UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Middleware de integração de diferentes plataformas e sistemas inteligentes IoT

Vasco Barreiros Ferreira

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Trabalho de Projeto orientado por:
Prof. Doutor José Manuel da Silva Cecílio
Prof. Doutor Pedro Miguel Frazão Fernandes Ferreira

2021

Agradecimentos

Ao meu orientador, Professor Doutor José Manuel da Silva Cecílio, que durante o desenvolvimento deste trabalho se mostrou sempre disponível para ajudar, motivando-me quando mais precisei e encaminhando-me na direção certa para a concretização deste trabalho. Ao meu co-orientador, Professor Doutor Pedro Miguel Frazão Fernandes Ferreira, que se demonstrou sempre disponível a garantir as melhores condições de trabalho e sendo crítico sobre a composição do trabalho final. Aos restantes professores envolvidos no projeto SATO que se demonstraram sempre disponíveis para dar auxílio no desenvolvimento deste trabalho.

Aos meus familiares e amigos, agradeço pelo apoio incondicional durante esta fase do meu percurso académico, mesmo sabendo que pouco ou nada poderiam fazer e acreditando sempre nas minhas capacidades. Deixando um especial agradecimento à Ana Rita, que durante os momentos mais difíceis deste trabalho deu-me forças para continuar. Ao Miguel e ao Guilherme, um obrigado por todos os momentos de apoio e paciência durante este período da minha vida. E por fim, aos meus pais pela constante preocupação e orgulho.

Dedicatória.

Resumo

Atualmente existe um grande foco na eficiência e flexibilidade energética, onde os dispositivos IoT têm uma grande importância devido às suas capacidades de monitorização e controlo. Considerado este contexto, o projeto SATO (Self Assessement Towards Optimization) tem como principais objetivos a auto-avaliação e otimização dos recursos energéticos dos edifícios, através de um sistema de gestão de edifícios autónomo que recorre ao uso de dispositivos inteligentes. Para que a plataforma desenvolvida atinja um maior número de dispositivos disponíveis no mercado, são necessárias soluções de integração dessas plataformas e sistemas inteligentes IoT, da qual resultam variadíssimos problemas.

No âmbito desta tese, desenvolveu-se uma solução de *middleware* que permite lidar com as questões de interoperabilidade resultantes da integração dos diferentes sistemas. O trabalho desenvolvido consistiu no desenho, desenvolvimento e teste de uma solução de *middleware* que permite a integração de serviços de um conjunto heterogéneo de plataformas e sistemas IoT. Esta solução é capaz de abstrair a complexidade da integração, fornecendo os serviços das plataformas e/ou sistemas IoT através de uma API unificadora. Dada a integração que esta solução garante, existe a necessidade de uniformização dos dados, das plataformas/sistemas integrados, para que estes possam ser facilmente interpretados por toda a plataforma SATO. Adicionalmente, esta solução oferece a capacidade de uniformização dos metadados produzidos pelas plataformas e/ou sistemas IoT, permitindo através destes a capacidade de enriquecimento de dados e ainda a possibilidade de extensão de novos serviços, de controlo e gestão dos dispositivos, que facilitam a otimização do consumo energético. No desenvolvimento da solução proposta é utilizada uma abordagem de micro-serviços que garante assim uma solução robusta, extensível e escalável, permitindo a integração de plataformas e/ou sistemas IoT de forma simples, baseada no desenvolvimento de um novo serviço por plataforma a integrar. Para concluir o trabalho, foi construído um protótipo experimental com base na especificação do *middleware* proposto, onde é possível avaliar a performance resultante da especificação proposta. Deste protótipo foi possível retirar resultados promissores e também alguns aspetos a melhorar para que o *middleware* atinja a totalidade dos requisitos do projeto SATO.

Palavras-chave: *Middleware*, Plataformas, Integração, IoT, Energia.

Abstract

Nowadays, there is a focus on energy efficiency and flexible energy where IoT devices can introduce some advantages due to their monitoring and remote control capabilities. In this context, the SATO (Self Assessment Towards Optimization) project was proposed. This project aims to develop a platform and a set of services for self-assessment and optimization of energy resources in buildings by using an autonomous management system for buildings through smart devices. For these devices to be available for the SATO platform to use, it relies on the integration of existing IoT energy-focused platforms and existing IoT smart systems. However, the integration of different parts of building management systems results in a common problem, due to the existence of diverse appliances, devices and technologies that must be integrated. To solve this issue, this thesis designs, implements and tests a middleware component that abstracts the specification of different proprietary solutions and exposes a generic API to ease the management of the integrated devices by abstracting the underlying details of the proprietary interfaces.

Additionally, since the platform intends to receive the data events generated by the devices and make optimizations through the usage of Machine Learning, this solution standardizes all the data injected by the integrated platforms/systems, for it to be easily interpreted by the remaining SATO components, and uses solutions which allow for the future extension of new services. For the development of the proposed solution, a micro-services approach was used. It allows developing a robust, extensible and scalable solution, where the integration of any platform can be done easily. For that integration, just a new service must be implemented.

In order to do a proof-of-concept, a prototype was developed. This prototype allowed us to test the solution, where we concluded that the proposed architecture enables the integration required to the SATO project. However, to improve the performance of a real platform, some parts of the prototype must be changed/improved.

Keywords: Middleware, Platforms, Integration, IoT, Energy.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O presente documento relata as atividades, experiências e conhecimentos obtidos durante a realização da Dissertação em Engenharia Informática, integrada no 2º e último ano do Mestrado em Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa.

Este projeto foi realizado no contexto do Projeto SATO (Self Assessment Towards Optimization of Building Energy) financiado pela União Europeia no âmbito do quadro Europeu H2020.

1.1 Motivação

Nos dias de hoje, o consumo de energia é um tema com grande importância. O elevado consumo energético tem originado deficiências no sistema, impedido-o de atingir os índices de eficiência energética definidos para 2020 [24].

Com isto, um dos principais objetivos no setor energético consiste em reduzir o consumo de energia, ou flexibilizá-lo para que este seja maioritariamente dependente de energias renováveis. Neste contexto, os dispositivos da Internet das Coisas (IoT) têm vindo a ajudar, oferecendo capacidades de monitorização e controlo remoto. Contudo, com a proliferação destes sistemas no mercado criou-se um problema de integração, configuração e operação dos mesmos. De forma a colmatar este problema, foi desenvolvida a plataforma SATO (Self Assessment Towards Optimization of Building Energy), que tem como principais objetivos a auto-avaliação e otimização dos recursos energéticos dos edifícios, garantidos através do uso de um sistema de gestão de edifícios autónomo.

O desenho e a implementação de sistemas de gestão de edifícios autónomos, baseiam-se no uso de dispositivos inteligentes através da integração de plataformas e sistemas IoT numa única plataforma, permitindo assim o controlo dos sistemas energéticos, promovendo a sua eficiência. Contudo, a integração de diferentes plataformas e sistemas IoT não é simples, originando grande parte das vezes problemas de interoperabilidade.

Dado o projeto SATO ter como objetivo a optimização energética através do controlo

de dispositivos inteligentes, é necessário colmatar o problema da interoperabilidade resultante da integração de dispositivos heterogéneos com diferentes interfaces, onde cada uma delas apresenta um desenho diferente, dificultando assim a sua utilização por parte dos componentes de optimização da plataforma SATO.

Para que esta optimização seja possível é necessária a utilização de um componente de *assessment*, que permita a avaliação do contexto dos dispositivos e respetivos edifícios, através dos dados que estes produzem. Esta necessidade de análise dos dados por parte do componente de *assessment* obriga a que o mesmo seja capaz de suportar todos os formatos proprietários de dados de cada uma das plataformas e sistemas integrados, não sendo esta uma solução adequada já que dificulta a integração de novos sistemas. Uma possível solução para este problema passa pelo desenvolvimento de uma camada de abstração das especificidades de cada plataforma e/ou sistema a integrar.

Por fim, existe a necessidade, tanto pelo componente de *assessment*, como pelo componente responsável pelos serviços de optimização, de consultar a caracterização dos dispositivos e da sua respectiva localização, para poder assim atingir os objetivos da plataforma SATO.

Através destes três principais problemas resultantes da integração de diferentes plataformas e sistemas IoT na plataforma SATO, surgiu a necessidade de desenvolver uma solução de *middleware* que facilitasse esta integração de diferentes plataformas e/ou sistemas IoT, oferecendo uma forma simplificada e unificadora que permita a recolha de informação e posterior análise da mesma.

1.2 Objectivos

O principal objetivo da solução de *middleware*, resultante da presença de diversas interfaces das plataformas e/ou sistemas integrados, passa pela definição e desenvolvimento de uma interface que ofereça todos os serviços disponíveis. A interface deve oferecer a maior variedade de serviços possíveis, sem que a sua execução necessite de expôr a complexidade dos serviços originais. Desta forma, através desta interface deve ser possível gerir e actuar sobre os dispositivos integrados independemente da plataforma subjacente que os suporta.

Após o desenho e desenvolvimento desta interface, surge a necessidade de uniformização dos dados provenientes das plataformas e sistemas integrados, visto estes produzirem dados em formatos proprietários que dificultam a sua interpretação e processamento pela plataforma SATO. Esta uniformização é de elevada importância para a plataforma SATO, porque os dados têm um enorme impacto na capacidade de *assessement* e optimização energética que a plataforma SATO pretende oferecer. Para que esta uniformização seja atingida, a solução apresenta um formato de dados único para qual os dados devem ser mapeados, reduzindo assim a complexidade na integração dos dados das soluções inte-

gradas.

A estes dois objetivos é adicionado um terceiro que passa pela disponibilização dos metadados dos utilizadores, dos dispositivos e do espaço onde estão, de forma a que estes estejam uniformizados e podendo assim correlacionar os metadados produzidos pelas diferentes plataformas e sistemas integrados.

Considerados os principais objetivos para a solução a desenvolver neste trabalho, é descrita na Figura 1.1 uma representação de alto nível do objetivo global desta solução, onde a ligação a azul representa a percepção que os componentes externos devem ter quando utilizam a solução para interagir com os dispositivos integrados.

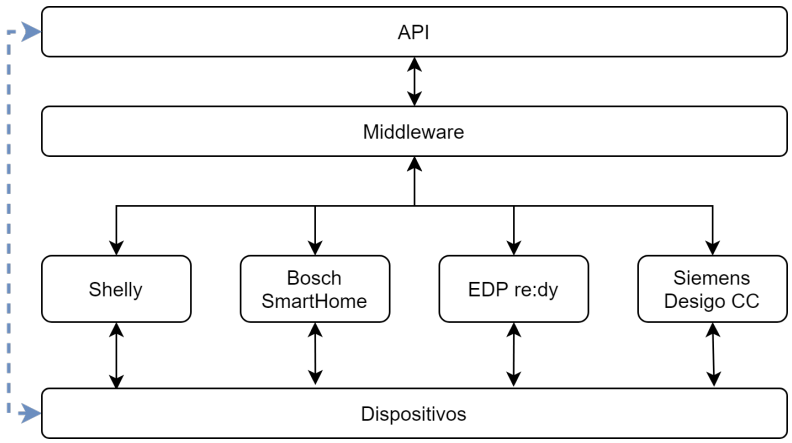

Figura 1.1: Desenho abstrato do objetivo da solução

1.3 Contribuições

A principal contribuição deste trabalho reside num *middleware* de integração que permite abstrair a complexidade existente na integração de plataformas e/ou sistemas IoT proprietários.

O foco da arquitetura proposta reside na integração de plataformas e sistemas inteligentes. No entanto, a arquitetura foi desenhada de forma a permitir a integração de fontes de dados externas, como por exemplo dados metereológicos ou preços da energia.

Para a concretização desta contribuição principal, foram também feitas diversas contribuições a um mais nível específico:

- **Desenho arquitetural**: Nesta tese desenhou-se uma solução arquitetural baseada em micro-serviços que oferece um alto grau de robustez e escalabilidade.

- **Interfaces**: Dada a diversidade de interfaces proprietárias, nesta tese definiu-se uma interface unificadora que agrega e oculta as especificidades de cada plataforma e/ou sistema integrado.

- **Modelo de dados comum**: Tendo por base a diversidade de plataformas e interfaces proprietárias, nesta tese foi definido e implementado um modelo de dados comum que é utilizado para unificar os modelos de dados proprietários, facilitando a integração de novas plataformas e/ou sistemas IoT.

- **Ontologia**: De forma a unificar os metadados dos dispositivos e edíficios, esta tese também contribuiu para a definição de uma ontologia de suporte à plataforma SATO. Esta ontologia irá servir para o enriquecimento dos dados, para estabelecer relações entre os dispositivos e os edíficos, bem como para enriquecer os *Assessments* que iram ser feitos pelos serviços da plataforma SATO.

- **Conectores**: Esta tese contribuiu também para a definição de um conjunto de conectores que permitem a integração de plataformas e/ou sistemas inteligentes IoT que não são nativamente compatíveis com a arquitetura SATO (plataforma orientada a dados em *stream*).

Todos estas contribuições fazem parte da plataforma SATO.

Com este conjunto de contribuições, a solução é responsável pela capacidade de interação de um conjunto abrangente de dispositivos na plataforma SATO, apresentando assim uma enorme contribuição para o projeto SATO.

Já no contexto ciêntifico, o trabalho apresenta uma contribuição significativa para a área de *middleware*, mais especificamente na qual se pretende concretizar a integração de plataformas e sistemas IoT.

1.4 Estrutura do documento

Este documento está organizado em 7 Capítulos, sendo que neste primeiro se descreveu a motivação, objetivos e contribuições desta tese. A restante tese está organizada da seguinte forma:

- **Capítulo 2** - Neste Capítulo é descrito o trabalho relacionado. No levantamento do trabalho relacionado foram revistos artigos ciêntificos onde são descritos trabalhos de *middleware* e *surveys* onde são feitas análises de diferentes soluções de *middleware* em IoT. Em complemento a esta revisão, foram identificados os standards da industria que servem ou podem ser adaptados ao domínio da energia.

- **Capítulo 3** - No Capítulo 3 é apresentada a arquitetura da plataforma SATO. Com base nessa arquitetura é definido um conjunto de casos de uso que a solução de *middleware* deve suportar. Por fim é feito um levantamento dos requisitos e analisados os dois principais problemas de interoperabilidade para os quais esta tese deve dar resposta.

- **Capítulo 4** - Através da análise feita no Capítulo 3, neste Capítulo detalha-se o desenho da solução de *middleware* proposta nesta tese. Na descrição da solução são apresentados os mecanismos que permitem endereçar os problemas de interoperabilidade resultantes da integração de diferentes plataformas ou sistemas IoT.

- **Capítulo 5** - Este Capítulo apresenta os detalhes de implementação do protótipo da solução desenvolvida, que utiliza como base a arquitetura desenhada no Capítulo 4.

- **Capítulo 6** - O Capítulo 6 especifica um conjunto de testes e apresenta os resultados obtidos da execução dos mesmos no protótipo de *middleware* desenvolvido. Através dos testes e dos resultados obtidos são retirados alguns indicadores de desempenho e apresentadas algumas das conclusões extraídas do protótipo.

- **Capítulo 7** - Por fim, neste Capítulo apresentam-se as principais conclusões retiradas deste trabalho e apresentam-se propostas de trabalho futuro.

Capítulo 2

Trabalho relacionado

O conceito de *middleware* no contexto de IoT é um tópico com muito desenvolvimento nos últimos anos devido à quantidade e heterogeneidade dos dispositivos que têm vindo a ser integrados neste tipo de sistemas. No entanto, grande parte das soluções atuais são, de alguma forma, limitadas a determinados contextos, não existindo por isso uma solução suficientemente genérica/universal.

De forma a desenhar-se uma solução de integração que seja o mais genérica possível, procedeu-se à identificação das principais características e diferenças das soluções existentes. Neste capítulo são analisados alguns *surveys* e soluções de *middleware* em IoT, tendo em consideração a avaliação de diferentes critérios. Desta forma são identificadas e descritas algumas das principais soluções existentes que apresentam interesse para o desenho da solução deste trabalho. Assim, para ficar a conhecer os requisitos e objetivos das soluções de *middleware* para IoT e desta forma avaliar as soluções existentes, foram analisados os trabalhos desenvolvidos em [26, 32, 39]. Estes trabalhos avaliam os principais requisitos e características desejáveis para este tipo de sistemas, comparando as diferentes plataformas avaliadas. Nesta análise é especificada a arquitetura seguida pelas plataformas assim como os protocolos suportados para comunicação. Por fim, os estudos são concluídos com uma definição de problemas ou limitações existentes nas atuais soluções. Destes trabalhos, o trabalho [39] apresenta a melhor categorização dos requisitos necessários para a construção deste tipo de sistemas, fazendo a separação entre requisitos funcionais, não funcionais e arquiteturais.

De forma a sistematizar a lista apresentada pelo autor, de seguida são descritos e discutidos os principais requisitos das plataformas IoT. O objetivo desta apresentação de requisitos passa por avaliar os requisitos mais importantes para o contexto da solução para que possam ser garantidos.

- **Funcionais**
  - **Deteção de Recursos**: Os dispositivos devem ser capazes de se anunciar ao sistema assim como anunciar as suas capacidades, para permitir uma integração sem intervenção humana.

  - **Gestão de Recursos**: O *middleware* deve ser capaz de gerir os recursos associados a si de forma a garantir a qualidade do serviço (QoS) desejada.
  - **Gestão de Dados**: Deve ser suportada a capacidade de gerir os dados produzidos pelos dispositivos integrados, seja a capacidade de os persistir, filtrar, processar ou ainda outras operações.
  - **Gestão de Eventos**: Capacidade de gerir simples eventos de forma a que passem a ter significado para o sistema que os processa.
  - **Gestão de Código**: Capacidade de migrar ou integrar novas funcionalidades nos dispositivos (nova versão de software), como por exemplo a atualização do mesmo.

- **Não Funcionais**

  - **Escalabilidade**: O *middleware* deve ser escalável para suportar o crescimento da rede e dos serviços/aplicações.
  - **Segurança**: A segurança em IoT deve ser considerada em todos os blocos funcionais e não funcionais incluindo a nível aplicacional.
  - **Privacidade**: O *middleware* deverá garantir mecanismos que garantam a privacidade dos utilizadores.
  - **Disponibilidade**: O *middleware* em sistemas de IoT, principalmente em sistemas críticos, deve estar sempre disponível.
  - **Confiabilidade**: Todos os componentes ou serviços do *middleware* devem ser fiáveis de forma a que juntos garantam a confiabilidade geral do *middleware*.
  - **Tempo Real**: O *middleware* deve fornecer serviços em tempo real onde a sua execução depende não só da exatidão de uma operação mas também da hora a que foi executada.
  - **Popularidade**: O *middleware* em IoT deve garantir que recebe atualizações e extensões regulares.
  - **Facilidade de implementação**: A integração ou atualização do *middleware* deve ser feita de forma simples.

- **Arquiteturais**

  - **Abstração**: O *middleware* deve fornecer interfaces que isolem a infraestrutura heterogénea subjacente.
  - **Interoperável**: O *middleware* deve ser capaz de interagir e integrar dispositivos, plataformas ou aplicações de todos os tipos sem grande dificuldade.

- **Contexto**: O *middleware* deve entender o contexto dos utilizadores, dispositivos e o meio que os rodeia para que possam fornecer serviços eficientes e úteis.
- **Autónomo**: O *middleware* deve permitir a interação entre dispositivos, tecnologias e aplicações sem interação humana.
- **Adaptável**: Devido às constantes mudanças num sistema IoT, o *middleware* deve ser capaz de se adaptar, assim como à mudança de contexto.
- **Orientado a Serviços**: O *middleware* deve seguir uma arquitetura orientada a serviços garantindo assim a flexibilidade do sistema e a abstração da infraestrutura subjacente.
- **Distribuído**: Uma implementação de *middleware* deve ser capaz de oferecer serviços/funções que estão distribuídas pela infraestrutura física de IoT.

Depois de avaliados os requisitos comuns às diferentes soluções de *middleware* para IoT, é importante considerar a quantidade de dados que estes dispositivos produzem, o que implica a estas soluções uma elevada capacidade de processamento, sendo comum a utilização da cloud para colmatar a falta de recursos para processamento. Com isto, é importante avaliar a comparação entre a utilização ou não utilização da cloud para a solução deste trabalho. Com este aspeto em mente foi avaliado o trabalho de Farahzadi et al. em [32], onde este apresenta a separação do *middleware* entre a utilização ou não da cloud, fazendo uma análise entre os dois tipos, e qual oferece melhor suporte a um conjunto de requisitos apresentados. Por fim os autores concluêm que o uso de computação na cloud é a opção mais viável para sistema de *middleware* em contextos de sistemas inteligentes que não necessitem de cumprir o requisito de tempo real, já que a comunicação com a cloud acrescenta um overhead que prejudicará o cumprimento deste requisito. Considerada a conclusão retirada pelos autores, é possível avaliar que a solução desenvolvida recorrerá à utilização da *cloud* já que na plataforma SATO não existe o requisito de tempo real.

Considerados os requisitos enumerados, nas secções seguintes são apresentadas as soluções de *middleware* que mais se assemelham à solução que se pretende desenvolver nesta tese, discutindo como é que os requisitos enumerados são, ou não, satisfeitos pelas soluções. Em complemento, foram identificados os principais mecanismos utilizados pelas soluções existentes e, sempre que estes se revelaram adequados aos objetivos da plataforma SATO, foram então considerados no desenho da solução final de *middleware* proposta nesta tese.

2.1 FIWARE

O FIWARE [16] é uma framework que oferece um conjunto de componentes que juntos têm a capacidade de compôr uma plataforma de IoT. O seu principal foco está nas *smart*

cities. São oferecidos vários componentes e estes são designados de Generic Enablers (GE). Para que a comunicação entre estes componentes seja possível é utilizado o protocolo NSGI, desta forma os componentes garantem a sua interoperabilidade. De seguida segue uma lista dos principais componentes e respetivas funcionalidades oferecidas pelos FIWARE:

- **Orion Context Broker** : Considerado como o principal componente, responsável por manter o estado atual de cada dispositivo integrado através de uma base de dados MongoDB. Grande parte dos componentes são integrados através da comunicação com este componente, sendo essa a principal razão para ser considerado o componente de maior importância. Oferece ainda uma API REST para que serviços externos possam consultar facilmente os seus serviços e/ou dados.

- **STH Comet**: Short Time Historic ou Comet é um dos componentes oferecidos que garante a persistência do histórico dos dados produzidos e a respectiva consulta temporal dos mesmos. Este componente foi desenvolvido para complementar o Orion, uma vez que este não tem a capacidade de persistir dados de longa duração. Para que esta ligação exista, este componente regista-se com o Orion como subscritor para ser notificado das modificações de estado dos dispositivos e, ao ser notificado, regista a informação recebida. O seu funcionamento tem duas configurações possíveis, o próprio trata de receber as notificações enviadas pelo Orion e inseri-las na base de dados Mongo ou deixa a cargo do Cygnus o registo na base de dados ficando o STH Comet responsável apenas pela disponibilização de uma interface que facilite a contulta dos dados. Sendo importante notar que atualmente apenas é suportada a persistência usando MongoDB.

- **Cygnus**: Faz parte do ecossistema Cosmos [8] e é baseado no Apache Flume. A sua função não passa pela persistência dos dados, mas sim por gerir onde são inseridos os dados de forma a poder criar a sua visualização histórica. Serve de conector entre o Orion e os serviços de armazenamento.

- **IDAS**: Para que seja possível a iteração de um sistema heterogéneo de dispositivos, este componente oferece agentes IoT que tratam do mapeamento dos protocolos mais usados dentro de IoT (por exemplo, LoRaWAN) com o protocolo NGSI. É ainda oferecida uma biblioteca que possibilita o desenvolvimento de um agente custumizado ao desejo do programador.

- **Keyrock**: Este componente permite adicionar a gestão de identidades de forma a garantir a autenticação e autorização das mesmas sendo baseado no protocolo OAuth2.

- **Wilma**: É um Policy Enforcement Point Proxy que garante a segurança ao servir de intermediário entre os serviços e os clientes sem que estes tenham conhecimento

que não estão a interagir diretamente com o serviço. Ao ser integrado juntamente com o Keyrock e AuthZForce, permite o controlo no acesso às componentes do sistema.

- **AuthZForce PDP/PAP**: Também conhecido por Access Control GE, este componente é responsável pela tomada de decisões de autorização, com base nas políticas de autorização e os pedidos de autorização feitos pelo PEP (Policy Enforcement Point). Para que outros componentes beneficiem desta camada de segurança oferecida por este componente, é disponibilizada uma API capaz de fornecer os seus serviços.

Sendo o FIWARE uma solução modular que se adapta a vários contextos, é possível retirar algumas ideias para o desenho da solução de *middleware* que se pretende neste trabalho. Desde o Orion, responsável por permitir a integração de todo o tipo de componentes com diferentes funcionalidades através da definição do protocolo NGSI, aos agentes oferecidos pelo IDAS, que permitem a integração de dispositivos com diferentes protocolos de comunicação. Outro aspeto importante que a plataforma deve oferecer é a capacidade de escalar, sendo que no caso do FIWARE esta é dividida devido à separação da plataforma por diversos componentes. Um bom exemplo disso é a documentação oferecida para permitir a escalabilidade do Orion Context Broker, que é executado sobre um container, o que facilita a replicação utilizando ferramentas de orquestração de containers. Como não basta que o servidor que expõe a API REST seja replicado, é também apresentada uma estratégia de replicação da base de dados Mongo, compondo assim uma solução que garante tanto a disponibilidade como a escalabilidade. O seguinte diagrama 2.1 apresenta um exemplo da arquitetura de replicação disponível para o Orion.

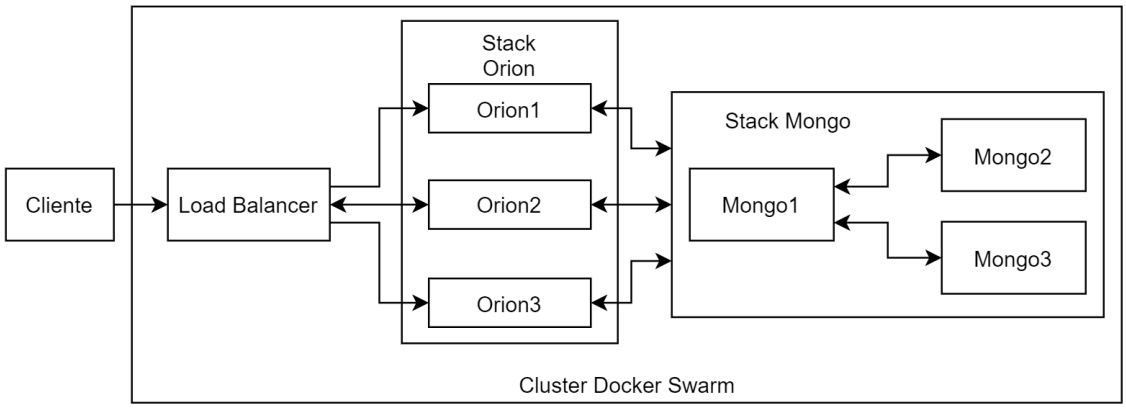

Figura 2.1: Exemplo de escalonamento do Orion Context Broker

2.2 EEBUS

EEBus [14] é uma família de protocolos para o domínio de Internet das Coisas (IoT), é composto pelo EEBus SPINE e EEBus SHIP tendo como objetivo a definição de uma linguagem comum com foco nos sistemas energéticos. O EEBUS consiste numa especificação, não existindo implementações *open-source*. Contudo, este está a ser utilizado por algumas plataformas comerciais, como por exemplo, a da Bosch [5], em que tanto o SHIP como o SPINE são suportados facilitando assim integrações de dispositivos que os suportem. A Figura 2.2 descreve a arquitetura de um sistema completo desenhado com recurso ao EEBus, onde se vê uma separação clara das suas camadas (EEBus SPINE e EEBus SHIP).

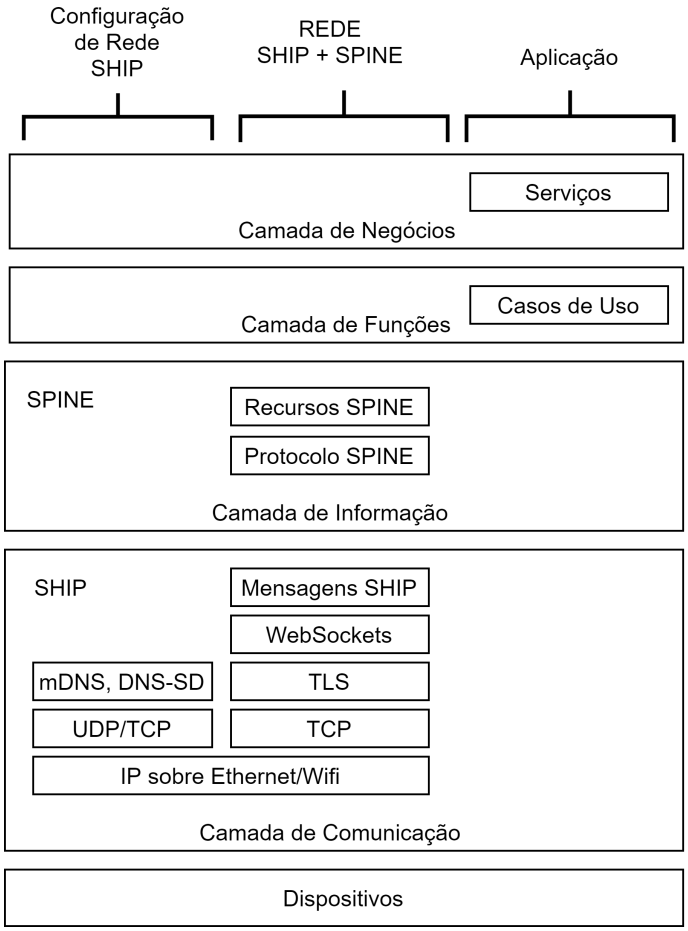

Figura 2.2: Arquitetura do EEBus

SHIP

O SHIP [31] (Smart Home IP) é um protocolo com base no protocolo IP e que se foca na interoperabilidade entre aparelhos domésticos inteligentes. É utilizado entre a camada de rede e a camada aplicacional seguindo o modelo OSI, suportanto tanto o protocolo TCP como UDP para o transporte das mensagens ao nível da camada de comunicação.

Foi desenvolvido para alcançar uma tecnologia de comunicação comum, tratando de todas as complexidades de estabelecimento de comunicações seguras e fiáveis entre dispositivos. Adicionalmente oference ainda diversos mecanismos para o registo de nós SHIP cobrindo assim diversos cenários reais.

SPINE

O SPINE (Smart Premises Interoperable Neutral-Message) estabelece um protocolo aplicacional, presente na camada 7 do modelo OSI, que especifica o formato das mensagens a ser adoptado pelos protocolos de comunicação, definindo estas mensagens como datagramas. A informação que estes contêm segue a Resource Specification [30], onde são definidos os diferentes modelos de dados, com foco no domínio de energia inteligente, a serem usados na troca de mensagens. Sendo que a definição da estrutura das mensagens em si, assim como a representação dos dispositivos, é definida no Protocol Specification do SPINE [31]. Assim, com estes dois componentes é possível obter uma estandardização no domínio de energia inteligente, sendo que para esta ser atingida é expectável que todos os dispositivos comuniquem através deste protocolo ou que pelo menos suportem o mapeamento entre o seu protocolo proprietário e o protocolo SPINE. Com este último cenário mencionado é possível a integração de dispositivos não SPINE em sistemas que adoptem SPINE.

Sendo mais objetivo, o protocolo SPINE oferece mecanismos de descoberta de serviços baseados numa representação própria orientada à representação dos dispositivos, estando estes mecanismos descritos na sua especificação. Especificação essa que detalha ainda outros mecanismos relativamente à troca de mensagens tais como *Binding*, *Subscription*, ou os diferentes modos de comunicação. O último mecanismo define que mensagens SPINE podem ser trocadas diretamente entre dois dispositivos ou usando um terceiro dispositivo que irá servir de reencaminhador. Outra das características do SPINE passa pela capacidade dos dispositivos descreverem os casos de uso que suportam, sendo esta uma forma de descrever a mais alto nível as suas capacidades. Uma nota importante relativamente ao SPINE e a sua utilização na troca de mensagens é que este não está restrito à utilização do SHIP para o transporte das mensagens, podendo ser transmitido por qualquer protocolo de comunicação TCP ou UDP.

Por fim, os datagramas SPINE são definidos usando XML Schemas (XSD), contudo os datagramas não estão restritos à utilização de XML podendo o JSON ou qualquer outro formato ser utilizado para representar os dados no sistema.

2.3 Ontologias

Uma das grandes vantagens do projeto SATO advém da possibilidade de se fazerem *Assessments* reais, considerando várias fontes de informação. Neste contexto, durante o desenho da solução arquitetural do projeto SATO foi identificada a necessidade de se

guardar um conjunto relevante de metadados que descrevem o contexto dos dispositivos e edifícios. Estes metadados descrevem o contexto do ambiente a optimizar e através dos quais é possível fazer uma avaliação mais rigorosa da eficiência, bem como fazer melhores optimizações energéticas. Baseado nesta necessidade, estudaram-se as ontologias do domínio energético existentes. Nesta secção são apresentadas as soluções consideradas mais adequadas para colmatar esta necessidade por parte da plataforma. Contudo, para que as ontologias avaliadas possam ser compreendidas é necessário detalhar o funcionamento e composição geral das ontologias. O principal conceito na definição de uma ontologia é a definição de classes, tendo elas uma função semelhante às classes no domínio de programação orientada a objectos, onde descrevem a composição de um objeto e permitem a criação de instâncias desse objeto. Para que seja possível a construção do contexto guardado pela ontologia é possível a conexão entre diferentes classes através de *object properties*, que como o seu nome indica permite que os objetos/instâncias das classes sejam ligados e criando assim o contexto dos dados. Juntamente a este tipo de propriedades e assim como na programação orientada a objetos, as classes podem ainda ter atributos, neste caso representados através das *data properties*, as quais são utilizadas para definir a ligação entre um objeto/instância de uma classe e os seus atributos, chamados de literais.

Na Figura 2.3 são apresentadas estas definições através de um exemplo de uma ontologia onde é definida a associação entre os dispositivos, os seus utilizadores e a sua localização (a que edifício pertence e as coordenadas desse edifício).

Neste exemplo é possível verificar que estes quatro conceitos estão associados a classes. Para relação entre classes são definidos *object properties* e para os seus atributos são definidas *data properties*.

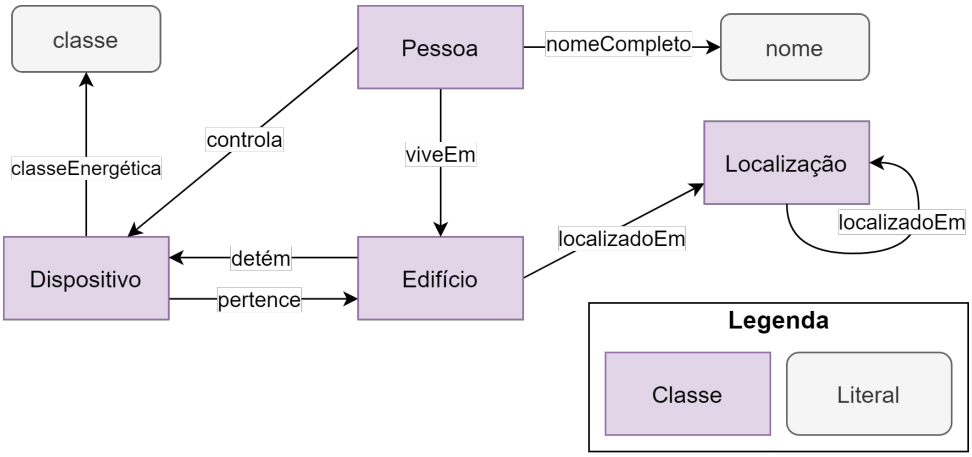

Figura 2.3: Exemplo de Ontologia

SAREF

Smart Appliances REFerecence (SAREF) [21] é uma ontologia que facilita a correlação entre os diferentes tipos de dispositivos no domínio de aplicações inteligentes com

as suas capacidades relacionadas à energia. Um dos principais objetivos do SAREF é cobrir o contexto energético através da representação das caracteristicas energéticas das aplicações inteligentes. Esta ontologia fornece uma representação única dos dispositivos e das suas características, mesmo que as suas tecnologias subjacentes sejam diferentes, sendo que uma das suas principais vantagens no contexto de optimização energética é o facto de ter sido desenvolvida com esse contexto em mente através da contribuição de *stakeholders* do setor energético [28]. A Figura 2.4 apresenta uma representação de alto nível dos conceitos cobertos pela ontologia SAREF, onde através desta definição é possível definir uma estrutura dos dados adequada para a representação do contexto dos dispositivos, descrevendo as suas propriedades, funções, medições, entre outras. Esta ligação entre as diferentes classes que se verifica na figura permite que as instâncias destas possam estar associadas a instâncias de outras classes, permitindo assim criar o contexto dos dispositivos.

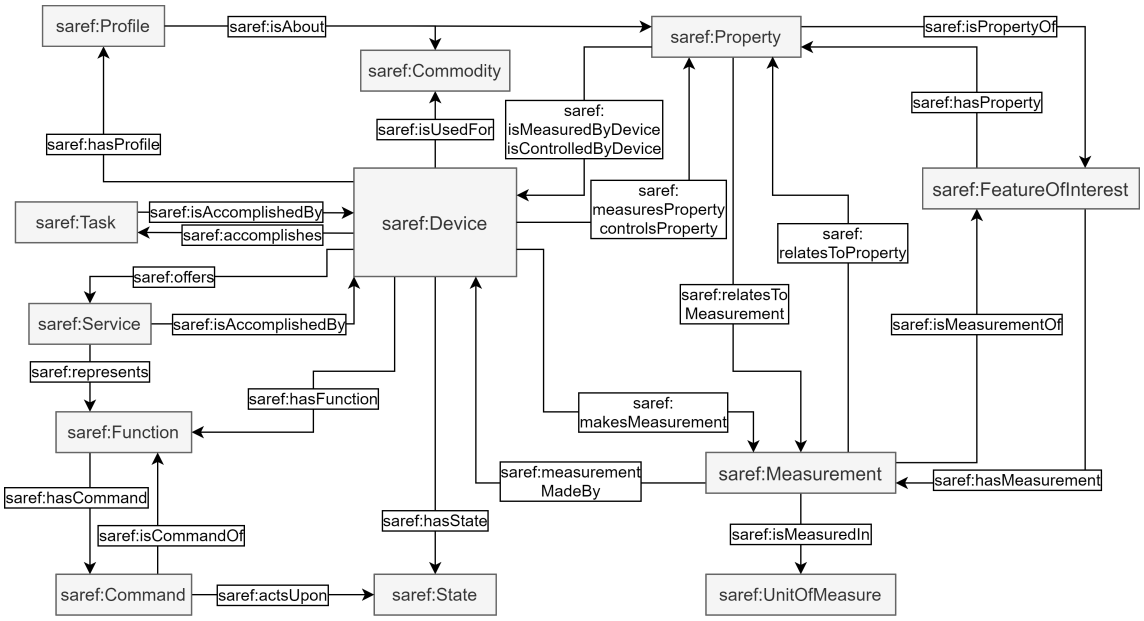

Figura 2.4: Representação da ontologia SAREF

SAREF4ENER

Devido à necessidade da definição de uma ontologia que representasse o domínio energético através da utilização de dispositivos IoT, uma vez que o SAREF não detalhava alguns dos conceitos necessários, foi proposto o SAREF4ENER. Esta ontologia é uma extensão do SAREF criada em colaboração com o EEBus e o Energy@home [15], dois *stakeholders* dentro do domínio da energia inteligente, tendo como objetivo estabelecer a ligação entre os seus diferentes modelos de dados [28]. Esta extensão garante todos os benefícios da ontologia SAREF ao mesmo tempo que permite o suporte de modelos de dados de iniciativas emergentes no mercado. Ao utilizar o SAREF4ENER, os disposi-

tivos que suportam os modelos de dados do EEBus e Energy@home podem facilmente comunicar entre eles utilizando qualquer sistema de gestão energética seja local ou na núvem.

Esta extensão permitiu não só adicionar mais propriedades associadas ao dispositivo, como também a definição do seu fabricante, do seu vendedor, do tipo de alimentação, entre outras. Sendo ainda adicionada a classe cuja função passa por representar as sequências energéticas e as respetivas propriedades, que servem para descrever os ciclos de funcionamento dos dispositivos. Complementarmente é também definida a classe Load Management, bem como as respetivas propriedades, que descrevem eventos energéticos associados a alguns tipos de dispositivos.

BOT

Considerando que 37% da energia final consumida na Europa provém dos edifícios [38], concluíu-se que seria benéfico, para o projeto SATO, a utilização de uma ontologia capaz de representar o domínio dos edifícios. Verificada esta necessidade, procurou-se na literatura por uma ontologia capaz de representar o domínio, tendo sido identificada a ontologia Building Topology Ontology (BOT) [6].

Esta ontologia engloba os conceito da topologia dos edifícios, conceitos esses importantes para permitir o *assessment* na localização dos dispositivos, facilitando assim a identificação dos dispositivos nas diversas zonas do edifício que os detém. A limitação desta ontologia passa pelo seu foco na topologia dos edifícios, não englobando conceitos como por exemplo as capacidades térmicas do edifício. No entando, esta limitação pode ser facilmente ultrapassada dada a facilidade de correlação e extensão que as ontologias permitem.

De forma a facilitar a compreensão dos conceitos incluídos nesta ontologia, sendo esta apresentada na Figura 2.5 e onde são descritos os principais conceitos definidos para representação de um edifício.

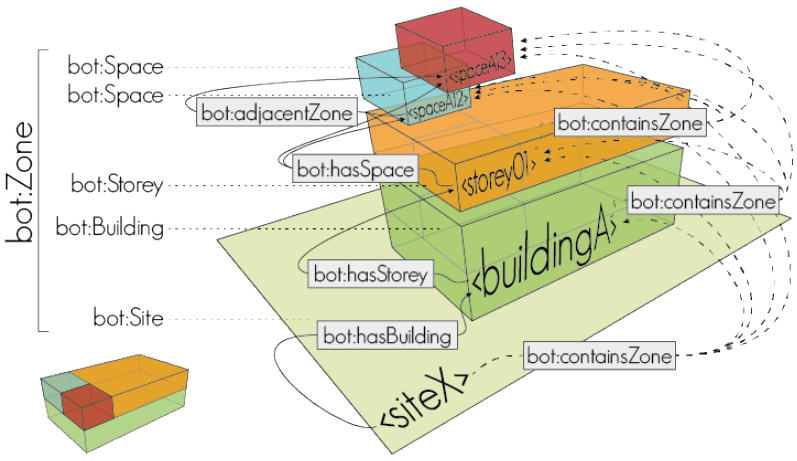

Figura 2.5: Representação da ontologia BOT [7]

2.4 RESPOND

Integrated demand REsponse Solution towards energy POsitive NeighbourhooDs (RESPOND) [20] é um projeto com ligação à comissão europeia do domínio da energia e da sua optimização, mais específicamente através de *Demand Response*, ou seja, através do ajuste do consumo energético durante os picos, redistribuindo esse consumo pelo restante período através de pequenos incentivos financeiros como a redução do custo energético. Apesar de apenas recorrer ao conceito de *Demand Response* para atingir a otimização energética, uma boa parte da sua arquitetura pode ser considerada para a solução à integração das plataformas no projeto SATO. Com isto, para o âmbito da integração das plataformas e respetivos dispositivos na plataforma SATO, é possível avaliar o estabelecimento de dois componentes que são essenciais para atingir os objetivos, sendo estes descritos de seguida.

Modelo Comum de Dados

Dado o ambiente heterogéneo resultante da integração de diferentes plataformas e dispositivos, a solução implementada no RESPOND irá receber ou enviar comandos e medições em diferentes formatos. Com este aspeto em mente, é estabelecido um formato único de dados a ser utilizado dentro da plataforma, de forma a abstrair todos os restantes componentes desta complexidade e deixando assim esta complexidade para os componentes responsáveis pela uniformização.

Este formato único de dados foi desenhado para permitir a uniformização de medições, de pedidos de atuações e dos estados dos dispositivos. Estes três formatos descritos nas Tabelas 2.1, 2.2, 2.3, pela respetiva ordem descrita.

Chave	Tipo	Valor
timestamp	integer	Tempo UNIX em milisegundos
value	float/boolean	Valor medido
measurementIndex	integer	Index da medição registada
deviceID	string	O identificador único do dispositivo que realizou a medição
measurementID	string	Identifica o tipo de medição a que o valor se refere

Tabela 2.1: RESPOND: Formato único de medições

Com os formatos de dados definidos, todos os componentes da plataforma necessitam apenas de reconhecer uma sintaxe, ficando abstraídos todos os formatos proprietários adjacentes às plataformas integradas.

Componente Semântico

Considerado o ambiente heterogéneo da solução desenvolvida, foi estabelecido um

Chave	Tipo	Valor
deviceID	string	O identificador único do dispositivo que realizou a medição
requestID	string	Identificador único utilizado para poder associar ao estado resultante da atuação
value	float/boolean	Valor medido

Tabela 2.2: RESPOND: Formato único para pedidos de atuação

Chave	Tipo	Valor
timestamp	integer	Tempo UNIX em milisegundos
value	float/boolean	Estado atual
status	integer	Representa o resultado da atuação, indicando se decorreu com sucesso ou se foi verificado algum erro
deviceID	string	O identificador único do dispositivo que realizou a medição
requestID	string	Identifica o pedido de atuação que provocou a modificação do estado representada pela mensagem

Tabela 2.3: RESPOND: Formato único de resultado de atuação

modelo semântico, cujo objetivo passa pela uniformização dos dados que compõem o contexto semântico. Para o desenvolvimento deste componente foram reutilizadas ontologias existentes, tendo sido definidas as relações entre elas, permitindo assim representar todo o domínio necessário para o projeto. Na Figura 2.6 representa-se a ontologia resultante do projeto.

Esta ontologia, uma vez baseada nas ontolgia SAREF e BOT, descritas na Secção 2.3, permite associar os dispositivos às suas localizações nos edifícios. Adicionalmente, a ontologia permite definir associações entre dispositivos inteligentes e dispositivos *legacy* que não oferecem qualquer funcionalidade de comunicação. Esta associação é feita através da conexão entre a classe seas:Property e a classe sear:FeatureOfInterest. Através desta abordagem, a solução desenvolvida permite a integração de um grande número de dispositivos não inteligentes. Por exemplo, a associação de uma tomada inteligente a qualquer tipo de dispositivo legacy, permite não só que este possa ser ligado e desligado remotamente, como também que seja feita a monitorização do seu consumo.

Ainda com base na ontologia resultante do projeto RESPOND, é possível estabelecer associações entre as propriedades controlados ou medidas pelos dispositivos e os IDs que as identificam numa base de dados.

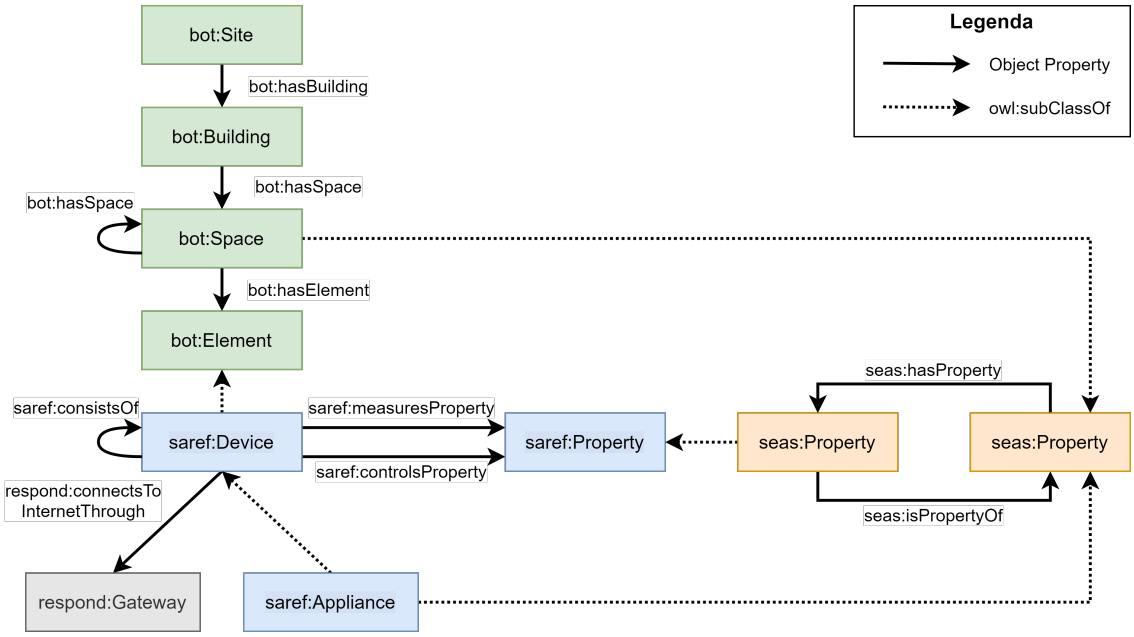

Figura 2.6: Respresentação da ontologia RESPOND

2.5 Outras Soluções

Para além das soluções já descritas, existem outras que não são do domínio da energia. Devido ao largo número de plataformas existentes, escolheu-se um conjunto limitado de plataformas que se acharam interessantes no contexto em que esta tese está inserida.

O DeviceHive [9] é uma plataforma *open-source* de IoT com uma arquitetura definida por micro-serviços, o que favorece a sua capacidade de disponibilidade e escalabilidade. Esta plataforma serve de *middleware*, suportando a integração de qualquer dispositivo que suporte comunicação REST, MQTT ou Websocket. A plataforma permite a sua costumização através do uso de plugins para permitir a integração de novos componentes, sendo por exemplo o suporte de MQTT feito através do uso de um plugin.

No DeviceHive é utilizado um Message Bus para tratar da comunicação dos serviços garantindo assim a distribuição de carga entre os mesmos. Este componente utiliza como base de desenvolvimento o Apache Kafka, mas para que a plataforma se tornasse independente de soluções externas foi desenhado o Proxy Websocket Kafka, sendo este implementado na forma de micro-serviço, garantindo assim os requisitos da plataforma.

Um dos pontos mais interessantes a analisar nesta plataforma, para o desenvolvimento deste trabalho, passa pelo uso de uma base de dados que apenas mantém em cache os mais recentes dados produzidos pelos dispositivos integrados, permitindo assim a consulta destes em tempo real.

São também fornecidos outros componentes que garantem a segurança dos dados e serviços através de mecanismos de autenticação dos dispositivos e dos clientes. Graças à sua arquitetura de microserviços, sendo cada serviço executado em containers, a escala-

bilidade e disponibilidade são requisitos facilmente cumpridos como qualquer solução de microserviços.

O Kaa [17] é uma plataforma de IoT que oferece a capacidade de integração de diversos dispositivos na sua plataforma, oferecendo também serviços como análise de dados, visualização, gestão de dispositivos, recolha de dados, entre outros. A sua arquitetura é orientada a serviços, permitindo assim uma fácil escalabilidade. O KAA é constituido pelos seguintes componentes:

- **Kaa Protocol Communication**: O KPC é responsável pela comunicação com os dispositivos integrados.

  Atualmente suporta comunicação com os dispositivos em claro ou usando TLS através de MQTT, WebSockets e HTTP. Permite ainda a comunicação através de MQTT sobre WebSockets com TLS ou então HTTPS, mas para estas opções serem viáveis é necessário fazer algumas configurações. O protocolo garante a interoperabilidade entre diferentes dispositivos através da utilização de outros serviços, apenas tratando do reencaminhamento das mensagens para os respetivos serviços de extensão de protocolos. Esta separação de serviços permite fornecer uma plataforma facilmente adaptavél e extensivel.

- **Credential Management**: Disponibiliza o serviço de Gestão de credenciais para clientes (CCM) e o serviço de gestão de credenciais para dispositivos (CM). Estes serviços estão diretamente ligados ao KPC, sendo estes os seus serviços de maior importância já que garantem a autenticação dos clientes e dispositivos.

- **Endpoint Lifecycle Service**: Trata da Monitorização do estado atual dos dispositivos, podendo ainda configurar de maneira a que atualize os metadados do dispositivo no EPR com o seu estado atual e o timestamp da mudança do estado.

- **Endpoint Register Service**: O EPR é responsável pela persistência dos dados dos dispositivos e dos seus respetivos metadados. Para a persistência utiliza uma MongoDB e disponibiliza a gestão dos dispositivos e respetivos metadados através de uma API REST.

Estes são os componentes que despertaram maior atenção e que permitem retirar algumas ideias para a solução desejada, sendo que os restantes componentes apesar de importantes acrescentam especificidades que não se aplicam no âmbito deste projeto. A solução de execução desta plataforma é feita através de *containers*, solução muito comum e benéfica para qualquer sistema, já que permite alcançar a escalabilidade, disponibilidade e a segurança.

2.6 Discussão

De forma a concluir o trabalho relacionado, foram ainda estudadas análises de algumas plataformas através da utilização de determinadas plataformas num determinado contexto. No trabalho descrito em [40], é feita uma análise do FIWARE através do desenvolvimento de uma pequena aplicação que permitisse a utilização dos principais componentes oferecidos pela plataforma. Segundo o autor, durante o trabalho desenvolvido, a sua experiência com os componentes foi algo variável, verificando que alguns dos componentes apresentam limitações nas suas funcionalidades, falta de documentação adequada ou mesmo bugs que necessitaram de ser corrigidos. Contudo, mesmo com os contratempos enfrentados o autor faz uma análise positiva da plataforma, já que excluindo alguns dos problemas, facilita o desenvolvimento de aplicações de IoT oferecendo componentes para diversas funcionalidades.

Um exemplo da utilização do FIWARE no contexto energético é a solução desenvolvida em [37], onde os autores utilizaram os diversos componentes fornecidos pelo FIWARE para permitir o controlo automático dos dispositivos num ambiente residencial. Sendo este trabalho destacado pelo controlo dos dispositivos através de um *gateway*, característica em comum com o projeto SATO, ao contrário de grande parte dos trabalhos que tratam da integração direta dos dispositivos. É ainda estabelecida uma arquitetura do sistema, apresentada na Figura 2.7, que é composta por alguns dos componentes oferecidos pelo FIWARE. A solução desenvolvida permite o controlo autónomo de eletrodomésticos e apesar de simples, permite analisar a arquitetura definida pelos autores dos componentes no mesmo contexto do projeto SATO.

Já T. Saref et al. através do trabalho [41] apresenta um sistema de controlo energético para edifícios utilizando o FIWARE, mais especificamente os agentes IoT e o Orion Context Broker, juntamente com a integração de um gateway que agrega todos os dispositivos. A arquitetura seguida do sistema é apresentada na Figura 2.8, onde é possível verificar que os dispositivos compõem uma rede sensorial BACnet que é gerida pelo *gateway* OpenMUC, responsável por manter toda a configuração dos dispositivos juntamente com os seus dados e metadados disponíveis, sendo também responsável pela comunicação com a cloud para que os dispositivos possam ser integrados com os componentes FIWARE.

Através da lista de dispositivos mantida no *gateway*, os autores fazem o seu registo dos dispositivos no Agente IoT garantindo assim o início da transmissão dos dados para o Orion através da API QuantumLeap que permite o registo dos dados na CrateDB. Já para garantir a visualização do sistema na web, é utilizada a framework Grafana, que recolhe os dados através da CrateDB. Após testes feitos, os autores concluem que a utilização do FIWARE juntamente com um *gateway* é uma solução eficiente, costumizável e de baixo custo para desenvolver sistemas de controlo energético para edifícios, oferecendo ainda a possíbilidade da utilização de *containers* o que garante uma configuração fácil e escalável do sistema.

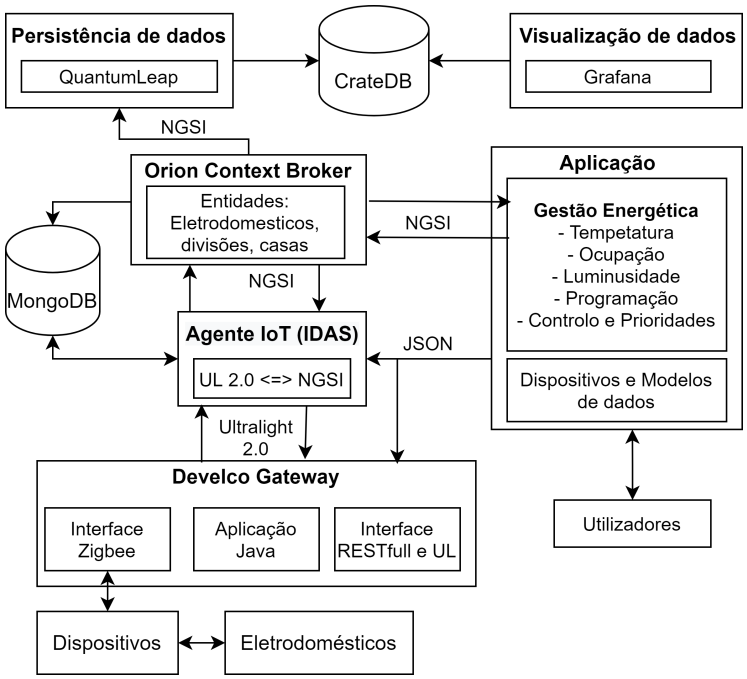

Figura 2.7: Arquitetura de sistema de controlo energético com componentes FIWARE em [37]

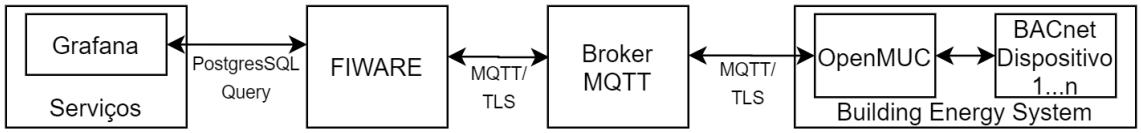

Figura 2.8: Arquitetura de sistema de controlo energético com FIWARE em [41]

Relativamente à realização de testes objetivos ao FIWARE, Araujo et al. em [25] realiza uma série de testes ao Orion Context Broker e aos Agentes IoT (IDAS) de forma analisar a performance da solução. Os autores realizaram testes de performance aos componentes na sua configuração base, seguido de uma estratégia de escalabilidade vertical, aumentando os recursos de cada componente, e ainda através de uma estratégia de escalabilidade horizontal, fazendo a replicação dos componentes tal como demonstrado na Figura 2.1. Através destes três tipos de configurações testadas os autores retiram as seguintes conclusões:

- O Agente IoT disponibilizado apresenta problemas ao lidar com alta carga de pedidos de atualização dos dados recebidos, tendo sido verificado um constante *crash* do agente na configuração base quando eram atingidos os 1000 pedidos de atualização.
- Deve ser desenvolvida uma implementação do Agente IoT que suporte um cenário de larga escala oferecendo também medidas de escalabilidade horizontal.
- O Agente IoT assim como o Orion Context Broker não retiram grande do aumento

dos seus recursos, dado que são desenvolvidos em Node.js que não retira proveito das capacidades multi-thread do hardware atual.

- Os componentes testados do FIWARE retiram grande proveito das medidas de escalabilidade horizontal suportadas, já que permitem um aumento considerável na capacidade de processamento do sistema.

- A escalabilidade do Orion deve ser pensada já que os autores verificaram que a replicação do Context Broker não aumentou o número de pedidos de atualização sem que fosse aumentada a capacidade da base de dados Mongo.

Sem apresentar nenhuma solução concreta, Mendes et al. em [35] apresenta uma arquitetura que tenta garantir a interoperabilidade dos dispositivos IoT no contexto de Smart Homes através de *middleware*. A arquitetura definida pelo autor é apresentada na Figura 2.9, na qual é possível verificar que segue uma arquitetura semelhante ao FIWARE, utilizando tradutores para os diferentes protocolos, uma interface que abstrai a infraestrutura subjacente fornecendo serviços de alto nível e a utilização de uma base de dados para guardar a configuração dos dispositivos e respetivas capacidades.

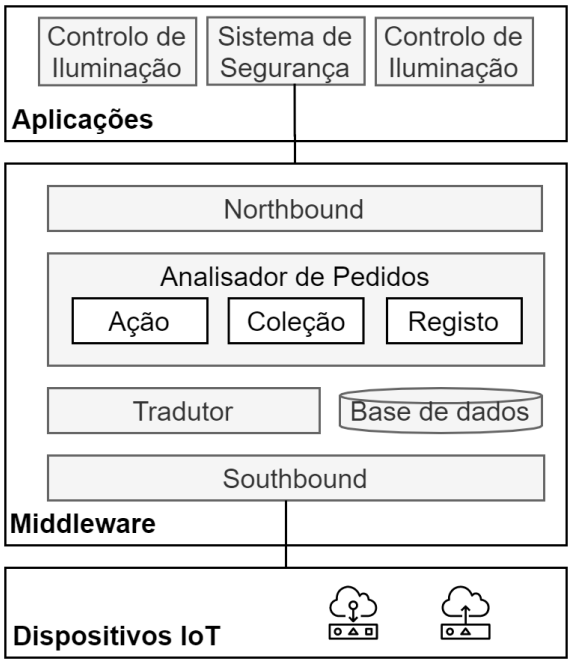

Figura 2.9: Arquitetura do *middleware* Smart Homes

Para concluir, considerando que grande parte dos *surveys* a soluções de *middleware* para IoT fazem análises qualitativas, o trabalho realizado por Da Cruz et al. em [27] destaca-se pela análise quantitativa das plataformas usando métricas como tamanho dos pacotes enviados, percentagem de erros e o tempo médio de resposta sobre uma carga específica. Nesta análise são estudadas 11 plataformas, onde apenas 5 foram analisadas

quantitativamente, onde o Sitewhere [23] se apresentou como a plataforma com melhor performance geral, seguida do FIWARE com os segundos melhores resultados. Estes testes foram realizados com a configuração base destas plataformas, sendo que um dos trabalhos futuros de interesse mencionados pelos autores é a utilização de *clusters* para testar as plataformas num cenário mais próximo de um cenário real.

Com esta revisão do estado da arte é possível compreender as atuais soluções desenhadas e desenvolvidas pela comunidade científica e soluções comerciais de forma a que seja adotada a melhor solução para o *middleware* do projeto SATO. Através da análise destas soluções, tendo cada uma delas o seu âmbito, serão retiradas conclusões de forma a compôr a solução para o projeto SATO.

Capítulo 3

Análise do problema e levantamento de Requisitos

3.1 Arquitetura SATO

O trabalho desenvolvido no âmbito desta tese assenta sobre a definição da arquitetura de integração que virá a integrar a arquitetura global desenhada para o projeto SATO. Esta definição tem como objetivo definir uma forma de integrar um conjunto heterogéneo de plataformas e sistemas inteligentes de IoT na solução final do projeto SATO. Com base nesta assunção, analisou-se a arquitetura desenhada para a plataforma SATO, de forma a identificar possíveis requisitos no desenho da solução de *middleware* a desenvolver para que se enquadre corretamente na arquitetura SATO. Esta arquitetura SATO é apresentada na Figura 3.1, onde foram destacados a verde os componentes que envolvem o *middleware* a desenvolver.

Ao avaliar a arquitetura proposta é possível verificar que a plataforma baseia-se numa solução de streaming, sendo esse um requisito importante para o desenvolvimento do *middleware* de integração, resultante da sua inclusão no projeto SATO.

Após avaliar a diversidade de soluções que é necessário integrar e ao contexto do projeto, foram identificados dois domínios semânticos que devemos considerar no desenvolvimento do *middleware* de integração, o domínio dos edifícios, devido à sua importância no cenário da optimização energética, e o domínio dos dispositivos. Através da Figura 3.1, é possível constatar que estes domínios são representados por dois componentes, o Device Semantics e o Building Semantics, onde os dois estão interligados de forma a estabelecer a ligação entre os dispositivos, o edifício em que estão localizados e as respetivas características.

Nesta mesma Figura 3.1, verifica-se que os dois componentes são utilizados pelo Device Services, componente esse que contém a interface de controlo para outros componentes SATO, como é o caso do Self-Optimization Services, permitindo assim o controlo e gestão dos dispositivos e sistemas integrados. Por fim, é necessário que a solução disponha de um componente responsável por receber todos os dados que são introduzidos na

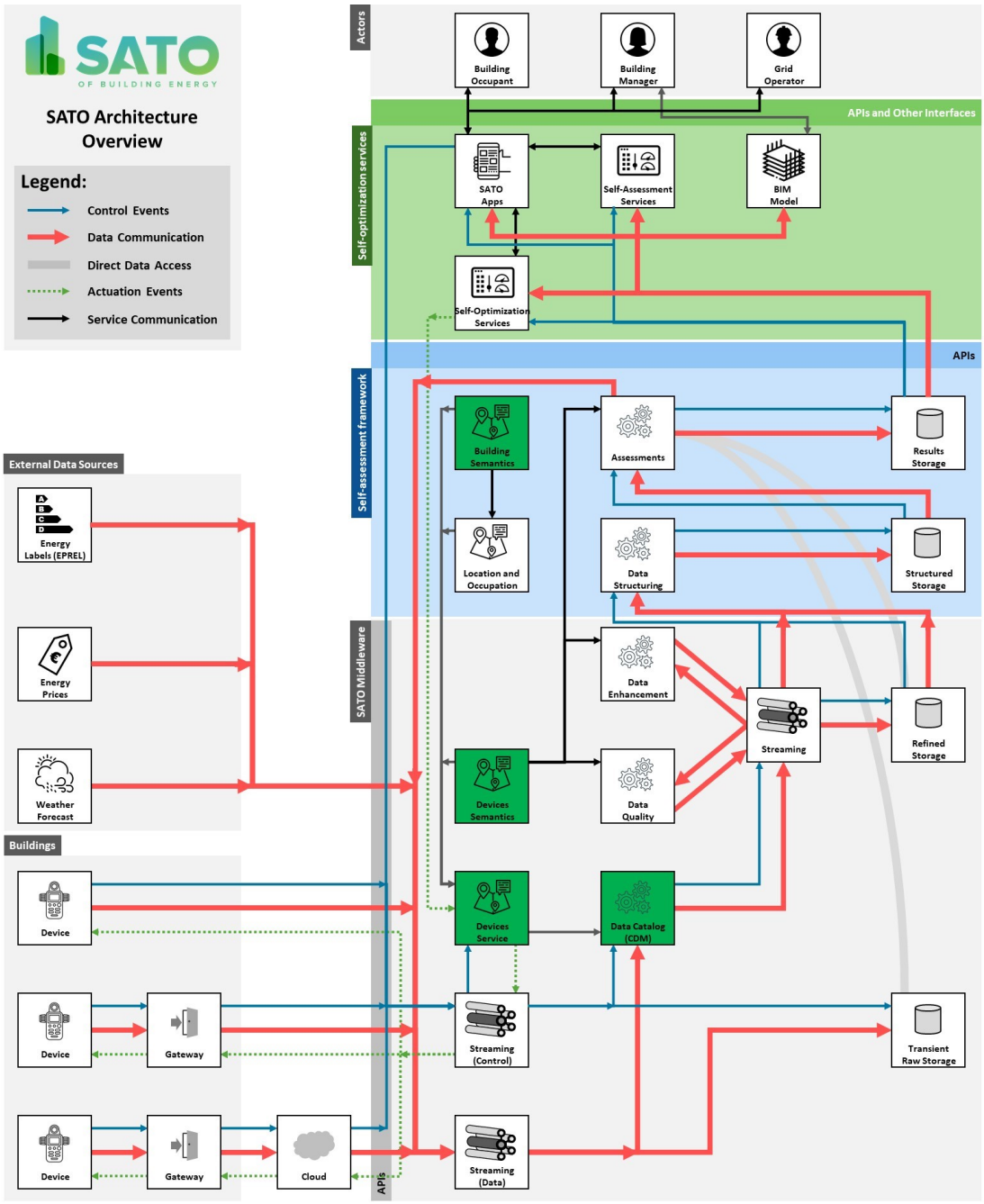

Figura 3.1: Arquitetura proposta para o projeto SATO

plataforma SATO pelos sistemas integrados e capaz de os uniformizar para que possam ser integrados na restante plataforma através do componente Streaming (Data Catalog).

Com esta análise é possível retirar alguns requisitos para a solução a desenhar e implementar, assim como uma visão de alto nível do fluxo dos dados entre os componentes do *middleware* para que sigam o fluxo necessário para a plataforma SATO.

3.2 Casos de Uso

Na análise da solução a desenvolver e considerando o contexto do projeto SATO, é possível retirar casos de uso que esta solução deve suportar, solucionando as complexidades resultantes dos mesmos. Na seguinte lista são apresentados alguns dos principais casos de uso que utilizam como referência a arquitetura SATO descrita na Figura 3.1.

1. **O utilizador registra-se na plataforma SATO como utilizador de uma das plataformas subjacentes:**

   Um utilizador da plataforma SATO regista-se como utilizador da plataforma integrada, através da chamada ao serviço de registo. Para tal, necessita de fornecer os dados necessários para o acesso à plataforma a integrar. O Device Services (componente que oferece o serviço de registo), realiza o pedido de acesso com a plataforma subjacente através do componente Streaming Control. De seguida é inserida, na plataforma SATO, a informação dos dispositivos do utilizador através do componente Streaming Control. Uma vez recebida a informação dos dispositivos, o Device Services regista-os, permitindo ao Data Catalog uniformizar a informação de forma a ser consumida pelos restantes componentes da plataforma SATO. Este processo de registo é representado na Figura 3.2.

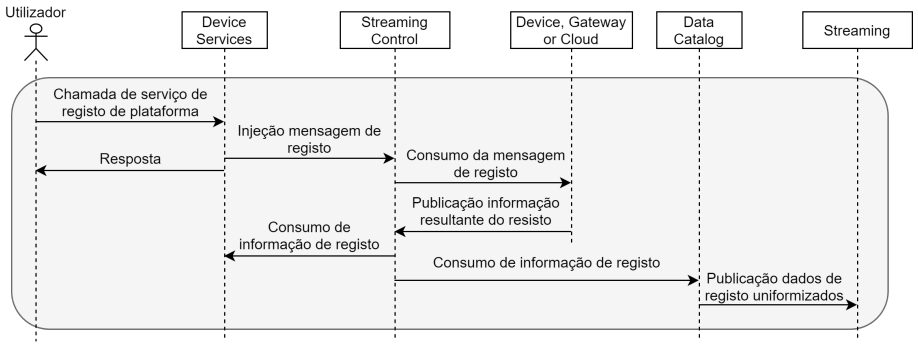

Figura 3.2: Caso de Uso 1: Registo de utilizador numa das plataformas integradas

2. **O utilizador regista um novo dispositivo diretamente na plataforma que o suporta e à qual está registado na plataforma SATO**:

   O utilizador tem um novo dispositivo suportado por uma plataforma à qual está registado na plataforma SATO, onde o dispositivo dispõe de mecanismos próprios de emparelhamento com o *gateway* ou com a núvem. Para que esse dispositivo possa ser registado na plataforma SATO, a sua plataforma de controlo deve inserir os dados de registo do dispositivo no componente Streaming Control, permitindo assim o seu registo pelo Device Services na plataforma, o que permitirá a uniformização da informação do dispositivo no Data Catalog. Este fluxo de dados durante este caso de uso é descrito pela Figura 3.3.

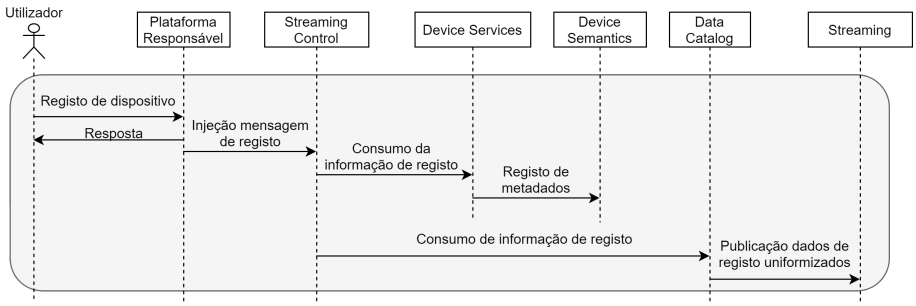

Figura 3.3: Caso de Uso 2: Registo de dispositivo numa plataforma integrada

3. **É recebido um novo evento de consumo energético numa tomada inteligente integrada na plataforma SATO**:

   É registada uma alteração no valor de potência medido numa tomada inteligente integrada na plataforma SATO. Este evento é então inserido na plataforma SATO no seu formato de dados proprietário, o que dificulta o seu processamento. Contudo, durante o registo deste dispositivo foi definido o modelo do dispositivo, o qual permite ao Data Catalog uniformizar os dados do dispositivo através da consulta destes dados, mantidos pelo Device Semantics. Seguido do mapeamento do evento, é enviado para um canal (Streaming) que se encarregará de canalizar os dados pela restante plataforma SATO. Todo o processamento e comunicação realizados neste caso de uso é representada na Figura 3.4.

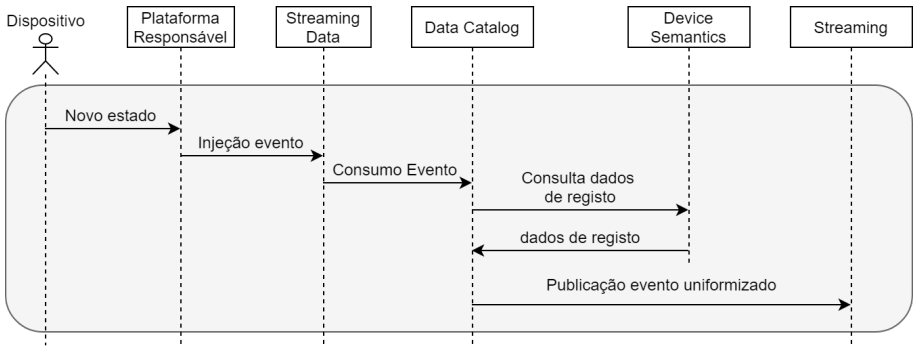

Figura 3.4: Caso de Uso 3: Processamento de evento de estado de um dispositivo

4. **É chamado um serviço através da API para realizar a atuação de um dispositivo**:

   Através da interface da solução é executado o serviço responsável para atuar num dispositivo com o ID indicado, independemente de qual a plataforma subjacente que suporta o dispositivo. Para que esta abstração seja feita a nível da interface é necessário que a mesma consulte os dados resultantes do registo do dispositivo, descobrindo qual a plataforma subjacente que integra o dispositivo, obtendo o ID proprietário do mesmo e ainda algum outro tipo de dados necessários para a atuação

sobre o dispositivo. De seguida, estes dados formam uma mensagem de atuação que será publicada no componente Streaming Control e que por sua vez será consumida pelo Device, Gateway ou Cloud alvo da atuação. Este caso de uso é representado pela Figura 3.5.

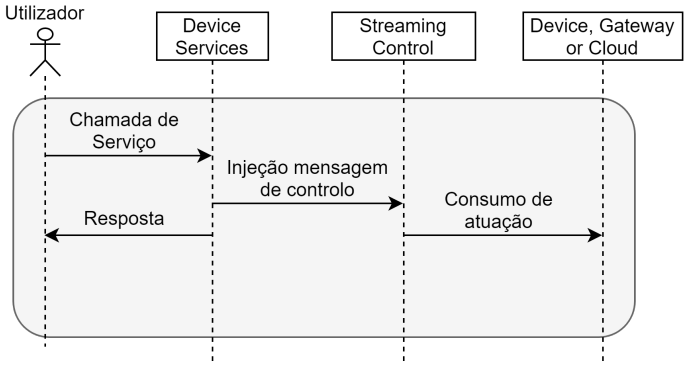

Figura 3.5: Caso de Uso 4: Chamada de serviço de atuação através da API genérica

3.3 Requisitos

Após a avaliação da arquitetura SATO, apresentada no Secção 3.1, é possível identificar os requisitos necessários à solução de *middleware*, sendo que esses requisitos são divididos entre requisitos funcionais, que descrevem as funcionalidades necessárias a cumprir por parte do sistema, e os requisitos não funcionais, que detalham os critérios de avaliação necessários a cumprir pelo sistema.

3.3.1 Funcionais

- **Interfaces de Serviços**: A solução deve expôr interfaces (APIs) de forma a facilitar a utilização dos serviços ao mesmo tempo que abstrai as plataformas que os fornecem.
- **Gestão de Dados**: O *middleware* deve ser capaz de gerir os dados enviados pelas plataformas/sistemas integrados e distribuí-los para os componentes que deles dependem.
- **Integração Plug & Play**: A solução desenvolvida deve facilitar a integração de novas plataformas ou sistemas inteligentes sem grandes alterações na plataforma SATO.
- **Fácil Integração de Serviços**: Deve ser possível a integração de novos serviços oferecidos pelas interfaces seja para integrar um novo serviço de uma plataforma integrada ou para a integração de uma nova plataforma.

3.3.2 Não Funcionais

- **Escalabilidade**: A solução deve ser capaz de aumentar a sua capacidade de processamento seja de forma linear ou exponêncial para suportar a carga sobre a mesma.
- **Segurança**: Dado o cariz dos dados enviados pelos sensores, a solução deve garantir a segurança dos mesmos através do estabelecimento de comunicações seguras entre os componentes que a ela se conectam.
- **Interoperabilidade**: Sendo um dos requisitos principais da solução, esta deve ser capaz de facilitar a integração de novos componentes, plataformas ou mesmo dispositivos, suportando um ambiente heterogéneo.
- **Disponibilidade**: A solução deve estar disponível para receber os dados e disponibilizar serviços para os componentes que dela dependem.
- **Tolerância a falhas**: A solução deve ter mecanismos que tornem possível a recuperação de dados não enviados/recebidos durante um período de falha do sistema.

3.4 Interoperabilidade Sintática

O projeto SATO pretende integrar um conjunto heterogéneo de dispostivos através das respetivas plataformas externas em que cada uma delas segue o seu modelo de dados proprietário. Tendo esta complexidade em consideração, a plataforma SATO e a solução a desenvolver neste trabalho necessitam de acomodar toda a diferente informação que recebe e para o fazer tem de recorrer a alguma solução para o ambiente heterogéneo dos dados. Uma solução comum, já vista anteriormente na Secção 2.5 relativamente ao projeto RESPOND, é a definição de um modelo de dados comum à plataforma SATO à qual todos os dados de entrada serão mapeados. Esta é uma solução vastamente adoptada dadas as suas vantagens, já que permite assim abstrair esta complexidade dos diferentes modelos de dados para os restantes componentes da plataforma ou sistema, permitindo ainda assim uma mais fácil integração de novas plataformas na plataforma SATO, dado que apenas necessitará de ser feito esse mapeamento para que os seus dados possam integrar a plataforma SATO.

A Figura 3.6 ilustra dois cenários, um em que não é estabelecido um modelo de dados comum e outro em que é definido um modelo de dados comum. Através da imagem é possível concluir os benefícios da solução e consecutiva utilização.

Tendo em conta a necessidade de estabelecer um modelo de dados único foram consideradas duas hipóteses, formular um novo modelo de dados de raiz, algo que necessita de ser bem pensado já que deve cobrir grande parte do mundo IoT no contexto energético e todos os que o englobam, ou optar por adotar um modelo de dados existente dentro do contexto energético podendo utilizá-lo e extendendo-o caso exista essa necessidade.

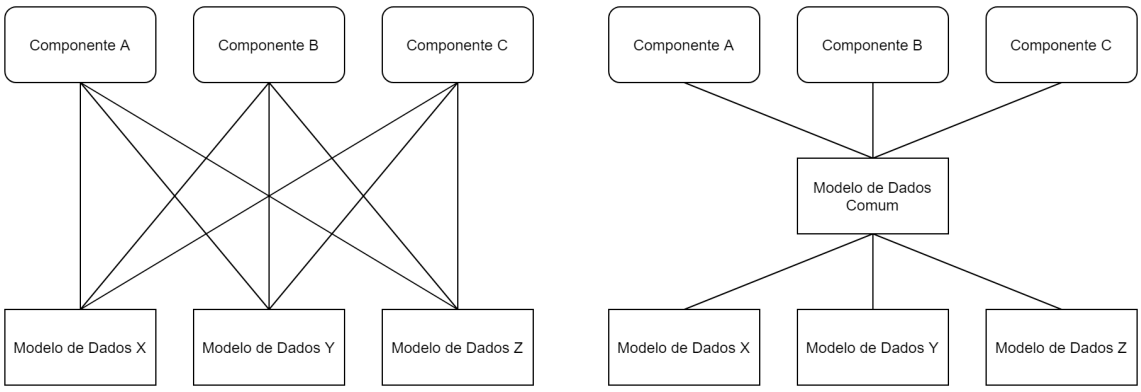

Figura 3.6: Representação do estabelecimento de modelo de dados comum

Dadas estas duas opções, optou-se pela reutilização de um modelo de dados existente, mais em concreto no EEBus SPINE, dado que este define um modelo de dados para o contexto energético, abrangendo grande parte deste e sendo amplamente utilizado a nível Europeu [42], o que prova também a sua utilidade e competência.

3.5 Interoperabilidade Semântica

Considerando o ambiente heterogéneo resultante da integração das diversas plataformas e respetivos dispositivos na plataforma SATO, existe alguma dificuldade em consultar os metadados, devido há existencia de diferentes formatos e contextos. A este problema, junta-se ainda a dificuldade resultante da integração dos diferentes dispositivos com a componente de optimização autónoma presente na plataforma SATO, uma vez que este componente deve conhecer os dispositivos integrados e as suas respetivas capacidades, propriedades e características, de forma a obter a melhor optimização possível dado o ambiente ou contexto em questão.

Sendo estes problemas frequentes no ambiente científico, é comum encontrarem-se soluções previamente desenvolvidas e formuladas. Do conjunto de soluções existentes, a mais comum passa pelo desenvolvimento de um componente que forneça e suporte os metadados dos dispositivos com um contexto semântico. Esta solução é possível através da definição e consequente utilização de uma ontologia. No caso do projeto SATO esta ontologia deve cobrir os conceitos do contexto energético, o contexto dos dispositivos IoT, assim como o contexto dos edifícios já que estes podem variar no seu tipo, topologia entre outros, tendo um impacto significativo no tipo de optimizações a fazer.

O problema da interoperabilidade semântica não é novo em sistemas que pretendam a integração de um conjunto heterogéneo de dados. Um exemplo disso é o projeto RESPOND [20]. Este projeto apresenta-se no mesmo domínio da plataforma SATO e apresenta também um problema com a integração de um conjunto heterogéneo de dados para a sua plataforma. No projeto RESPOND é definida uma ontologia que contém os con-

ceitos necessários para descrever o contexto dos dispositivos integrados e assim permitir uma representação única deste contexto, garantido assim a interoperabilidade semântica de acordo com os requisitos da sua plataforma.

Capítulo 4

Middleware de Integração SATO

4.1 Arquitetura

De forma a desenvolver uma solução de integração para um conjunto heterogéneo de plataformas e sistemas inteligentes IoT, é necessário definir uma arquitetura que permita resolver os problemas resultantes desta integração.

A Figura 4.1 apresenta a arquitetura do *middleware* de integração desenhada para o projeto SATO. Esta arquitetura foi desenhada para suportar a integração de diferentes tipos de plataformas IoT, principalmente as utilizadas no domínio da energia, para suportar grandes ritmos de chegada/manipulação de dados e para servir diferentes tipos de serviços (monitorização e atuação). De forma a garantir este suporte, a arquitetura foi desenhada com base numa arquitetura de micro-serviços, que permite escalar facilmente qualquer componente e satisfazer as necessidades de ritmos de dados.

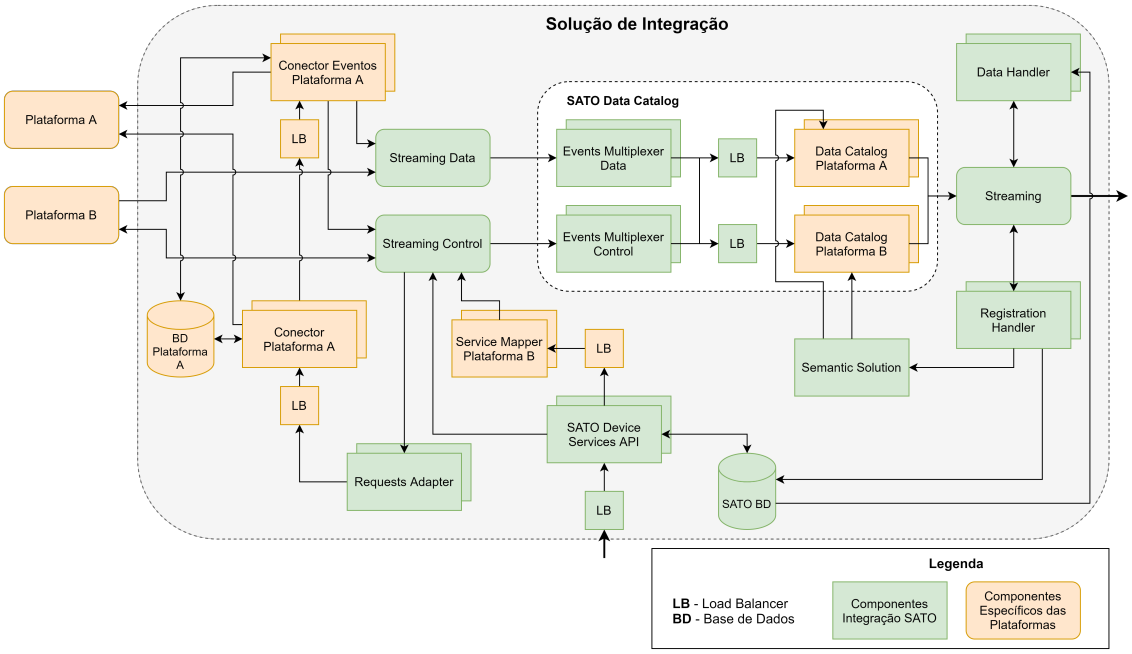

Figura 4.1: Arquitetura da solução de integração SATO

Quando pensamos na integração de plataformas surge logo o problema da heterogeneidade das plataformas e dos dados a integrar. Uma das possíveis soluções em termos de dados passa pela uniformização sintática dos dados.

Para concretizar esta uniformização sintática é necessário desenhar o componente responsável pelo mapeamento dos modelos de dados específicos para o modelo de dados genérico definido para a arquitetura apresentada.

Na Figura 4.1, este componente tem o nome SATO Data Catalog e tem a função de uniformizar os dados que provêm das diferentes plataformas integradas. Este componente é constituido por 2 subcomponentes, Events Multiplexers, que são consumidores das soluções de Streaming e que analisam os dados identificando a que plataforma pertencem e que de seguida reencaminham, através de um pedido HTTP, os dados para o Load Balancer (LB) que tratará de distribuir esse pedido para a réplica do Data Catalog da plataforma em questão. O segundo sub-componente representa a modelação (modelo de dados) específica à plataforma a integrar e é este o sub-componente que permite ao SATO Data Catalog uniformizar os dados heterogéneos das plataformas integradas. Desta forma, é possível separar a lógica deste mapeamento dos multiplexers, sendo também uma grande vantagem para a futura integração de novas plataformas, necessitando apenas de receber um componente (de *software*) que tenha uma interface igual aos restantes da plataforma e que seja também capaz de publicar os dados mapeados no formato único de dados para o componente de *streaming*.

Para além da uniformização dos dados que fluem na plataforma, é também necessário estabelecer um formato único para os metadados dos dispositivos e a descrição das capacidades de cada um, para que estes possam ser consultados quando necessário sem conhecer a sua representação proprietária. Ao analisar soluções para este problema, feito por outros trabalhos existentes, foi estabelecida a utilização de um componente semântico (Semantic Solution), responsável por manter os metadados e modelo do dispositivo criados no registo de cada dispositivo. Desta forma, é disponibilizado um componente que uniformiza estes dados provenientes também das plataformas integradas.

Um último requisito resultante desta integração está na interação com os dispositivos das diferentes plataformas, já que utilizam diferentes interfaces para o seu controlo e gestão. O problema acresce dado que a plataforma SATO terá um componente de otimização, que tratará de mudar os estados dos dispositivos, de forma a atingir uma maior eficiência energética. Ou seja, não seria prático que o componente tivesse de conhecer todas as interfaces subjacentes, já que desta decisão resultaria a necessidade de modificação do componente para cada nova integração. De forma a solucionar o problema, foi desenvolvido um componente, o SATO Device Services API, que uniformiza todas as interfaces subjacentes disponibilizando-as de forma unificada numa só interface

A tarefa deste componente é semelhante à de um *proxy*, onde este identifica qual a plataforma subjacente que irá executar o serviço, reencaminhando-o para o respetivo

componente, onde será executada a lógica proprietária da plataforma e garantindo assim o requisito de integração plug & play da solução.

Após uniformizados os dados que são injetados na plataforma, foi encontrada a necessidade de abstrair os IDs proprietários dos dispositivos para que estes não fossem expostos para a restante plataforma SATO, utilizando IDs SATO gerados no registo de cada novo dispositivo. Sendo que esta abstração dos IDs proprietários feita depois dos dados estarem uniformizados. Isto implica que esta abstração acontecesse à saída dos Data Catalogs das Plataformas, o que implicaria a necessidade de permitir o acesso à Base de Dados SATO, algo que deve ser evitado já que a base de dados deve ser apenas gerida por componentes SATO. Em oposição a esta solução, foi desenvolvido o Registration Handler, que é assim responsável não só pelo registo dos metadados e respetivos modelos dos dispositivos no componente responsável pela uniformização semântica, mas também pelo registo do mapeamento, entre os IDs proprietários dos dispositivos e os IDs SATO dos dispositivos, na base de dados utilizada pelo SATO Device Services. Após este registo, é necessário que os eventos das plataformas integradas e consequentemente mapeados para o MCD, utilizem este novo ID SATO (SATODeviceID) em vez de o ID proprietário. Para que isso seja possível, e de forma a que não sejam os Data Catalogs a tratar desta substituição, foi definido um novo serviço, designado de Data Handler que é o responsável por substituir o ID proprietário pelo SATODeviceID.

Desta forma, o Registration Handler e o Data Handler, adicionam mais uma camada de abstração às plataformas integradas, ocultando os IDs proprietários para a restante plataforma, o que para além da simplificação da gestão de IDs também permite adicionar um nível de segurança extra (ocultando os IDs proprietários e evitando o seu contacto direto por parte de terceiros). Sendo que, uma importante decisão foi a da separação entre o componente responsável pelo registo e o outro responsável pelo mapeamento dos IDs. Deste modo, é possível que estes sejam facilmente replicados em diferentes números, o que é importante dado que é esperado que a plataforma receba um maior número de dados de eventos do que dados de registo. Ambos os componentes são consumidores do componente Streaming, logo não necessitam de um *Load Balancer* para distribuir os dados entre as réplicas.

Ao avaliar todo o tipo de plataformas e sistemas inteligentes IoT existentes e possíveis de serem integrados na plataforma SATO, é possível concluir que nem todas elas estarão dispostas a interagir diretamente com as soluções de streaming, dado que utilizam as suas próprias interfaces. Tendo isso em consideração, foi necessário resolver esse problema na integração deste tipo de plataformas. Para isso optou-se pela utilização de conectores que podem ser facilmente adicionados à solução de integração, facilitando assim a integração deste tipo de plataformas na solução SATO. Com isto, para cada plataforma a integrar que não use *streaming*, é necessário adicionar dois conectores específicos, o Conector da Plataforma e o Conector de Eventos da Plataforma.

O conector da plataforma (Conector Plataforma A) disponibiliza uma interface REST que fornece os serviços da plataforma integrada, adota a especificação definida para todos estes conectores e realiza a lógica necessária para executar os serviços através da interface proprietária da plataforma. Desta forma este componente permite a integração de plataformas que apenas suportam uma comunicação através das suas interfaces proprietárias, não sendo capazes de comunicar diretamente com as soluções de Streaming.

Já o conector para eventos da plataforma (Conector Eventos Plataforma A) é responsável por obter os novos eventos dos utilizadores registados a esta plataforma integrada, injetando-os na solução Streaming Data/Control. Esta abordagem permite a integração dos seus dados de forma similar às plataformas que permitem a injeção direta na solução de Streaming como é o caso da Plataforma B representada na Figura 4.1.

Por fim para garantir o requisito de integração *plug & play*, foram desenhados os Service Mappers cujo a funcionalidade se aplica apenas a plataformas que interagem diretamente com as soluções Streaming Data e Streaming Control da plataforma SATO, sendo a plataforma B na Figura 4.1, um exemplo deste tipo de plataformas integradas. Devido a esta interação direta, estes devem conhecer a sintaxe em que é feito o pedido de serviços, retirando esta responsabilidade do SATO Device Services API, mapeando os serviços para o formato que as suas plataformas suportam no consumo dos dados de controlo. Assim, através destes componentes a integração de uma nova plataforma necessita apenas que este componente seja desenvolvido e deste modo a alteração no SATO Device Services API consiste unicamente no registo de uma nova plataforma.

4.2 Modelo Comum de Dados

Como descrito na Secção 3.4 e 4.1, um dos problemas resultantes da integração de um diverso conjunto de plataformas e sistemas IoT está nos diferentes formatos dos dados utilizados por cada uma, o que dificulta a sua utilização para os restantes componentes da plataforma SATO. Para colmatar este problema é necessário adotar um Modelo de Comum de Dados (MCD), de forma a que todos os dados provenientes das plataformas e respetivos dispositivos o adotem, tornando mais fácil a compreensão dos dados por todos os componentes. Para que este modelo possa ser interpretado por todos os componentes é necessário que este seja definido com uma estrutura comum, que seja capaz de lidar com os dados que este uniformiza. Desta forma, após análise das alternativas possíveis, foi escolhida a reutilização do EEBus SPINE, ao qual foram feitas algumas modificações e extensões de forma a que se torne o mais adequado possível à utilização dada pela plataforma SATO.

Ao iniciar esta reutilização é necessário pensar na utilização deste modelo, começando pela representação dos dispositivos na plataforma, ou seja, como são as funcionalidades ou serviços destes dispositivos representados e identificados para que assim possam

também ser uniformizados. Considerando este aspeto, o EEBus SPINE apresenta na sua especificação um modelo de representação dos dispositivos e respetivos serviços. Desta forma, a partir desta uniformização resulta a consequentemente remoção de complexidade de suporte das diferentes representações provenientes das plataformas integradas. Assim, é possível que este modelo único para a representação dos dispositivos cubra os casos de uso das diferentes plataformas e resultando assim numa maior cobertura de serviços e por consequência, dispositivos. Este modelo é composto por três camadas, dispositivo, entidade e funcionalidade. Cada uma destas tem um tipo, sendo alguns destes já especificados pelo SPINE e podendo ainda serem extendidos. Neste modelo a entidade representa sub-dispositivos lógicos do dispositivo físico, para que esse dispositivo seja dividido assim como as suas respetivas funcionalidades, que estão associadas às entidades e não diretamente ao dispositivo. Estas três camadas e os seus respetivos tipos são representados na seguinte Figura 4.2 que descreve um breve exemplo da representação de um frigorífico seguindo este modelo.

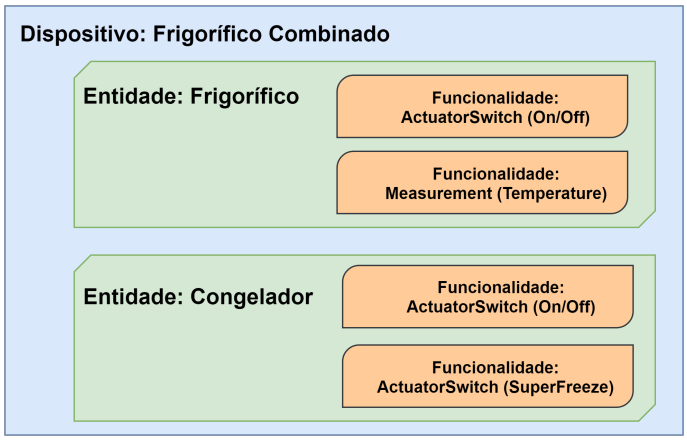

Figura 4.2: Exemplo de modelo de dispositivos do EEBus

Com o modelo dos dispositivos é possível definir as capacidades de qualquer dispositivo, necessitando apenas de fazer o mapeamento entre o modelo de dispositivos da plataforma a integrar e o modelo definido na Figura 4.2. Assim, a integração de novas plataformas torna-se simples para o resto da plataforma SATO dado que esta já suporta o modelo de dispositivos definido. Desta forma, cada dispositivo seguirá o modelo definido em que irá conter as entidades do dispositivo físico e as respetivas funcionalidades associadas e onde cada um terá um endereço que é utilizado nas respostas a pedidos ou na produção de eventos por parte dos dispositivos.

Tendo estes aspetos em consideração, o mapeamento dos dados para o modelo comum de dados deve ser feito nas três seguintes ocasiões:

- **Registo dos dispositivos e respetivos serviços** - No momento do registo dos dispositivos, ou seja quando uma plataforma é registada ou é efetuado o registo individual

de um dispositivo, a solução de integração deve mapear as funcionalidades e características do dispositivo de acordo com o modelo de dispositivos definido e esse modelo deve ser mantido para uso futuro.

- **Resposta a chamadas de serviços de informação à API** - Na chamada de serviços à API deve ser consultado o modelo do dispositivo criado no seu registo, de forma a consultar a informação pedida pelo serviço e a mapear a resposta do formato próprio da plataforma integrada para o modelo de dados único definido pelo SPINE. O modelo do dispositivo será útil para a identificação dos endereços da entidade e funcionalidade que representam os serviços em questão.

- **Produção de eventos por parte dos dispositivos** - Semelhante à chamada de serviços de informação, os eventos produzidos pelos dispositivos registados na plataforma SATO devem ser mapeados de acordo com o modelo único de dados de forma a que os dados sejam facilmente interpretados pelos restantes componentes SATO. Tal como na chamada de serviços, este mapeamento requer a consulta do modelo do dispositivo criado no respetivo registo na plataforma SATO.

Contudo, apesar da especificação ser adequada para o projeto SATO, alguns elementos da especificação no formato das mensagens não foram adotados por não serem necessários e desta forma reduzindo o tamanho de cada mensagem que siga o modelo comum de dados. Um exemplo da remoção de campos é verificado na ausência do endereço do destinatário no cabeçalho da mensagem, já que na plataforma SATO os eventos apenas necessitam de indicar qual o dispositivo a que pertence o evento. Tal como este exemplo, existem outros elementos que não foram reutilizados, desta forma reduzindo o tamanho das mensagens e promovendo assim um impacto positivo na performance. Este desenho genérico da estrutura utilizada pelo MCD nas mensagens é apresentado pela Listagem 4.1 utilizando a sintaxe do JSON.

```
{
  "payload":{
    "cmd":{
      "function": functionId,
      functionId: functionObject
    }
  },
  "header":{
    "addressSource":{
      "feature": featureId,
      "device": satoDeviceId,
      "entity": entityId
    },
    "cmdClassifier":"notify",
    "specificationVersion": spineVersion,
    "timestamp": value
  }
}
```

Listing 4.1: Estrutura datagrama MCD SATO

Em todos os casos de uso da uniformização dos dados a estrutura permanece a mesma, composta por um cabeçalho e um corpo de mensagem. No cabeçalho é indentificado o remetente, através do seu SATODeviceID, o ID da *entity* e da *feature* a que o evento se refere, o *timestamp* do evento e ainda a classificação do tipo de mensagem.

Já o corpo da mensagem, seguindo a especificação do SPINE, contém a indicação da função que o corpo detalha seguido do objeto JSON com os valores da função. No âmbito da definição desta estrutura, foi estabelecido que cada datagrama contém apenas uma única função para representação dos dados, seguindo assim a especificação do SPINE relativamente ao corpo do datagrama. Através da estrutura apresentada será assim possível para a plataforma que os dados inseridos na plataforma SATO sejam uniformizados facilitando a sua interprestação por parte dos restantes componentes.

4.3 Solução Semântica

Dada a necessidade de um componente capaz de persistir os metadados resultantes das plataformas integradas e respetivos dispositivos, a solução para este problema está na utilização de uma ontologia, já que estas oferecem mecanismos de pesquisa de dados poderosos resultante da conexão entre os dados inseridos que seguem uma ontologia definida. Esta ontologia deve então ser desenhada considerando os contextos em que o projeto SATO se enquadra, ou seja, o contexto energético associado ao contexto IoT devido aos dispositivos utilizados e ainda ao contexto dos edifícios, que descreve as localizações em que os dispositivos se encontram. Considerando os domínios a cobrir foram consideradas para servir de base as três seguintes ontologias:

- **SAREF** [21] - A Smart Applications REFerence ontology dispõe de um conjunto de classes e propriedades que facilita a representação dos dados no domínio de aplicações inteligentes.
- **SAREF4ENER** [22] - Dada a necessidade de englobar conceitos previamente não incluidos na ontologia SAREF, foi criada a extensão SAREF4ENER que se foca no domínio energético e acrescentando ainda um desenho de acordo com o modelo de dados do EEBus.
- **BOT** [6] - Building Topology Ontology foi a ontologia escolhida como base para suportar os requisitos semânticos relacionados com os edifícios, já que apresenta uma ontologia simples mas concisa que define os conceitos necessários para descrever as topologias dos edifício.

Baseado nestas três ontologias, é possível definir uma solução que satisfaz os requisitos para o componente semântico da solução aqui proposta. Esta reutilização foi feita sem que todos os conceitos das ontologias fossem necessariamente utilizados, dado que a ontologia deve ser simples e ao mesmo tempo deve ser capaz de descrever o contexto do projeto SATO. Contudo, a ontologia definida deve ser mantida apenas ao nível necessário para o projeto SATO, ou seja, utilizando apenas os conceitos essenciais. Tendo este aspeto em conta, na ontologia SAREF e SAREF4ENER foram reutilizadas as seguintes classes, seguindo a funcionalidade descrita:

- **s4ener:Device** - Extensão da classe saref:Device, sendo esta a principal classe da ontologia, onde grande maioria das restantes classes utilizadas estão de forma direta ou indireta ligadas à classe Device. Tem a função de representar os dispositivos na ontologia e a persistência de alguns metadados relativos às sua criação (i.é. marca, modelo, fabricante).
- **saref:Property** - Representa as propriedades medidas ou controladas por um dispositivo, permitindo assim identificação dos dispositivos pelas suas propriedades.
- **saref:Task** - Representa a tarefa de um determinado dispositivo.
- **saref:Function** - Representa uma função oferecida pelo dispositivo.
- **saref:Command** - Associada à classe Function, representa os comandos disponíveis pertencentes a uma determinada função do dispositivo.
- **saref:Profile** - Representa uma especificação do dispositivo associada a uma propriedade para o qual o perfil tenta optimizar.
- **s4ener:PowerProfile** - Sendo uma subclasse de Profile, tem o mesmo objetivo, sendo específica para a optimização da propriedade de energia, ou seja, a optimização energética de um dispositivo.

- **saref:FeatureOfInterest** - Representa qualquer tipo de entidade real do qual uma propriedade é medida, por exemplo, uma máquina de lavar sem capacidades inteligentes.

Nesta lista são apenas descritas as principais classes reutilizadas desta ontologia, em que algumas delas contêm sub-classes que são utilizadas para uma representação mais detalhada. Alguns exemplos são a específicação de sub-classes da classe Device, para descrever diferentes tipos de dispositivos, a especificação de diferentes tipos dentro da classe Function, por exemplo para detalhar se a função é de medição ou atuação, e ainda as sub-classes da classe Property. Com isto, foi também necessário a reutilização do domínio dos edifícios onde a Building Topology Ontology (BOT) foi reutilizada, reaproveitando as seguintes classes:

- **bot:Zone** - Classe genérica que representa uma zona, podendo ser de diversos tipos representados pelas suas sub-classes (Site, Building, Storey e Space).
- **bot:Site** - Representação de um local que contém um ou mais edifícios.
- **bot:Building** - Representa um edifício.
- **bot:Storey** - Serve o propósito de definir os pisos que fazem parte de um edifício.
- **bot:Space** - Representa um espaço tridimensional que pode conter outros sub-espaços.
- **bot:Element** - Representação genérica de um objeto ou componente presente num edifício.

Através das classes descritas na lista acima é possível a representação da topologia de um edifício assim como a associação entre elementos e a sua localização na topologia. Estes elementos podem ser considerados dispositivos, onde através desta ligação é possível que exista também uma conexão entre a descrição dos dispositivos e a topologia do edifício que os detém. Ainda no BOT, foram reutilizadas todas as *object properties* e *data properties* associadas às classes descritas na lista.

Depois de escolhidas as classes a reutilizar foi necessário definir novas classes que sejam capazes de representam os conceitos em falta, nomeadamente as unidades relativas às grandezas medidas pelos dispositivos, onde apesar de existir a classe UnitOfMeasure na ontologia SAREF esta não foi reutilizada, devido à ontologia SATO ter como objetivo a separação entre unidade de medição e atuação dos dispositivos. Dadas as necessidades relativamente à unidade de medição ou atuação no contexto em questão, foram criadas as seguintes classes:

- **sato:Unit** - Representa o conceito de unidade, seja ela de medida ou de atuação.

- **sato:PrefixSI** - Representa todos os prefixos segundo o Sistema Internacional.

Para que estas novas classes possam ser integradas na ontologia existente foi necessária a criação de novas *object properties*, que permitem a ligação entre o Device e a Unit e Unit e a PrefixSI. Estas propriedades são descritas na lista:

- **sato:actuatingUnit** - Permite a ligação entre a classe Device e Unit, especificando que a unidade em questão se aplica à unidade de atuação do dispositivo.
- **sato:measuringUnit** - Permite a especificação da unidade de medição utilizada por um dispositivo.
- **sato:prefix** - Associa à unidade de medição ou atuação o prefixo utilizado.

É importante perceber a razão pela escolha de criação do domínio de unidade, já que este é coberto por algumas ontologias existentes. Esta decisão foi tomada devido à complexidade das ontologias existentes, com dependências internas que não facilitam a sua reutilização e ainda devido à sua definição se focar apenas para o contexto de medições. Ainda no âmbito dos dispositivos, foram ainda adicionadas algumas *data properties* necessárias para identificar os dados através de identificadores conhecidos pela plataforma SATO. Propriedades essas descritas na seguinte lista:

- **sato:userId** - Identifica o utilizador SATO a que um dispositivo pertence.
- **sato:platformId** - Identifica a plataforma que permite a integração do dispositivo na plataforma SATO.
- **sato:proprietaryId** - Associa ao dispositivo o seu ID na plataforma subjacente que o integra.

Relativamente ao domínio de edifícios, foi acrescentada a classe BuildingType e as suas respetivas sub-classes que representam os diferentes tipos de edifícios já que estes metadados podem ser determinantes nas optimizações feitas.

A Figura 4.3 apresenta uma visão geral da ontologia SATO, expondo as principais classes e propriedades. Esta ontologia serve para manter os metadados necessários à plataforma SATO, seja para a integração dos dispositivos ou para outro tipo de serviços necessários por outros componentes da plataforma.

É possível verificar pela representação da ontologia SATO na Figura 4.3, a ligação entre o domínio dos dispositivos e o domínio dos edifícios. Esta ligação permite-nos assim associar um dispositivo a uma zona, e uma zona a um edificio, sendo essa informação crucial do ponto de vista da optimização energética, já que permite esta separação física entre os dispositivos, tendo em conta o seu contexto físico. Assim, este tipo de informação, que não estará incluída em grande maioria das medições produzidas, poderá ser utilizada tanto

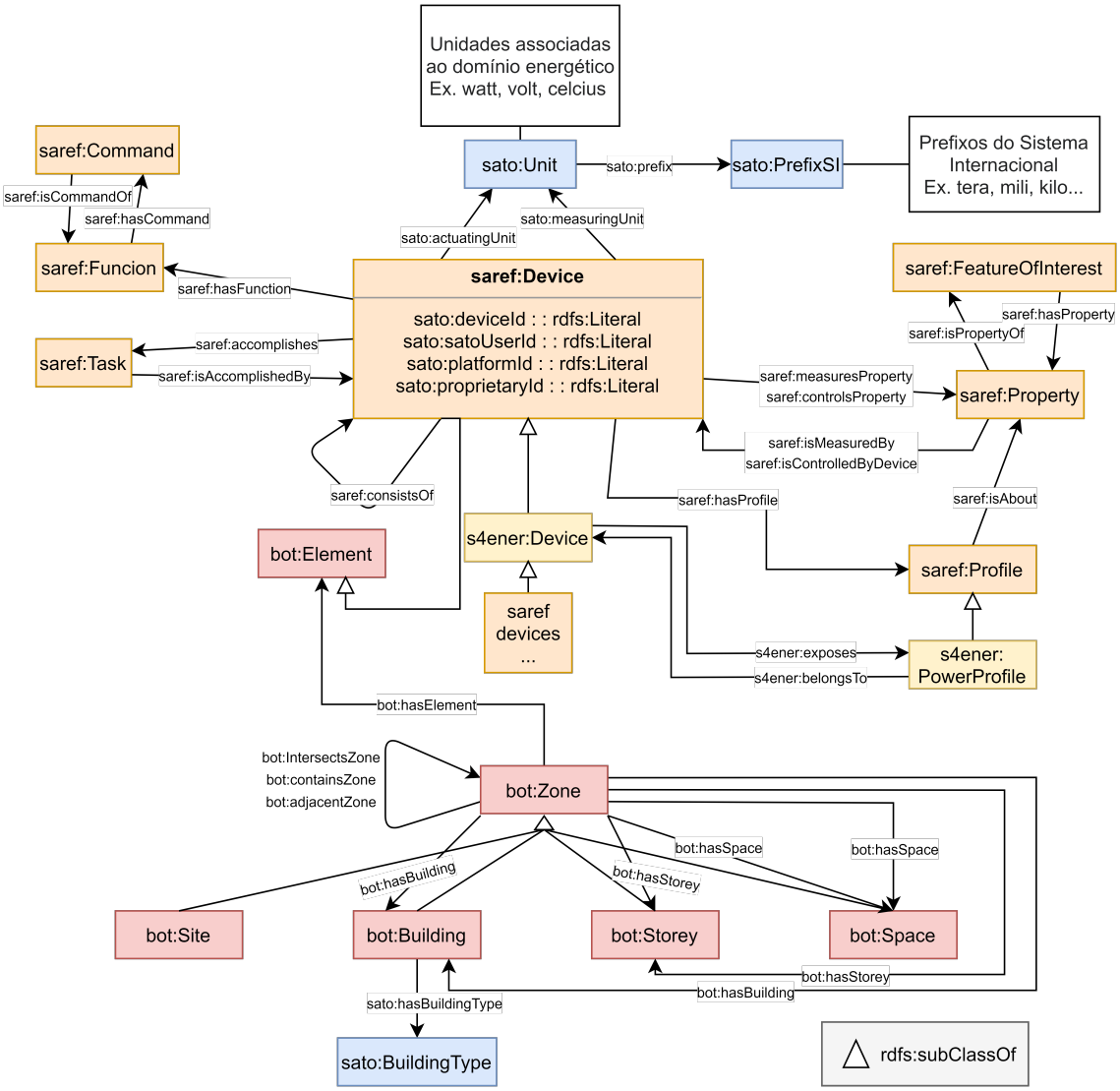

Figura 4.3: Principais classes e propriedades da ontologia SATO

para enriquecer alguns dados que fluem pela plataforma, como até mesmo para disponibilizar novos tipos de serviços, por exemplo de localização de dispositivos, que podem ser úteis a outros componentes da plataforma.

De notar que, apesar de um dos objetivos estar em manter a ontologia compacta mas completa, algumas das classes reutilizadas ou criadas foram desenhadas para uso futuro.

4.4 Interface Comum

Considerando o papel de grande importância que tem a interface do componente SATO Device Services API, é necessário definir a sua especificação de desenho para facilitar a sua utilização e compreensão. Ao avaliar a funcionalidade da interface a desenhar foram retirados os seguintes três *resources* a suportar:

- **Users** - Utilizadores SATO aos quais será atribuído um ID para que possam associar as suas diferentes plataformas e sistemas inteligentes que desejam registar na plataforma SATO.

- **Devices** - Dispositivos suportados pela plataforma SATO, em que lhes é atribuído um novo ID único para a plataforma utilizada de maneira a que possa ser identificado independemente da plataforma/sistema subjacente que o suporta.

- **Services** - Conjunto de serviços que a interface disponibilizará que são serviços resultantes das capacidades dos dispositivos integrados na plataforma.

Através dos três conceitos é possível desenhar a estrutura da interface REST a seguir pelo SATO Device Services API, que segue a lógica de associação dos dispositivos aos utilizadores e dos serviços aos respetivos dispositivos, sendo que, desta lógica, resultaram os seguites serviços:

- **POST /users** - Cria um novo utilizador SATO para que obtenha um SATO User ID que será utilizado em todos os futuros acessos à interface.

- **GET /users/{user-id}/devices/{device-id}/services/{service-id}** - Obtém o estado atual do serviço relativo ao dispositivo indicado e pertencente ao utilizador com dado user-id.

- **POST /users/{user-id}/devices/{device-id}/services/{service-id}** - Realiza a alteração do estado do serviço do dispositivo indicado.

Contudo, tal como foi analisado na Secção 4.1, nem todas as plataformas podem injetar diretamente os dados dos seus utilizadores na plataforma. Por essa mesma razão, este tipo de plataformas necessitam que exista um registo do utilizador na interface e assim se possa registar as credenciais de acesso à interface proprietária da plataforma em questão. Para que este passo seja possível, é necessário adicionar alguns *paths* para permitir a integração deste tipo de plataformas. Esses *paths* são apresentados na seguinte lista:

- **POST /users/{user-id}/platforms** - Registo de uma nova plataforma a associar ao utilizador em questão. Pelo corpo do pedido são passadas as credenciais necessárias para realizar este registo, assim como a identificação da plataforma a registar.

- **DELETE /users/{user-id}/platforms/{platform-id}** - Remoção do registo feito anteriormente pelo utilizador relativamente a uma plataforma específica, apagando também assim todos os dados dos dispositivos do utilizador para a plataforma indicada e dessa forma parando a receção de eventos para este utilizador.

- **POST /users/{user-id}/platforms/{platform-id}/devices** - Registo de um dispositivo para plataformas e sistemas inteligentes IoT que não dispõem de uma plataforma que permita o registo do utilizador.
- **DELETE /users/{user-id}/platforms/{platform-id}/devices/{device-id}** - Elimina os dados de registo de um dispositivo em específico, parando a respetiva receção de eventos por parte da plataforma SATO.

Por fim, considerando este conjunto de serviços disponibilizados pela interface, esta satisfaz as necessidades da plataforma SATO, sendo que poderá ser extendida no futuro para satisfazer novos requisitos e serviços. Uma possível extensão passa pela disponibilização de um serviço de atuação genérico. Este serviço recebe através do pedido HTTP a mensagem de atuação, seguindo o modelo comum de dados definido, onde indica no cabeçalho a *entity* e *feature* a que o pedido de atuação se refere e passando os dados de atuação no corpo dessa mesma mensagem. Desta forma, seria oferecida uma alternativa de interface com os dispositivos, facilitando a atuação por parte dos serviços de optimização da plataforma SATO.

Capítulo 5

Implementação

5.1 Arquitetura

Durante o desenvolvimento do trabalho, foi necessário criar uma prova de conceito da arquitetura descrita no Capítulo 4.1, e que os seus componentes estivessem de acordo com a arquitetura SATO, descrita na Secção 3.1. Com esta prova de conceito é possível analisar os componentes da solução e cada uma das decisões de implementação tomadas. Assim, através da arquitetura desenhada e da utilização de soluções, como o Bosch Smart Home, a Shelly Cloud e o SATO Monitoring Solution, foi implementada a arquitetura apresentada na Figura 5.1.

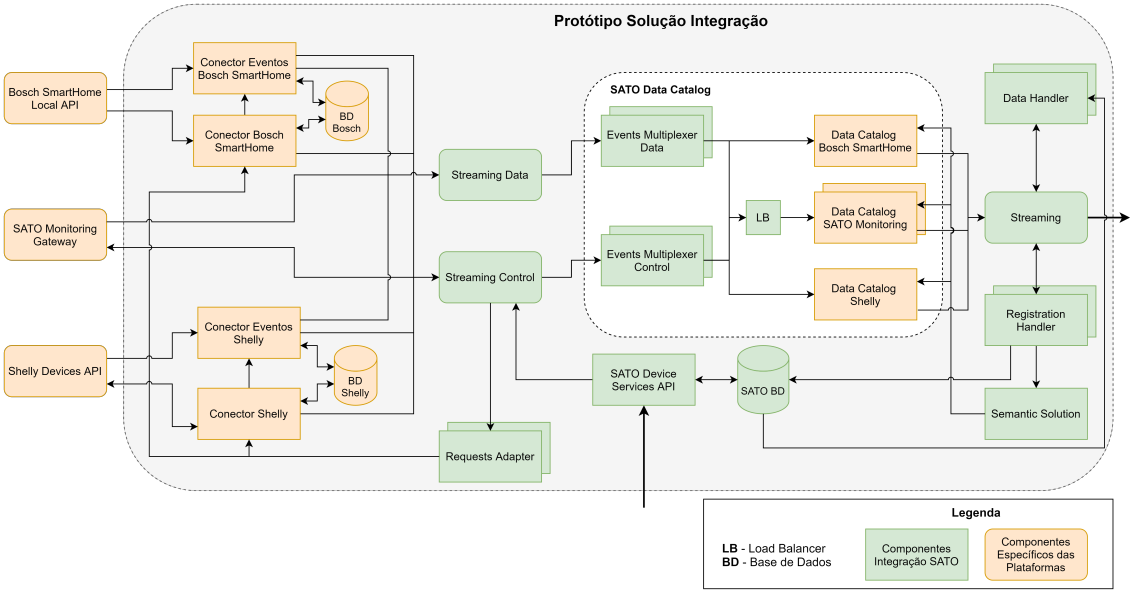

Figura 5.1: Arquitetura implementada no protótipo desenvolvido

Ao analisar a figura é possível perceber que a implementação segue a arquitetura definida para a solução de integração, desde a separação do SATO Data Catalog por diversos Data Catalogs proprietários, à utilização de conectores que possibilitam a integração de plataformas/sistemas IoT em que não é suportada a comunicação através da solução de

Streaming da plataforma SATO. Assim, através desta arquitetura implementada é possível concretizar o conceito da integração de diversas plataformas/sistemas para além dos que se conectam diretamente à plataforma SATO.

Tendo em consideração a arquitetura apresentada, a implementação da solução pode ser dividida pelos seus diversos componentes. Cada um destes componentes tem a sua lógica de processamento dos dados, sendo que deve ser descrita em detalhe para que seja possível a sua compreensão e por consequência facilite futuras expansões. Esta especificação da implementação da arquitetura começa por um dos pontos principais da solução, o SATO Device Services API. Este componente oferece uma API REST para gestão e controlo dos dispositivos integrados, para que esta interação possa ser feita sem a complexidade da integração das diversas plataformas/sistemas subjacentes. Durante a execução de um serviço, o seu princípal objetivo está em identificar a plataforma/sistema integrado que irá executá-lo. Por exemplo, na execução de um serviço de actuação num dispositivo, este componente identifica o `SATODeviceId` indicado no pedido HTTP e através da BD SATO, obtém a informação do dispositivo que distingue a plataforma que o integra, assim como seu ID proprietário. Assim, através desta informação o SATO Device Services API é capaz de realizar o pedido de serviço necessário à plataforma/sistema em questão e coloca-o no Streaming Control para que este seja consumido e consequentemente executado. Este fluxo de processamento por parte da API é apresentado, na Figura 5.2, num exemplo genérico de serviços orientados aos dispositivos. Sendo também importante notar que na implementação desta arquitetura não foram implementados Service Mappers, desenhados na Figura 4.1, já que o SATO Monitoring Gateway, que é a única solução que se adequa à utilização deste componente, não oferece nenhum serviço de atuação.

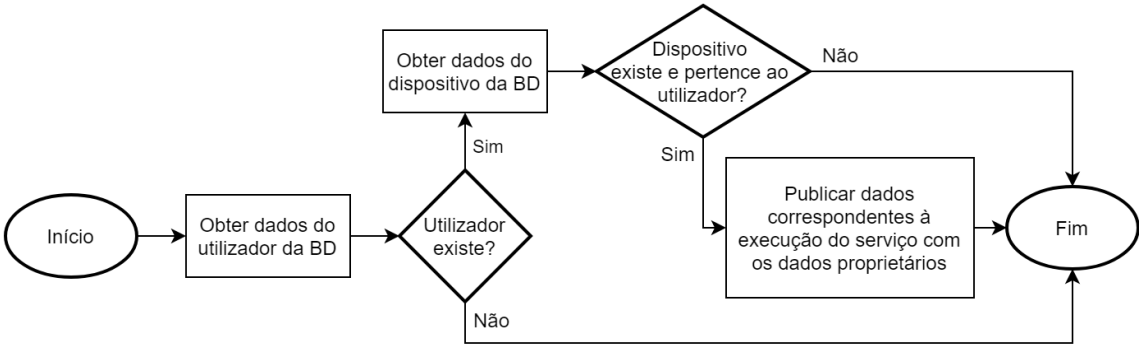

Figura 5.2: Diagrama de fluxo SATO Device Services API

Considerando a execução de serviços nas plataformas integradas, tal como foi analisado anteriomente, esta execução não é feita de igual forma para todas as plataformas. Algumas destas não permitem a interação direta com as soluções de Streaming da plataforma SATO, sendo nesses casos necessário o desenvolvimento de conectores que permitam a sua integração. No caso dos Conectores de Serviços das plataformas, estes dispõem

de uma interface que permite receber os pedidos de serviços provenientes, indiretamente, do SATO Device Services API. Nesta interação com o conector é indicado o SATOUserId a que está associado o pedido, desta forma podendo obter possíveis credenciais de acesso à interface da plataforma subjacente, credenciais essas que foram guardadas no registo do utilizador.

O conector realiza o registo com a interface da plataforma subjacente e se o registo for bem sucedido, este obtém a lista de dispositivos do utilizador de forma a que estes sejam integrados na plataforma SATO. Durante este processo, o conector realiza um pedido ao único serviço do conector de eventos da mesma plataforma e regista as credenciais do utilizador na sua base de dados. O funcionamento deste conector de serviços durante o registo de um utilizador é descrito pela Figura 5.3.

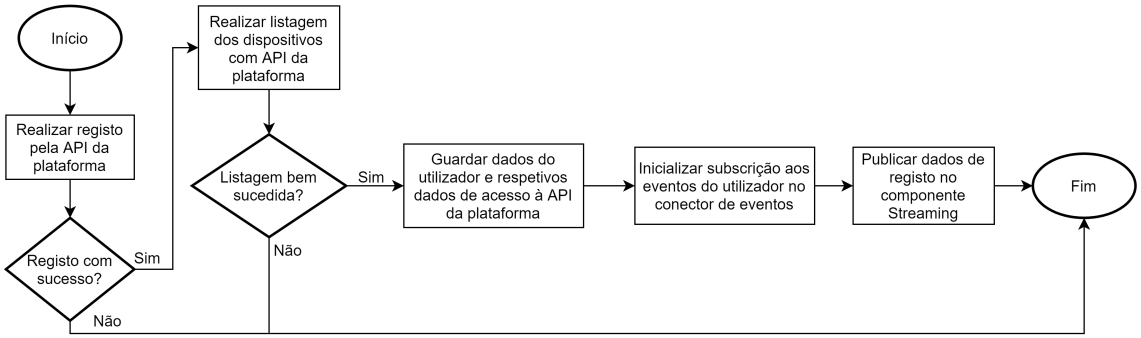

Figura 5.3: Diagrama de fluxo para registo no Conector da uma Plataforma

No serviço disponibilizado, o Conector de Eventos regista o novo utilizador e começa a executar um pedido periódico de novos eventos do utilizador, de forma a poder injetar esses eventos no componente Streaming Data. Desta forma, este componente oferece uma capacidade similar ao Publish/Subscribe, abstraindo dos restantes componentes da solução de integração a complexidade de aquisição dos novos eventos dos utilizadores. Esta lógica de funcionamento por parte do Conector de Eventos durante o registo de um novo utilizador é apresentada no diagrama de fluxo na Figura 5.4.

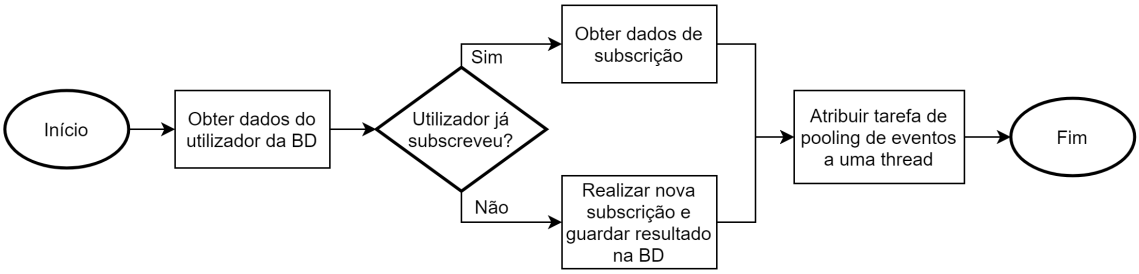

Figura 5.4: Diagrama de fluxo para registo no Conector de Eventos da uma Plataforma

Em ambos os conectores dos sistemas integrados (Bosch e Shelly), foram utilizadas as mesmas tecnologias da interface genérica do SATO Device Services, sendo esta des-

crita em detalhe na Secção 5.4. Já no funcionamento de execução de serviços associados à atuação dos dispositivos, apenas o conector principal é necessário, onde este obtém as credenciais de acesso do utilizador guardadas no seu registo e através delas contacta diretamente a interface proprietária da plataforma.

Após a implementação do SATO Device Services API, dos conectores dos sistemas integrados e analisando a arquitetura SATO, é possível verificar que a interação entre estes dois componentes deve ser feita através do componente Streaming Control. Contudo, os conectores disponibilizam os seus serviços através de uma interface REST. Dado este requisito, foi necessário desenvolver um adaptador que permitisse aos conectores continuarem a ter uma interface, semelhante à especificada na Secção 4.4, para execução dos seus serviços. Este adaptador tem o nome de Requests Adapter, cuja função passa pelo consumo de dados do Streaming Control e respetiva formulação de um pedido HTTP conforme os mesmos. Esta tarefa torna-se possível devido ao SATO Device Services API, já que este disponibiliza os pedidos de serviços num formato que facilita o processamento deste adaptador. Um exemplo desse formato é apresentado no Listing 5.1, onde é representado um exemplo de um pedido de atuação para ligar um dispositivo pertencente à plataforma da Bosch.

```
{
  "path": "http://sato-bosch:8080/users/userid/devices/bosch-
   deviceid/services/power",
  "header": {
    "Content-Type": "application/x-www-form-urlencoded"
  },
  "body": {
    "status": "1"
  }
}
```

Listing 5.1: Formato genérico de pedidos para conectores

Ainda analisando o registo de utilizadores nas plataformas, independentemente se necessitam de conectores, este devolve a lista dos dispositivos do utilizador, onde os novos dispositivos devem ser registados para que possam ser futuramente utilizados. De forma a desconectar este registo de algum componente específico a uma plataforma, foi implementado o Registration Handler. Este consome do componente Streaming as mensagens de registo dos dispositivos, uniformizadas pelo SATO Data Catalog, analisando o modelo de cada dispositivo descrito nos dados, juntamente com o SATOUserId a que este pertence e assim podendo associar o ID proprietário do dispositivo ao utilizador.

Contudo, para que este ID proprietário não seja exposto para as camadas superiores expondo as plataformas/sistemas integrados, um novo `SATODeviceId` deve ser gerado para que este seja utilizado em todas os futuros dados pertencentes a este dispositivo. Esta informação de cada dispositivo e do utilizador é registada no BD SATO, podendo assim na futura chamada de serviço ao SATO Device Services API ser consultado o ID

proprietário correspondente ao `SATODeviceId` indicado e assim como a identificação da plataforma/sistema que integra o dispositivo. Para além do registo dos IDs dos dispositivos na BD SATO, é também necessário persistir os metadados e os modelos de cada dispositivo, criado pelos Data Catalogs proprietários de acordo com a especificação do EEBus SPINE. Após os respetivos registos, os dados com os novos `SATODeviceIds` são publicados novamente no componente Streaming de forma a poderem seguir para a restante plataforma SATO. Para facilitar a compreensão do funcionamento do Registration Handler, este é descrito no diagrama de fluxo apresentado na Figura 5.5.

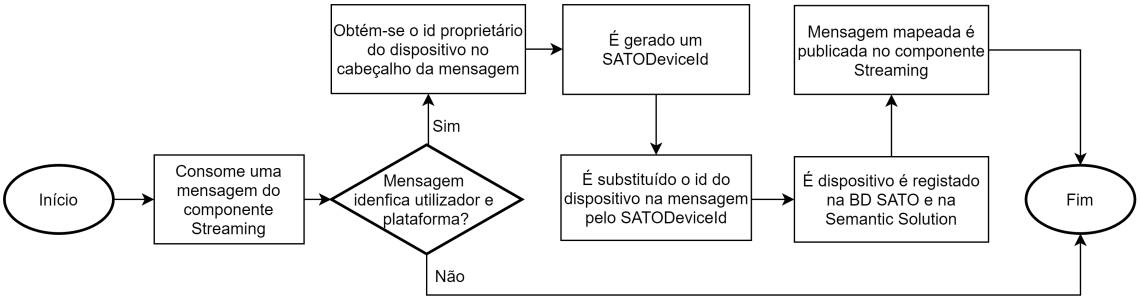

Figura 5.5: Diagrama de fluxo para registo de dispositivos no Registration Handler

Após este registo dos dispositivos na plataforma SATO, estes podem começar a injetar dados na plataforma de forma a serem processados e provocando possíveis optimizações energéticas. No entanto, devido à necessidade de tornar os Data Catalogs proprietários desconectados da BD SATO, estes componentes ao uniformizar os eventos recebidos identificam o dispositivo através do ID proprietário em vez do `SATODeviceId`. No entanto, dada a necessidade de desconectar a BD SATO de componentes proprietários, os Data Catalogs proprietários ao uniformizarem os eventos recebidos identificam o dispositivo associado ao evento através do ID proprietário em vez do `SATODeviceId`. Para solucionar este problema foi implementado o Data Handler, cuja tarefa passa por consumir os dados inseridos no Streaming pelo SATO Data Catalog, obter de cada evento o Id proprietário do dispositivo a que se refere e através da BD SATO obter o respetivo `SATODeviceId`. Com este Data Handler, os dados consumidos são modificados utilizando agora este `SATODeviceId` e de seguida colocando-os novamente no componente Streaming para que possam ser consumidos pela restante plataforma SATO. Este processamento dos dados por parte do DataHandler é representado pelo diagrama de fluxo apresentado na Figura 5.6.

Por fim, existem ainda outros componentes, dentro da arquitetura SATO, para os quais não foi detalhada a sua implementação. Isso deve-se ao facto destes terem uma maior importância na composição da solução assim como uma maior quantidade de detalhes da sua implementação e por isso seguem detalhados nas secções seguintes.

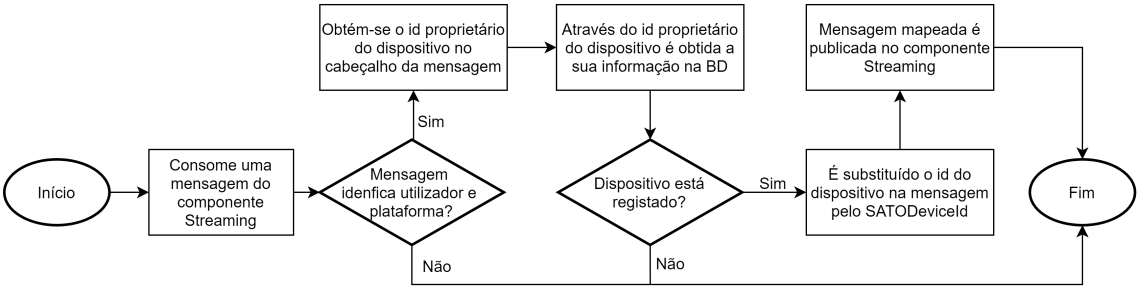

Figura 5.6: Diagrama de fluxo para mapeamento de dados do Data Handler

5.2 Modelo Comum de Dados

O Modelo Comum de Dados (MCD) da plataforma SATO está ao cargo do componente SATO Data Catalog para que este realize a uniformização dos dados da plataforma. Este MCD é baseado no EEBus SPINE, de forma a reutilizar a existente especificação podendo assim abranger mais facilmente o âmbito do projeto em que a solução se incluí. Desta reutilização surgiu a primeira decisão a tomar, já que ao estudar a especificação do SPINE foi verificado que toda ela utiliza Extensible Markup Language (XML), assim como todos os exemplos disponibilizados dos datagramas que contêm os dados. Contudo, é possível verificar que atualmente a linguagem de troca de dados mais utilizada é JSON, uma linguagem intercambiável de dados desenvolvida mais recentemente quando comparado com o XML. Para que uma decisão fosse tomada quanto à solução a utilizar foi utilizado o artigo [36], que realizou a avaliação de ambas as linguagens. Neste artigo o autor conclui, através do conjunto diversificado de testes entre as duas linguagens, que o JSON é substancialmente mais rápido e necessita de menos recursos. Tendo estes aspetos em consideração foi optado pela adoção de JSON como linguagem intercambiável de dados na plataforma.

Após a escolha da linguagem utilizada na uniformização dos dados, é necessário especificar a composição destes dados de maneira a que estes possam ser facilmente interpretados. Assim, nesta implementação foram considerados os dois seguintes tipos de mensagens:

- **Modelo de Dispositivo**: Este modelo descreve o dispositivo tanto pelo seu tipo, como pelas suas funcionalidades oferecidas. Esta é a mensagem gerada quando existe um registo do dispositivo na plataforma SATO.
- **Evento de Dispositivo**: Cada evento do dispositivo terá de ser uniformizado, sendo que estes devem identificar o dispositivo a que está associado o estado que contém, assim como a que funcionalidade do dispositivo os dados se referem.

Considerando a estrutura base do MCD apresentada no Listing 4.1, é necessário descrever a sua utilização para os dois principais casos de uso do protótipo, a mensagem

que descreve o modelo do dispositivo, gerada durante o seu registo, e a representação de um evento do dispositivo relativamente a uma das suas funcionalidades, de acordo com o modelo criado no seu registo.

Ao avaliar o caso de registo dos dispositivos e a respetiva uniformização destes dados, a especificação do SPINE define que cada dispositivo deve ter a primeira *entity* do tipo *DeviceInformation*, onde esta tem pelo menos a sua primeira *feature* do tipo *NodeManagement*. Este requisito deve-se à facilitação na interpretação das mensagens, podendo assim apenas através do cabeçalho concluir se a mensagem é referente ao anúncio do dispositivo e respetivas funcionalidades. Com esta regra, segue-se um exemplo deste tipo de mensagem, sendo o exemplo sobre uma tomada inteligente com a capacidade de monitorização energética e de poder ser desligada ou ligada. Este exemplo é apresentado na Listing 5.2.

```
{
  "payload": {
    "cmd": {
      "function": "nodeManagementDetailedDiscoveryData",
      "nodeManagementDetailedDiscoveryData": {
        "entityInformation": [
          {
            "description": {
              "entityAddress": {
                "entity": "0"
              },
              "entityType": "DeviceInformation"
            }
          },
          {
            "description": {
              "entityAddress": {
                "entity": "1"
              },
              "entityType": "SubMeterElectricity"
            }
          },
          {
            "description": {
              "entityAddress": {
                "entity": "2"
              },
              "entityType": "Relay"
            }
          }
        ],
        "featureInformation": [
          {...},
          {...},
```

```
35            {...},
36            {
37              "description": {
38                "role": "server",
39                "supportedFunction": {
40                  "function": "actuatorSwitchData"
41                },
42                "featureAddress": {
43                  "feature": "0",
44                  "featureType": "ActuatorSwitch",
45                  "entity": "2"
46                }
47              }
48            }
49          ],
50          "deviceInformation": {
51            "description": {
52              "deviceType": "SmartPlug",
53              "deviceAddress": {
54                "device": "satoDeviceId"
55              }
56            }
57          }
58        }
59      }
60    },
61    "header": {...}
62  }
```

Listing 5.2: Exemplo MCD em registo de tomada inteligente

No desenvolvimento do registo dos dispositivos na plataforma, foi encontrada a necessidade de extender a lista de tipo de dispositivos e de entidades para que esta servisse para os tipos utilizados nesta prova de conceito. Um exemplo disto é apresentado pelo exemplo da tomada inteligente, no Listing 5.2, em que o `deviceType` é do tipo `SmartPlug` e a entidade apresentada é do tipo `Relay`. Estes dois tipos não estão incluidos na especificação do SPINE, contudo é explícito na mesma que esta extensão pode ser feita nas implementações, já que estas servem para dar contexto aos dados que estes produzem através das funções das suas *features*.

É nestas funções das *features* que passa grande parte da uniformização suportada pela especificação do SPINE. Cada uma das funções dispõe de uma estrutura bem definida, onde o conteúdo das mesmas não é afetado pelo tipo de dispositivo, ou seja, do ponto de vista sintático a função que descreve uma medição de potência instântanea é a mesma que para a medição da temperatura. Isto permite que exista a uniformização dos dados sendo que a sua combinação com as *entities* e o seu correspondente tipo é que lhes acrescenta o contexto necessário para interpretar os dados. Com isto, é possível na Listing 5.3 ana-

lisar um exemplo desta uniformização para o caso da mudança de estado de uma tomada inteligente em que esta passou a estar ligada, ou seja, independentemente do contexto do interruptor, os seus dados são representados com a mesma estrutura do exemplo na figura apresentada. Este exemplo é correspondente ao dispositivo descrito na Listing 5.2.

```
{
  "payload":{
    "cmd":{
      "actuatorSwitchData":[
        {
          "function":"on"
        }
      ],
      "function":"actuatorSwitchData"
    }
  },
  "header":{
    "addressSource":{
      "feature":"0",
      "device":"satoDeviceId",
      "entity":"2"
    },
    "cmdClassifier":"notify",
    "specificationVersion":"1.1.1",
    "timestamp": timestamp
  }
}
```

Listing 5.3: Exemplo MCD para evento de tomada inteligente

No exemplo do Listing 5.3 é possível verificar a dependência entre o *header* da mensagem e o modelo de dispositivo criado durante o registo. Esta dependência resulta da utilização dos Ids das *entities* e *features* do dispositivo. Considerada esta dependência, é necessário que o SATO Data Catalog seja capaz de consultar o modelo do dispositivo criado no registo. A interação entre estes dois componentes é descrita na Secção 5.3, onde é possível analisar como é que o SATO Data Catalog obtém os Ids correspondentes ao evento que pretende mapear. Desta forma é possível a uniformização dos dados de cada evento recebido.

Com os exemplos analisados pode ganhar-se a ideia do tipo de mapeamento necessário entre os formatos proprietários das plataformas/sistemas para o modelo comum de dados SATO. Este mapeamento, na arquitetura interna do SATO Data Catalog, é atribuído a cada Data Catalog proprietário, onde cada plataforma/sistema tem o seu, sendo este responsável pela integração dos seus dados na plataforma SATO através desta uniformização. Para que estes componentes recebam dados necessitam de uma interface que forneça um serviço para receber dados de controlo e de eventos, sendo esses serviços utilizados pelos dois Events Multiplexers. Estes dois componentes são responsáveis pelo consumo de da-

dos dos componentes Streaming Data e Control, por interpretar a que plataforma os dados correspondem e assim reencaminhar para o Data Catalog proprietário adequado para realizar o mapeamento. Sendo que esta interação com os Data Catalogs proprietários é feita através de pedidos HTTP, passando por um *Load Balancer* que permite assim a replicação destes componentes e consequente distribuição de carga.

5.3 Solução Semântica

Através da solução semântica e respetiva ontologia desenhada na Secção 4.3, foi necessário realizar a implementação deste componente. Nesta implementação foi necessário escolher uma *framework* capaz de suportar o uso de ontologias e ainda que fornecesse uma interface capaz de interagir com qualquer tipo de componente, independemente das tecnologias que este utiliza. Através desta necessidade de interface resultou um novo requisito. O requisito passa por permitir a interação com os seus dados através de pedidos HTTP, facilitando assim a sua interação com outros componentes.

Tendo este requisito em mente e através da leitura de outros projetos e artigos científicos, que recorreram à utilização de um componente com as mesmas funcionalidades, optou-se pela utilização do Apache Jena Fuseki [1]. Esta solução disponibiliza um servidor de SPARQL, linguagem de *queries* e protocolo de bases de dados RDF, permitindo guardar os dados em triplos de acordo com a ontologia apresentada na Figura 4.3 e disponibilizando a execução de *queries* através de pedidos HTTP. Adicionalmente, o Fuseki permite a configuração de persistência dos dados em memória ou através da base de dados Apache Jena TDB [2]. Uma solução também pertencente ao Apache Jena responsável por garantir a persistência dos dados.

Depois de definida a solução a utilizar para o componente, seguiu-se a concretização da interação com este componente, onde este apenas considerou o domínio dos dispositivos devido aos dados produzidos pelas plataformas utilizadas para o desenvolvimento da prova de conceito implementada. Posto isto, o objetivo deste componente passa por guardar os metadados e os respetivos modelos de cada dispositivo registado na plataforma SATO, ou seja, uniformizar a representação dos mesmos, por exemplo através da caracterização das propriedades, funcionalidades e características de fabrico. A uniformização destes dados inicia-se quando é feito um novo registo dos dispositivos na plataforma, através do qual as *entities* e *features* resultantes da uniformização dos dados de registo do dispositivo são mapeadas para as propriedades (Property) e funcionalidades (Function) dos dispositivos abrangidos pela ontologia descrita na Secção 4.3.

Assim, neste registo é possível guardar esta informação dos dispositivos uniformemente, permitindo assim abstrair esta representação dos dispositivos das plataformas/sistemas integrados, e sendo essa informação representada na forma de um diagrama que descreve a ligação de alguns desses dados através da Figura 5.7.

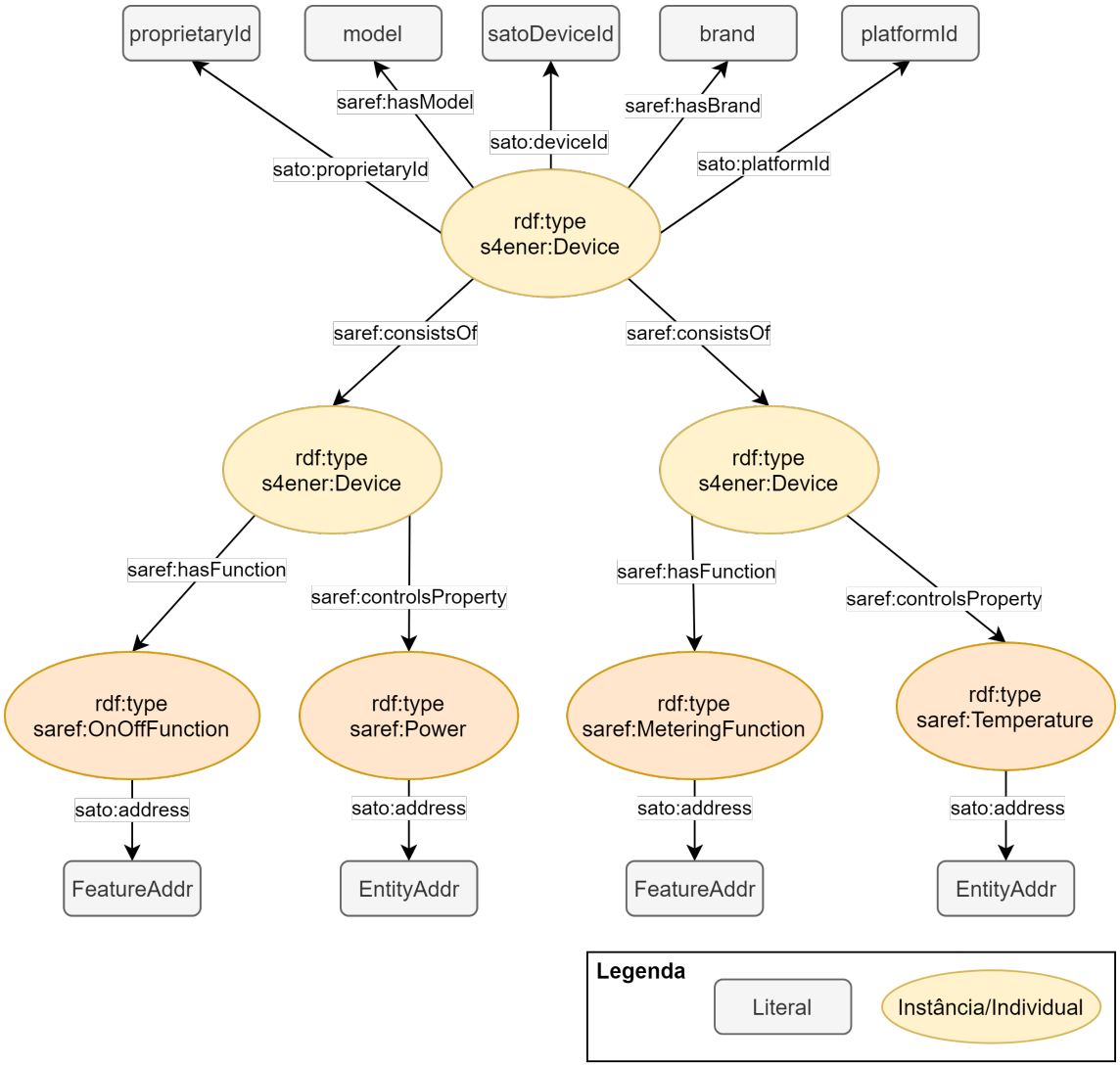

Figura 5.7: Exemplo do modelo de dispositivo de um frigorífico inteligente

Após o registo do dispositivo, este começerá a produzir dados sobre o estado das suas funcionalidades, por exemplo a temperatura e humidade medidas, devendo ser estas medições mapeadas para o MCD. Contudo, como foi detalhado na Secção 5.2, este mapeamento requer a consulta dos dados de registo guardados no Fuseki. Esta consulta é feita através de um *query* em SPARQL, onde no Listing 5.4 é possível obter um exemplo de uma pesquisa dos IDs registados por um dispositivo em específico para uma determinada funcionalidade.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns
   #>
PREFIX sato: <http://www.semanticweb.org/satoOnt#>
PREFIX s4ener: <https://saref.etsi.org/saref4ener/>
PREFIX saref: <https://saref.etsi.org/core/>

SELECT ?entityAddr ?featureAddr
WHERE
{
  ?device rdf:type s4ener:Device;
        saref:consistsOf ?subDevice;
        sato:deviceId "satoDeviceId".
  ?subDevice saref:measuresProperty ?property;
        saref:hasFunction ?function.
  ?property rdf:type saref:Power;
        sato:address ?entityAddr.
  ?function rdf:type saref:OnOffFunction;
        sato:address ?featureAddr.
}
```

Listing 5.4: Exemplo de pesquisa de IDs de entity e feature do dispositivo

Na *query* apresentada na Figura 5.4, é possível verificar que existe uma equivalência entre a *Entity* e *Feature*, do Modelo Comum de Dados SATO, com as *Property* e *Function*, da Ontologia SATO definida. Na implementação a *Entity* corresponde à classe *Property* e a *Feature* corresponde à classe *Function* na ontologia definida. É através desta ligação que é possível guardar os modelos dos dispositivos juntamente dos seus restantes metadados.

5.4 Interfaces

Após análise do desenho da interface descrita na Secção 4.4 é retirado o principal requisito desta, que passa pela concretização de uma interface REST. Considerado este requisito, resultaram duas decisões a tomar, a definição de uma *framework* que facilite a implementação da interface, tratando dos detalhes de implementação como a complexidade de conexão com os serviços, como por exemplo a utilização de sockets, e ainda a definição de uma solução que forneça um servidor web capaz de utilizar o código de implementação, descrito através da *framework* escolhida, e disponibilize os seus serviços.

Com estes requisitos em mente e considerando a escolha de Java como linguagem de programação, foi feita a análise das *frameworks* existentes que facilitem a concretização de APIs REST. Desta análise foi estabelecido que a *framework* a utilizar deve seguir a especificação JAX-RS, descrita em [34].

Esta especificação suporta a criação de serviços web, seguindo o padrão arquitetural REST, através de anotações e assim facilitando a definição dos serviços. É então necessário optar por uma solução que implemente esta especificação. Após alguma pes-

quisa sobre as implementações existentes, foi escolhido o Eclipse Jersey [12] por este ser *open-source* e amplamente utilizado por parte da comunidade no contexto de desenvolvimento de serviços web RESTful. De forma a demonstrar a implementação usando a especificação JAX-RS, é descrito um excerto de um serviço web, no Listing 5.5, responsável por executar comandos associados ao serviço Power dos dispositivos integrados.

```
1  @Path("/users/{sato_user_id}/devices/{sato_device_id}/services
2  /power")
3  public class PowerController {
4
5          @Inject
6          private Publisher producer;
7
8          @Inject
9          private DevicesDAO db;
10
11         @POST
12         @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
13         public void setDevicePowerStatus(@PathParam(value = "
               sato_user_id") String satoUserId,
14                 @PathParam(value = "sato_device_id") String
                       satoDeviceId,
15                 @FormParam(value = "status") int status) {
16                 ...
17         }
18 }
```

Listing 5.5: Exemplo de um serviço utilizando JAX-RS

Neste exemplo de implementação de um serviço utilizando as soluções escolhidas, é possível verificar a utilização de anotações para a descrição do serviço e seus respetivos componentes. Para começar é identificado o *path* da classe, usando o `sato_user_id` e `sato_device_id` como variáveis já que este serviço aplica-se para diferentes utilizadores e dispositivos. Assim, todos os serviços desta classe se aplicam para o *path* definido ou para extensões deste. Depois de definido o *path* e considerando que por defeito cada chamada do serviço cria novos atributos da classe, é necessária a anotação @Inject para deixar ao cargo da *framework* a inicialização destes atributos. Esta injeção depende da configuração dada à *framework*, onde nos exemplos mostrados é definido o padrão *singleton* já que estes dispõem de mecanismos de *connection pooling*, ou seja estes objetos permitem a reutilização de objectos previamente criados em vez de realizar a inicialização destes a cada chamada do serviço. Após a anotação dos atributos, é verificado a definição de um serviço web onde este é definido por diferentes tipos anotações. Neste caso é inicialmente definido que este serviço recebe apenas pedidos POST e que apenas recebe dados do tipo `application/x-www-form-urlencoded`. Dada a ausência da anotação `@Path` para este serviço, é então herdada a anotação `@Path` da classe a que pertence, ficando assim responsável por tratar dos pedidos recebidos, quando estes se tratam de pedidos `POST` e quando os dados enviados no corpo do pedido são as definidas pela anotação `@Consumes`. Considerando ainda que grande parte dos serviços

necessitam de *input* na forma de parâmetros, é necessário que esta especificação os suporte. Para começar, existem os parâmetros passados através do *path* do serviço, onde o exemplo contém o `sato_user_id` e o `sato_device_id`. Estes tipos de parâmetros são obtidos através da anotação `@PathParam`, no qual se indica o `value`, valor esse que deve corresponder ao nome dado na anotação `@Path` que se aplica ao serviço. Adicionalmente existem ainda os parâmetros passados através do corpo do pedido, estes são obtidos através da anotação `@FormParam` à qual se aplica a mesma lógica do value que se aplica ao `@PathParam`. Em ambos os tipos de parâmetros, caso este `value` não corresponda ao nome dado ao valor que se pretende obter, a *framework* trata de preencher o valor da variável a `null`. Assim, com o exemplo apresentado é possível verificar a utilização das anotações seguindo a especificação JAX-RS e através do qual se comprova o *decoupling* com solução responsável pela implementação da especificação.

Após a descrição do serviço exemplo e da definição da *framework* a usar, é necessário estabelecer uma solução para servidor web da implementação dos serviços criados pelo Apache Jersey. Para cumprir esta tarefa foi escolhido o Apache Tomcat [4] para a disponibilização dos serviços web HTTP devido este ser uma solução *open-source* amplamente utilizada pela comunidade para servidor web. Para finalizar, é importante notar que todos os componentes do protótipo que dispõem de uma interface, recorrem a este conjunto de soluções para a implementação das suas interfaces e através destas é possível a concretização de uma interface web capaz de cumprir os seus requisitos.

5.5 Ambiente de execução

Depois de finalizar a implementação de todos os componentes da solução de integração desenvolvida no trabalho desta tese, foi concluído que estes deveriam ser executadas usando tecnologias atuais. Com este requisito foi então estabelecido que estes componentes iriam ser executados através de *containers* Docker [11].

Os *containers* podem ser vistos como pequenas máquinas virtuais, sendo capazes de isolar a aplicação e as suas dependências do sistema operativo subjacente e ainda de outros *containers*.

A sua utilização torna-se ainda mais interessante quando são executadas diferentes aplicações na mesma máquina, onde neste caso a utilização de múltiplas máquinas virtuais se tornaria mais complicada. Com a capacidade dos *containers* de se isolarem do sistema operativo subjacente estes tornam-se portáveis, podendo ser executados em qualquer máquina desde que esta disponha do Docker runtime. Com isto, através da separação da solução entre diversos componentes, sendo cada um executado num *container*, é garantido assim atingir uma solução modular o que promove a sua escalabilidade.

Durante o desenvolvimento desta prova de conceito não foram utilizadas soluções como o Docker Swarm [10] ou Kubernetes [18], que permitiriam organizar os componen-

tes por serviços e podendo assim mais facilmente realizar a escalabilidade de cada um. Contudo, através da utilização de *containers* já funcionais para cada componente, esta transição passa a ser um passo relativamente simples de fazer.

Esta não separação de componentes por serviços provocou a necessidade de implementar um *Load Balancer*, para que os componentes que disponibilizem interfaces web possam ser replicados e acedidos através de um único endereço. A solução utilizada para este foi o NGINX [19], cuja função quando utilizado é apenas redirecionar os pedidos HTTP para as réplicas nele registadas. Na sua configuração foi definida a estratégia de redirecionamento de `least_conn`, já que esta configuração permite que o *Load Balancer* tenha em consideração o número de conexões de cada instância e assim distribuir para a que apresenta maior disponibilidade para responder.

Com estes aspetos em consideração a solução desenvolvida pode ser dividida pelos seus dois tipos de componentes, os consumidores dos componentes de *streaming* que ficam constantemente à escuta de novas mensagens e os componentes que disponibilizam os seus serviços através de uma interface web. Com isto, todos os componentes utilizam Java como linguagem de programação, havendo assim a necessidade a que as imagens base utilizadas para cada um dos componentes tenha a Java Virtual Machine para que estes possam ser executados. Contudo, no caso dos componentes com uma interface web, estes componentes utilizam o Apache Tomcat como servidor web, ou seja a sua imagem base deve conter ainda esta solução, para ser capaz de executar o servidor e disponibilizar assim os seus serviços. Adicionalmente, ambos os dois tipos de componentes utilizam Apache Maven [3] para garantir que as bibliotecas especificadas como depedências estejam incluídas nas aplicações.

Capítulo 6

Resultados

6.1 Introdução

Este capítulo tem o propósito de apresentar os testes do protótipo desenvolvido neste trabalho. Este protótipo contém as principais características arquiteturais desenhadas para a solução final do projeto SATO e para o qual solucionará o problema de integração de plataformas e sistemas inteligentes heterogéneos de IoT. Para que os valores retirados dos testes se tornem úteis para análise de performance considerando os requisitos da plataforma SATO, os mesmos foram executados na máquina do projeto SATO que será responsável pela hospedagem dos diversos componentes da solução final. Assim, os resultados do protótipo desenvolvido permitiram analisar a performance da solução no ambiente real de funcionamento. Adicionalmente, e para tornar a avaliação o mais realista possível, foram utilizados geradores de dados, externos ao *middleware*, que permitem simular grandes volumes de dados a chegar ao *middleware*. Tendo em conta estes aspetos, o computador utilizado nos testes contém as especificações apresentadas na Tabela 6.1.

Parâmetro	Valor
Processador	2x Intel® Xeon® Silver 4214 12C/24T 2.20GHz
Memória RAM	128 GB
Disco	1 TB
Sistema Operativo	Ubuntu 20.04.2 LTS

Tabela 6.1: Especificações da máquina de testes

Relativamente ao volume de dados esperado, para os testes realizados nesta tese, foi considerado o valor de 100000 eventos por minuto. Este valor foi definido para a plataforma SATO como o pior cenário possível onde a plataforma terá de operar.

A versão atual do protótipo apenas utiliza uma das máquinas disponiveis no cluster da plataforma SATO. Neste sentido, o valor considerado como máximo para a injeção de dados foi de 50000 eventos por minuto (metade do que será requirido quando a plataforma estiver toda em funcionamento).

Contudo, não seria plausível a injeção de dados no protótipo usando dispositivos físicos, tendo sido desenvolvido um gerador de dados que simula a solução SATO Monitoring Gateway, enviando o número definido de eventos por minuto.

Este gerador calcula a frequência a enviar eventos, dado o valor de *thoughput* desejado e através deste valor injeta os dados no componente Streaming Data.

Adicionalmente, este gerador é capaz de injetar um número definido de novos dispositivos na plataforma SATO, desta forma testando a capacidade do protótipo em registar estes dispositivos e fazer todo o consequente processamento.

Juntamente ao gerador, foi também desenvolvido um consumidor de dados na saída do *processo* para que seja possivel calcular o *throughput* do protótipo de forma mais precisa.

Assim, com estes dois componentes implementados, são então definidos os dois casos de teste do protótipo, a capacidade de processamento de eventos de dispositivos na plataforma, descrito na Secção 6.2, e a capacidade de processamento da solução para o registo de novos dispositivos, descrito na Secção 6.3.

6.2 Avaliação de processamento de eventos

Considerando o contexto da solução dentro da plataforma SATO, um dos principais objetivos será a capacidade de processamento e respetiva uniformização dos dados produzidos pelas plataformas e dispositivos integrados.

Esta tarefa, de acordo com a Figura 5.1, segue um fluxo de dados comum que se inicializa pela injeção dos dados no Streaming Data, seguido do respetivo consumo por parte do Events Multiplexer Data, onde este analisa a que plataforma pertencem os dados e reencaminha-os para o respetivo Load Balancer.

Considerando que nos testes realizados apenas é utilizado o SATO Monitoring Gateway, então o Load Balancer distribuí o pedido para uma das réplicas do SATO Monitoring Data Catalog, sendo este componente responsável pela principal uniformização dos dados.Assim, após a sua uniformização, o SATO Monitoring Data Catalog insere os dados no componente Streaming para que estes possam ser processados pelo Data-Handler. Como é esperado, este componente consome os dados publicados e realiza substituição pelos respetivos SATODeviceIds, colocando assim os dados novamente no componente Streaming para que possam seguir para os restantes componentes da plataforma SATO.

Com a descrição do fluxo comum de dados no caso de teste, é necessário avaliar o tempo de processamento médio por cada um destes componentes. Os tempos resultantes da comunicação pela rede interna do protótipo, dos componentes de Streaming e Load Balancer são desprezados dado o seu valor ser muito reduzido (na ordem dos nanosegundos).

Esta análise permite a verificação da capacidade de processamento de cada um dos componentes desenvolvidos e assim verificar qual o componente que apresenta uma maior

dificuldade na capacidade de processamento da solução como um todo.

Para que a análise possa ser feita por etapas, foi estabelecida uma configuração base na qual nenhuns dos componentes foi replicado, desta forma obtendo um resultado base de referência do protótipo. Contudo, para que fosse possível a avaliação do impacto no *throughput* provocado pelo número de dispositivos registados na solução , foram realizados dois casos de teste. No primeiro apenas o dispositivo que o gerador de dados simula está registado como dispositivo na solução e no segundo foram testadas as mesmas capacidades de processamento quando se encontram 5000 dispositivos registados na plataforma. Com os casos de teste definidos, foi definido para os testes a injeção de 50000 eventos por minuto durante um período de 10 minutos, realizando 5 repetições para verificar que o seu comportamento é constante e assim resultando na injeção total de 500000 eventos por repetição. A Figura 6.1 apresenta assim os valores médios de eventos processados por cada um dos componentes ao longo deste fluxo de processamento para cada um dos casos de teste, seguindo as normas de teste definidas.

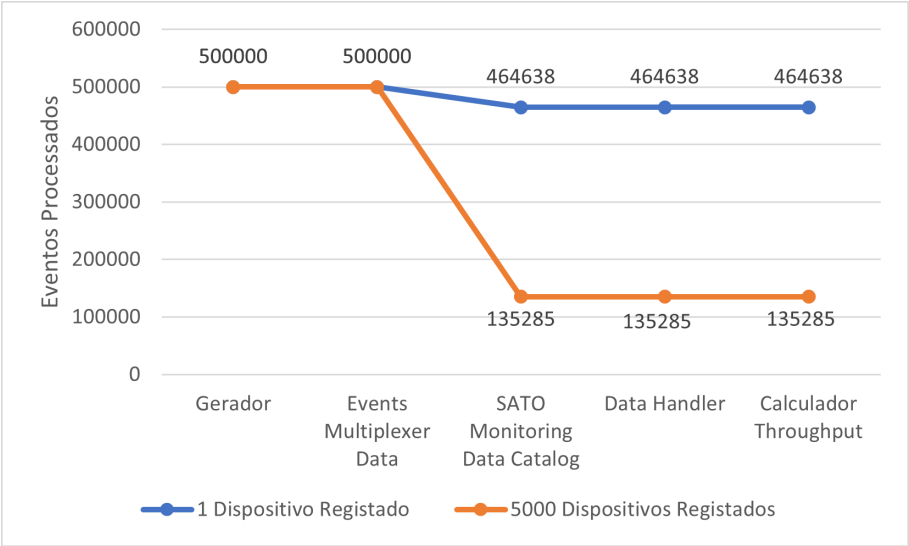

Figura 6.1: Configuração Base: Capacidade de processamento de eventos

A partir da Figura 6.1, verifica-se que o registo de dispositivos na solução tem um impacto significativo na sua performance de processamento e uniformização dos dados. Verificou-se ainda que nesta configuração base, mesmo no caso com apenas um dispositivos registado, não foi possível atingir o *throughput* desejado. Em ambos os casos, o *throughput* desejado não é atingido devido à perda de eventos por parte do SATO Monitoring Data Catalog. Este *bottleneck* é também percetível quando apresentados, na Figura 6.2, os valores médios de tempo de processamento total por parte de cada componente em ambos os casos testados nesta configuração.

Juntamente com a análise dos eventos processados por componente, é possível através da Figura 6.2 confirmar o problema por parte do SATO Monitoring Data Catalog. Este

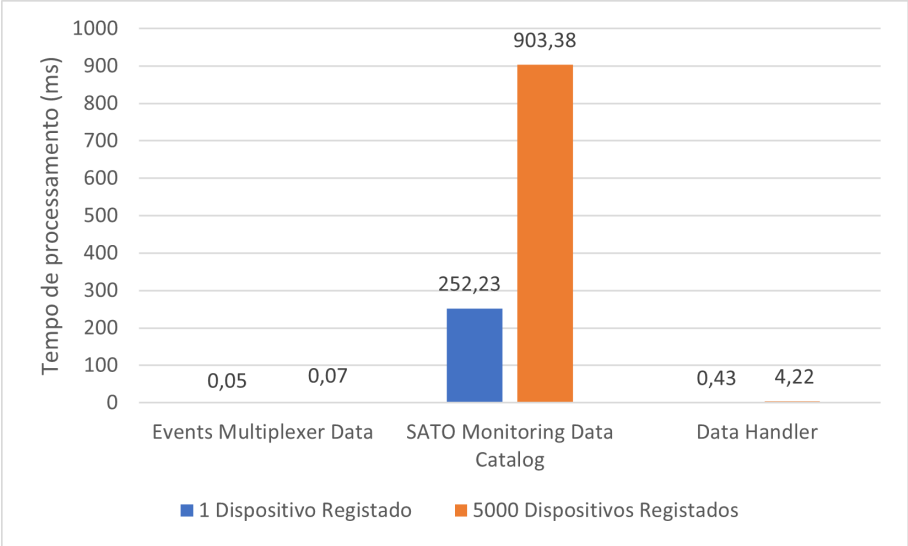

Figura 6.2: Configuração Base: Latência por componente no processamento de eventos

componente é responsável pelo mapeamento do formato proprietário para o modelo comum de dados através da consulta do modelo criado no seu registo e persistido na base de dados que suporta os dados de acordo com a ontologia definida. Esta dependência por parte deste componente, em que é necessária a consulta à Semantic Solution para cada evento, torna-se a principal razão para a presença de *overhead* no processamento dos eventos, onde nos dois tipos de testes realizados na configuração base esta dependência representa 99.9% do tempo de processamento deste componente.

Com isto, o *overhead* resultante da interação com o Semantic Solution provoca a perda de eventos. Isto verifica-se na Figura 6.1 e deve-se à sua interação através de pedidos HTTP. Nesta interação, o Events Multiplexer realiza pedidos assíncronos para o Load Balancer do SATO Monitoring Data Catalog a uma frequência superior à dos eventos processados por ele, o que resulta em *timeouts* dos pedidos devido à sua indisponibilidade de resposta e a consequente perda de eventos.

Tendo em conta estes resultados, procedeu-se à optimização da configuração base, replicando o SATO Monitoring Data Catalog para assim testar o desempenho nesta nova configuração. Foram novamente executados os dois tipos de testes relativamente ao número de dispositivos registados na plataforma. Desta forma verifica-se o desempenho desta configuração na interação com o Semantic Solution. Os resultados dos eventos processados por cada componente nesta nova configuração são apresentados na Figura 6.3

Através desta segunda configuração, é possível verificar que com a replicação deste componente os resultados foram significativamente melhores. Ao avaliar o caso com apenas um dispositivo registado é possível concluir que neste, através desta replicação, o SATO Monitoring Data Catalog é capaz de processar e uniformizar todos os eventos

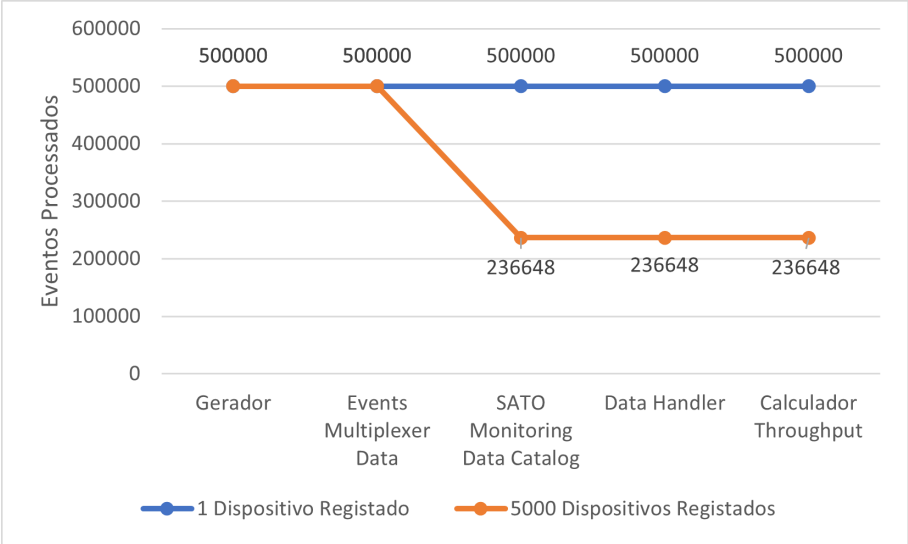

Figura 6.3: Configuração Replicada: Capacidade de processamento de eventos

injetados na plataforma, não se verificando qualquer perda ou atraso significativo.

Contudo, no caso em que existem 5000 dispositivos registados, os resultados apesar do aumento de cerca de 75% relativamente ao mesmo teste na configuração base, continuam a não ser os ideais, já que se verifica em média uma perda de 53% dos eventos injetados. Esta perda de eventos tem a mesma causa que a dos testes na configuração base, devendo ser avaliado o tempo de processamento por evento de cada componente para que se possa comparar entre as duas configurações testadas.

Nesta avaliação existem duas réplicas do SATO Monitoring Data Catalog, sendo que para a sua representação na Figura 6.4 é feita a média dos seus tempos de processamento, sendo esse o valor apresentado na figura para este componente.

Considerando a replicação do SATO Monitoring Data Catalog, na Figura 6.4 verifica-se que o tempo de processamento por evento deste componente no caso de um único dispositivo registado é menor do que na configuração base. Este resultado deve-se ao facto de, através da replicação, ter uma maior capacidade de processamento de mensagens, realizando a sua uniformização mais rapidamente e resultando assim na maior capacidade de *throughput*.

Já no caso em que existem 5000 dispositivos registados, verifica-se um aumento do tempo de processamento comparativamente à configuração base. Este aumento deve-se ao maior número de eventos processados que por consequência provoca um aumento no tempo de processamento do Semantic Solution, dado que este tem responder a aproximadamente o dobro dos pedidos quando comparado com o número de eventos processados para o mesmo teste na configuração base.

Os testes realizados para avaliar a capacidade de processamento de eventos por parte da solução desenvolvida permitiram concluir que a dependência dos Data Catalogs proprietários sobre o componente Semantic Solution deve ser evitada. Caso esta dependência

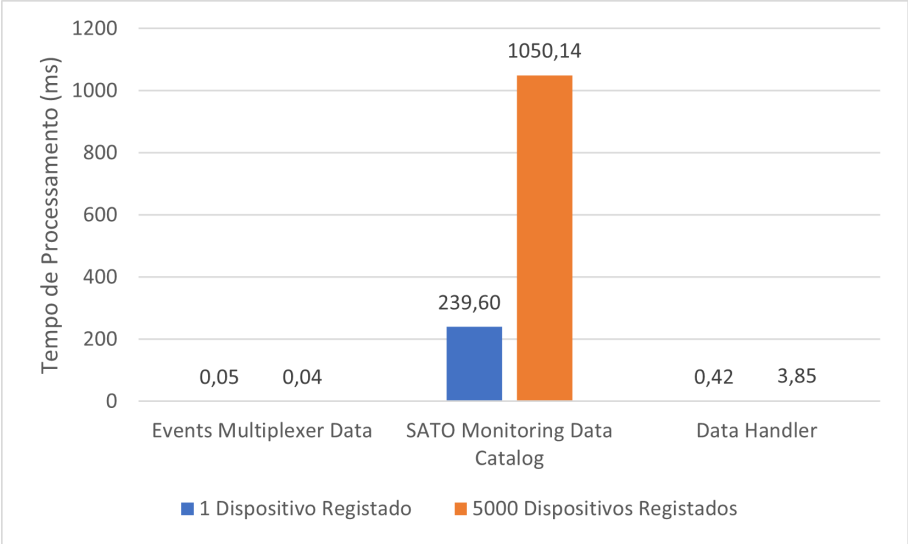

Figura 6.4: Configuração Replicada: Latência por componente no processamento de eventos

seja eliminada, o processamento dos eventos de dados, por parte das plataformas e dispositivos integrados, passa a ser realizada sem que seja necessária a constante consulta dos dados de registo na ontologia. Como demonstrado pelos resultados, este fenómeno agrava-se com o aumento do número de dispositivos registados.

6.3 Avaliação de processamento de registo de dispositivos

No funcionamento da solução, para que os eventos possam ser injetados na plataforma é necessário que os respetivos dispositivos estejam registados. Neste sentido definiu-se um teste que permite avaliar o desempenho da solução quando exposta a grandes quantidades de registo de novos dispositivos.

Para analisar este desempenho foi utilizada a configuração base, em que existe apenas uma instância de cada componente, e ainda uma configuração optimizada, onde são utilizadas 3 réplicas do componente Replication Handler, e assim demonstrar a capacidade de melhorar os resultados obtidos com a configuração base. Em ambas as configurações são injetados 10000 novos dispositivos por minuto na solução durante o período de 5 minutos e fazendo 3 repetições deste mesmo teste, para que seja possível calcular os valores estatísticos que permitem aferir o desempenho do sistema.

De maneira a que os resultados possam ser mais facilmente interpretados é importante conhecer o fluxo dos dados pelos componentes da solução, durante o registo de um dispositivo.

Este fluxo começa pela injeção dos dados de registo no componente de Streaming Control, onde o Events Multiplexer Control é responsável por consumir e por redirecionar estes dados para o Load Balancer adequado e onde esse tratará de distribuir o pedido para

as réplicas registadas. No caso de teste, dada a configuração base, este apenas dispõe de uma instância do SATO Monitoring Data Catalog, para a qual serão reencaminhados todos os eventos de registo injetados no teste. De seguida, o SATO Monitoring recebe os dados através do pedido enviado pelo seu *load balancer*, prossegue com a uniformização da descrição das capacidades dos dispositivo. Após a uniformização, estes dados são colocados no componente Streaming, prontos para serem processados pelo Registration Handler. Este componente consome os dados disponíveis no Streaming, interpreta-os de acordo com o modelo de dispositivo criado pelo Data Catalog e regista o dispositivo na Semantic Solution e na SATO BD. Por fim, o Registration Handler, após os respetivos registos do novo dispositivo, coloca o modelo de dispositivo com o novo SATODeviceId novamente no componente Streaming, para que este possa ser consumido pela restante plataforma.

Depois de analisado o fluxo de dados de registo na solução, foram realizados os testes em ambas as configurações descritas acima e nas quais não se verificou qualquer perda de dados durante o processo de registo. Contudo, neste processamento dos dados de registo é possível verificar na Figura 6.5 que existe um elevado *overhead* no processamento dos mesmos, na configuração base. Contudo, após análise dos resultados da configuração base, foi identificada a necessidade de realizar os mesmos testes numa configuração optimizada, sendo os seus resultados apresentados na mesma figura.

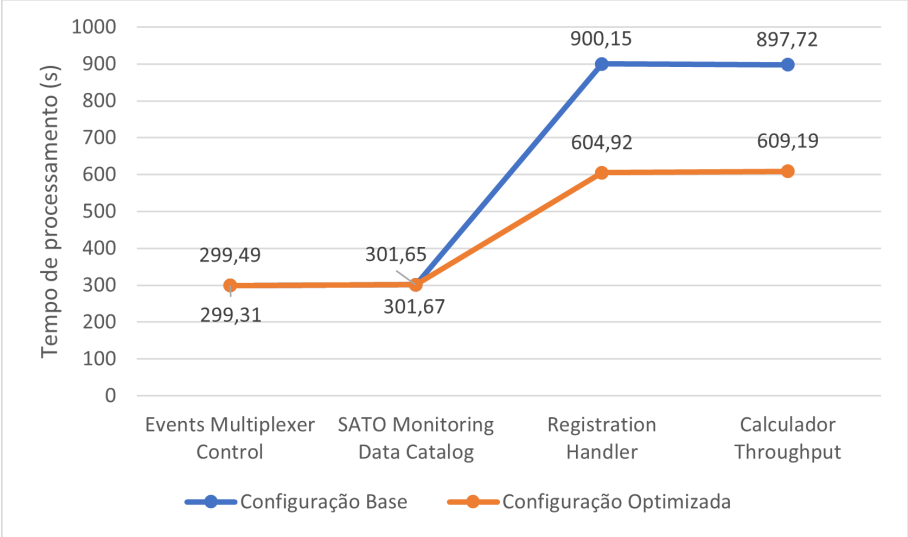

Figura 6.5: Configuração Base: Tempo total de processamento de registo por componente

A Figura 6.5 permite verificar o *bottleneck* na capacidade de processamento dos dados por parte do Registration Handler, em ambas as configurações, sendo este um componente Single-Threaded que consume os dados e processa o respetivo registo.

O *overhead* existente neste componente deve-se à sua implementação Single-Threaded, onde este não é capaz de registar tantos dispositivos quantos os que estão a ser inseridos na solução. Porém, devido à utilização de containers, este componente é fácilmente

replicável tal como se verificou na configuração optimizada onde foram utilizadas 3 réplicas do Registration Handler para, desta forma, reduzir o tempo de processamento dos registos e consequenctemente aumentar o *throughput* da solução. Desta forma, este valor aumentou de 3341 registos por minuto na configuração base, para 4925 registos por minuto na configuração optimizada. Após realizados este conjunto de testes, é importante perceber a razão pela qual não se verifica perda de dados mesmo sendo o *throughput* destes componentes inferior ao do gerador de dados. Para isso, ao analisar a perda de dados verificada nos testes da Secção 6.2, concluiu-se que esta resulta da interação entre os componentes do SATO Data Catalog através de pedidos HTTP, onde a cadência de processamento de dados foi inferior à cadência de chegada dos mesmos.

Já nos testes realizados de registo, apesar do fluxo continuar a passar pelo SATO Monitoring Data Catalog, este não necessita de interagir com outro componente durante o registo, sendo apenas responsável por manipular os dados para realizar a uniformização dos dados de registo. Sendo assim, este componente, como se verifica na Figura 6.5, deixa de provocar o *bottleneck* no sistema, passando a verificar-se agora no Registration Handler pelas razões descritas anteriormente.

Contudo, este *bottleneck* não resulta na perda de dados de registo, sendo este fenómeno graças ao componente Streaming. Este componente oferece a capacidade de manter os dados que por ela passam, acabando por fazer o paper de um *buffer* destes dados. Assim, o Registration Handler acaba por processar aos dados com a cadência que é capaz e o Streaming mantém os restantes ainda não consumidos. Com isto, é necessário ter em conta que este *buffer* não é infinito e dependendo das configurações deste componente, os dados podem acabar por ser perdidos. No entanto, o componente Streaming permite ao Registration Handler alguma folga num cenário de elevada carga, permitindo que não se percam dados quando estes aumentam e podendo caso necessário replicá-lo para lidar com a maior afluência de registos.

6.4 Resumo

De forma a sumarizar os resultados obtidos construiram-se as Tabelas 6.2 e 6.3. Começando pela análise da Tabela 6.2, esta representa a capacidade de processamento de eventos. Os valores apresentados correspondem a um total de 500000 eventos gerados em 10 minutos, e correspondem também às estatísticas determinadas a partir das diferentes execuções (5 repetições de cada teste).

Estes resultados permitem verificar que a capacidade de processamento da solução, apesar de capaz de escalar, não atingiu os valores pretendidos. Adicionalmente, verifica-se que o *throughput* é afetado pelo número de dispositivos registados, o que se deve à dependência entre os componentes Data Catalogs proprietários e o Semantic Solution. Com isto, concluí-se que para a solução atingir melhores valores de *throughput* é ne-

Teste Realizado	Eventos Processados	Eventos Perdidos	Latência Média por Evento (ms)	Desvio Padrão (ms)
Configuração Base: 1 Dispositivo Registado	464638	35362	252,71	2,45
Configuração Base: 5000 Dispositivo Registado	135285	364715	907,68	22,70
Configuração Optimizada: 1 Dispositivo Registado	500000	0	240,07	42,37
Configuração Optimizada: 5000 Dispositivo Registado	236648	263352	1054,03	46,31

Tabela 6.2: Resumo dos resultados de processamento de eventos

cessária a remoção desta dependência, já que esta representa em média 99,7% do tempo de processamento total por evento.

Dado que para que os eventos de dados sejam produzidos é necessário que existam dispositivos registados na plataforma SATO, foi então testada a capacidade de processamento de registos da solução de integração (Tabela 6.3). De notar que nestes testes foram gerados 50000 registos de dispositivos. Cada teste teve a duração de 5 minutos e fizeram-se 3 repetições. Mais uma vez, os valores apresentados correspondem à estatística resultante das várias repetições.

Teste Realizado	Registos Processados	Latência Média por Evento (ms)	Desvio Padrão (ms)	Tempo Total de Processamento (s)
Configuração Base	50000	9,89	0,06	900,14 (15min)
Configuração Optimizada	50000	19,00	0,05	609,19 (10min)

Tabela 6.3: Resumo dos resultados de processamento de registos

Através dos resultados, é possível verificar que todos os registos foram realizados com sucesso. Contudo, os mesmos resultados mostram-nos que o tempo total de processamento dos registos, é muito superior ao tempo de execução do teste, sugerindo a necessidade de se escalarem componentes da arquitetura.

Ao contrário do que aconteceu no processamento de eventos, o ponto de falha deixou de ser no componente SATO Monitoring Data Catalog e passou a ser no componente Registration Handler. Contudo, ao replicar este componente verificou-se um aumento significativo no *throughput*, e consequentemente, uma redução acentuada no tempo total de processamento dos registos gerados.

Capítulo 7

Conclusão e Trabalho Futuro

7.1 Conclusão

Considerando a atual importância do consumo energético, cada vez mais são necessárias novas abordagens que o permitam reduzir e que ao mesmo tempo promovam a sua flexibilidade para que seja maioritariamente dependente de energias renováveis. Neste contexto, os dispositivos da Internet das Coisas (IoT) têm vindo a ajudar, oferecendo capacidades de monitorização e controlo remoto. Contudo, com a proliferação destes sistemas no mercado criou-se um problema de integração, configuração e operação dos mesmos.

De forma a colmatar estes problemas, nesta tese foi desenhada, implementada e testada uma solução de *middleware* que permite a integração de diferentes plataformas de IoT usadas na monitorização e controlo de energia, numa única plataforma, a plataforma SATO (Self Assessment Towards Optimization of Building Energy). A plataforma SATO é uma plataforma de *Assessments* que tem como objetivo a auto-avaliação e otimização dos recursos energéticos dos edifícios através da utilização de dispositivos inteligentes.

O *middleware* proposto nesta tese oferece capacidades de abstração dos serviços proprietários de cada plataforma, tratando da comunicação com os sistemas integrados de forma unificada através de serviços genéricos. A exposição destes serviços é feita através de APIs e de um modelo de dados único que permite compreender a informação proveniente de cada uma das plataformas integradas. Adicionalmente, a solução desenvolvida apresenta mecanismos que permitem colmatar problemas de interoperabilidade, tanto semântica como sintática, resultantes dos conjuntos de dados heterogéneos das diferentes plataformas/sistemas.

De forma a colmatar os problemas de interoperabilidade sintática foi utilizado um componente que faz o mapeamento entre o modelo de dados específico utilizado pela plataforma integrada e o modelo de dados interno da plataforma SATO. O modelo de dados interno utilizado na plataforma SATO foi proposto e desenvolvido no âmbito desta tese. No caso dos problemas de interoperabilidade semântica, foi desenhada uma ontologia que permite armazenar metadados relativos aos dispositivos e aos edifícios onde os mesmos

estão instalados. Esta informação permite enriquecer todas as análises que são feitas relativas aos dispositivos, edifícios ou à conjugação de ambos, de forma a compreender os consumos energéticos e desenhar novos modelos de optimização energética.

Para além da definição destes mecanismos, foi criado um protótipo que permite demonstrar e avaliar a capacidade da arquitetura proposta nesta tese. Com base neste protótipo foram definidos e executados testes que permitiram aferir a capacidade da solução em lidar com diversas plataformas integradas, gerando grande volume de eventos por minuto.

O protótipo desenvolvido seguiu a abordagem de micro-serviços, onde cada componente foi executado através de *containers* Docker. Esta separação em micro-serviços permite implementar a arquitetura de forma modular e melhora a possibilidade de escalar cada um dos componente de forma a melhorar o desempenho global do sistema. Na implementação deste protótipo não foram utilizadas soluções como o Docker Swarm ou Kubernetes. A possibilidade do uso destas soluções não era foco desta tese, sendo no entanto importante considerá-las de forma a garantir um melhor desempenho de toda a plataforma SATO.

Tendo em consideração o protótipo implementado, executaram-se um conjunto de testes que permitiram perceber o desempenho da solução e identificar possíveis pontos de falha e/ou limites. Neste cojunto de testes, os resultados obtidos mostraram-se promissores, onde através da replicação de alguns componentes foi possível processar 50000 eventos por minuto. Contudo, como identificado e discutido ao longo do documento, existem algumas limitações na solução desenvolvida, sendo que a principal limitação advém da solução semântica que foi implementada. O acesso a este componente mostrou-se demorado, provocando perda de eficiência em todo o sistema. Assim, com estes resultados podemos concluir que a arquitetura proposta é eficiente e adequada, permitindo a resolução de questões de interoperabilidade que grandes partes dos sistemas têm quando tentam integrar soluções de IoT provenientes de terceiros.

Por fim, esta tese permitiu-me desenvolver competências de análise e desenho de soluções arquiteturais de *middleware* para IoT, promovendo a capacidade de trabalho autónomo e a capacidade de análise crítica dos diferentes mecanismos envolvidos numa solução completa. Por outro lado, a integração desta tese num projeto Europeu permitiu-me estar integrado num ambiente de investigação, trabalhar numa equipa com diferentes pessoas, de diversas instituições. Adicionalmente, este envolvimento permitiu-me aprimorar as capacidadades de documentação através da participação na escrita de *deliverables* do projeto.

7.2 Trabalho Futuro

Através dos resultados obtidos concluiu-se que a solução desenvolvida carece de algumas optimizações para atingir uma maior capacidade de processamento.

Um dos componentes que poderá ser optimizado é o SATO Data Catalog. Dada a constante necessidade de consulta dos dados de registo por parte deste componente, a sua capacidade é limitada, acabando por restringir significativamente a performance de todo o sistema. Uma possivel alteração passaria pela modificação do cabeçalho das mensagens para que seja eliminada a dependência dos identificadores associados às *entities* e *features* de cada dispositivo.

Relativamente aos serviços disponibilizados, a utilização da ontologia permite a criação de novos serviços e funcionalidades extendendo assim as existentes, oferecidas pelas plataformas e sistemas integrados. Um exemplo destes serviços seria a definição de um serviço que permita desligar todos os dispositivos com a capacidade de aquecimento e localizados num determinado espaço. Este tipo de serviço torna-se possível através da informação de registo das plataformas e respetivos dispositivos que é mantida no componente responsável pela ontologia.

Com estas duas optmizações será possível tornar a solução ainda mais completa e capaz de lidar com um elevado número de dispositivos e grandes quantidades de dados, que poderão vir a ser integrados na plataforma SATO.

Complementarmente e para tornar o sistema ainda mais escalável, este deverá ser configurado de forma a executar num sistema distríbuido de *containers*, recorrendo à utilização de um orquestador de *containers*. Todos os micro-serviços deverão ser implementados seguindo a especificação do *composer*, onde a escalabilidade e o *load balancing* são geridos pela solução de orquestação (Docker Swarm, Kubernetes ou outra semelhante).

Por fim, e de forma a tornar a plataforma SATO mais completa, a solução proposta deverá integrar um maior número de plataformas e sistemas IoT, validando a utilidade da solução proposta e oferecendo a capacidade de desenvolver novos serviços utilizando essas mesmas plataformas integradas.

Bibliografia

[1] Apache Jena Fuseki. https://jena.apache.org/documentation/fuseki2/.

[2] Apache Jena TDB. https://jena.apache.org/documentation/tdb/.

[3] Apache Maven Website. https://maven.apache.org/.

[4] Apache Tomcat Website. http://tomcat.apache.org/.

[5] Bosch IoT Gateway Software. https://developer.bosch-iot-suite.com/service/gateway-software/.

[6] Building Topology Ontology (BOT). https://w3c-lbd-cg.github.io/bot/.

[7] Building Topology Ontology (BOT) Overview. https://w3c-lbd-cg.github.io/bot/#Zones-Overview.

[8] Cosmos Documentation. https://fiware-cosmos.readthedocs.io/en/latest/.

[9] DeviceHive Documentation. https://docs.devicehive.com/docs.

[10] Docker Swarm Overview. https://docs.docker.com/engine/swarm/.

[11] Docker Website. https://www.docker.com/.

[12] Eclipse Jersey Website. https://eclipse-ee4j.github.io/jersey/.

[13] EEBus Technology. https://www.eebus.org/technology/.

[14] EEBus Website. https://www.eebus.org/.

[15] Energy@home Project. http://www.energy-home.it/.

[16] FIWARE Website. https://www.fiware.org.

[17] Kaa Website. https://www.kaaproject.org/.

[18] Kubernetes Website. https://kubernetes.io/.

[19] NGINX Website. https://www.nginx.com/.

[20] RESPOND. http://project-respond.eu/.

[21] SAREF Website. https://sites.google.com/site/smartappliancesproject/ontologies/reference-ontology.

[22] SAREF4ENER Ontology. https://saref.etsi.org/saref4ener/v1.1.2/.

[23] Sitewhere Website. https://sitewhere.io/en/.

[24] Eurostat news release. https://ec.europa.eu/eurostat/documents/2995521/9549144/8-07022019-AP-EN.pdf/4a5fe0b1-c20f-46f0-8184-e82b694ad492, 2 2019.

[25] Victor Araujo, Karan Mitra, Saguna Saguna, and Christer Åhlund. Performance evaluation of fiware: A cloud-based iot platform for smart cities. *Journal of Parallel and Distributed Computing*, 132:250 – 261, 2019.

[26] M. A. A. da Cruz, J. J. P. C. Rodrigues, J. Al-Muhtadi, V. V. Korotaev, and V. H. C. de Albuquerque. A reference model for internet of things middleware. *IEEE Internet of Things Journal*, 5(2):871–883, 2018.

[27] Mauro A.A. da Cruz, Joel J.P.C. Rodrigues, Arun Kumar Sangaiah, Jalal Al-Muhtadi, and Valery Korotaev. Performance evaluation of iot middleware. *Journal of Network and Computer Applications*, 109:53 – 65, 2018.

[28] Laura Daniele, Monika Solanki, Frank den Hartog, and Jasper Roes. Interoperability for Smart Appliances in the IoT World. pages 21–29, 10 2016.

[29] EEBus Initiative e.V. *EEBus SPINE Technical Specification Protocol Specification*, 12 2018. Version 1.1.1.

[30] EEBus Initiative e.V. *EEBus SPINE Technical Specification Resource Specification*, 12 2018. Version 1.1.1.

[31] EEBus Initiative e.V. *EEBus Technical Specification Smart Home IP*, 11 2019. Version 1.0.1.

[32] Amirhossein Farahzadi, Pooyan Shams, Javad Rezazadeh, and Reza Farahbakhsh. Middleware technologies for cloud of things: a survey. *Digital Communications and Networks*, 4(3):176 – 188, 2018.

[33] Nikolaos Georgantas, Georgios Bouloukakis, Sandrine Beauche, and Valérie Issarny. Service-oriented distributed applications in the future internet: The case for interaction paradigm interoperability. In Kung-Kiu Lau, Winfried Lamersdorf, and Ernesto Pimentel, editors, *Service-Oriented and Cloud Computing*, pages 134–148, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[34] Marc Hadley and Paul Sandoz. Jax-rs: Java™ api for restful web services. *Java Specification Request (JSR)*, 311, 2009.

[35] E. J. Mendes, M. M. Silveira, M. B. Araújo, J. Celestino, and R. L. Gomes. Abstraction of Heterogeneous IoT Devices for Management of Smart Homes. In *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6, 2019.

[36] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: A case study. pages 157–162, 01 2009.

[37] E. J. Palacios-Garcia, B. Arbab-Zavar, J. C. Vasquez, and J. M. Guerrero. Open iot infrastructures for in-home energy management and control. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pages 376–379, 2019.

[38] Luis Pérez-Lombard, José Ortiz, and Christine Pout. A review on buildings energy consumption information. *Energy and Buildings*, 40(3):394 – 398, 2008.

[39] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(1):70–95, 2016.

[40] Peter Salhofer. Evaluating the FIWARE Platform. In *HICSS*, 2018.

[41] T. Storek, J. Lohmöller, A. Kümpel, M. Baranski, and D. Müller. Application of the open-source cloud platform FIWARE for future building energy management systems. *Journal of Physics: Conference Series*, 1343:012063, nov 2019.

[42] Dennis van Goch, Marc Eulen, Stefan Lodeweyckx, and Chris Caerts. Rennovates, flexibility activated zero energy districts, h2020. *Impact*, 2017(5):29–31, 2017.

[43] Apostolos Zarras, Nikolaos Georgantas, Thiago Teixeira, Sara Hachem, Valérie Issarny, Fabio Kon, Pierre Châtel, Amira Ben Amida, Santos Carlos Eduardo Moreira Dos, Rafael Correia, Daniel Cukier, Nelson Lago, Dionysis Athanasopoulos, and Panos Vassiliadis. CHOReOS Middleware Specification (D3.1). working paper or preprint, October 2011.