoção segura e responsável da tecnologia, englobando desde segurança corporativa, segurança nativa dos dispositivos e criptografia de dados, até privacidade, resiliência e proteção, incluindo especificamente questões de autenticação, autorização e controle de acesso.

Dentro dessa iniciativa, o DTC também propôs a Tabela Periódica de Capacidades de Gêmeos Digitais [van Schalkwyk et al. 2025]. Essa tabela atua como um *framework* agnóstico de arquitetura e tecnologia para a definição de requisitos para DTs. Nela, as seis categorias anteriormente mencionadas são expandidas em um conjunto de 62 requisitos de alto nível e capacidades técnicas necessárias para compor soluções complexas de DTs.

Apesar da abrangência da proposta do DTC, algumas considerações sobre as relações entre os requisitos merecem atenção. Por exemplo, a sincronização entre uma instância de DT e seu ativo físico (isto é, a manutenção da correspondência bidirecional e em tempo real entre o mundo físico e sua representação digital) é um requisito que atravessa as categorias de Integração e Gerenciamento. Do ponto de vista da conectividade e do transporte de dados, a sincronização é um problema de Integração; já do ponto de vista da consistência de estado e do monitoramento da qualidade dessa correspondência ao longo do tempo, ela se aproxima do Gerenciamento. Outro aspecto a ser considerado diz respeito às capacidades de orquestração e alocação de recursos, que envolvem decisões sobre onde implantar instâncias de DT e como alocar recursos computacionais para sua execução. A orquestração pode ser associada à categoria de Inteligência, mas os mecanismos de orquestração pertencem ao Gerenciamento, uma vez que diversas estratégias de orquestração operam como políticas de infraestrutura e governança do ecossistema.

As seis categorias de requisitos anteriormente apresentadas podem orientar a avali-

ação e o desenvolvimento de soluções para sistemas de DTs. Contudo, identificar como esses requisitos podem ser concretamente atendidos na prática ainda representa um grande desafio. Diante desse amplo espectro de capacidades, é necessária uma abordagem em que capacidades técnicas individuais das entidades possam ser modeladas em DTs específicos e orquestradas como um ecossistema de DTs. Nesse sentido, plataformas de *middleware* emergem como uma camada de abstração estratégica, capaz de encapsular e prover parte dessas capacidades de forma padronizada e reutilizável, sendo esse o foco das próximas seções deste capítulo.

5.3. Uso de Middleware em Sistemas de Gêmeos Digitais

A evolução das plataformas de *middleware* está fortemente associada ao surgimento de sistemas distribuídos e à necessidade crescente de abstrair sua inerente complexidade. Desde os anos 1990, com o avanço das redes de computadores, tornou-se evidente a demanda por camadas intermediárias de software capazes de facilitar a comunicação, a coordenação e a integração entre componentes heterogêneos, muitas vezes executando em diferentes plataformas e localizações [Bernstein 1996]. Nesse contexto, o papel primordial de um *middleware* é preencher a lacuna funcional entre os programas de aplicação e a complexa infraestrutura de comunicação e distribuição subjacente.

Um *middleware* atua para coordenar a conexão e a interoperabilidade entre as partes de uma aplicação, além de simplificar a integração de componentes distribuídos desenvolvidos por múltiplos provedores [Schantz and Schmidt 2002]. Ao abstrair a heterogeneidade e proteger os desenvolvedores de detalhes complexos de baixo nível, essa camada torna viável e eficiente o desenvolvimento e a evolução de sistemas distribuídos robustos. Em essência, o *middleware* é projetado para isolar os desenvolvedores das complexidades inerentes à distribuição e à heterogeneidade, que são fontes recorrentes de erros em sistemas distribuídos.

Apesar da aplicabilidade e eficácia das tecnologias de *middleware* de propósito geral, certos domínios exigem serviços específicos não tratados pelas plataformas convencionais. Essas plataformas são tipicamente genéricas e, portanto, não endereçam nativamente os requisitos semânticos complexos de novos paradigmas, principalmente no contexto de aplicações industriais. Para cobrir essa lacuna, na última década foram propostas soluções de *middleware* voltadas para domínios específicos, como a IoT, incorporando serviços para lidar com o novo contexto de integração com objetos físicos, altíssima heterogeneidade dos dispositivos, grande dinamismo, necessidade de ciência de contexto e restrição de recursos.

Sistemas de DTs apresentam requisitos que extrapolam os tradicionalmente encontrados em aplicações de IoT. Enquanto um *middleware* para IoT, em geral, foca na conectividade de dispositivos e no transporte de dados, um *middleware* para DT precisa tratar aspectos como semântica, modelagem de estado e sincronização contínua entre os mundos físico e o virtual, a chamada conexão causal. Além da simples coleta, transporte e processamento de dados de sensores, esses sistemas devem manter uma representação digital (instância de DT) continuamente sincronizada com o mundo físico, permitindo a observação, a análise e, em muitos casos, a atuação sobre as entidades representadas. Em outras palavras, enquanto um *middleware* convencional concentra-se primordialmente

na troca de mensagens e na abstração de heterogeneidade de hardware e de protocolos, um *middleware* para DT necessita dar suporte ao ciclo de vida integral da representação virtual. Isso implica garantir a sincronização bidirecional persistente e a consistência de estado entre a entidade do mundo real e sua contraparte digital, além de lidar com os demais requisitos já apresentados. Portanto, para tratar toda essa complexidade de forma sistemática, a literatura e as iniciativas industriais têm convergido para o uso de arquiteturas de *middleware* especializadas em sistemas de DTs.

Esta seção discute como soluções de *middleware* podem apoiar a construção de sistemas de DTs, descrevendo brevemente alguns exemplos da literatura e da indústria, tanto comerciais quanto de código aberto. O intuito é ilustrar que tais soluções permitem separar responsabilidades, promover interoperabilidade e facilitar a construção e a evolução de sistemas de DTs, ao atender parte ou todos os requisitos de tais sistemas ao longo de seu ciclo de vida.

5.3.1. Requisitos Atendidos por Middleware para Gêmeos Digitais

Integração. Como visto na Seção 5.2, um requisito essencial para sistemas de DTs refere-se à integração em diferentes níveis (entre entidades de DT, instâncias de DT e com sistemas externos), ao qual qualquer solução de *middleware* para DT deve atender. Diversas plataformas de *middleware* existentes utilizam a estratégia de desenvolver *drivers* específicos para permitir a comunicação tanto com ativos físicos quanto entre DTs [Redelinghuys et al. 2020, Barbie et al. 2024]. Nesse contexto, a interoperabilidade entre diferentes protocolos exige a implementação de manipuladores dedicados para cada protocolo, o que aumenta a complexidade e dificulta a escalabilidade das soluções. Outras propostas utilizam o protocolo MQTT e argumentam que o próprio *broker* de mensagens, componente central em arquiteturas baseadas nesse protocolo, permite a intercomunicação de DTs.

Há ainda abordagens que incorporam *frameworks* ou *middleware* de comunicação existentes [Lin and Lu 2024], como o serviço de distribuição de dados (DDS), a uma arquitetura mais completa de *middleware* para DT. Grande parte das soluções adota arquiteturas baseadas no modelo publicação-subscrição, devido à sua capacidade de desacoplar produtores e consumidores de dados, permitindo comunicação assíncrona e escalável entre componentes. Esse modelo facilita a interoperabilidade em ambientes heterogêneos, reduz a complexidade de integração e possibilita a evolução dinâmica do sistema, características essenciais para cenários com múltiplos DTs e as entidades do mundo real distribuídas.

Serviços de Dados. Serviços de dados também são centrais para sistemas de DTs e providos pela maior parte das soluções de *middleware*. Eles abrangem requisitos de representação e modelagem de DTs, armazenamento, aquisição e ingestão de dados, e processamento em tempo real. Soluções existentes adotam diferentes estratégias para representar DTs, relacionadas a metadados, identidade, atributos (estáticos ou dinâmicos), informações derivadas e relações entre DTs. Quanto à aquisição e ingestão de dados, em um ambiente onde o número de dispositivos e dados gerados pode aumentar rapidamente, recursos de coleta e ingestão devem estar presentes juntamente com o gerenciamento de fluxos de dados e processamento em lote. Algumas propostas utilizam soluções baseadas

em *brokers* de mensagens para lidar com a ingestão de dados, enquanto outras utilizam *brokers* para o armazenamento de médio prazo dos dados coletados. Para a persistência de dados a longo prazo, diferentes estilos de bancos de dados, incluindo abordagens na nuvem, têm sido empregados.

Inteligência. Com relação aos serviços de inteligência, um dos principais usos de DTs é imitar processos do mundo real por meio de simulações, e técnicas de inteligência computacional e análise de dados podem enriquecer a contraparte virtual desses processos. Entre outros objetivos, a incorporação de técnicas de aprendizado de máquina a sistemas de DTs permite prever falhas em ativos físicos, enquanto a análise de dados pode apoiar a tomada de decisões em tempo real, incluindo cenários de simulação com dados ingeridos em tempo real. Todavia, apesar de sua relevância e da forte integração entre aprendizado de máquina e DT, poucas propostas existentes tratam dessa integração em nível de *middleware*.

Gerenciamento. Requisitos de gerenciamento relacionam-se a funcionalidades de monitoramento do sistema, registro de *logs*, governança de dados e ao gerenciamento e à orquestração de recursos. A capacidade de monitorar continuamente o ambiente e seus elementos é crucial para garantir o funcionamento adequado de um DT. Por meio do monitoramento e de alertas do sistema, é possível observar DTs, aplicações e serviços, coletando, analisando e agindo sobre os dados para maximizar sua disponibilidade e desempenho. Além disso, a implementação de estratégias de registro de *logs* pode fornecer dados essenciais para análise e geração de melhorias adicionais para o sistema.

Governança. Outra capacidade de gerenciamento está relacionada à governança de dados, que envolve o gerenciamento da disponibilidade, usabilidade, integridade e segurança dos dados em DTs, com base em padrões e políticas que controlam seu uso. Poucas propostas de *middleware* na literatura incluem soluções completas de governança de dados. Na literatura, observa-se que a governança de dados pode ser facilitada isolando bancos de dados em microsserviços e utilizando *frameworks* e ferramentas de observabilidade para ajudar a garantir a disponibilidade, usabilidade, integridade e segurança dos dados.

Outras propostas adotam abordagens de sistemas de gerenciamento de banco de dados (SGBD) para armazenar os dados coletados de ativos físicos e aplicações externas e prover funcionalidades centralizadas de governança, como a rastreabilidade de dados. A orquestração é outra funcionalidade relacionada ao gerenciamento que o *middleware* pode fornecer para coordenar interações e fluxos de dados entre os diversos elementos de um sistema de DT, os quais podem variar de ativos físicos a aplicações corporativas. Mecanismos de registro e descoberta são igualmente essenciais para viabilizar estratégias de orquestração adequadas em DTs. Nesse contexto, muitos *middleware* adotam processos manuais para registrar um DT, fornecendo *endpoints* de API RESTful específicos.

Confiabilidade. Considerando a confiabilidade, requisitos de segurança, privacidade e criptografia de dados são essenciais para sistemas de DTs, principalmente em domínios de aplicação críticos em que têm sido utilizados. No entanto, ainda se observa uma carência de soluções voltadas para esse domínio e da incorporação de recursos de segurança como componentes essenciais de *middleware* para DT.

A seguir, descrevemos exemplos de *middleware* para DT que atendem a um ou mais desses requisitos. Observa-se que nenhuma plataforma existente atende a todos os requisitos simultaneamente. Além disso, há muitas soluções de *middleware* voltadas para domínios específicos de aplicação de DT. Na Seção 5.3.2 ilustramos exemplos de plataformas da indústria e na Seção 5.3.3 exemplos de soluções reportadas na literatura.

5.3.2. Iniciativas Proprietárias e de Código Aberto

***Azure Digital Twins* (Microsoft).** A *Azure Digital Twins* (ADT)[2] é uma plataforma como serviço (do inglês, *platform-as-a-service* – PaaS) que permite a criação de grafos de DTs baseados em modelos digitais de ambientes inteiros, tais como edifícios, fábricas, fazendas, redes de energia, ferrovias, estádios ou mesmo cidades. Esses modelos digitais podem ser usados para obter *insights* úteis para a melhoria de produtos, otimizações de operações, redução de custos e para proporcionar experiências inovadoras para as pessoas usuárias. Com foco em integração, a ADT pode ser usada para projetar uma arquitetura de sistemas de DTs na qual dispositivos físicos de IoT são representados no contexto de uma solução mais ampla, baseada na nuvem. Um importante diferencial da ADT é sua capacidade de integração nativa com o *Azure IoT Hub*[3], um serviço que funciona como um *hub* central de mensagens para soluções de IoT baseadas em nuvem, permitindo comunicação confiável e segura em larga escala entre aplicações de IoT e seus dispositivos conectados.

Na ADT, é possível definir entidades de DT que representam pessoas, lugares e objetos em um ambiente físico desejado usando tipos customizados de DTs, denominados modelos. Tais definições de modelos atuam como um vocabulário especializado para descrever um dado domínio de negócio. Modelos são definidos em uma linguagem similar a JSON, a *Digital Twins Definition Language* (DTDL)[4]. Modelos DTDL descrevem tipos de entidades de acordo com suas propriedades de estado, componentes e relacionamentos. Uma vez definidos os modelos, eles podem ser usados para criar gêmeos digitais (instâncias) que representem cada entidade específica no ambiente. Por exemplo, pode-se usar a definição de um modelo "Edifício" para criar vários DTs do tipo "Edifício" (por exemplo, "Edifício 1", "Edifício 2" e assim por diante). Podem-se também usar relacionamentos nas definições de modelos para conectar DTs entre si, formando um grafo conceitual. É importante destacar que tais modelos digitais, uma vez instanciados, são representações dinâmicas continuamente sincronizadas com o mundo real. A Figura 5.1 ilustra um modelo DTDL e uma instância correspondente, bem como o grafo de representação dos elementos do DT.

Para manter as propriedades da instância de DT atualizadas em relação ao ambiente físico, pode-se usar o *Azure IoT Hub*. Os dispositivos gerenciados pelo *hub* são representados como parte do grafo de DTs e fornecem os dados que mantêm o modelo atualizado. Pode-se criar um novo *hub* IoT para usar com a ADT ou conectar um *hub* existente juntamente com os dispositivos que ele já gerencia. Pode-se ainda alimentar a ADT a partir de outras fontes de dados, usando APIs RESTful ou conectores para outros serviços da plataforma Azure. A ADT fornece também um sistema de eventos robusto

[2]https://azure.microsoft.com/en-us/products/digital-twins

[3]https://azure.microsoft.com/en-us/products/iot-hub

[4]https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v3/DTDL.v3.md

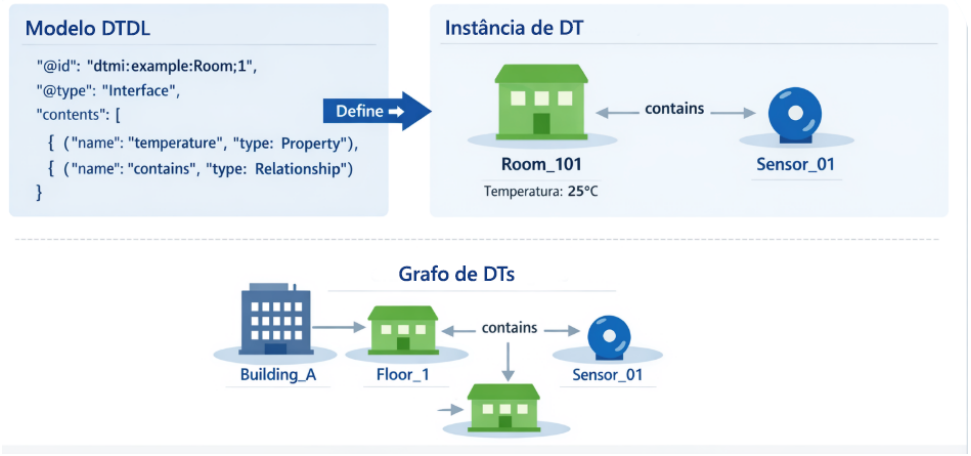

Figura 5.1. Exemplo de um modelo DTDL, de uma instância de DT e de um grafo de DTs

para manter o grafo atualizado, incluindo processamento de dados que pode ser personalizado para corresponder a cada lógica de negócios específica.

A ADT também oferece uma API de consulta que possibilita extrair *insights* do ambiente de execução em tempo real. Essa API permite realizar consultas com critérios de pesquisa abrangentes, incluindo valores de propriedades, relacionamentos, propriedades de relacionamento, informações do modelo, entre outras. Outra funcionalidade disponível é a visualização de grafos de DTs usando o *Azure Digital Twins Explorer*[5], que fornece uma interface para a construção e interação com o grafo conceitual. Por fim, o *Azure Digital Twins 3D Scenes Studio*[6] é um ambiente visual 3D imersivo em que pessoas usuárias podem monitorar, diagnosticar e investigar dados operacionais de DTs com o contexto visual de ativos 3D. Com um grafo de DTs e um modelo 3D, especialistas no domínio podem usar o componente construtor do estúdio para mapear os elementos 3D para DTs no grafo da ADT e definir a interatividade da interface do usuário e a lógica de negócios para uma visualização 3D de um ambiente de negócios. As cenas 3D podem então ser consumidas no *Azure Digital Twins 3D Scenes Studio* hospedado ou em uma aplicação personalizada que utiliza o componente de visualização 3D embarcado.

Eclipse BaSyx. A concretização da Indústria 4.0 demanda uma interoperabilidade em nível semântico que permita aos sistemas industriais interpretarem, de forma autônoma, as capacidades e o estado de cada elemento da cadeia produtiva. Nesse contexto, o *Asset Administration Shell* (AAS)[7] tem se estabelecido como a especificação técnica padronizada para a representação digital de um ativo. Concebido para solucionar o problema da heterogeneidade de protocolos no contexto da manufatura, o AAS atua como uma interface digital universal que abstrai a complexidade técnica dos equipamentos físicos e permite sua integração direta em ecossistemas conectados [Bader et al. 2022]. Para operacionalizar essa integração, um I4.0 *Component* é definido como a agregação lógica de dois elementos distintos e complementares: (i) o *Asset*, o qual se refere a qualquer entidade física

[5]https://learn.microsoft.com/en-us/azure/digital-twins/concepts-azure-digital-twins-explorer
[6]https://learn.microsoft.com/en-us/azure/digital-twins/quickstart-3d-scenes-studio
[7]https://reference.opcfoundation.org/I4AAS/v100/docs/4.1

ou lógica que possua valor para a organização (por exemplo, um motor, um sensor, um plano de produção ou um software), e; (ii) o AAS propriamente dito, que constitui a representação virtual que encapsula o *asset* e provê uma interface padronizada para o gerenciamento de suas informações e funcionalidades. Dessa forma, o AAS atua como a camada digital que permite ao *asset* físico ser reconhecido, conectado e gerenciado dentro do ecossistema da Indústria 4.0.

O Eclipse BaSyx[8] fornece um conjunto de componentes de software projetados para materializar o conceito de AAS em ambientes de produção, podendo, portanto, ser considerado um *middleware* para sistemas de DTs. Essa plataforma foi originalmente concebido pelo Fraunhofer IESE, da Alemanha, e atualmente é mantido pela Eclipse Foundation. O Eclipse BaSyx adota uma arquitetura orientada a serviços e microsserviços, composta por módulos independentes que interagem por meio de APIs padronizadas para viabilizar a persistência, a localização e a integração dos gêmeos digitais. A Figura 5.2 ilustra uma visão geral do Eclipse BaSyx. A seguir, descrevemos brevemente os componentes arquiteturais fundamentais que compõem o ecossistema do Eclipse BaSyx e suas respectivas responsabilidades.

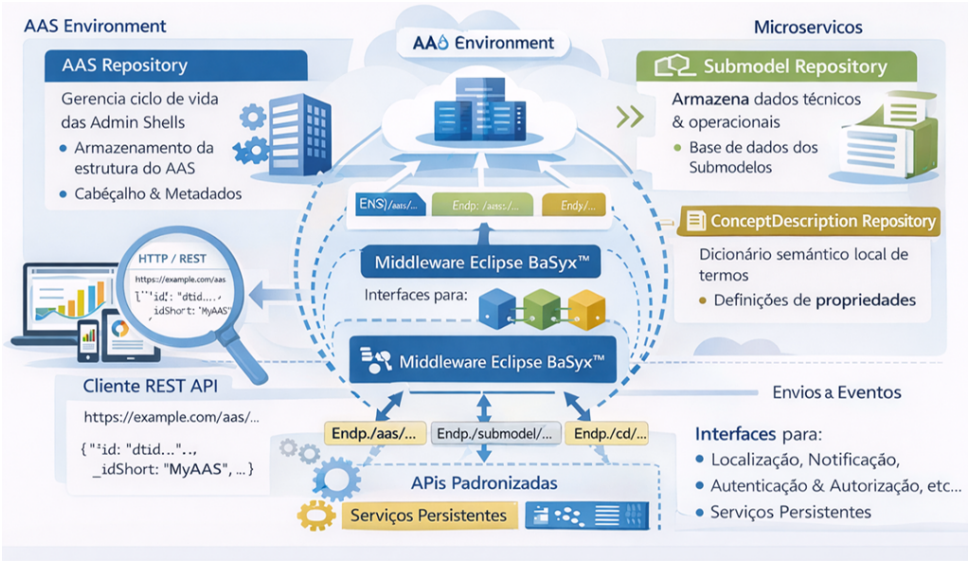

Figura 5.2. Visão geral do Eclipse BaSyx

AAS Environment: Atua como o servidor de aplicação principal da arquitetura, sendo responsável pela hospedagem e persistência de DTs. Ele consiste na agregação lógica de três interfaces de repositório distintas, acessíveis via protocolo HTTP/REST: *AAS Repository*, *Submodel Repository* e *ConceptDescription Repository*. O *AAS Repository* gerencia o ciclo de vida das entidades, armazenando a estrutura do AAS e permitindo operações de criação, leitura, atualização e remoção sobre o cabeçalho e os metadados do DT. O *Submodel Repository* é responsável pelo armazenamento efetivo dos dados técnicos e operacionais, gerenciando os submodelos e seus elementos internos e atuando como o banco de dados onde as propriedades estáticas e dinâmicas do ativo residem. O *ConceptDescription Repository* gerencia os dicionários semânticos locais, armazenando

[8]https://basyx.org

as definições que atribuem significado às propriedades (como unidades de medida e referências normativas) e permitindo que sistemas clientes consultem o significado exato de cada dado armazenado.

AAS Registry: Atua como um diretório de endereços do sistema, mantendo um mapeamento entre o identificador único da entidade de DT e o seu endereço de rede (*endpoint*). Dessa forma, aplicações clientes consultam o *AAS Registry* para resolver a localização de um AAS específico antes de estabelecer conexão direta com o repositório que o hospeda.

AAS Discovery Service: Complementar ao *AAS Registry*, oferece mecanismos de busca baseados em atributos. Enquanto o *AAS Registry* exige o conhecimento prévio do identificador da entidade para localizá-la, o *AAS Discovery Service* permite que aplicações pesquisem por DTs que possuam características específicas, por exemplo, listar todas as entidades de DT que contenham um Submodelo de "Consumo Energético". Esta funcionalidade viabiliza a descoberta dinâmica de recursos na rede industrial sem a necessidade de um inventário estático de identificadores.

DataBridge: Responsável pela conexão entre o ambiente virtual e os ativos físicos, ou seja, entre entidades e instâncias de DT. Ele atua como um *gateway* de tradução de protocolos, projetado para integrar equipamentos que não possuem suporte nativo ao padrão AAS. Sua operação baseia-se na configuração de rotas de sincronização que mapeiam sinais de protocolos industriais (como MQTT, OPC UA, HTTP ou Modbus) para propriedades específicas dentro dos submodelos. Esta operação ocorre em dois fluxos distintos. No sentido do físico para o digital, o componente monitora uma fonte de dados externa, como um tópico MQTT, e atualiza automaticamente a propriedade correspondente no servidor AAS sempre que um novo valor é recebido. No sentido do digital para o físico, ele observa alterações em uma propriedade do submodelo e propaga o novo valor para o dispositivo físico, permitindo o envio de comandos para atuadores.

BaSyxWeb User Interface: Atua como um painel de visualização genérico, permitindo que operadores e gestores inspecionem o estado dos ativos e naveguem pela estrutura hierárquica dos modelos AAS sem a necessidade de manipular diretamente chamadas de API ou *scripts* de código. O funcionamento da interface baseia-se na integração dinâmica com os componentes de registro e persistência. Ao ser inicializada, a aplicação conecta-se ao *AAS Registry* para recuperar a lista de DTs disponíveis na rede industrial. Quando um ativo é selecionado, a interface consulta automaticamente o *AAS Environment* correspondente para carregar e renderizar os submodelos e suas propriedades. Além da visualização passiva de telemetria, a ferramenta oferece suporte a operações de interação ativa, permitindo que pessoas usuárias autorizadas modifiquem valores de propriedades configuráveis ou acionem operações técnicas diretamente pelo navegador.

Eclipse Ditto. O Eclipse Ditto[9] é uma tecnologia de código aberto que tem se destacado como uma das principais referências para a implementação de DTs no contexto da IoT. Ele fornece representações digitais, denominadas *things*, de dispositivos físicos conectados, com suporte à sua modelagem, controle e monitoramento. A plataforma expõe APIs e protocolos Web e possui integração com módulos de fila e mensageria compatíveis com

[9]https://github.com/eclipse-ditto/ditto

diversos modelos amplamente difundidos, permitindo que os DTs sejam manipulados como serviços comuns e abstraindo complexidades de conectividade e infraestrutura.

A arquitetura do Eclipse Ditto é concebida segundo princípios modernos de engenharia de software, adotando o modelo hexagonal, no qual a lógica de domínio é isolada por meio de portas e adaptadores que intermediam as interações com o ambiente externo, promovendo modularidade e desacoplamento entre os componentes do sistema. A plataforma segue ainda uma abordagem baseada em microsserviços, distribuindo responsabilidades entre unidades especializadas que operam de forma autônoma e assíncrona. Além disso, o Eclipse Ditto emprega JSON como linguagem de interface entre artefatos e adota padrões como *Command Query Responsibility Segregation* (CQRS) e *event sourcing*, garantindo interoperabilidade, escalabilidade e desacoplamento para soluções distribuídas e heterogêneas baseadas em DTs.

O Eclipse Ditto implementa os seguintes serviços, conforme ilustra a Figura 5.3:

- *Gateway:* Atua como ponto de entrada principal, expondo interfaces REST e WebSocket para consumidores externos, além de realizar a autenticação inicial e o roteamento interno das mensagens em formato JSON.
- *Things:* É o núcleo funcional responsável pelo ciclo de vida dos DTs, mantendo estados, aplicando comandos e emitindo eventos persistidos e distribuídos aos serviços interessados.
- *Policies:* Gerencia regras de autorização, associando permissões específicas a identidades (pessoas usuárias, sistemas ou dispositivos) sobre os recursos.
- *Things Search:* Indexa e projeta eventos do serviço de *Things*, otimizando consultas complexas sobre atributos e metadados das entidades.
- *Connectivity:* Permite a integração com sistemas e dispositivos externos, oferecendo suporte nativo a diversos protocolos, como MQTT, AMQP e HTTP.

Figura 5.3. Serviços do Eclipse Ditto

No Eclipse Ditto, a comunicação consiste no mecanismo que assegura a coerência entre o mundo físico e sua representação digital. É por meio da troca de mensagens que telemetrias são incorporadas ao DT, comandos são propagados aos dispositivos físicos e o sistema mantém tanto reatividade em tempo real quanto consistência histórica. Para viabilizar essa comunicação de forma unificada e independente do protocolo de transporte, o Eclipse Ditto adota uma padronização semântica baseada em um modelo de mensagens estruturadas, garantindo interoperabilidade entre componentes heterogêneos. Essa abordagem permite que diferentes meios de comunicação coexistam sob um mesmo modelo

lógico, além de prover mecanismos para assegurar a integridade dos dados, a robustez diante de desconexões e o atendimento a requisitos de baixa latência quando necessários. Todos esses objetivos são atingidos por meio do *Ditto Protocol*.[10], que é uma especificação semântica baseada em JSON para estruturar e interpretar comandos, eventos e interações sobre *things*, de forma independente do protocolo de transporte adotado. Sua adoção viabiliza uma representação padronizada para operações como leitura, modificação e exclusão de propriedades, bem como para o envio de comandos a dispositivos físicos, garantindo uniformidade semântica em toda a arquitetura.

Um aspecto central do *Ditto Protocol* é a distinção entre dois canais de comunicação complementares. O canal denominado *Twin* é responsável por acessar e manipular o estado persistente da *thing*, isto é, o modelo digital armazenado na plataforma. As informações lidas ou alteradas por meio desse canal permanecem dissociadas da conectividade imediata com o dispositivo físico, permitindo a manutenção de um espelho confiável e histórico do estado do artefato. Já o canal denominado *Live* estabelece a comunicação em tempo real com o dispositivo físico, possibilitando o envio direto de comandos e a recepção instantânea de eventos. Por operar de forma síncrona com o mundo físico quando há conectividade ativa, este canal é indispensável em cenários que exigem baixa latência e controle imediato. Ao integrar esses dois canais em uma única especificação semântica, o *Ditto Protocol* garante que o modelo digital possa simultaneamente preservar estados históricos e responder de forma imediata a interações, assegurando consistência, rastreabilidade e flexibilidade operacional. A Figura 5.4 ilustra esses conceitos.

Figura 5.4. Canais *Twin* e *Live* relacionados a *Thing* no contexto do Eclipse Ditto

Comparando ADT, Eclipse Basyx e Eclipse Ditto. Os três exemplos de *middleware* discutidos anteriormente destacam-se como plataformas consolidadas voltadas para sistemas de DTs, sendo o Eclipse BaSyx e o Eclipse Ditto soluções de código aberto e a ADT uma solução proprietária baseada em nuvem. A seguir, traçamos um breve comparativo entre as três plataformas em termos dos seguintes aspectos:

- **Estratégia de sincronização:** O Eclipse Ditto destaca-se pela separação clara entre o canal *Twin* (acesso ao estado histórico e persistente) e o canal *Live* (interação direta e síncrona com o dispositivo), o que garante reatividade e consistência

[10]https://eclipse.dev/ditto/protocol-overview.html

histórica simultaneamente. Já o Eclipse BaSyx foca na tradução de protocolos de equipamentos para atualizar as propriedades técnicas dos submodelos do AAS.

- **Modelo de mensageria:** O Eclipse Ditto adota um protocolo semântico próprio, o *Ditto Protocol*, operando sobre uma arquitetura de microsserviços e mensageria orientada a eventos. A ADT utiliza um sistema de eventos robusto para manter seu grafo de representação atualizado conforme a lógica de negócio.
- **Segurança e acesso:** O Eclipse Ditto oferece controle de autorização granular por meio do serviço *Policies*, permitindo configurar permissões detalhadas de leitura e escrita para pessoas usuárias ou sistemas sobre recursos específicos. O Eclipse BaSyx, por sua vez, delega essas funções para provedores de identidade externos, como OAuth2.
- **Escopo e modelagem de dados:** A ADT adota a linguagem DTDL, baseada em JSON, enquanto o Eclipse Ditto define modelos de *things* também via JSON. Já o Eclipse BaSyx adota o AAS como padrão de modelagem. Enquanto a ADT é mais adequada para visualizar e consultar grandes grafos de relacionamentos, o Eclipse BaSyx é o mais rigoroso na definição semântica industrial em termos de dicionários semânticos e unidades de medida normatizadas. O Eclipse Ditto, por sua vez, foca na abstração da complexidade do hardware, permitindo que as instâncias de DTs sejam manipulados como serviços Web comuns.
- **Foco de aplicação:** A ADT é adequada para representar, por meio de grafos, ambientes complexos, como cidades e redes de energia. O Eclipse Ditto foi concebido com foco em ambientes de IoT, enquanto o Eclipse BaSyx é voltado para ambientes fabris e para lidar com o desafio de interoperabilidade na Indústria 4.0.

5.3.3. Propostas da Literatura

Considerando o estado da arte atual, a pesquisa sobre *middleware* para DTs ainda é relativamente recente, considerando que a necessidade de soluções especificamente projetadas para esses ambientes passou a receber maior atenção apenas nos últimos anos. Em geral, as abordagens existentes concentram-se em aspectos pontuais, como protocolos de comunicação, padrões de integração industrial ou mecanismos de sincronização, mas frequentemente carecem de uma plataforma unificada capaz de abordar, de forma integrada, questões de modelagem, orquestração, integração e execução. Além disso, muitas soluções são direcionadas a domínios específicos de aplicação, o que limita sua generalização para diferentes cenários de sistemas de DTs. A seguir, apresentamos algumas propostas representativas da literatura, destacando como elas atendem aos requisitos previamente analisados.

Tao e Zhang [Tao and Zhang 2017] propõem um *middleware* para DTs voltado a ambientes fabris no contexto da Indústria 4.0. O objetivo principal do *middleware* é utilizar dados provenientes de sensores para realizar simulações em tempo real da linha de montagem, permitindo o ajuste de parâmetros de produção sem a necessidade de interromper as máquinas. O *middleware* é responsável pela chamada Unidade de Processamento de Dados do DT, que executa funções como limpeza, compressão e fusão de dados oriundos de sensores industriais, além de atualizar, quase instantaneamente, modelos 3D e de simulação com base nos dados físicos.

Mahdi et al. [Mahdi et al. 2025] apresentam uma arquitetura baseada em DTs específica para aplicações do tipo *Wire Arc Additive Manufacturing* (WAAM). Nesse contexto, o DT consiste em um conjunto de ferramentas voltadas ao aumento da precisão, eficiência e garantia de qualidade do processo produtivo. A arquitetura incorpora redes neurais convolucionais (CNNs) para detecção de anomalias, recursos de visualização 3D para representação detalhada do processo e técnicas de visão computacional para identificação precisa de parâmetros das peças. Os autores destacam que a implementação de DTs nesse cenário exige o tratamento de grandes volumes de dados, provenientes de múltiplas fontes, como sensores inteligentes, imagens, sinais analógicos e digitais e registros de inventário. Para viabilizar a comunicação entre entidades físicas e seus respectivos DTs, é utilizado o padrão OPC UA [Zhang et al. 2023], que oferece interoperabilidade semântica, independência de plataforma e comunicação segura em tempo real. A arquitetura proposta combina OPC UA com técnicas de análise de dados e armazenamento em banco de dados não relacional em nuvem, oferecendo suporte a comunicação, gerenciamento de dados estruturados, segurança e controle em tempo real. Além disso, integra mecanismos avançados de visualização e predição de defeitos.

Kutz et al. [Kuts et al. 2019] exploram o uso de DTs no gerenciamento e controle de sistemas de produção, com ênfase em robótica industrial, sendo derivado da adaptação de uma solução previamente desenvolvida para a Internet Industrial das Coisas (IIoT) [Modoni et al. 2018]. A proposta incorpora o uso de realidade virtual para simular a presença humana, permitindo interação remota e colaborativa. Os autores identificam requisitos fundamentais para sistemas de DTs, incluindo aquisição de dados via sensores, configuração precisa do modelo digital, uso de modelos de dados representativos e sincronização entre os sistemas físico e digital. O foco principal do trabalho está na sincronização, sendo proposto um *middleware* orientado a mensagens baseado no modelo publicação-subscrição, responsável por propagar dados de telemetria entre os ativos físicos e suas representações digitais.

André et al. [André et al. 2020] investigam a separação de interesses relacionadas à comunicação em relação a outros aspectos da aplicação, com o objetivo de aumentar a flexibilidade, adaptabilidade e evolutividade dos sistemas de DT. O trabalho propõe um *framework* que atua como camada intermediária de comunicação entre sistemas de controle de manufatura e os componentes de fábrica, incluindo DTs e simuladores.

Yun et al. [Yun et al. 2017] apresentan uma *plataforma* de DT voltada a sistemas ciberfísicos críticos, como veículos autônomos com sistemas avançados de assistência à direção. A arquitetura proposta contempla um ambiente distribuído de cooperação entre DTs, um *middleware* de comunicação centrado em dados baseado no padrão DDS e suporte a co-simulação por meio do padrão *Functional Mock-Up Interface* (FMI)[11]. Os autores destacam a importância da interoperabilidade em tempo real e da distribuição eficiente de dados como requisitos centrais para sistemas de DTs em larga escala. Além disso, enfatizam a necessidade de suporte a requisitos de qualidade de serviço (QoS), sincronização temporal entre simuladores e gerenciamento de dados complexos e multimídia. Nesse contexto, o *middleware* desempenha papel fundamental não apenas na comunicação entre entidades físicas e digitais, mas também na integração entre diferentes

[11] https://opensimulationplatform.com/co-simulation/

sistemas e simuladores.

Por fim, Bellavista et al. [Bellavista et al. 2024] abordam os desafios relacionados à sincronização e à orquestração de DTs, introduzindo o conceito de *entanglement*, originalmente proposto no trabalho de Minerva et al. [Minerva et al. 2020]. Esse conceito representa o grau de alinhamento entre o DT e sua contraparte física, sendo considerado essencial para garantir a confiabilidade da representação digital. Com base nesse conceito, os autores propõem um *middleware entanglement-aware*, capaz de monitorar e manter o nível de sincronização entre os sistemas. A solução utiliza conteinerização e considera aspectos como proximidade física entre DT e ativo real, bem como a competição por recursos computacionais entre diferentes DTs. A arquitetura é composta por um nó central, responsável pelo controle global, e por nós trabalhadores, que executam os DTs em diferentes camadas, na borda ou na nuvem. O sistema inclui ainda interfaces para monitoramento, orquestração, gerenciamento e visualização. Ademais, a arquitetura dispõe de componentes para gerenciamento de dados, armazenando métricas, eventos, configurações e artefatos de DTs.

A Tabela 5.1 sintetiza a cobertura dos requisitos fundamentais para DTs pelas soluções analisadas com base nas categorias propostas pelo DTC. Como podemos observar, nenhuma das soluções analisadas atende completamente a todas as categorias. As categorias *Integração* e *Serviços de Dados* são cobertas de forma consistente, especialmente nas plataformas mais maduras. Por outro lado, as categorias *Inteligência* e *Confiabilidade* concentram as lacunas mais expressivas. Apesar da evolução, a integração de técnicas de aprendizado de máquina e análise avançada de dados ainda é tratada de forma periférica pela maioria das soluções, e os mecanismos de segurança, quando presentes, são geralmente provenientes de *frameworks* genéricos, sem especialização para o contexto de DTs. A categoria *Gerenciamento*, especialmente no tocante a orquestração e alocação de recursos, é atendida de forma satisfatória apenas por soluções que o elegem como foco central, como o caso do Eclipse BaSyx e da proposta de Bellavista et al.

Além da diferença de maturidade em relação a soluções proprietárias, é possível perceber que as atuais soluções de *middleware* para DTs abordam os requisitos de forma generalista, deixando lacunas em inteligência e segurança, ou são altamente especializadas para nichos específicos, atendendo bem a aspectos de um contexto industrial, por exemplo. Esses resultados reforçam que o desenvolvimento de *middleware* para DTs ainda é uma área em evolução, com espaço para contribuições que integrem as seis categorias, requisitos de alto nível e capacidades técnicas necessárias de forma mais abrangente.

5.4. Um Middleware para Sistemas de Gêmeos Digitais: Componentes e Implementação

A análise das soluções comerciais, de código aberto e das propostas da literatura, apresentada na Seção 5.3.3, evidencia que, apesar dos avanços significativos no desenvolvimento de *middleware* para sistemas de DTs, ainda persistem lacunas importantes, especialmente no que se refere à integração simultânea dos diferentes requisitos discutidos anteriormente neste capítulo. De modo geral, observa-se que essas soluções tendem a se concentrar em aspectos específicos, como integração de dispositivos, gerenciamento de dados ou mecanismos de comunicação, sem oferecer uma abordagem unificada que contemple, de forma

Tabela 5.1. Comparativo entre soluções de *middleware* para DTs

Requisitos / Soluções	Serviços de Dados	Integração	Inteligência
Microsoft ADT	✔	✔	✕
Eclipse BaSyx	✔	✔	✕
Eclipse Ditto	◯	✔	✕
[Yun et al. 2017]	◯	✔	✕
[Tao and Zhang 2017]	✔	✕	✕
[Kuts et al. 2019]	◯	✔	✕
[André et al. 2020]	✕	✔	✕
[Bellavista et al. 2024]	✔	✕	◯
[Mahdi et al. 2025]	✔	✔	✔

Requisitos / Soluções	Experiência de Usuário	Gerenciamento	Confiabilidade
Microsoft ADT	✔	◯	✕
Eclipse BaSyx	◯	✔	◯
Eclipse Ditto	✕	◯	◯
[Yun et al. 2017]	✕	✕	✕
[Tao and Zhang 2017]	✕	✕	✕
[Kuts et al. 2019]	✕	✕	✕
[André et al. 2020]	✕	✕	✕
[Bellavista et al. 2024]	◯	✔	✕
[Mahdi et al. 2025]	◯	✕	◯

✔ = requisito atendido; ◯ = requisito parcialmente atendido; ✕ = requisito não atendido

integrada, as dimensões de modelagem, orquestração, sincronização e operação dos DTs. Além disso, muitas propostas apresentam forte acoplamento a domínios específicos ou dependem de configurações complexas para garantir a interoperabilidade entre componentes heterogêneos.

Esta seção apresenta o MidDiTS [Pereira et al. 2024], um *middleware* concebido para apoiar, de forma integrada, o desenvolvimento e a operação de sistemas de DTs. A seguir, detalhamos sua arquitetura e seus principais componentes de implementação.

5.4.1. Arquitetura

A arquitetura do MidDiTS foi projetada para apoiar a modelagem, o desenvolvimento, a sincronização e a execução de sistemas de DTs. Sua concepção baseia-se na premissa de que o desenvolvimento de sistemas de DTs requer a abstração das complexidades de comunicação de baixo nível, o gerenciamento da heterogeneidade dos dispositivos e a garantia de consistência causal bidirecional entre os mundos físico e virtual.

Para fundamentar sua operação, o MidDiTS apoia-se em três conceitos principais: (i) a *entidade de DT*, que representa o ativo físico no mundo real; (ii) a *instância de DT*, que corresponde à contraparte digital em execução e continuamente sincronizada pelo *middleware*; e (iii) o *sistema de DT*, que denota a aplicação de mais alto nível responsável por agregar e orquestrar múltiplas instâncias interconectadas. Adicionalmente, o MidDiTS permite que múltiplas aplicações compartilhem a mesma infraestrutura subjacente, mantendo, contudo, o isolamento de seus modelos, regras de negócio e instâncias.

A Figura 5.5 apresenta a arquitetura do MidDiTS, cujas funcionalidades são organizadas em componentes lógicos distribuídos em quatro camadas principais: *Things Facade*, *Orchestration Layer*, *DT Gateway* e *Security Management*.

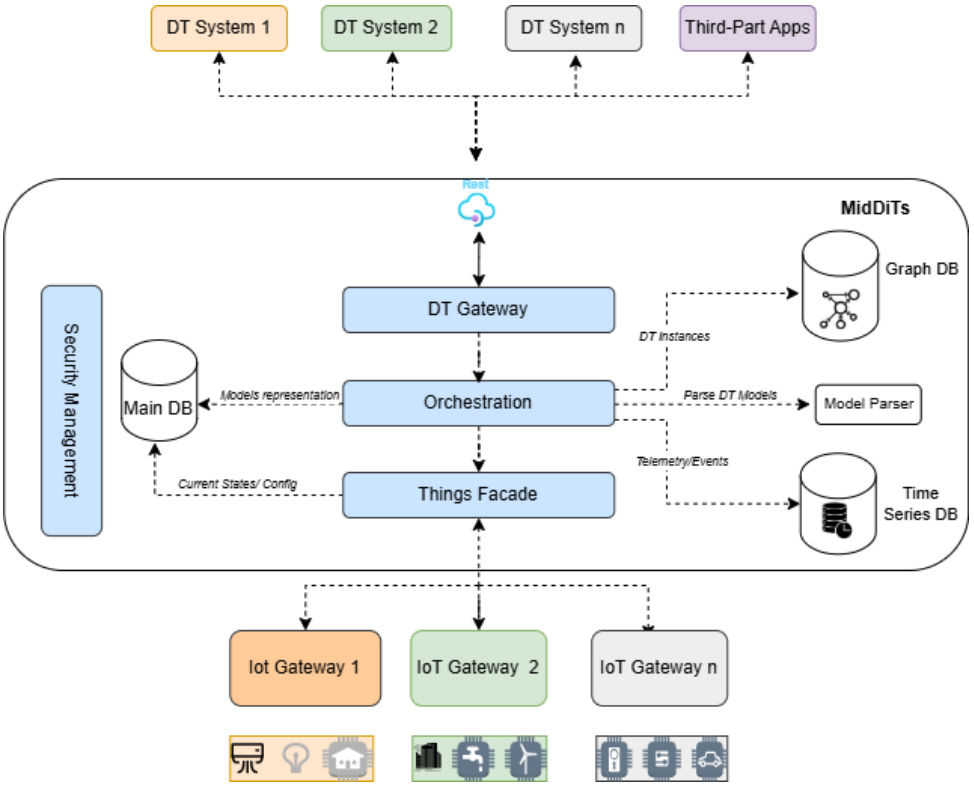

Figura 5.5. Visão geral da arquitetura do MidDiTS

A camada *Things Facade* atua como a interface de conexão entre o MidDiTS e as entidades físicas. Em vez de implementar diretamente protocolos de comunicação com dispositivos, essa camada integra-se a um *gateway* IoT externo. Sua principal função é mapear metadados de dispositivos (incluindo tipos e propriedades) para representações internas no *middleware*. Além disso, provê serviços de descoberta de dispositivos, gerencia o fluxo de telemetria e viabiliza o envio de comandos.

Devido ao fluxo contínuo e bidirecional de dados, a *Things Facade* requer suporte de sistemas de armazenamento otimizados para séries temporais, com o objetivo de registrar o histórico das propriedades e monitorar métricas como latência de comunicação. O MidDiTS incorpora ainda um gerenciador de eventos capaz de não apenas agendar requisições periódicas, mas também reagir dinamicamente a mudanças operacionais na infraestrutura.

O núcleo do MidDiTS é a *Orchestration Layer*. Sua principal responsabilidade é manter o estado e os modelos dos DTs continuamente sincronizados com suas contrapartes físicas. Essa camada armazena definições de modelos, gerencia a organização dos DTs por meio de relacionamentos dinâmicos entre instâncias e trata da conectividade por meio de mecanismos de *binding* explícitos e semânticos. Além disso, garante a propagação consistente de mudanças de estado para as aplicações consumidoras.

A interface de comunicação com aplicações externas é provida pelo *DT Gateway*. Baseado no padrão *API Gateway*, ele centraliza a exposição de serviços RESTful, gerenciando o roteamento e a composição de requisições, bem como a integração com aplicações de terceiros e sistemas de DTs independentes. O *gateway* oferece uma interface estável para acesso a dados agregados (por exemplo, retornando uma instância de DT juntamente com sua telemetria recente) e oferece suporte a interações síncronas e assíncronas, incluindo a emissão de identificadores para processamento em segundo plano.

De forma transversal, a camada *Security Management* garante o controle centralizado de autenticação e autorização. Considerando que sistemas de DTs frequentemente operam em ambientes críticos, essa camada gerencia políticas de acesso, emite credenciais e tokens de curta duração e assegura a comunicação segura entre o middleware, os *gateways* IoT e os clientes da API.

5.4.2. Implementação

No âmbito da implementação, todos os componentes do MidDiTS são empacotados como *containers*, o que facilita a implantação, a orquestração e o isolamento de dependências (ver Figura 5.6). A maior parte do *middleware* foi desenvolvida em Python, utilizando o *framework* Django. Para suportar as funcionalidades do *DT Gateway* e da camada de segurança, utiliza-se o nginx[12] como *proxy* reverso, balanceador de carga e mecanismo adicional de proteção, garantindo maior resiliência e escalabilidade.

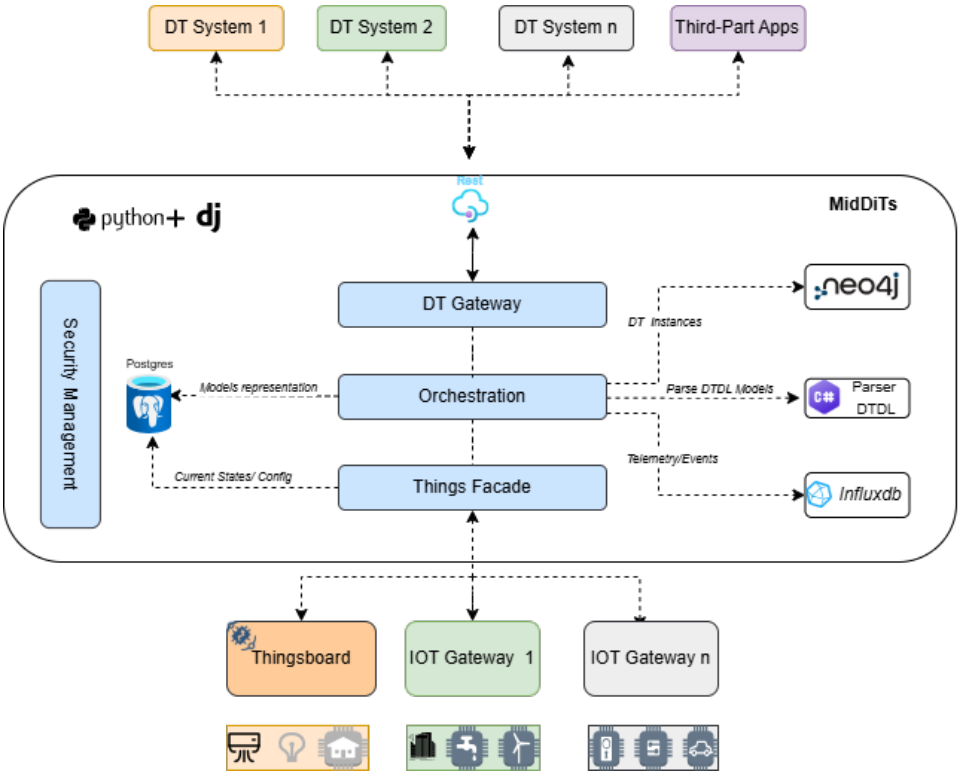

Figura 5.6. Visão geral da implementação atual da arquitetura do MidDiTS

[12]https://nginx.org

Para padronizar a definição das instâncias, o MidDiTS adota a DTDL, permitindo especificar interfaces, telemetrias, comandos e relacionamentos de ativos físicos. A conversão dessas definições para representações estruturadas utilizadas internamente é realizada por meio de um motor de *parsing* (*DTDL Parser*), desenvolvido pelo DTC[13], executado em *container* dedicado e acessado via REST.

A persistência de dados no MidDiTS reflete a heterogeneidade das informações manipuladas em um ecossistema de DTs, sendo implementada por meio da integração de três tecnologias distintas. O PostgreSQL é utilizado como repositório relacional para definições estruturais, modelos convertidos e dados administrativos. O Neo4j[14] é empregado para modelar a estrutura dinâmica do sistema como um grafo, permitindo consultas complexas e navegação eficiente entre instâncias e seus relacionamentos. Por sua vez, o InfluxDB[15] é utilizado para armazenamento de séries temporais, registrando dados de telemetria e permitindo análise de latência e sincronização.

A comunicação bidirecional com a contraparte física é delegada ao *ThingsBoard IoT Gateway*[16], que traduz comandos do *middleware* para protocolos nativos dos dispositivos e expõe o estado da infraestrutura física. Para evitar sobrecarga de rede associada a técnicas de *polling*, o MidDiTS adota um modelo orientado a eventos baseado em WebSockets, garantindo baixa latência na propagação de atualizações.

Para suprir limitações da modelagem com a DTDL no que diz respeito à sincronização bidirecional, o MidDiTS introduz a diretiva *Causal*. Quando aplicada a uma propriedade, essa diretiva ativa um mecanismo dedicado de sincronização, garantindo a propagação consistente de alterações entre os domínios físico e digital.

Por fim, o acoplamento entre propriedades do DT e sensores/atuadores do *gateway* IoT é realizado por meio do conceito de *binding*. Esse processo pode ser definido explicitamente pelo desenvolvedor ou automatizado por meio de um mecanismo de *binding* semântico, que utiliza técnicas de Inteligência Artificial para gerar representações vetoriais (*embeddings*) a partir de descrições textuais. A similaridade semântica é então calculada (por exemplo, via similaridade de cosseno), permitindo a inferência automática de correspondências entre elementos físicos e digitais. Essa abordagem reduz o esforço de configuração e melhora a integração em ambientes IoT heterogêneos e de larga escala.

5.5. Processo de Desenvolvimento de Sistemas de Gêmeos Digitais com o MidDiTS

Como detalhado na seção anterior, o MidDiTS atua como uma camada de abstração entre o mundo físico e o virtual. Sob a perspectiva de quem projeta sistemas de DTs, seu principal valor está em permitir que o desenvolvimento se concentre na lógica de negócio e nas aplicações finais, enquanto aspectos como comunicação de baixo nível, interoperabilidade e consistência de dados são gerenciados pelo *middleware*.

O processo de desenvolvimento de sistemas de DTs com o MidDiTS segue uma

[13]https://github.com/digitaltwinconsortium/DTDLParser
[14]https://neo4j.com/product/neo4j-graph-database/
[15]https://www.influxdata.com/products/influxdb/
[16]https://thingsboard.io/docs/iot-gateway/

abordagem estruturada em quatro fases principais (ver Figura 5.7): (i) *Projeto*, responsável pela definição dos modelos; (ii) *Configuração*, na qual o ambiente é preparado e integrado aos dispositivos físicos; (iii) *Desenvolvimento*, que materializa instâncias e relacionamentos; e (iv) *Operação*, que garante a execução contínua e sincronizada do sistema. A seguir, cada uma dessas fases é detalhada a partir do cenário de um condomínio inteligente, baseado na *Smart Cities Ontology*[17], composto por casas, cômodos, sensores e atuadores interconectados, conforme ilustrado na Figura 5.8.

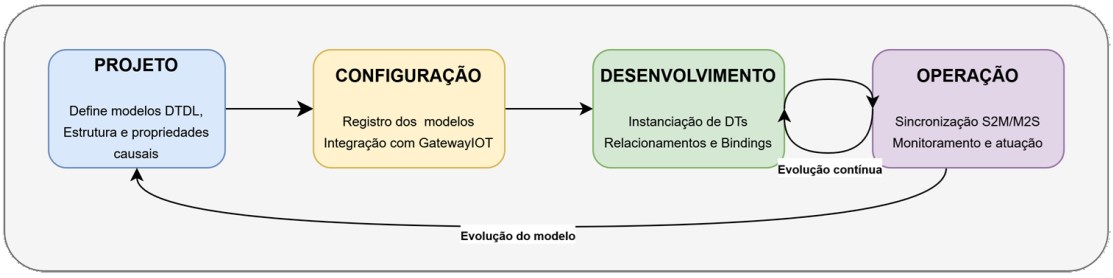

Figura 5.7. Processo de desenvolvimento de sistemas de DTs com o MidDiTS

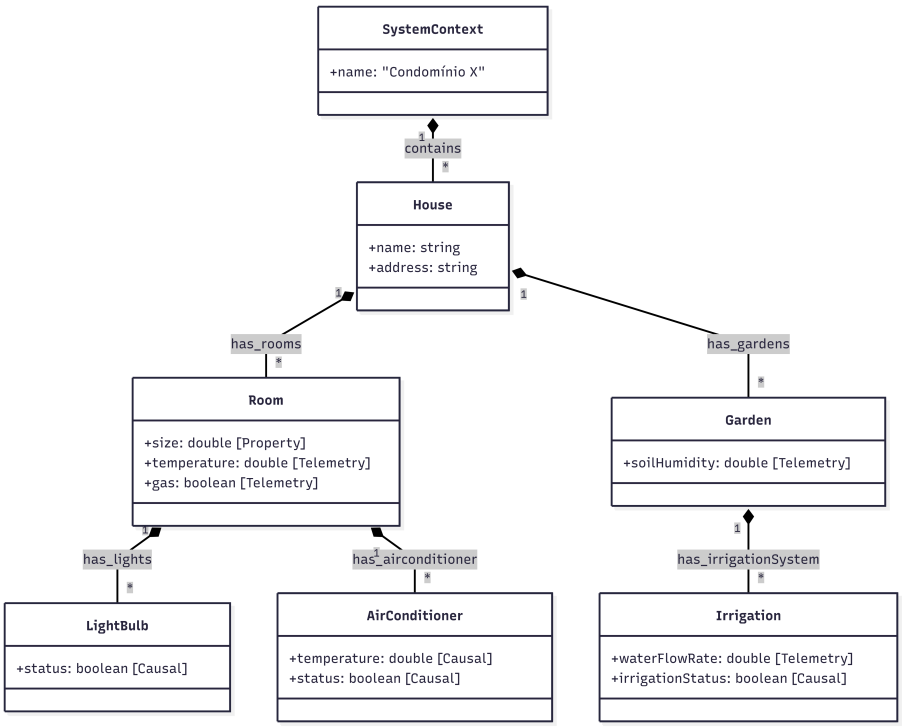

Figura 5.8. Condomínio inteligente baseado na *Smart Cities Ontology*

5.5.1. Projeto – Modelagem com DTDL

A fase de projeto estabelece a base conceitual de um ecossistema de DTs. Nela, o desenvolvedor define as representações digitais dos ativos físicos utilizando a linguagem

[17] https://github.com/Azure/opendigitaltwins-smartcities

DTDL, que permite descrever, de forma padronizada e processável por máquina, propriedades, telemetrias, comandos e relacionamentos. De forma geral, o desenvolvedor deve estruturar os modelos considerando quatro dimensões principais:

- **Modelos estruturais e espaciais**: representam a organização lógica e física do ambiente, como *House*, *Room* e *Garden*.
- **Modelos de dispositivos**: representam sensores e atuadores conectados, como *LightBulb* e *AirConditioner*.
- **Propriedades e fluxos de dados**: descrevem atributos monitorados por sensores ou controlados por atuadores.
- **Relacionamentos**: definem como as entidades se conectam, formando a estrutura do DT.

Essa organização permite compreender o modelo como uma estrutura conectada ao ambiente físíco e não apenas como um conjunto isolado de atributos, refletindo assim tanto sua organização quanto seu comportamento. Com base nessa modelagem, o MidDiTS introduz extensões e mecanismos que enriquecem a semântica desses modelos, permitindo uma integração mais direta e automatizada entre o mundo físico e o virtual.

Telemetria vs. propriedades causais. Modelos tradicionais em DTDL contemplam a definição de propriedades, comandos e telemetrias, porém não explicitam, de forma semântica, quais propriedades devem manter sincronização ativa com o ambiente físico. Na prática, essa relação costuma ser delegada à lógica da aplicação, o que dificulta a padronização e a compreensão do comportamento do DT. O MidDiTS trata essa limitação ainda na fase de projeto ao introduzir *propriedades causais* como extensão à DTDL, permitindo declarar explicitamente quais atributos possuem relação direta com dispositivos físicos. Assim, uma telemetria representaria dados observacionais, sem garantia de sincronização contínua, enquanto uma propriedade causal seria um atributo com sincronização bidirecional consistente entre os mundos físicos e virtual. Essa abordagem fortalece a coerência do modelo e viabiliza a automação da sincronização.

Para habilitar esse comportamento, o desenvolvedor inclui a extensão causal `dtmi:dtdl:extension:causal` no início do arquivo JSON correspondente ao modelo. O Código 5.1 mostra um modelo DTDL de um sistema de irrigação. Nesse trecho de código, a extensão causal é declarada no atributo `@context` (linha 2), permitindo que determinadas propriedades do modelo sejam tratadas como causais. A telemetria denominada `waterFlowRate` (linhas 8 a 12) representa um fluxo de dados proveniente do mundo físico, sendo utilizada para o monitoramento do fluxo de água. Em contraste, a propriedade denominada `irrigationStatus` (linhas 15 a 19) é definida como uma propriedade causal, o que indica que seu valor está diretamente associado ao estado de um dispositivo físico, por exemplo, uma válvula de irrigação, permitindo não apenas a leitura, mas também a atuação sobre o ambiente. Dessa forma, alterações realizadas no modelo digital podem ser automaticamente propagadas para o dispositivo físico, enquanto mudanças no ambiente físico atualizam o estado do DT, caracterizando uma relação causal contínua entre os dois domínios.

Ontologia e tipos quantitativos. No nosso cenário do condomínio inteligente, a modelagem é fundamentada na *Smart Cities Ontology*. A partir dessa ontologia, entidades

```
{
    "@context": ["dtmi:dtdl:context;3","dtmi:dtdl:extension:causal;1"
        ↪ ],
    "@id": "dtmi:housegen:Irrigation;1",
    "@type": "Interface",
    "displayName": "Irrigation",
    "contents": [
        {
            "@type": "Telemetry",
            "name": "waterFlowRate",
            "schema": "double",
            "displayName": "Water Flow Rate",
            "description": "The rate of water flow in liters per
                ↪ minute"
        },
        {
            "@type": ["Property","Causal"],
            "name": "irrigationStatus",
            "schema": "boolean",
            "displayName": "Set the Irrigation status",
            "description": "Defines if the irrigation system is ON or
                ↪  OFF"
        }
    ]
}
```

Código 5.1. Modelo DTDL com extensão causal para sistema de irrigação

como *House*, *Room* ou *Garden* passam a ser tratadas como objetos espaciais (*SpatialObject*), permitindo a incorporação de conceitos geoespaciais por meio de padrões como o GeoSPARQL[18]. Elementos fundamentais como *Geometry* e *Feature* possibilitam representar não apenas a estrutura lógica do ambiente, mas também sua organização espacial, o que é essencial em cenários urbanos e de cidades inteligentes.

Para representar grandezas físicas de forma precisa, o MidDiTS utiliza a extensão de tipos quantitativos (`dtmi:dtdl:extension:quantitativeTypes`). Essa extensão permite que propriedades sejam associadas a unidades e classificações físicas padronizadas. Por exemplo, ao modelar um cômodo (*Room*), a propriedade *size* pode ser definida como uma área, medida em metros quadrados, enquanto sensores de temperatura podem ser explicitamente definidos considerando temperatura em graus Celsius. Isso reduz ambiguidades e facilita a interoperabilidade entre sistemas.

Definição da estrutura do sistema de DTs. Outro aspecto fundamental da fase de projeto é a definição explícita da estrutura do sistema de DTs, ou seja, como os diferentes componentes relacionam-se entre si. Essa estrutura é descrita diretamente nos modelos DTDL por meio do atributo `Relationship`. Por exemplo, o modelo *House* pode definir um relacionamento *has_rooms*, apontando para instâncias do modelo *Room*. Da mesma forma, o modelo *Room* pode estabelecer o relacionamento *has_lights* com uma multiplicidade definida para indicar a quantidade máxima de dispositivos associados. Essa definição não é apenas estrutural, mas também operacional, pois estabelece a forma como as in-

[18]https://www.ogc.org/standards/geosparql/

stâncias de DTs se conectam e interagem dentro do sistema. No MidDiTS, essas relações são utilizadas pela *Orchestration Layer* para materializar relacionamentos no banco de dados orientado a grafos (Neo4j), evidenciando o sistema de DTs como uma rede de entidades interconectadas. Dessa forma, decisões tomadas ainda na fase de modelagem impactam diretamente a forma como consultas, navegação e análises serão realizadas posteriormente.

Ao final da fase de projeto, o desenvolvedor terá em mãos um conjunto de modelos DTDL semanticamente enriquecidos, capazes de representar não apenas as propriedades e comportamentos dos ativos físicos, mas também suas relações estruturais, espaciais e causais. Esses modelos constituem a base para as etapas seguintes do processo de desenvolvimento de um sistema de DTs com o MidDiTS, permitindo que o *middleware* automatize a criação, sincronização e operação dos DTs no ambiente de execução.

5.5.2. Configuração

A fase de configuração no MidDiTS pode ser compreendida como um *pipeline* de ativação do sistema de DTs, no qual os modelos definidos na fase de projeto são progressivamente conectados ao ambiente de execução e aos serviços do *middleware*. Diferentemente da fase de projeto, que se concentra na definição estrutural e semântica dos DTs, a configuração estabelece as condições necessárias para que essas definições sejam efetivamente executadas, integrando modelos digitais, dispositivos físicos (ou simulados) e os mecanismos internos do *middleware*. Esse processo organiza-se em três etapas principais: (i) definição do contexto de execução e controle de acesso; (ii) integração com a infraestrutura física por meio da camada *Things Facade*; e (iii) preparação do ambiente para execução dos DTs. Cada uma dessas etapas é detalhada a seguir.

Definição do contexto de execução e controle de acesso. O processo de configuração inicia-se com a autenticação do desenvolvedor e a definição do contexto organizacional no qual os DTs serão gerenciados. Esse contexto estabelece o escopo de isolamento das aplicações, permitindo que diferentes sistemas de DTs coexistam na mesma infraestrutura de forma independente.

Nesse momento, são definidos elementos como organização, pessoas usuárias e papéis, gerenciados pela camada *Security Management*, responsável por garantir autenticação, autorização e controle de acesso às operações sobre os DTs. A Figura 5.9 ilustra o cadastro de uma organização e a associação de seus membros. Esse mecanismo garante o isolamento de contexto entre diferentes aplicações no MidDiTS.

Integração com a infraestrutura física (*Things Facade*). Uma vez estabelecido o contexto de execução, o próximo passo consiste na integração com o ambiente físico por meio do registro de um *gateway* IoT externo, como a *ThingsBoard*. Essa etapa ativa a camada *Things Facade*, responsável por abstrair os detalhes de comunicação com dispositivos físicos e consolidar suas representações no *middleware*.

O registro do *gateway* inclui a configuração de credenciais de acesso e a definição da URL do serviço. A Figura 5.10 apresenta um exemplo do cadastro de um *gateway* IoT no MidDiTS.

Após o registro, o MidDiTS realiza a descoberta automática de dispositivos disponí-

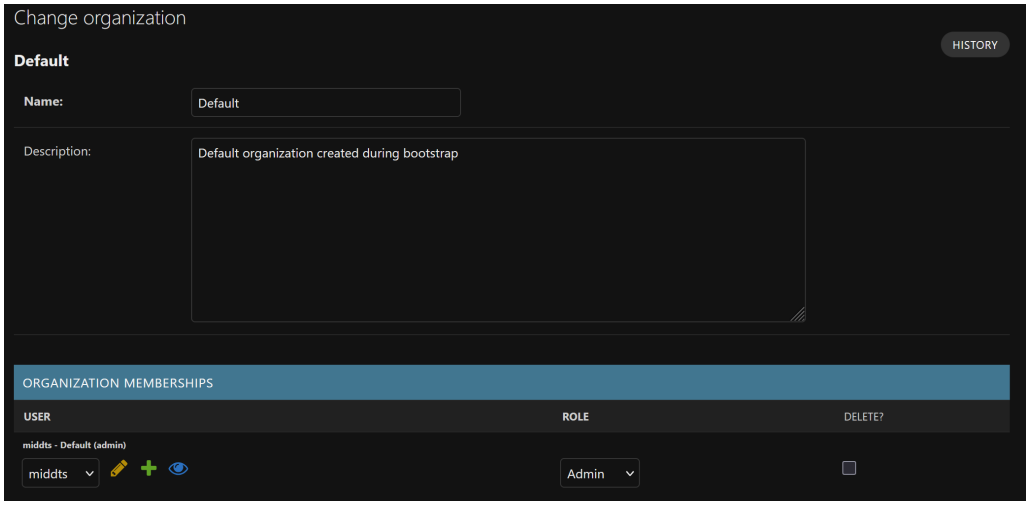

Figura 5.9. Cadastro de organização e gerenciamento de membros no MidDiTS

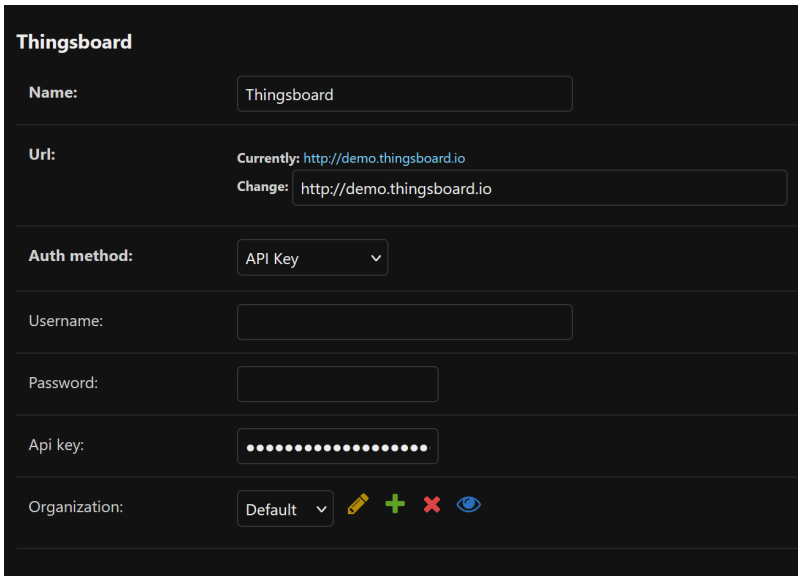

Figura 5.10. Cadastro de *gateway* IoT para integração com a *ThingsBoard*

veis no *gateway*. Esse processo consiste na consulta a APIs para obtenção de dispositivos, seus identificadores e metadados associados. Esses metadados incluem nomes, rótulos, atributos e chaves de telemetria, sendo fundamentais para a identificação e posterior associação com os elementos do modelo digital.

A Figura 5.11 ilustra o processo de descoberta de dispositivos. Nesse processo, o MidDiTS, por meio da camada *Things Facade*, realiza uma requisição ao *gateway* IoT para obtenção dos dispositivos cadastrados. Como resposta, o *gateway* retorna os metadados associados, que são então processados e armazenados pelo *middleware*, permitindo a construção de uma representação local dessas entidades.

Preparação do ambiente de execução dos DTs. Com a infraestrutura física integrada, a configuração avança para a preparação do ambiente de execução dos DTs. Nessa etapa,

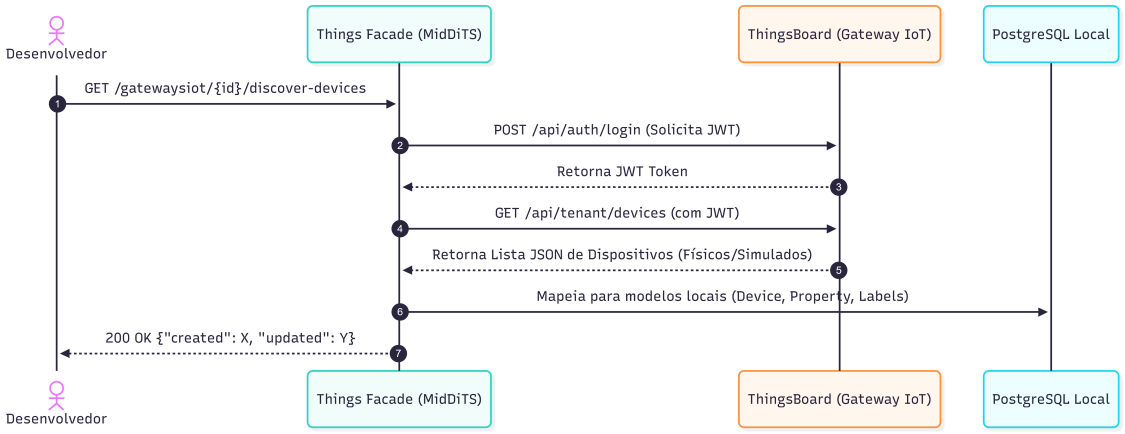

Figura 5.11. Fluxo de descoberta de dispositivos a partir do *gateway* IoT

os modelos previamente definidos em DTDL são registrados no *middleware*, permitindo que o sistema compreenda as estruturas, propriedades, telemetrias e relacionamentos das entidades.

Além do registro dos modelos, essa etapa envolve a organização das propriedades dos DTs e dos dispositivos físicos em um espaço comum de representação, no qual cada propriedade virtual pode ser potencialmente associada a uma propriedade física correspondente. Para isso, o sistema organiza os metadados provenientes do *gateway* IoT e os relaciona com as definições estruturais dos modelos digitais, estabelecendo o conjunto de candidatos que será utilizado no processo de *binding*.

Por fim, o sistema é conectado a um ambiente de execução, que pode ser composto por dispositivos reais ou por um simulador IoT. No contexto deste trabalho, utilizamos um simulador IoT para permitir a execução de experimentos de forma controlada, sem dependência de infraestrutura física.

Ao final desta fase, o sistema encontra-se plenamente configurado, com modelos registrados, dispositivos descobertos e ambiente preparado. O MidDiTS está, portanto, pronto para a materialização das instâncias e o estabelecimento dos vínculos com o mundo físico, que ocorrem na fase de desenvolvimento.

5.5.3. Desenvolvimento – Instanciação, Relacionamento e *Binding* dos DTs

Com o ambiente configurado e os modelos previamente definidos, a fase de desenvolvimento no MidDiTS corresponde à materialização dos DTs como entidades operacionais. É nesse momento que as abstrações definidas na fase de projeto tornam-se instâncias concretas, interconectadas e vinculadas ao mundo físico.

Criação de instâncias de DTs. A partir dos modelos definidos em DTDL, o desenvolvedor cria as instâncias de DTs, representando a transição do modelo abstrato para uma entidade executável dentro do sistema. O MidDiTS utiliza o banco de dados baseado em grafos Neo4j para armazenar essas instâncias, o que permite representar como os diferentes DTs se conectam entre si. Dessa forma, em vez de tratar cada elemento de forma isolada, o sistema passa a enxergar casas, cômodos e dispositivos como parte de

uma rede interligada, facilitando a navegação, a análise contextual e a realização de consultas baseadas nessas conexões. Cada instância de DT torna-se então um nó com o rótulo `DIGITALTWIN` e cada uma de suas propriedades torna-se um nó satélite `TWINPROPERTY`, ligado ao DT pela aresta `HAS_PROPERTY`.

Para facilitar o desenvolvimento de sistemas com muitas instâncias de DTs (como um bairro inteiro, por exemplo), o MidDiTS expõe um *endpoint* de criação hierárquica. Em vez de criar os nós um a um, o desenvolvedor pode enviar um único objeto JSON representando a árvore do ambiente para realizar um cadastro em lote, conforme o Código 5.2.

```
{
  "House 1": {
      "Room 1": {
          "LightBulb 1": {},
          "Air conditioner 1": {}
      },
      "Garden 1": {
          "Irrigation 1": {}
      }
  }
}
```

Código 5.2. Especificação em JSON de uma casa de forma hierárquica

Estabelecendo relacionamentos no grafo. Sistemas de DTs não são apenas listas de dispositivos, mas sim redes de entidades interconectadas. Embora seja possível representar parte dessa organização de forma hierárquica (por exemplo, casa → cômodo → dispositivo), essa abordagem não é suficiente para capturar todas as relações existentes no sistema. Isso ocorre porque, em cenários reais, as conexões entre entidades não se limitam a estruturas em árvore. Um mesmo tipo de dispositivo, como uma lâmpada, pode estar associado a diferentes contextos (por exemplo, a um cômodo específico ou a uma área funcional como uma cozinha) dependendo da modelagem adotada. Dessa forma, o desenvolvedor pode estabelecer explicitamente os relacionamentos entre os DTs, conforme as regras definidas nos modelos DTDL. No Neo4j, o MidDiTS materializa essas conexões por meio da criação de arestas entre os nós, permitindo representar o sistema como uma rede flexível, navegável e semanticamente rica de entidades interconectadas. A Figura 5.12 mostra a representação de um sistema completo de uma casa e seus relacionamentos.

***Binding* entre DTs e dispositivos físicos.** O passo fundamental desta fase é o *binding*, no qual propriedades dos DTs são associadas a dispositivos físicos registrados no sistema. A partir dessa conexão, o MidDiTS se encarrega de sincronizar automaticamente as contrapartes físicas e digitais. O MidDiTS suporta duas abordagens para o estabelecimento de *binding*:

1) *Binding* explícito (manual): O desenvolvedor associa diretamente um dispositivo físico a uma propriedade do DT por meio de chamadas à API do *middleware*.
2) *Binding* semântico: O MidDiTS utiliza técnicas de processamento de linguagem natural para inferir automaticamente correspondências entre propriedades dos DTs e dispositivos físicos com base na similaridade entre seus contextos.

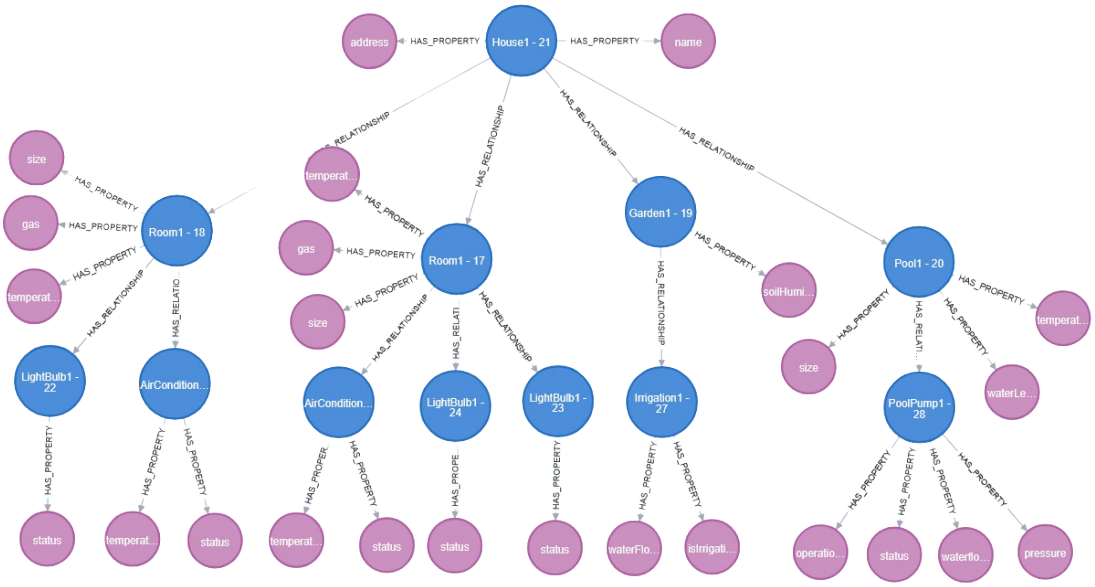

Figura 5.12. Representação de uma casa, sua hierarquia e relacionamentos no Neo4j

O MidDiTS considera o conjunto de metadados provenientes do *gateway* IoT, incluindo nomes, rótulos, atributos, descrições e chaves de telemetria. Em cenários com grande número de dispositivos, o processo de mapeamento individual dos dispositivos físicos às propriedades dos DTs torna-se inviável. Dessa forma, o *binding* semântico é realizado por meio de um modelo de *embeddings* de sentenças [Reimers and Gurevych 2019] (como o *all-MiniLM-L6-v2*[19]) para transformar tanto o contexto virtual quanto os metadados dos dispositivos físicos em representações vetoriais comparáveis. A similaridade entre esses elementos é calculada utilizando métricas como a similaridade de cosseno [Manning et al. 2008], permitindo identificar automaticamente correspondências mesmo quando há inconsistências ou variações na nomenclatura dos dispositivos.

Além da similaridade semântica capturada pelo modelo de *embeddings*, o MidDiTS implementa um *score* híbrido de correspondência. Esse *score* combina a proximidade semântica entre os textos descritivos com evidências adicionais extraídas dos nomes das instâncias, dos dispositivos e de formas canônicas das propriedades comparadas, reduzindo diferenças causadas por variações de escrita, prefixos, sufixos ou convenções distintas de nomeação. Esse mecanismo torna-se particularmente relevante em cenários com múltiplas entidades semelhantes. Por exemplo, em uma residência com *Room1* e *Room2*, ambas contendo um dispositivo chamado *AirConditioner1*, apenas a similaridade textual pode não ser suficiente para distinguir corretamente os vínculos. Ao considerar também o contexto da instância, o MidDiTS consegue associar a propriedade de temperatura de *Room1* ao *AirConditioner1* pertencente ao mesmo cômodo, evitando associações com dispositivos de outros ambientes. Com isso, o processo de sugestão de *binding* torna-se menos dependente apenas da similaridade textual e passa a considerar também sinais estruturais e contextuais presentes nos metadados.

[19]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Com base nesse *score* híbrido, o MidDiTS gera um conjunto de candidatos a *binding*, representando possíveis associações entre propriedades de DTs e dispositivos físicos. Esses candidatos são então ordenados conforme seu grau de correspondência e o *binding* pode ser estabelecido considerando um limiar mínimo de aceitação configurável pelo desenvolvedor. Esse limiar define o nível de confiança necessário para a associação e assume valores no intervalo $[0, 1]$, permitindo ajustar o nível de restrição das sugestões: valores mais altos retornam apenas associações de maior confiança, enquanto valores mais baixos ampliam o conjunto de candidatos considerados.

A Figura 5.13 ilustra o processo de estabelecimento de *binding* semântico, destacando a correspondência entre o contexto virtual e os metadados dos dispositivos físicos. Para operacionalizar esse processo, o MidDiTS disponibiliza APIs de *preview* e de efetivação de *binding*. A API de *preview* retorna uma lista ordenada de candidatos, permitindo ao desenvolvedor avaliar as sugestões antes de sua aplicação. Adicionalmente, o sistema permite restringir a análise a propriedades não vinculadas, considerar apenas propriedades causais, limitar o número de resultados e controlar o reúso de propriedades físicas.

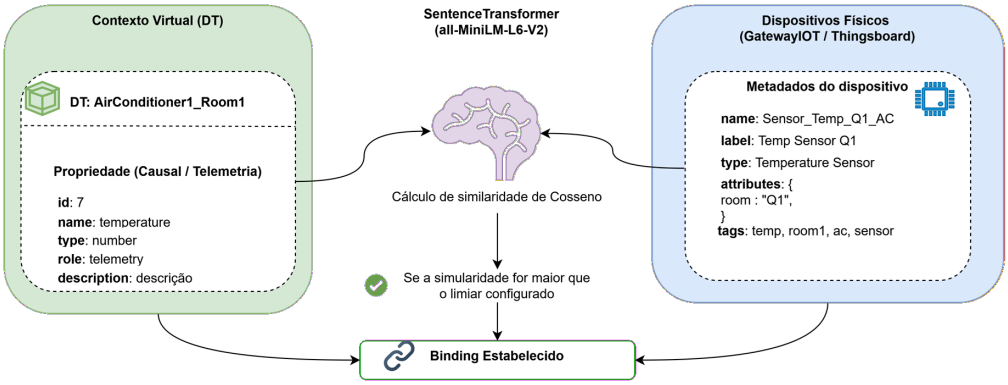

Figura 5.13. Representação do processo de *binding* semântico entre propriedades de DTs e dispositivos físicos.

Uma vez estabelecidos os *bindings*, o sistema passa a operar de forma contínua, com sincronização entre os estados físico e virtual. Nesse ponto, os DTs deixam de ser apenas representações informacionais e passam a atuar como entidades ciberfísicas, capazes de refletir e influenciar o comportamento do ambiente real, caracterizando o início da fase de operação.

5.5.4. Operação

Com a conclusão da fase de desenvolvimento, os DTs passam a operar como entidades ativas, mantendo uma sincronização contínua com suas contrapartes físicas. Na fase de operação, o foco desloca-se da construção estrutural para a manutenção da consistência, atualidade e fidelidade dos estados representados.

A operação no MidDiTS é sustentada por um fluxo bidirecional de dados, composto por dois processos principais: a propagação de telemetria do mundo físico para o digital (*sensor-to-middleware* – S2M) e a execução de comandos do modelo digital sobre os dispositivos físicos (*middleware-to-sensor* – M2S). Esse comportamento caracteriza

um ciclo contínuo de monitoramento, decisão e atuação, no qual aplicações externas observam o estado do ambiente físico, processam informações e influenciam diretamente esse ambiente por meio do DT, utilizando as APIs expostas pelo *middleware*. Internamente, essas operações são processadas por diferentes camadas do MidDiTS. A camada *Orchestration Layer* é responsável por interpretar as requisições sobre os DTs, enquanto a *Things Facade* gerencia a comunicação com o *gateway* IoT e a execução dos comandos nos dispositivos físicos.

Fluxo bidirecional. No fluxo S2M, dispositivos físicos enviam dados de telemetria, geralmente por meio de um *gateway* IoT (como a *ThingsBoard*). Esses dados são periodicamente capturados pelo MidDiTS, que atualiza as propriedades correspondentes nos DTs. Quando associadas a propriedades causais, essas atualizações não são meramente informativas, mas representam mudanças de estado que devem ser refletidas de forma consistente no modelo digital. Esse mecanismo garante que o DT permaneça alinhado com o estado atual do ambiente físico.

No sentido inverso, o fluxo M2S permite que alterações realizadas no modelo digital sejam propagadas para os dispositivos físicos por meio de comandos RPC. Por exemplo, ao alterar o estado de uma lâmpada no DT, o MidDiTS emite automaticamente uma requisição para o dispositivo correspondente, garantindo que a ação seja efetivamente executada no mundo físico. Esse comportamento é diretamente suportado pelas propriedades causais definidas na fase de modelagem.

Interface de operação via APIs. Durante a fase de operação, a interação com os DTs ocorre principalmente por meio das APIs expostas pelo MidDiTS. Essas APIs abstraem a complexidade da comunicação com os dispositivos físicos e permitem que aplicações externas manipulem os estados dos DTs de forma padronizada. De forma geral, o MidDiTS disponibiliza operações para: (i) consulta de estado de instâncias de DTs; (ii) atualização de propriedades, incluindo propriedades causais; (iii) execução de comandos associados a atuadores físicos; e (iv) monitoramento de eventos e telemetria.

Diferentemente de sistemas REST tradicionais, onde operações afetam apenas representações de dados, no contexto de DTs essas interações podem desencadear efeitos físicos no ambiente monitorado. Um exemplo de interação via requisições HTTP para consulta e atualização de propriedades de um DT é ilustrado nos Códigos 5.3 e 5.4, incluindo os respectivos *payloads* e respostas retornadas pelo *middleware*. Essas requisições seguem o padrão RESTful adotado pelo MidDiTS, permitindo que aplicações externas consultem estados e atuem sobre dispositivos físicos de forma transparente.

O Código 5.3 ilustra o consumo da API do MidDiTS para consulta de uma propriedade do DT e o respectivo reflexo no dispositivo físico. Quando essa propriedade é definida como causal, também é possível, por meio da API, manipular o valor de uma propriedade de um DT.

```
1 curl -X GET \
2 http://[host]/api/orchestrator/systems/1/instances/24/properties/7/
      value/ \
3 -H "Content-Type: application/json"
4 Resposta:
5 {
```

```
6     "value": "true"
7   }
```

Código 5.3. Consulta ao estado de uma propriedade do DT

O Código 5.4 mostra um exemplo de uma requisição HTTP PUT no MidDiTS alterando o status de uma propriedade de um DT e, consequentemente, do dispositivo físico.

```
1 curl -X PUT \
2 http://[host]/api/orchestrator/systems/1/instances/24/properties/7/
      value/ \
3 -H "Content-Type: application/json" \
4 -d '{"value": false}'
```

Código 5.4. Atualização de propriedade causal

A execução dessa requisição desencadeia uma sequência de interações internas no *middleware* e no *gateway* IoT, que resultam na atuação sobre o dispositivo físico. A Figura 5.14 ilustra esse fluxo de forma detalhada.

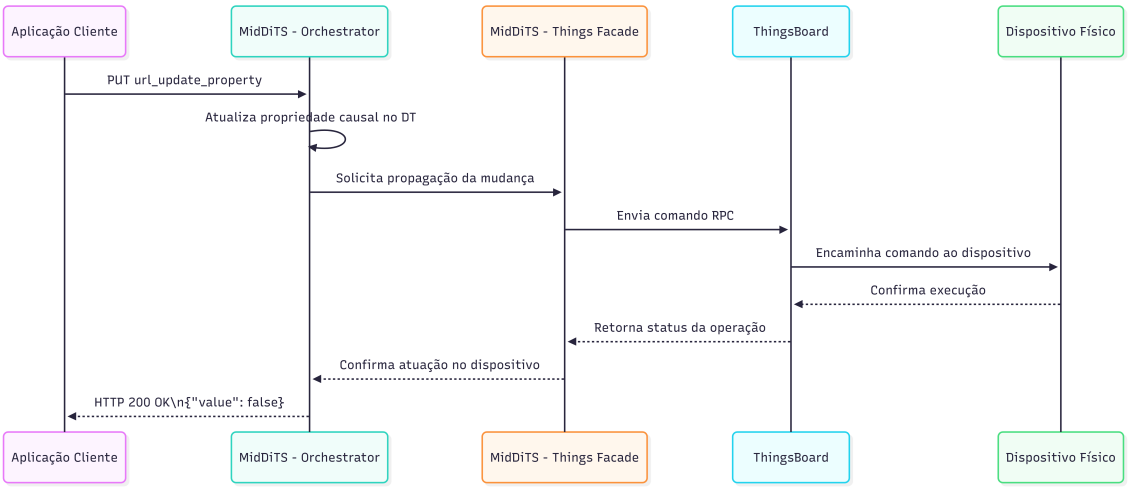

Figura 5.14. Representação do fluxo de atuação do modelo digital sobre o dispositivo físico (M2S)

Observa-se que não é necessário que o desenvolvedor conheça os mecanismos específicos de comunicação com o dispositivo físico, uma vez que o MidDiTS abstrai essa complexidade e gerencia automaticamente a propagação das alterações. Dessa forma, a API do MidDiTS não manipula apenas representações de dados, mas influencia diretamente o estado do mundo físico quando associada a propriedades causais.

O MidDiTS atua como elemento central na orquestração desses fluxos, sendo responsável por: (i) monitorar continuamente as atualizações de telemetria; (ii) identificar mudanças relevantes nas propriedades causais; (iii) garantir a consistência entre os estados físico e digital; e (iv) executar comandos e validar suas respostas. Essa operação contínua transforma o sistema em uma arquitetura ciberfísica, na qual decisões podem ser tomadas com base em um modelo digital atualizado em tempo quase real.

A fase de operação consolida o DT como um sistema dinâmico e responsivo. Diferentemente de uma representação estática, o DT passa a atuar como um agente ativo no ecossistema, capaz de refletir, analisar e influenciar o ambiente físico em tempo real. Nesse contexto, a operação não é apenas uma etapa do ciclo de vida, mas o elemento que sustenta a própria essência dos DTs: a manutenção de uma relação causal contínua e operacional entre o mundo físico e o digital, permitindo que decisões e ações sejam realizadas de forma consistente, confiável e responsiva. Esse comportamento operacional aproxima os DTs de sistemas ciberfísicos reativos, nos quais o modelo digital não apenas representa, mas participa ativamente da dinâmica do ambiente físico.

5.6. Conclusão

Este capítulo apresentou os principais conceitos, desafios e tecnologias relacionados a sistemas de DTs, bem como o uso de plataformas de *middleware* como estratégia para viabilizar sua construção de forma sistemática. Como discutimos ao longo do texto, o sucesso de um sistema de DT não reside apenas na coleta de dados, mas na capacidade de manter uma sincronia bidirecional e causal entre os domínios físico e digital.

Através deste minicurso, foi possível identificar a série de desafios técnicos e tecnológicos envolvidos na construção de um sistema de DTs, de modo a lidar com seus requisitos específicos. Observou-se ainda como um *middleware* atua como importante facilitador nesse contexto: o uso de camadas de abstração, como exemplificado pelo MidDiTS, é essencial para gerenciar a heterogeneidade e reduzir a complexidade do desenvolvimento. Nesse cenário, descrevemos e comparamos algumas soluções proprietárias e de código aberto, bem como outras propostas na literatura. A análise comparativa evidenciou que nenhuma dessas plataformas atende simultaneamente a todos os requisitos de um ecossistema de DT, motivando o desenvolvimento de soluções complementares.

Por fim, demonstramos o uso do MidDiTS como *middleware* para a construção de sistemas de DTs ao longo de quatro fases do ciclo de vida: projeto, configuração, desenvolvimento e operação. Na fase de projeto, apresentamos como construir modelos utilizando a linguagem DTDL, especificando elementos do ambiente do mundo real, dispositivos, propriedades de telemetria e causais e relacionamentos entre as entidades existentes. Na fase de configuração, foram apresentados o registro de tais modelos DTDL, a integração com o *gateway* IoT *ThingsBoard* e o uso de um simulador para validação independente de hardware físico. Na fase de desenvolvimento, foram abordadas a instanciação de DTs, o estabelecimento de relacionamentos entre entidades e o *binding* entre propriedades virtuais e dispositivos físicos. Na fase de operação, foram descritos os fluxos bidirecionais de sincronização e o papel central do MidDiTS na orquestração contínua dos estados físico e digital, transformando o sistema em uma arquitetura ciberfísica dinâmica e responsiva.

Em suma, o conteúdo deste capítulo demonstra que a adoção de um *middleware* especializado, orientado por um processo sistemático de desenvolvimento, é uma abordagem viável e eficaz para mitigar alguns dos desafios inerentes à construção de sistemas de DTs, promovendo interoperabilidade, reduzindo a complexidade de integração e habilitando a sincronização contínua entre os mundos físico e o digital ao longo de todo o ciclo de vida da entidade-alvo. Esperamos que este material sirva como guia para desenvolve-

dores e pesquisadores explorarem o potencial transformador dos DTs na próxima década de transformação digital.

Agradecimentos

Este trabalho possui apoio do Instituto Nacional de Ciência e Tecnologia em Redes de Comunicação e Internet das Coisas Inteligentes – INCT ICoNIoT (http://iconiot.ic.unicamp.br), financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq, processo nº 405940/2022-0, e pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES, processo nº 88887.954253/2024-00.

A elaboração deste capítulo contou com o apoio de ferramentas de Inteligência Artificial generativa para produzir algumas figuras e para a revisão do fluxo textual. Ressalta-se que todos os elementos gerados foram cuidadosamente revisados e validados pelos autores, que permanecem integralmente responsáveis pelo conteúdo e pela qualidade final deste capítulo, em plena conformidade com o Código de Conduta para Autores em Publicações da Sociedade Brasileira de Computação (SBC).

Referências

Almeida, A., Batista, T., Cavalcante, E., Delicato, F., Motta, R., and Vieira, M. (2023). Middleware for digital twins: A systematic mapping study. In *Proceedings of the 1st International Workshop on Middleware for Digital Twin*, pages 19–24, USA. ACM.

André, P., Azzi, F., and Cardin, O. (2020). Heterogeneous communication middleware for digital twin based cyber manufacturing systems. In Borangiu, T., Trentesaux, D., Leitão, P., Boggino, A. G., and Botti, V., editors, *Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future*, volume 853 of *Studies in Computational Intelligence*, pages 146–157. Springer International Publishing, Switzerland.

Bader, S., Barnstedt, E., Bedenbender, H., Berres, B., Billmann, M., and Ristin, M. (2022). Details of the Asset Administration Shell: Part 1 – The exchange of information between partners in the value chain of Industrie 4.0. Technical report, Federal Ministry for Economic Affairs and Climate Action (BMWK), Germany.

Barbie, A., Hasselbring, W., and Hansen, M. (2024). Digital twin prototypes for supporting automated integration testing of smart farming applications. *Symmetry*, 16(2):221.

Bellavista, P., Bicocchi, N., Fogli, M., Giannelli, C., Mamei, M., and Picone, M. (2024). An entanglement-aware middleware for digital twins. *ACM Transactions on Internet of Things*, 5(4):25.

Bernstein, P. A. (1996). Middleware: A model for distributed system services. *Communications of the ACM*, 39(2):86–98.

Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011). *Distributed systems: Concepts and design*. Addison-Wesley, USA, 5th edition.

Fuller, A., Fan, Z., Day, C., and Barlow, C. (2020). Digital twin: Enabling technologies, challenges and open research. *IEEE Access*, 8:108952–108971.

Iranshahi, K., Brun, J., Arnold, T., Sergi, T., and Müller, U. C. (2025). Digital twins: Recent advances and future directions in engineering fields. *Intelligent Systems with Applications*, 26.

ISO (2023). ISO/IEC 30173:2023: *Digital twin – Concepts and terminology*. https://www.iso.org/standard/81442.html.

Kritzinger, W., Karner, M., Traar, G., Henjes, J., and Sihn, W. (2018). Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022.

Kuts, V., Modoni, G., Otto, T., Sacco, M., Tähemaa, T., Bondarenko, Y., and Wang, R. (2019). Synchronizing physical factory and its digital twin through an IIoT middleware: a case study. *Proceedings of the Estonian Academy of Sciences*, 68(4):364–370.

Lin, C.-T. and Lu, H.-J. (2024). An intelligent product-driven manufacturing system using data distribution service. *IEEE Access*, 12:16447–16461.

Liu, M., Fang, S., Dong, H., and Xu, C. (2021). Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, 58:346–361.

Mahdi, M. M., Bajestani, M. S., Noh, S. D., and Kim, D. B. (2025). Digital twin-based architecture for wire arc additive manufacturing using OPC UA. *Robotics and Computer-Integrated Manufacturing*, 94.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, USA.

Minerva, R., Lee, G. M., and Crespi, N. (2020). Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models. *Proceedings of the IEEE*, 108(10):1785–1824.

Modoni, G. E., Veniero, M., Trombetta, A., Sacco, M., and Clemente, S. (2018). Semantic based events signaling for AAL systems. *Journal of Ambient Intelligence and Humanized Computing*, 9(5):1311–1325.

Pereira, L. S., Almeida, A., Delicato, F. C., Cavalcante, E., Batista, T., Motta, R., and Vieira, M. (2024). Leveraging middleware to support digital twin system development. In *Proceedings of the 2024 IEEE Smart World Congress*, pages 2081–2088, USA. IEEE.

Pylianidis, C., Osinga, S., and Athanasiadis, I. N. (2021). Introducing digital twins to agriculture. *Computers and Electronics in Agriculture*, 184.

Redelinghuys, A. J. H., Basson, A. H., and Kruger, K. (2020). A six-layer architecture for the digital twin: a manufacturing case study implementation. *Journal of Intelligent Manufacturing*, 31(6):1383–1402.

Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 3980–3990, USA. Association for Computational Linguistics.

Schantz, R. E. and Schmidt, D. C. (2002). Research advances in middleware for distributed systems: State of the art. In Chapin, L., editor, *Communication Systems*, vol-

ume 92 of *IFIP – The International Federation for Information Processing*, pages 1–36. Springer New York, USA.

Shahat, E., Hyun, C. T., and Yeom, C. (2021). City digital twin potentials: A review and research agenda. *Sustainability*, 13(6).

Tao, F., Zhang, H., Liu, A., and Nee, A. Y. C. (2019). Digital twin in industry: State-of-the-art. *IEEE Transactions on Industrial Informatics*, 15(4):2405–2415.

Tao, F. and Zhang, M. (2017). Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing. *IEEE Access*, 5:20418–20427.

van Schalkwyk, P., Whiteley, S., and Goldman, M. (2025). Digital Twin Capabilities Periodic Table - Version 1.2. https://www.digitaltwinconsortium.org/initiatives/capabilities-periodic-table/.

Yun, S., Park, J.-H., and Kim, W.-T. (2017). Data-centric middleware based digital twin platform for dependable cyber-physical systems. In *Proceedings of the 2017 Ninth International Conference on Ubiquitous and Future Networks*, pages 922–926, USA. IEEE.

Zhang, X., Lim, S., Lee, C., Song, W. S., Kim, Y. C., Yu, M., Hong, S. H., Yoo, N. H., and Wei, M. (2023). Integration of 5G and OPC UA for smart manufacturing of the future. In *Proceedings of the 2023 IEEE/SICE International Symposium on System Integration*, USA. IEEE.