P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

M

MESTRADO
ENGENHARIA INFORMÁTICA

Apoio à Definição de Arquiteturas IIoT Inteligentes

Bruno Cunha

06/2021

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

M

MESTRADO
ENGENHARIA INFORMÁTICA

Apoio à Definição de Arquiteturas IIoT
Inteligentes
Bruno Cunha

06/2021

Bruno Cunha. Apoio à Definição de Arquiteturas IIoT Inteligentes

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

P.PORTO

M

MESTRADO
ENGENHARIA INFORMÁTICA

Apoio à Definição de Arquiteturas IIoT Inteligentes

Bruno Cunha

Prof. Doutor Cristóvão Dinis Sousa

Agradecimentos

Gostaria de começar por agradecer ao meu orientador, o Professor Doutor Cristóvão Sousa, pela disponibilidade, paciência e exigência ao longo do meu percurso académico e na realização da tese de mestrado.

Ao centro de investigação onde sou bolseiro, o INESC TEC, mais concretamente aos meus colegas do CESE. Adicionalmente, aos Engenheiros Rui Rebelo e Elder Hernández, pela colaboração e apoio prestado na elaboração desta dissertação e na orientação nos meus primeiros passos do meu percurso profissional.

E por fim agradecer à minha família e amigos, em especial aos meus pais, pelo apoio que me deram nesta caminhada, e por me encorajarem sempre a colocar dedicação em tudo o que faço na vida.

Resumo

As plataformas IIoT (Industrial Internet-of-Things) são um facilitador na transformação digital, no âmbito da indústria 4.0, promovendo a flexibilidade, para uma adaptação mais rápida às necessidades do mercado, e permitindo às organizações ter uma visão clara sobre o seu estado atual. No entanto, as PMEs (Pequenas e Médias Empresas) estão a encontrar dificuldades na mudança para este novo paradigma, devido à falta de: i) recursos qualificados que são necessários para desenvolver e implementar as suas próprias soluções de digitalização; ii) um entendimento claro sobre a reengenharia necessária que envolve a digitalização, na adoção de soluções IIoT, e iii) modelos adequados para a especificação de soluções IIoT orientadas às PMEs. Com intuito de ultrapassar estes desafios, discute-se uma solução numa dupla perspetiva, em que: i) por um lado, procura-se a automatização da especificação das arquiteturas de plataformas IIoT, de acordo com as necessidades específicas do negócio, reduzindo o investimento necessário para desenvolver este tipo de facilitadores I4.0, e; ii) por outro lado, fomentar o entendimento partilhado do IIoT, entre os especialistas do domínio e as organizações, promovendo o envolvimento de ambas as partes neste processo de especificação. Neste contexto, a semântica desempenha um papel importante, permitindo a acomodação do conhecimento multidisciplinar das arquiteturas IIoT num modelo semântico alavancado por capacidades de raciocínio. Esta solução foi avaliada num caso de estudo, em que a arquitetura produzida pela solução foi comparada, em termos de utilidade, com a arquitetura implementada. O resultado foi que a arquitetura produzida correspondia aos requisitos impostos, pelo que esta foi aprovada pelos especialistas do domínio do caso de estudo, validando a solução.

Palavras-chave – Plataformas IIoT; Arquiteturas IIoT Semânticas; Ontologia; Transformação Digital.

Abstract

IIoT (Industrial Internet-of-Things) platforms are an enabler for the digital transformation in the scope of industry 4.0, promoting flexibility for a faster adjustment to market and allowing organisations to have a clear vision over its current status. However, SMEs (Small and Medium Enterprises) are struggling to shift into this new paradigm, due to the lack of: i) qualified resources needed to develop and implement their digitalization solutions; ii) a clear understanding about the digitalisation reengineering related IIoT adoption, and; iii) suitable models for SME-oriented IIoT solutions specification. In order to overcome these challenges, a solution is discussed within a twofold perspective: i) on one hand, it looks for the automation of the specification of IIoT platform's architectures, according to the specific business needs, reducing the investment needed to develop these kind of I4.0 digital enablers, and; ii) on the other hand, it fosters a shared understanding of the IIoT between the domain expert's and the organisations, promoting the involvement of both parties in this specification process. In this context, semantics plays an impacting role, allowing the accommodation of the multidisciplinary knowledge of IIoT architectures in a semantic model leveraged by reasoning capabilities. This solution was evaluated in a case study, where the architecture produced by the solution was compared, utility wise, with the implemented architecture. The result was that the produced architecture matched the imposed requirements, and so, it was approved by the case study's domain experts, validating the solution.

Keywords – IIoT Platform; Semantic IIoT Architectures; Ontology; Digital Transformation.

Índice

Lista de Tabelas

Lista de Figuras

Glossário

- API – Application Programming Interface;
- BI – Business Intelligence;
- ERP – Enterprise Resource Planning;
- FAMEST – Footwear, Advanced Materials, Equipments and Software Technologies;
- I4.0 – Indústria 4.0;
- IIoT – Industrial Internet-of-Things;
- IoT – Internet-of-Things;
- IT – Information Technology;
- JSON – JavaScript Object Notation;
- MQTT – Message Queuing Telemetry Transport;
- OPC – OLE (object linking and embedding) for Process Control;
- OPC UA – OPC Unified Architecture;
- OT – Operational Technology;
- OWL – Ontology Web Language;
- PME – Pequena ou Média Empresa;
- RAMI4.0 – Reference architectural Model Industrie 4.0;
- RDF – Resource Description Framework;
- REST – Representational State Transfer;
- XML-RPC – Extensible Markup Language - Remote Procedure Call;

1. Introdução

1.1. Motivação

O novo paradigma industrial associado à quarta revolução industrial, denominada de indústria 4.0 (I4.0), coloca a orientação para o cliente e a reengenharia de processos para a transformação digital na indústria, como objetivos estratégicos organizacionais, oferecendo maior eficiência e flexibilidade produtiva [1], [2].

A definição de transformação digital, no contexto deste trabalho, refere-se à abordagem sistemática na implementação de estratégias inerentes ao paradigma da indústria 4.0, tipicamente conduzidas por tecnologias de informação e comunicação, para o processamento de dados, referentes a processos, produtos e pessoas, em tempo real e de forma mais inteligente. As iniciativas de transformação digital, incluem atividades de inovação, otimização e automatização, desenvolvidas de acordo com os princípios da indústria 4.0 [1]–[6]. Apesar de, teoricamente consistente e exequível, a implementação de estratégias de transformação digital é um desafio para as empresas em geral, mas em particular para as PMEs [2], [7], [8]. Tipicamente, as PMEs são as que mais podem beneficiar com este novo paradigma da I4.0, mas também são aquelas que estão menos preparadas para o adotar com sucesso [2], [6], [9], [10]. Na génese da sua impreparação está, acima de tudo, a falta de metodologias adaptadas/customizadas à realidade das pequenas empresas, de forma a acrescentar valor ao negócio [2]. Neste contexto, o conceito de *Internet-of-Things* (IoT) emergiu para os ambientes industriais, potenciando a implementação da I4.0 [2], [4], [6], [7], [9]–[15]. Atualmente, esta será uma das mais conhecidas estratégias de transformação digital [4], [14]. Concretamente, uma abordagem IoT em ambiente industrial é um meio para alcançar a integração vertical intraempresarial, permitindo monitorizar todos os ativos industriais e de negócio em tempo real, oferecendo uma visão clara do estado atual da organização, promovendo um ajuste rápido às mudanças do mercado, e , consequentemente, maior flexibilidade [1], [2], [7], [15], [16].

Estas soluções de integração vertical, denominadas de plataformas *Industrial Internet-Of-Things* (IIoT), são focadas na interoperabilidade e na integração de sistemas e dispositivos [4], [17], [18], e a sua adoção tornou-se uma tendência entre as empresas [4], [16]. No entanto, estas soluções tecnológicas não são uma *silver bullet* no contexto da transformação digital [14]. Dada a sua complexidade tecnológica, é bastante comum estas plataformas estarem desalinhadas com as necessidades reais das organizações. Isto acontece quer devido a uma especificação em demasia, onde é dada demasiada atenção à vertente tecnológica, ou devido a uma especificação insuficiente, tendo em conta a falta de compreensão por parte das empresas do valor de negócio real das plataformas IIoT [4], [14].

Tipicamente, uma arquitetura IIoT é composta por um grande conjunto de componentes que permitem a troca de dados dentro das organizações, bem como para fornecer e consumir dados de ambientes externos (por exemplo, outros sistemas e serviços) [17]. Dentro deste aglomerado de componentes, a escolha dos mais adequados para uma necessidade empresarial específica pode não ser uma tarefa fácil. No contexto das PMEs este aspeto é crítico para a sua jornada de transformação digital, sendo que estas necessitam de soluções customizadas às necessidades específicas de cada empresa [2]. Esta customização implica não só a compreensão das necessidades do negócio, mas também das principais características dos componentes, das suas capacidades de interoperabilidade, e dos seus protocolos de comunicação [19]. Isto faz com que o processo de especificação das plataformas IIoT seja complexo, demorado e consequentemente, caro [20], dificultando assim adoção destas soluções por parte das PMEs [21].

Para além disso, o sucesso das plataformas IIoT depende, não só do alinhamento correto com as necessidades empresariais, mas também da compreensão clara dos conceitos e das capacidades instaladas, e de que forma é que estes acrescentam valor ao negócio [4], [14], [21]. No entanto isto nem sempre é claro para as empresas, em particular nas PMEs [2], [8], [10], [22].

As arquiteturas IIoT são complexas e extensas, caracterizadas pela sua modularidade, escalabilidade e por serem distribuídas e interoperáveis [17], [23], [24]. Se por um lado estas soluções tecnológicas atingiram um interessante estado de maturidade técnico-científico, assistimos também a uma proliferação de plataformas e tecnologias que dificultam o processo de identificação e seleção dos artefactos mais adequados às necessidades empresariais [8], [22], [25]. Adicionalmente, é comum encontrarmos implementações de arquiteturas IIoT que se desviam dos modelos de referência existentes, dificultando as atividades de suporte e integração [21]. Por fim, assistimos à apresentação, por parte dos grandes *players* tecnológicos (Google, IBM, Microsoft, Amazon), do desenvolvimento das suas próprias arquiteturas e modelos [25]–[27]. Tudo isto, são ingredientes que condicionam a adoção de tecnologias IIoT pelas PMEs [2], [8]. Reconhecendo o valor estratégico destas soluções para a competitividade das PMEs, urge revisitar o domínio técnico-científico do IIoT do ponto de vista destas empresas, (re)organizando o conhecimento por forma a que este possa ser interpretado e consumido pelas mesmas no decurso das suas iniciativas/projetos de transformação digital. Esta perspetiva requer uma abordagem semântica para a representação do conhecimento útil, para a especificação assistida de arquiteturas IIoT, orientadas para as PMEs [28]–[30]. Em conformidade, este trabalho, seguiu pressupostos semânticos no sentido de acomodar os conhecimentos multidisciplinares das arquiteturas IIoT num artefacto tecnológico. Este artefacto estaria ao serviço tanto dos especialistas do domínio IIoT como do negócio, e permitindo a articulação, de

forma automática [31], dos vários conceitos do domínio em soluções específicas para cada empresa.

1.2. Objetivos

O principal objetivo desta dissertação é a criação de um modelo semântico que, por um lado, simplifique e organize os conceitos inerentes às arquiteturas IIoT, com o intuito de garantir um entendimento partilhado dos mesmos entre os especialistas do domínio e as empresas. E, por outro, permita a automatização do processo de especificação das plataformas IIoT, através da articulação das necessidades, com os componentes e as capacidades instaladas, numa arquitetura específica às necessidades de negócio de cada empresa. Este artefacto, alicerçado no conhecimento, será estruturado por um repositório baseado em híper-grafos, apoiando a especificação de arquiteturas IIoT, e, oferecendo capacidades de raciocínio através de um motor de inferência. Um segundo objetivo é a avaliação da implementação deste modelo utilizando híper-grafos como formalismo para a representação do conhecimento.

Concretamente, os objetivos a alcançar com o modelo desenvolvido no âmbito desta dissertação passam por:

- Partilhar conhecimento sobre plataformas IIoT entre os especialistas do domínio e as empresas;
- Alinhar os conceitos das plataformas IIoT com as necessidades empresariais;
- Reduzir o tempo de especificação de uma plataforma IIoT;
- Mitigar o risco de uma especificação em demasia ou insuficiente das plataformas IIoT;
- Fomentar a customização modular destas plataformas;
- Agilizar os processos de desenvolvimento das plataformas IIoT.

Como discutido na secção da motivação, as PMEs são as que poderão tirar maior partido da I4.0 [2]. Por esta razão, entende-se que o modelo proposto será de maior utilidade para estas empresas, como um facilitador para a adoção das plataformas IIoT de forma a alcançar a integração vertical. No entanto, este esforço técnico-científico é mais amplo que apenas na aplicação específica no domínio das PMEs, podendo também ser estendido a aplicações em empresas de maior dimensão.

1.3. Resultados

Os resultados obtidos nesta dissertação passam pela sistematização dos *digital enablers* (componentes, tecnologias e protocolos) no âmbito da integração vertical, em particular no domínio das plataformas IIoT. A conceptualização do domínio do IIoT, envolvendo os especialistas deste domínio, de forma a desmistificar e simplificar os conceitos inerentes às arquiteturas IIoT. A partir da conceptualização foi articulado um modelo semântico, sendo este

formalizado em híper-grafos e implementado num protótipo para o apoio e automatização à especificação de plataformas IIoT. Este protótipo foi experimentado no âmbito de um caso de estudo e a utilidade dos resultados obtidos foi validada pelos especialistas do domínio.

Consoante as lacunas identificadas no estado de arte, estes resultados constituem um incremento à base de conhecimento científica atual, no âmbito da descrição de arquiteturas IIoT. O protótipo desenvolvido cumpre os objetivos propostos, permitindo a automatização do processo de especificação de plataformas IIoT, segundo as necessidades de negócio específicas de cada empresa.

1.4. Estrutura do Documento

Este documento encontra-se dividido nas seguintes secções:

- **Introdução**: a presente secção inicia-se com a motivação desta dissertação no contexto do IIoT e das PMEs, de seguida são apresentados os objetivos específicos a alcançar com a mesma, terminando com os resultados obtidos;
- **A jornada digital das PMEs na adoção de soluções IIoT**: apresenta a revisão da literatura dos desafios no IIoT no contexto das PMEs e que forma estes podem ser minimizados pelo modelo proposto nesta dissertação; De seguida são elencadas as aplicações típicas de semântica no IIoT, segundo a literatura, sendo estas classificadas segundo objetivo da sua aplicação e comparadas com o modelo proposto; Esta termina com as conclusões da revisão do estado de arte, onde inclui as lacunas, identificadas na literatura, a colmatar com esta dissertação;
- **Abordagem Socio-Semântica para especificação de arquiteturas IIoT**: corresponde à secção mais extensa deste documento; Esta inicia-se com a abordagem e a metodologia de trabalho utilizada no desenvolvimento do modelo semântico; De seguida, descreve-se o processo de conceptualização deste modelo, que inclui a articulação do modelo de referência RAMI4.0 numa arquitetura conceptual de plataformas IIoT; Posteriormente, são apresentados os conceitos que formam o pilar conceptual das arquiteturas IIoT, o método utilizado para a validação dos mesmos e a descrição da arquitetura *baseline*; Para além disto, é também apresentada a implementação da solução e dos componentes que a constituem, abordando em mais detalhe a tecnologia utilizada para implementação do modelo semântico e o seu formalismo de representação de dados; Seguidamente, detalha-se a implementação dos conceitos do modelo, das regras e restrições do domínio e a forma como decorre automatização do processo de especificação das arquiteturas IIoT; Esta secção termina com as conclusões e as limitações encontradas durante o desenvolvimento do protótipo da solução;

- **Caso de Estudo**: esta secção principia-se com a descrição do caso de estudo, inserido no âmbito do projeto FAMEST, para a validação da utilidade do modelo por um conjunto de especialistas do domínio; Incluí também a exposição da arquitetura da plataforma IIoT, utilizada neste projeto, e a metodologia de avaliação da arquitetura IIoT, gerada pela solução, para este caso de estudo; Esta termina com a discussão de resultados e a avaliação do protótipo da solução;
- **Conclusão**: esta secção procura fazer uma reflexão crítica dos resultados obtidos face aos objetivos propostos inicialmente; São também descritos os contributos científicos obtidos por esta dissertação e definidos futuros melhoramentos e projetos que possam derivar deste modelo semântico.

2. A jornada digital das PMEs na adoção de soluções IIoT

Neste capítulo é apresentada a revisão da literatura que visa verificar a relevância científica da dissertação. Aqui são explorados os desafios enfrentados pelas PMEs na implementação do IIoT, e que forma é que estes são endereçados pelo modelo proposto. De seguida, é feita uma revisão do estado de arte das aplicações típicas de semântica no domínio do IIoT, sendo estas classificadas e enquadradas com o modelo semântico.

2.1. Desafios na implementação do IIoT

Foi conduzida uma revisão da literatura dos principais desafios da implementação do IIoT, no contexto da transformação digital nas PMEs. De estes, foram identificados 6 desafios cujo modelo proposto neste documento contribui para a minimização do obstáculo que estes impõem no processo de digitalização das empresas. Os desafios identificados são:

- Necessidade de Investimentos Elevados;
- Falta de mão-de-obra qualificada;
- Falta de Standardização;
- Falta de integração e capacidade de interoperabilidade;
- Falta de estratégia clara para a digitalização;
- Falta de conhecimento e transparência acerca do IIoT.

Necessidade de Investimentos Elevados

Este refere-se ao elevado investimento que é necessário na implementação de tecnologias de suporte à indústria 4.0, como soluções de integração vertical, e nos recursos e infraestruturas subjacentes [2], [8]–[10], [13], [21], [22], [32]–[35]. Enquanto este desafio não se manifesta de forma regular nas empresas de maior dimensão, nas PMEs este é particularmente impeditivo no seu processo de digitalização [2], [8], [22]. Nestas empresas, a capacidade de investir nos recursos necessários para explorar estas tecnologias, e perceberem quais é que são efetivamente relevantes para o seu negócio, é que dita a sua maturidade na implementação da indústria 4.0 [8], [22], [33].

O modelo proposto tem como objetivo reduzir o tempo de especificação das plataformas IIoT, sendo estas uma das soluções de integração vertical que permitem a digitalização das empresas. Reduzindo o tempo que demora a especificar uma plataforma, diminuímos o custo final para as empresas, sendo necessário um menor investimento. Para além disto, este modelo também automatiza a seleção das tecnologias mais relevantes, a implementar a arquitetura específica de cada empresa, reduzindo o investimento necessário para a exploração destas tecnologias.

Falta de mão-de-obra qualificada

A especificação e implementação de soluções para a digitalização das empresas envolve conhecimento multidisciplinar. Isto faz com que seja essencial existirem recursos humanos com as qualificações, as *skills* e o *mindset* necessários não só para o desenvolvimento, mas também para a interação com este tipo de soluções. Este é considerado um dos desafios enfrentados pelas empresas no seu processo de digitalização [2], [8]–[11], [13], [21], [22], [32], [34]–[37]. Este faz com que surjam novos requisitos para os trabalhadores, havendo um foco nas *skills* tecnológicas, o que poderá ser um entrave para os recursos humanos existentes. Isto leva as empresas a terem que fazer investimentos na formação destes recursos humanos e/ou a contratação de novos trabalhadores. No entanto, existe falta de mão-de-obra qualificada no mercado, dificultando esta contratação [8], [9], [13], [34], [37]. No caso concreto das PMEs isto é agravado porque estas competem com outras PMEs e empresas de maior dimensão na contratação destes recursos humanos [2], [8], [22].

Com este modelo, espera-se abstrair esta falta de conhecimento do domínio do IIoT na especificação das arquiteturas, por parte das empresas e dos seus trabalhadores. A escolha dos componentes, das tecnologias e dos protocolos a utilizar na arquitetura são, em grande parte, alavancadas pelo modelo com base nas necessidades de negócio da empresa. Desta forma, reduzindo a necessidade de mão-de-obra qualificada por parte das empresas na especificação e implementação das plataformas IIoT.

Falta de Standardização

Este desafio é visto de duas formas. Do ponto de vista das arquiteturas IIoT, e do próprio processo de digitalização, em que não existe uma arquitetura standardizada para a implementação destas soluções de integração vertical, criando ambiguidades nas empresas [13], [21]. E isto, como afirmado em [21], é uma barreira significativa para o sucesso da adoção de arquiteturas eficientes, ou seja, adequada às necessidades reais das empresas.

Um segundo ponto de vista é a falta de standardização da comunicação e troca de dados entre sistemas e dispositivos [2], [8], [9], [13], [22], [32], [33]. Neste sentido, o OPC-UA contribuí para a universalização da comunicação entre dispositivos no chão-de-fábrica e outros sistemas e aplicações. No entanto, ainda não existem standards, que sejam claros e acordados a nível europeu ou internacional, para este efeito [9], [13]. As PMEs são particularmente afetadas por este desafio, dado o risco de investir no standard “errado”. Estas acabam por tipicamente implementar o standard utilizado pelas empresas de maior dimensão com qual interagem, com o intuito de facilitar a troca de informação entre ambas as partes [2], [8].

Neste sentido, o modelo proposto alinha as arquiteturas especificadas com os standards, as arquiteturas de referência e a literatura no âmbito das arquiteturas IIoT. A nível da comunicação entre os dispositivos e os sistemas empresariais, a própria plataforma IIoT tem como intuito ser um facilitador na integração, possibilitando a troca e a transformação de dados entre os mesmos, selecionando os protocolos de comunicação mais adequados durante a especificação.

Falta de integração e capacidade de interoperabilidade

Este desafio revela-se quase exclusivamente nas PMEs, em que os seus dispositivos e máquinas são muitas vezes de diferentes fornecedores, utilizando cada um diferentes protocolos de comunicação, que podem ser proprietários ou *legacy* [8], [10], [13], [21], [38]. Para além disso, por vezes existem máquinas de eras diferentes, em que é necessário fazer um *retrofit* de uma máquina para esta ser compatível com o paradigma da i4.0. Estas circunstâncias criam um panorama heterogéneo no chão-de-fábrica, dificultando a integração destes dispositivos com outros sistemas ou até mesmo entre si [8], [10], [21], [38].

Este pode ser minimizado pela utilização de plataformas IIoT. Estas são um veículo para a interoperabilidade entre dispositivos heterogéneos, permitindo a conexão com estes dispositivos suportando diferentes protocolos de comunicação [7]. No entanto, a plataforma só consegue assegurar a interoperabilidade se for possível comunicar com estas máquinas, se tal não for possível, o *retrofit* das mesmas é inevitável.

Falta de estratégia clara para a digitalização

O processo de digitalização visa permitir às empresas aumentar a sua capacidade produtiva reduzindo os custos. No entanto, é necessário existir uma estratégia clara para o sucesso da digitalização [2], [8]–[11], [13], [22], [32], [33], [35], [37]. Este envolve a alocação de tempo e recursos, sendo este uma tarefa extensiva que inclui a restruturação do processo e da própria empresa para se adaptar a este novo paradigma, e por isso, deveria envolver planeamento substancial a médio/longo prazo [8], [9], [33], [37]. Esta restruturação acaba por ser um problema mais comum no caso das empresas de maior dimensão, dada a complexidade das suas estruturas, do que nas PMEs. Tipicamente, as PMEs conseguem implementar esta digitalização de forma mais rápida, dada a sua flexibilidade na implementação das estruturas IT. No entanto, estas empresas não são imunes a este desafio [2], [9], [10].

Uma das razões pelas quais não são preparadas estratégias para a digitalização das empresas, é a falta de suporte da gestão de topo. Isto pode ser pelo facto de não terem o conhecimento suficiente sobre a indústria 4.0, e/ou os benefícios concretos que esta pode trazer para o seu o negócio [13], [22], [33]. Outra das razões poderá ser por existir um foco, por parte da gestão, na parte operacional e não no desenvolvimento da empresa, em que é descrito em [33] como uma gestão a curto prazo, uma cultura de inibição por parte da gestão. Em [37] é apresentado um

caso semelhante ao anterior, em que a gestão tem como prioridade a digitalização, no entanto, não veem este processo como uma forma de obter lucros. Estes são referidos como investimentos defensivos, que visam proteger o negócio, em vez de desenvolver o seu futuro. Este comportamento está associado ao conceito de *choice overload*, em que quando existem vários caminhos por quais se pode optar, por vezes acaba-se por não se seguir por nenhum [37]. Isto impede o investimento em soluções de integração vertical, o que facilitaria o processo de digitalização das empresas [13].

Com o modelo proposto pretende-se consciencializar as empresas para as necessidades de negócio que podem ser colmatadas através do IIoT, e de que forma é que isto é alcançado, e assim clarificar os benefícios que a digitalização pode trazer para estas empresas. Para além disso, este modelo tem como o intuito a redução do *choice overload*, através da seleção dos componentes e das tecnologias, que devem constituir a arquitetura IIoT, com base nas necessidades específicas de cada empresa.

Falta de conhecimento e transparência acerca do IIoT

O domínio do IIoT é um pântano conceptual, que inclui tecnologias, protocolos, equipamentos e sistemas. O entendimento destes conceitos, e de que forma estes trazem valor para o negócio, nem sempre é claro para as empresas, sendo particularmente impactante nas PMEs [2], [8], [10], [22]. A falta de conhecimento do domínio do IIoT, e no entendimento dos seus conceitos, é um dos desafios a ultrapassar para o sucesso da adoção das soluções de integração vertical [8], [13], [21], [22], [32], [33]. Isto leva a que as empresas não tenham conhecimento das diferentes soluções e tecnologias existentes para a sua digitalização, e quais é que são as mais adequadas para o seu caso de uso [8], [22]. Esta falta de conhecimento reflete-se nos vários níveis organizacionais, desde o nível mais estratégico ao operacional, sendo um entrave para alcançar integração vertical [8], [10], [22], [32], [33].

Ultrapassar este desafio é crítico para as empresas, sendo afirmado em [13], que é urgente a existência de uma maior transparência e partilha de experiência do domínio do IIoT. Em [22] é realçado, por um dos participantes do inquérito, que é necessário existir uma visão holística destes conceitos, abordada da perspetiva do negócio, e que permita a partilha do conhecimento entre os especialistas do domínio e as empresas.

Neste contexto, a semântica é o próximo passo para a criação deste entendimento claro e partilhado, acerca do IIoT, sendo o foco deste modelo ultrapassar este desafio. Isto seria alcançado através da organização e a desmistificação dos conceitos envolvidos no IIoT, e de que forma é que estes ajudam a colmatar as necessidades específicas de negócio das empresas. Desta forma é possível envolver os especialistas do domínio e as empresas na especificação das

arquiteturas IIoT, e, clarificar os benefícios e valor que estas soluções de integração vertical acrescentam ao negócio.

Considerações Finais

Finalizando a revisão da literatura, entende-se que o artefacto semântico proposto nesta dissertação endereça pelo menos estes 6 desafios que as PMEs encontram na adoção e implementação do IIoT durante o seu processo de digitalização. De estes desafios, este modelo tem como principal foco a partilha do conhecimento do domínio do IIoT entre os especialistas do domínio e as empresas. A semântica é a chave para alcançar este entendimento partilhado, e por essa razão foi conduzida uma revisão do estado de arte das aplicações típicas de semântica no IIoT, apresentada na subsecção seguinte.

2.2. Soluções IIoT baseadas em princípios semânticos

O paradigma da indústria 4.0 impulsionou a transformação digital nas empresas e com ela sistemas cada vez mais complexos, abrangentes e holísticos, em que a tecnologia é apenas uma parte da equação. Estes sistemas, tipicamente sociotécnicos [39], [40], são compostos por múltiplas partes, elementos ou componentes e pessoas ou grupos, conectados entre si de forma mais ou menos complexa, e cuja interação requer mecanismos de estruturação e orquestração cada vez mais sofisticados [41].

As soluções IIoT integram este tipo de sistemas complexos, sendo estas constituídas por um grande conjunto de componentes, assentes nos conceitos e tecnologias subjacentes ao IIoT, centrados nos utilizadores e na utilidade da informação, que interagem entre si de forma a interligar as várias camadas empresariais [4], [14], [17], [24]. Estas soluções são focadas em alcançar a integração vertical como forma de ultrapassar o desafio da digitalização das empresas [4], [17], [18].

Para lidar com este tipo de sistemas é fundamental não só a descrição dos vários elementos que estes envolvem e de que forma é que estes se relacionam, mas também o estabelecimento semiautomático de novas relações que possam incrementar a interpretabilidade da informação oferecida por este tipo de sistemas [41]. Neste contexto, a semântica poderá ter um papel preponderante na representação dos sistemas complexos.

O recurso a modelos semânticos, entre os quais, ontologias, permitem a definição dos conceitos e as suas respetivas relações que caracterizam um domínio específico [30], [42]–[44]. Estes modelos são utilizados, por um lado, na integração de conhecimento, através da inferência de novo conhecimento, ou, como forma de desambiguar conceitos, visando a criação de um entendimento partilhado entre os vários agentes intervenientes [28], [42], [43]. Por outro lado,

estes podem ser utilizados como forma de estruturar e organizar o conhecimento, como por exemplo, através da formalização da experiência dos especialistas do domínio [43], [44].

Um exemplo de representação de um sistema complexo através de modelos semânticos é o caso do BioGrakn[1]. Este exemplo insere-se na área da biomédica, em que é utilizada uma ontologia para descrição de genes e proteínas e de que forma é que estes interagem ao nível metabólico em diferentes tipos de cancro. A utilização de uma ontologia neste caso permitiu não só a estruturação e integração do conhecimento do domínio, disperso por diversas fontes de dados, mas também como forma de garantir uma uniformização do significado/entendimento destes conceitos [45].

A extensão da semântica ao domínio do IIoT entende-se como um meio para a formalização da estrutura destas soluções, ou seja, a sua arquitetura, composta pelos componentes e as respetivas ligações, num artefacto semântico. Este seria um veículo para a desmistificação e a organização dos conceitos do IIoT, algo atestado na secção anterior como um desafio no processo de transformação digital das empresas, em particular nas PMEs.

Neste sentido, foi averiguado, segundo a literatura, quais são as aplicações típicas de semântica no IIoT. Para isso, foi conduzida uma pesquisa, na base de dados bibliográfica Scopus[2], em que foram utilizadas uma combinação de termos para a realização da mesma. As combinações utilizadas foram: "*industrial IoT"* e "*ontology"*, e a combinação seguinte, *"industrial IoT"* e "*semantics"*. Foram também utilizadas outras combinações de termos, mas os resultados obtidos eram irrelevantes para a pesquisa. Dos resultados obtidos, e após uma análise realizada pelos autores, foram selecionados 18 artigos que evidenciavam a aplicação de semântica em casos de uso industriais.

Os artigos selecionados foram depois classificados segundo os tipos de aplicações semânticas que estes demonstram. As classificações utilizadas são baseadas no trabalho de [46], em que é feito um levantamento dos diferentes cenários que demonstram a importância da semântica no IoT. Estes cenários foram agregados e generalizados em três objetivos que estas aplicações, a nível semântico, pretendem alcançar, nomeadamente, interoperabilidade, transformação de dados e conceptualização do domínio do IIoT, sendo estas usadas para a classificação dos artigos. As descrições de cada uma das aplicações de semântica, assim como a classificação dos artigos selecionados segundo as mesmas, são apresentadas na Tabela 1.

O trabalho proposto neste artigo, é classificado como conceptualização do domínio do IIoT, em que são descritas as entidades do domínio das arquiteturas de plataformas IIoT, nomeadamente,

[1] https://github.com/vaticle/biograkn

[2] https://www.scopus.com/

os componentes da arquitetura, as funções disponibilizadas pela plataforma, e as necessidades que levaram à sua especificação. Esta modelação de alto-nível, mais próxima da visão dos utilizadores, dos conceitos associados às arquiteturas IIoT, e a sua posterior formalização, possibilita a automatização do processo de especificação das plataformas IIoT.

Com o intuito de encontrar semelhanças entre os artigos de conceptualização do domínio do IIoT e o trabalho proposto, foi conduzida uma análise comparativa, mais detalhada, dos mesmos, sendo esta apresentada de seguida.

Aplicação de Semântica	Descrição	Artigos
Interoperabilidade	Possibilitar a comunicação e a partilha de dados entre *things* heterogéneas. **Exemplo:** Uso de uma *gateway* semântica para uniformizar a comunicação com dispositivos que usem diferentes protocolos de comunicação.	[47]–[57]
Transformação de Dados	Conversão e integração de dados heterogéneos, e ainda, o uso de anotações semânticas para o enriquecimento e a contextualização de dados. **Exemplo:** Adição de uma variável, aos dados de telemetria de um sensor de temperatura, com a unidade SI utilizada no local de medição.	[47], [48], [50]–[52], [55], [58]–[61]
Conceptualização do Domínio do IIoT	Modelação semântica de entidades e conceitos do domínio, viabilizando interações de alto-nível entre *things* a partir da inferência e interpretação de conhecimento. **Exemplo:** Criação de uma ontologia para a descrição das características, das funções, e do contexto dos dispositivos IoT.	[47]–[49], [54], [57], [61]–[64]

Tabela 1 Classificação de artigos segundo o tipo de aplicação semântica

Nos trabalhos [48], [54], [61], são focados no caso de uso de descoberta e registo de dispositivos ou serviços, através da especificação das características e capacidades destas entidades. O intuito deste caso de uso é de facilitar a integração destes dispositivos ou serviços em sistemas e promover a utilização das suas capacidades. No entanto, estes trabalhos não consideram o que está a montante e a jusante dos dispositivos e serviços, ou seja, as estruturas de componentes que compõem a arquitetura do sistema, e definem as suas funcionalidades.

Em [57] é apresentada uma ferramenta para acelerar o desenvolvimento de aplicações IoT interoperáveis com dispositivos heterogéneos. Isto é alcançado através do mapeamento semântico, de forma automática, entre as *skills* necessárias por parte da aplicação e os dispositivos que implementam essas *skills*. Este trabalho assemelha-se ao proposto no mapeamento semântico, em que este articula as necessidades específicas de uma empresa com os componentes que colmatam estas necessidades através do conceito de funções. Esta articulação é semelhante ao que é feito neste trabalho com o mapeamento das *skills* entre as aplicações e os dispositivos.

No trabalho [64] é apresentado um *knowledge graph* que visa formalizar e partilhar o conhecimento acerca das falhas de várias máquinas, espalhadas pelo mundo, entre os operadores, os técnicos e os fornecedores das mesmas. O único ponto comum entre este trabalho e o trabalho proposto é no uso de semântica para a criação de um entendimento partilhado entre os vários atores do domínio.

Em [47] é proposto um motor de semântica cujo o foco é a integração com diferentes dispositivos, e o enriquecimento e interpretação dos dados recolhidos, segundo as necessidades dos utilizadores. Em comparação com o trabalho proposto, apesar da partilha da orientação às necessidades dos utilizadores, este foca-se fundamentalmente nos dados e na inferência dos mesmos.

No trabalho [63] é apresentada uma ferramenta de modelação de requisitos e necessidades a nível semântico para apoiar a decisão no desenho de *layouts* de chão-de-fábrica. Este trabalho encontra-se a um nível diferente do trabalho proposto, e por essa razão estes não podem ser comparados de forma direta. O trabalho analisado está ao nível das tecnologias operacionais (OT), enquanto o trabalho proposto apresenta-se ao nível das tecnologias de informação (IT). No entanto, a modelação de necessidades é um ponto comum entre os dois trabalhos, sendo que no trabalho proposto esta modelação não é apenas para apoiar a decisão no processo, mas sim para o orientar e automatizar.

Em [62], é utilizada modelação semântica para descrever um conjunto de tarefas a desempenhar por diferentes dispositivos. Esta modelação possibilita a realização das tarefas de forma automática, sendo esta a única semelhança com o trabalho proposto neste artigo.

Por fim, em [49], é apresentada uma arquitetura que permite a integração dinâmica de componentes (sistemas e dispositivos) de forma automática, através da especificação das suas interfaces de comunicação. O trabalho proposto assemelha-se a este, no conceito de integração automática de componentes, que é algo fulcral na especificação de arquiteturas IIoT. No entanto, o trabalho analisado fica-se apenas pela descrição das interfaces dos componentes, não

fazendo uma descrição mais detalhada desses componentes, das suas funcionalidades e das necessidades que levaram à sua instanciação.

Considerações Finais

Concluindo a análise do estado de arte, no domínio do tema proposto, é possível entender que o âmbito da aplicação de semântica no domínio do IIoT tem sido muito focado na criação de interoperabilidade entre sistemas, em 11 dos 18 artigos analisados, e na transformação de dados, em 10 dos 18 artigos analisados. Existe pouco foco na descrição de arquiteturas no domínio do IIoT, apenas em 1 dos 18 artigos analisados, e nas que existe esse foco, em [49], a descrição é pobre, tornando-a limitada no seu âmbito.

2.3. Conclusões

Na revisão da literatura, foi identificado que a falta de conhecimento sobre o domínio do IIoT efetivamente é um desafio para as empresas, em especial as PMEs. Neste sentido, a semântica poderá desempenhar um papel fundamental permitindo a criação de um artefacto semântico, que esteja ao serviço dos especialistas do domínio e das empresas, fomentando a partilha do conhecimento e da experiência do domínio. No entanto, as aplicações de semântica no IIoT não têm sido neste sentido, havendo um foco predominante no uso de semântica para a interoperabilidade e a transformação de dados. Ao que se pôde apurar, na revisão do estado de arte, a utilização de semântica para descrição de arquiteturas ainda é escassa no IIoT, sendo que, na opinião do autor, é onde se pode retirar maior valor da semântica. A arquitetura é o que permite fazer a mediação entre a parte mais técnica, envolvendo as tecnologias e as suas capacidades, e a parte de negócio, contendo as necessidades específicas das organizações. Desta forma, é possível a criação de um modelo que abstraia a parte técnica da arquitetura, e que seja focado no valor acrescentado ao negócio que esta traz.

Após a revisão do estado de arte, entende-se que existe uma lacuna na aplicação de semântica ao nível das arquiteturas IIoT. Por esta razão, considera-se o modelo proposto relevante do ponto de vista científico, sendo que o cariz inovador é a descrição semântica dos conceitos subjacentes às arquiteturas IIoT. Este modelo, orientado às necessidades específicas das organizações, possibilita a automatização do processo de especificação de plataformas IIoT, facilitando a adoção deste tipo de soluções de integração vertical por parte das PMEs, e assim fomentar a sua transformação digital.

3. Abordagem Socio-Semântica para especificação de arquiteturas IIoT

O desenvolvimento do modelo proposto nesta dissertação seguiu uma abordagem socio-semântica [65] na criação de representações de conhecimento, semanticamente enriquecidos através de interações sociais com especialistas do domínio da IIoT.

A fundamentação socio-semântica advém da combinação da experiência dos especialistas do domínio e do conhecimento explícito, específico do domínio técnico-científico da IIoT. Concretamente, esta abordagem foca o processo extração e formalização do conhecimento através da sistematização e definição dos conceitos e tecnologias emergentes da IIoT, articulados num único artefacto semântico.

O modelo resultante ambiciona reduzir a distância entre as empresas e os especialistas de domínio no decorrer do processo de especificação de plataformas IIoT, fomentando a partilha de conhecimento entre ambas as partes e seu o envolvimento na customização/adaptação das plataformas IIoT às necessidades específicas da(s) empresa(s). Esta abordagem, permitirá incrementar a utilidade do resultado da especificação. A abordagem prevê a implementação do modelo num motor de suporte à automatização da especificação de plataformas IIoT, adequadas à realidade de cada empresa.

No decorrer desta secção será abordada a metodologia que esteve na base do desenvolvimento deste trabalho e, consequentemente, no desenvolvimento do modelo semântico. Descreve-se ainda em detalhe o modelo e o seu processo de conceptualização e formalização. A secção termina com uma breve reflexão sobre o modelo desenvolvido.

3.1. Metodologia

A metodologia de desenvolvimento do modelo semântico, apresentada na Figura 1, está dividida em 3 Fases:

- Fase I: verificação da relevância do problema através da revisão da literatura, identificando os principais desafios associados à aplicação de IIoT nas PMEs e, adicionalmente, a análise do estado da arte no que concerne à aplicação de semântica no domínio do IIoT;
- Fase II: conceptualização do modelo, através de uma abordagem iterativa, onde o modelo conceptual foi mapeado com as arquiteturas de referência, nomeadamente o RAMI4.0, e validado recorrendo a *templates* semânticos;
- Fase III: formalização do modelo recorrendo a híper-grafos e a sua utilização num caso de estudo.

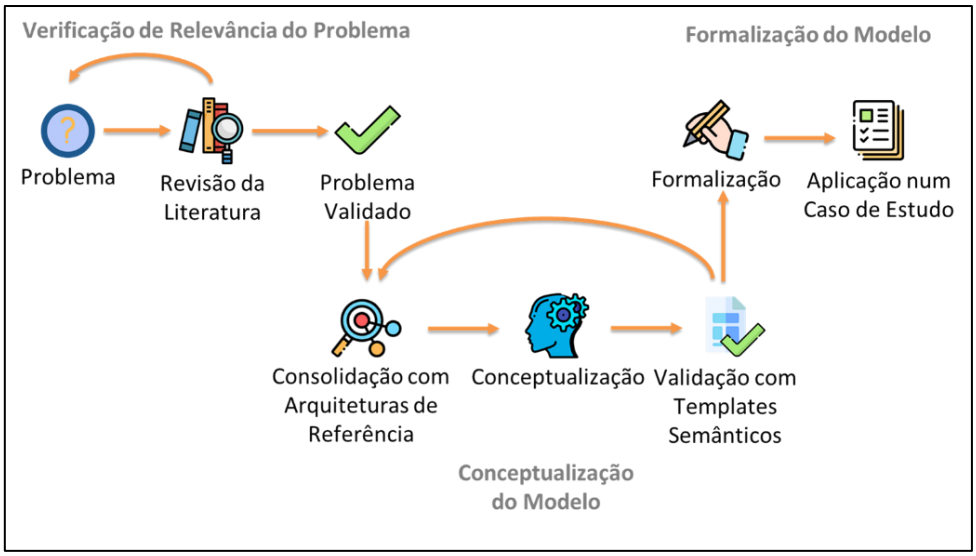

Figura 1 Metodologia de Trabalho

Cada fase da metodologia ocorreu de forma iterativa, até à validação dos objetivos identificados no início de cada fase. Durante uma fase, é possível regressar à fase anterior para ajustes e correções necessárias.

A base lógica para progressão do trabalho segundo a metodologia representada na Figura 1 está sistematizada na Tabela 2.

A Tabela 2 representa não só os critérios associados à metodologia de trabalho, mas constituiu uma base fundamental para controlo e monitorização do mesmo. No âmbito desta metodologia há que destacar que as fases II e III, obedeceram a algumas dinâmicas colaborativas, nomeadamente, durante o processo de conceptualização e no caso de estudo, com o intuito do enriquecimento da utilidade do modelo proposto.

Concretamente, e a fim de alcançar uma visão comum sobre as arquiteturas IIoT, foi conduzido um processo de construção colaborativa de conhecimento envolvendo os especialistas do domínio e enquadrado com os modelos de referência da literatura, nomeadamente o modelo RAMI4.0 [23].

Fase	Objetivos	Resultados Esperados
Ciclo de verificação de relevância do problema	- Validação da relevância do problema; - Caracterização do problema; - Clarificação dos objetivos e âmbito do trabalho.	- Necessidades e desafios associados à aplicação do IIoT nas PMEs; - Objetivos definidos.
Ciclo de conceptualização do artefacto de engenharia	- Sistematização dos conceitos inerentes ao domínio da plataforma IIoT, mantendo o alinhamento com os modelos de referência existentes; - Desenvolvimento de um modelo conceptual em colaboração com os especialistas do domínio.	- Lista de conceitos, definições e caracterização; - Ontologia conceptual.
Ciclo de formalização	- Identificação e descrição das restrições associadas à ontologia conceptual e que estarão na base do motor de inferência; - Representação do modelo através híper-grafos; - Verificação da utilidade do modelo através de um caso de estudo; - Ajuste, se necessário, do resultado do ciclo anterior.	- Implementação da ontologia no Grakn; - Condução de um caso de estudo.

Tabela 2 Tabela dos objetivos e dos resultados esperados de cada fase da metodologia

3.2. Articulação do modelo com as arquiteturas de referência

O *Reference architectural Model Industrie 4.0* (RAMI4.0), é um modelo de referência arquitetural, proposto pela *Platform Industrie 4.0*, cujo seu intuito é guiar a transformação e implementação da indústria 4.0 nas empresas europeias. Este modelo está dividido em três eixos, conforme apresentado na Figura 2, nomeadamente: i) o eixo vertical, das Camadas de IT; ii) o eixo horizontal esquerdo, do Ciclo de vida de sistemas e produtos; iii) o eixo horizontal direito, da Hierarquia Funcional [23].

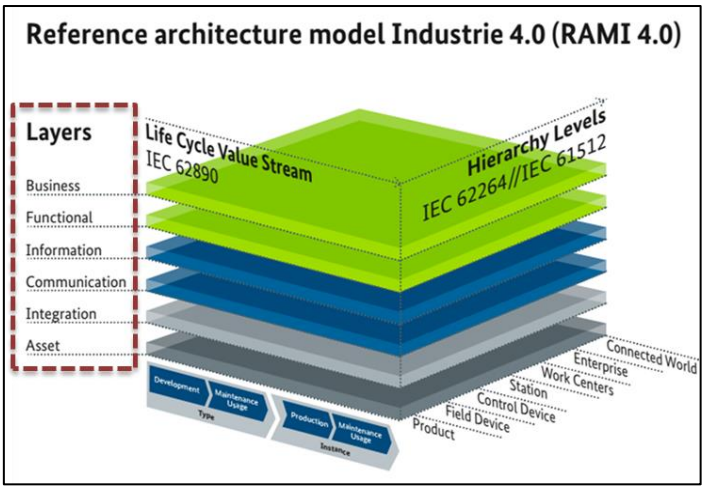

Figura 2 Representação tridimensional do RAMI4.0 [23]

No contexto do RAMI4.0, as plataformas IIoT estão associadas ao eixo vertical, mais concretamente, nas camadas de integração, comunicação e informação [7]. Com base neste pressuposto, foi concebida uma arquitetura conceptual de alto nível para plataformas IIoT, apresentada na Figura 3, sendo esta estruturada em camadas e níveis.

Tipicamente, os níveis que enquadram a plataforma IIoT são:

- ***Edge***: nível que se situa na periferia dos produtores de dados [66], [67], nomeadamente os ativos no chão de fábrica, como por exemplo, máquinas, sensores, controladores ou pessoas;
- ***Platform***: nível que compreende a plataforma IIoT;
- ***Business***: onde estão localizados os sistemas e aplicações de apoio aos processos empresariais, como por exemplo, sistemas de informação ou aplicações de BI.

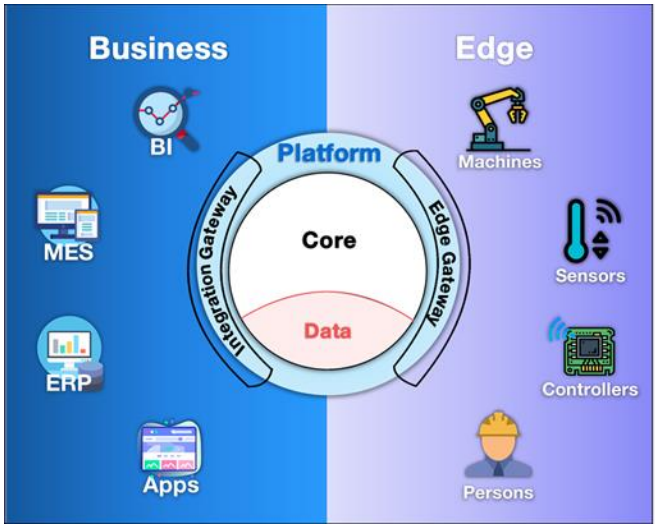

Figura 3 Arquitetura Conceptual de uma plataforma IIoT

As camadas da plataforma IIoT são definidas como:

- ***Core***: contém os serviços básicos e fundamentais de suporte à plataforma IIoT, proporcionando resiliência, a alta disponibilidade e a consistência na entrega e na coleta de dados;
- ***Data***: garante o armazenamento, consulta, manipulação e gestão dos dados recolhidos pela plataforma IIoT;
- ***Edge Gateway***: camada de abstração que permite a transição de dados entre os níveis do *Edge* e *Platform*;
- ***Integration Gateway***: camada de abstração que permite a transição de dados entre os níveis do *Business* e *Platform*.

A arquitetura conceptual descrita encontra-se alinhada com o modelo de referência RAMI 4.0, concretamente com o eixo vertical, como evidenciado pela Figura 4. O enquadramento da base do modelo com os *standards* existentes é fundamental, considerando que o propósito assenta na desmistificação e adaptação dos modelos existentes para as PMEs, e não, o desenvolvimento de apenas mais um modelo. Deste modo, o nível *edge* da arquitetura apresentada, corresponde à camada dos ativos do modelo RAMI 4.0. Já o nível *platform*, agrega as camadas de integração, comunicação e informação, e por fim, o nível de *business*, é constituído pelas camadas funcional e de negócio do RAMI4.0. Ao nível das camadas, o *edge gateway* e o *integration gateway* estão relacionados com a camada de integração do RAMI4.0, constituindo a fronteira entre o nível do *edge* e *platform*, e, entre o nível *platform* e *business*, respetivamente. A camada do *core* corresponde à camada de comunicação, e a camada *data* é mapeada na camada de informação do RAMI4.0.

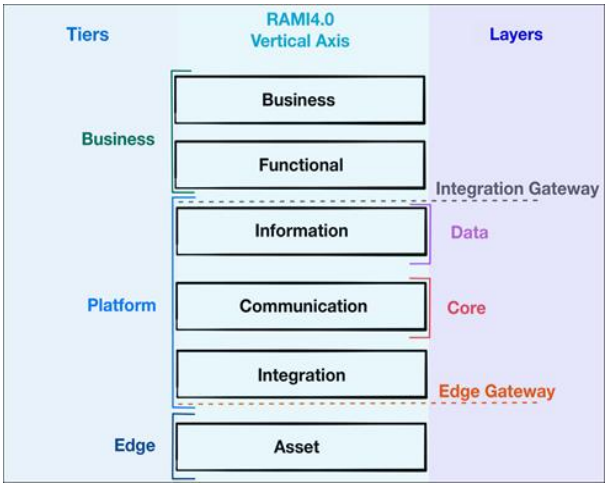

Figura 4 Enquadramento da arquitetura conceptual de plataformas IIoT com o eixo vertical do RAMI4.0

3.2.1. Conceptualização do domínio

Um processo de conceptualização pretende estruturar um domínio de conhecimento através de um modelo conceptual, descrevendo um determinado problema e/ou solução em termos do seu vocabulário de domínio [68]. O processo de conceptualização poderá ainda ser visto como um mecanismo de suporte à aprendizagem organizacional, considerando-o como um processo de reflexão colaborativa, articulando o conhecimento tácito (exemplo: experiência) em conceitos e relações entre conceitos de forma explícita [69]–[71].

De um ponto de vista mais formal, a "*conceptualização contém os objetos, os conceitos e as outras entidades que se presume existirem numa determinada área de interesse e as relações entre os mesmos. Uma conceptualização é uma visão abstrata e simplificada do mundo que desejamos representar para um dado propósito. Cada base de conhecimento, sistema baseado no conhecimento, ou agente de nível de conhecimento, está comprometido com alguma conceptualização, explícita ou implicitamente."* [72] .

Adicionalmente, o termo concetualização é frequentemente associado a ontologias, sendo que, segundo [42], *"Uma ontologia é uma especificação explícita de uma conceptualização*".

Neste trabalho, seguiu-se uma abordagem colaborativa no processo de conceptualização, cujo resultado (modelo conceptual da arquitetura) foi posteriormente articulado numa ontologia. Este representa a "*knowledge base backbone*" e fornece apoio ao processo de especificação de arquiteturas IIoT.

O processo de conceptualização contemplou as fases de:

- Identificação e extração de informação, em que o processo foi facilitado pela análise da literatura;
- Extração de conceitos relevantes do domínio, identificação dos termos que referenciam os conceitos e descrição dos mesmos (ver Anexos A, B, C, D, E, F);
- Definição dos conceitos através da definição de estruturas "Conceito-Relação-Conceito";
- Negociação das estruturas conceptuais (Conceito-Relação-Conceito) com os especialistas.

O resultado do processo de conceptualização constitui um modelo conceptual representando o conhecimento acerca da especificação de arquiteturas IIoT adaptadas para PMEs. Este processo foi conduzido com base no pressuposto de articular necessidades, meios e capacidades. As necessidades derivam das lacunas entre o estado atual do negócio e os seus objetivos a curto e médio prazo, correspondendo a requisitos de negócio a implementar na plataforma IIoT. Os meios correspondem aos artefactos digitais, que facilitam ou suportam o desenvolvimento de

capacidades para a transição do estado atual do negócio para o seu estado futuro, no âmbito de uma estratégia de integração vertical.

Neste processo de conceptualização, um dos principais desafios é a definição do conceito de *componente* e de *necessidade*, de modo que seja possível a identificar os componentes mais adequados a uma necessidade empresarial específica. Implica não só compreender as necessidades do negócio, mas também compreender as principais características dos componentes, as suas capacidades de interoperabilidade e os seus protocolos de comunicação. O modelo discutido nesta secção representa o conhecimento sobre os componentes mais comuns a serem utilizados numa arquitetura IIoT. Define o contexto de utilização de cada componente através das relações conceptuais dos componentes e das restrições subjacentes (por exemplo, regras), fornecendo capacidades de raciocínio através de um motor de inferência.

3.2.2. Modelo conceptual para especificação de arquiteturas IIoT

Do processo de conceptualização resultou o modelo conceptual, representando o vocabulário comum para especificação de arquiteturas IIoT, seguindo uma abordagem pragmática de articulação entre necessidades – meios – capacidades. Deste modo, é possível identificar os requisitos técnicos associados aos requisitos de negócio de uma forma standardizada. Este modelo de alto-nível é apresentado na Figura 5, em que as relações são representadas pelos círculos mais pequenos e as entidades pelas formas ovais.

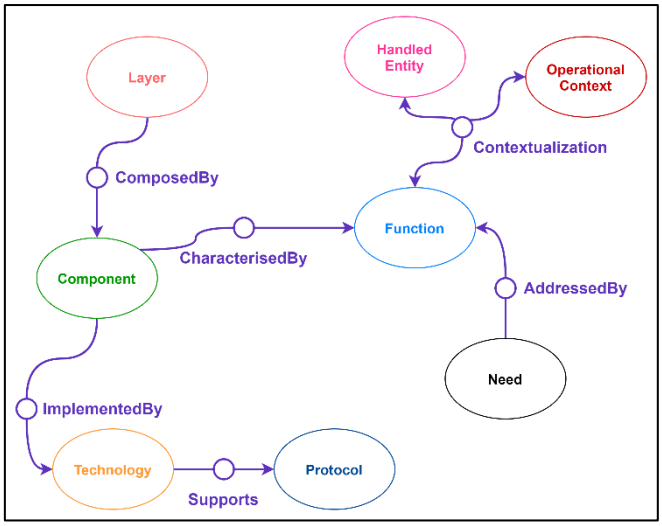

Figura 5 Excerto da estrutura conceptual de alto-nível

O modelo conceptual apresentado, resulta de um esforço de sistematização exaustiva dos artefactos tecnológicos tipicamente relacionados com arquiteturas IIoT (ver Anexos A, B, C, D, E, F). Contudo, descrevem-se de seguida o racional associado a cada um dos conceitos, subconceitos e relações representadas no modelo.

Componente

Neste contexto, um componente é uma entidade abstrata que desempenha um determinado papel na arquitetura IIoT. Este papel é caracterizado por um conjunto de funções desempenhadas por esse componente. Estes são implementados por uma determinada tecnologia, sendo que todos os componentes têm de estar enquadrados, em pelo menos, numa das camadas da arquitetura conceptual. Cada componente tem associado um tipo que pode ser, por exemplo, *message-broker*, *timeseries-database* ou *dashboard-portal*. Para além disso, podem existir relações de dependência entre componentes, em que um componente ***a*** depende de um componente ***b***. Assim o componente ***a*** é designado como *depender* e o componente ***b*** como *dependee*. Mais concretamente, *depender* e *dependee* correspondem a papeis (*roles)*, que os conceitos assumem na relação conceptual.

Este conceito é ainda categorizado segundo as funcionalidades que este disponibiliza. A categorização funcional utilizada, apresentada na Figura 6, é dividida em 9 categorias, denominadas no modelo como domínios. Esta categorização tem por base o trabalho apresentado em [25], no âmbito do projeto UNIFY-IOT[3] .

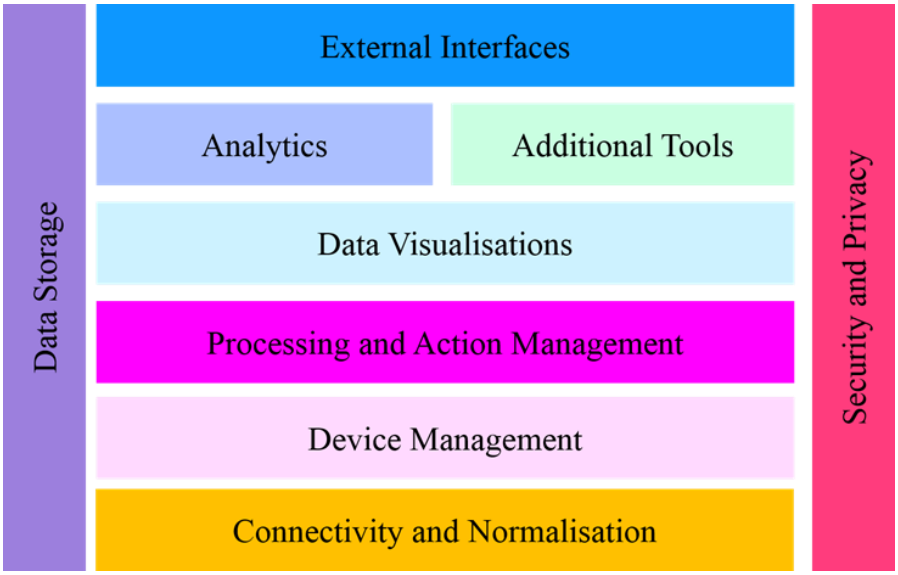

Figura 6 Categorização funcional dos componentes das arquiteturas IIoT [25]

- ***Connectivity and normalisation***: engloba a obtenção e o envio de dados em tempo real para os diferentes dispositivos, sistemas ou componentes da plataforma IIoT, assegurando a normalização do formato dos dados;

[3] https://cordis.europa.eu/project/id/688369

- ***Device management***: permite a descoberta, o registo e monitorização de dispositivos IIoT, possibilitando também a atualização do *software/firmware* dos mesmos quando necessário;
- ***Processing and action management***: conjunto de operações sobre os fluxos de dados recebidos, dos diferentes dispositivos de IIoT, permitindo a geração de eventos e a tomada de ações, através da definição e execução de regras;
- ***Data storage***: assegura o armazenamento dos dados recolhidos pela plataforma IIoT para posterior processamento e utilização por parte dos componentes;
- ***Data visualisations***: permite a exploração dos dados da plataforma IIoT de forma a identificar relações entre dados e apoiar a tomada de decisão através da criação de *dashboards*;
- ***Analytics***: permite gerar conhecimento, a partir dos dados recolhidos pela plataforma, através da análise de históricos ou processamento de *streams* de dados, incluindo funcionalidades de *data mining* e *machine learning*;
- ***External Interfaces***: permite o desenvolvimento de aplicações e a sua integração com a plataforma IIoT através da disponibilização de APIs;
- ***Security and Privacy***: possibilita a definição e aplicação de políticas de acesso e utilização dos componentes e dados da plataforma IIoT;
- ***Additional tools***: representam componentes adicionais que ativam outros tipos de funcionalidades, como por exemplo, a gestão de alterações, orquestração de componentes ou monitorização da infraestrutura.

O Componente tem ainda um subconceito – *Entrypoint.* O *Entrypoint*, define os mecanismos de interação entre a plataforma IIoT e os níveis contíguos, nomeadamente com os níveis de *Business* e *Edge*. Este tem dois portos, o *North Port*, que é utilizado para comunicar com os outros componentes, que compõem a plataforma IIoT. E o *South Port*, que é utilizado para comunicar com *things*, nomeadamente, ativos e sistemas, que estão fora da plataforma IIoT. Em cada um destes portos são utilizados protocolos de comunicação, que são suportados pelas tecnologias que implementam este *entrypoint*.

Função

As funções são representadas por termos que correspondem a verbos, conjugados no presente, e que definem o objetivo/ação da função, como por exemplo, *collect*, *present*, *query* ou *transform*.

A definição das funções, estão contextualizadas com o tipo de entidades que manipulam (Figura 7) e com o contexto operacional a que pertencem essas mesmas entidades. Para além disso, uma

função contribui para a definição de um ou mais componentes, ou, tecnologias, e, colmata uma ou mais necessidades de negócio existentes.

As entidades manipuladas, apresentadas na Figura 7, são:

- *Component-Containers:* são contentores lógicos que encapsulam os componentes e que permitem que estes sejam manipulados por outros componentes, garantindo a integridade da configuração dos mesmos.
- *Data:* define um conjunto de dados abstrato, que pode ser subdividido em:
    - *Files:* são dados materializados em ficheiros
    - *Logs:* representa os *logs* dos vários componentes da plataforma
    - *Metrics:* representa as métricas dos vários componentes da plataforma
    - *Alerts:* constitui os alertas imitidos a partir da ocorrência de eventos
    - *Datasets:* são um tipo de dados genérico que representa os dados coletados fora da plataforma IIoT
        - *Timeseries-Datasets:* conjunto de dados de series temporais
        - *Tabular-Datasets:* conjunto de dados que segue uma estrutura tabular
        - *Document-Datasets:* conjunto de dados estruturados em documentos
- *Actors:* composto pelos vários atores que interagem com a plataforma, sendo estes:
    - *Systems:* os sistemas empresariais ou aplicações externas que estão ligados à plataforma IIoT
    - *Users:* os utilizadores que interagem com os componentes da plataforma IIoT
    - *Devices:* os dispositivos do chão de fábrica que estão ligados à plataforma IIoT

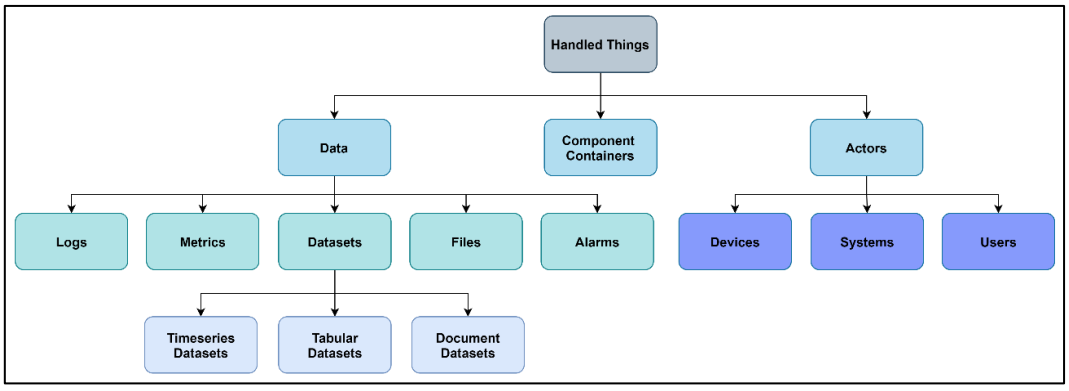

Figura 7 Hierarquia das Entidades Manipuladas

No que diz respeito aos contextos operacionais destas entidades, estes são divididos em:

- *External-Systems:* composto pelos sistemas empresariais ou aplicações externas que estão ligados à plataforma IIoT;
- *Platform:* a plataforma IIoT e os seus componentes;
- *Platform-Infrastructure:* a infraestrutura tecnológica onde a plataforma IIoT assenta;

- *Assets:* representa o conjunto de dispositivos ligados à plataforma IIoT.

A relação entre estas 3 entidades: a função, a entidade manipulada e o contexto operacional; formam uma função composta, como por exemplo, *Present Platform-Infrastructure's Metrics*, *Collect Platform's Logs*, ou, *Connect Assets's Devices*. Em todo o modelo, as funções utilizadas estão sempre compostas com a respetiva entidade manipulada e seu o contexto operacional.

Necessidade

As necessidades representam requisitos específicos a colmatar com a adoção da plataforma IIoT, intimamente associados com o uso e utilidade da plataforma, como por exemplo, visualizar os dados dos ativos no chão-de-fábrica (*visualise status data from the assets*), desencadear eventos da plataforma (*trigger platform events*) e integrar com sistemas empresariais (*integrate enterprise systems*).

Por usa vez, as necessidades são colmatadas por uma ou mais funções, e têm um tipo de necessidade associado, que agrupam as necessidades pelo tipo de funcionalidade oferecida pelas mesmas.

Os tipos de necessidades, e as suas respetivas funcionalidades, estão divididas da seguinte forma:

- *Edge Integration*: necessidades relacionadas com integração de ativos, no chão de fábrica, com a plataforma IIoT, como por exemplo, *Integrate with Assets*.
- *Business Integration*: necessidades relacionadas com integração de sistemas e aplicações de negócio com a plataforma IIoT, como por exemplo, *Integrate with External Systems*.
- *System Functionality*: necessidades relacionadas com funcionalidades de infraestrutura da plataforma IIoT, como por exemplo, *Monitor the Platform*.
- *Security Functionality*: necessidades relacionadas com a segurança da plataforma IIoT, como por exemplo, *Control Users Access*.
- *Business Functionality*: necessidades relacionadas com as funcionalidades orientadas ao negócio da plataforma IIoT, como por exemplo, *Visualize Status Data from the Assets*.

Tecnologia

As tecnologias são o meio através do qual se concretizam os componentes, como por exemplo, Apache Kafka, Node-Red e Grafana.

Uma tecnologia pode implementar um ou mais componentes e suportar vários protocolos de comunicação. Esta pode ainda ser caracterizada por uma ou mais funções, específicas de uma implementação de um componente, adicionando funções extras ao componente. Para além disso, estas podem participar em relações de dependência entre implementações de

componentes, em que uma implementação ***a*** depende de uma implementação ***b***. A implementação ***a*** é designada como *depender* e a implementação ***b*** como *dependee*.

Protocolo

Os protocolos de comunicação são o meio que permite aos componentes comunicar entre si e com o exterior, sendo estes suportados por uma ou mais tecnologias. Para além disso também têm um tipo associado correspondente ao nível em que são usados, sendo estes:

- *Edge*: Protocolos tipicamente utilizados para comunicar com os ativos no nível do *Edge*.
- *Business*: Protocolos tipicamente utilizados para comunicar com os sistemas e aplicações no nível do *Business*.
- *Platform*: Protocolos tipicamente utilizados para comunicação entre os componentes da plataforma IIoT no nível *Platform*.

Camada

As camadas da plataforma IIoT são as mesmas apresentadas na arquitetural conceptual, como por exemplo, *Core*, *Edge Gateway* ou *Data*, sendo estas compostas por um ou mais componentes.

Os conceitos apresentados constituem as pedras basilares para a definição de uma arquitetura, seja ela orientada para IIoT ou para um outro *middleware*. Deste modo, a partir dos conceitos primários, será necessário representar uma arquitetura *baseline* que permita inferir arquiteturas específicas. É esta arquitetura *baseline* que representa o sistema de organização do conhecimento para especificação de arquiteturas IIoT. Para o efeito o modelo de alto nível foi estendido com um novo tipo de conceito, *component-instance*.

A Figura 8 ilustra a organização do modelo atual, que tem um nível inferior, onde está a estrutura conceptual do modelo de alto nível. Um nível central, onde temos a arquitetura *baseline* que é constituída por instâncias dos conceitos do modelo de alto-nível. E por fim, o nível superior, em que temos a arquitetura específica que é constituída por instâncias do conceito *component-instances*.

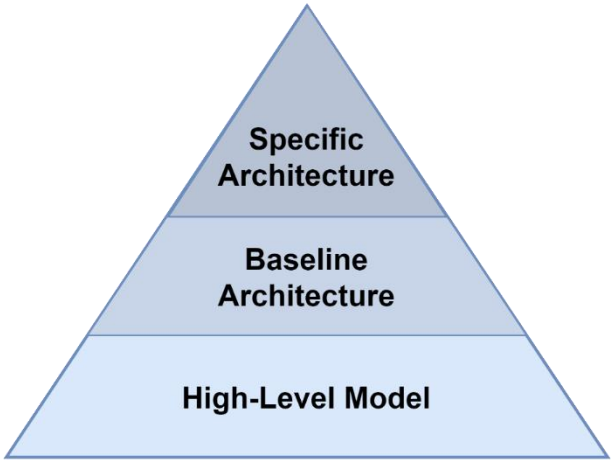

Figura 8 Organização do modelo semântico

3.2.2.1. Arquitetura Baseline

Para concretizar o objetivo de automatizar o processo de especificação das plataformas IIoT é necessário existir uma arquitetura *baseline*. Esta serve como uma base para a derivação de novas arquiteturas, que são customizadas a um determinado conjunto de necessidades. A mesma deve ser suficientemente genérica, *standard-oriented*, e com um âmbito e leque de funcionalidades alargado, permitindo a sua implementação em diferentes casos de uso industriais.

A arquitetura *baseline* segue uma abordagem baseada em componentes, fomentando a customização modular e a sua reutilização. Esta é formada a partir de uma sistematização de componentes, e da tecnologia subjacente, que está ao dispor das organizações para fazerem as suas escolhas, segundo as suas necessidades de negócio.

A automatização do processo de especificação das plataformas IIoT é alcançada através da aplicação de regras de inferência na arquitetura *baseline*. Estas regras permitem inferir novo conhecimento a partir do conhecimento existente na ontologia [31]. Neste caso, as regras permitem a derivação de novas arquiteturas IIoT através da redução do âmbito das funcionalidades, da arquitetura *baseline*, às necessidades específicas de negócio de uma determinada organização, como exemplificado na Figura 9. Isto resulta na seleção dos componentes, caracterizados pelas funções que endereçam essas necessidades específicas. A partir dos componentes selecionados são criadas instâncias de *component-instance*, que passam a integrar a nova arquitetura, e desta forma, é gerada uma arquitetura de uma plataforma IIoT, específica ao caso de uso daquela organização.

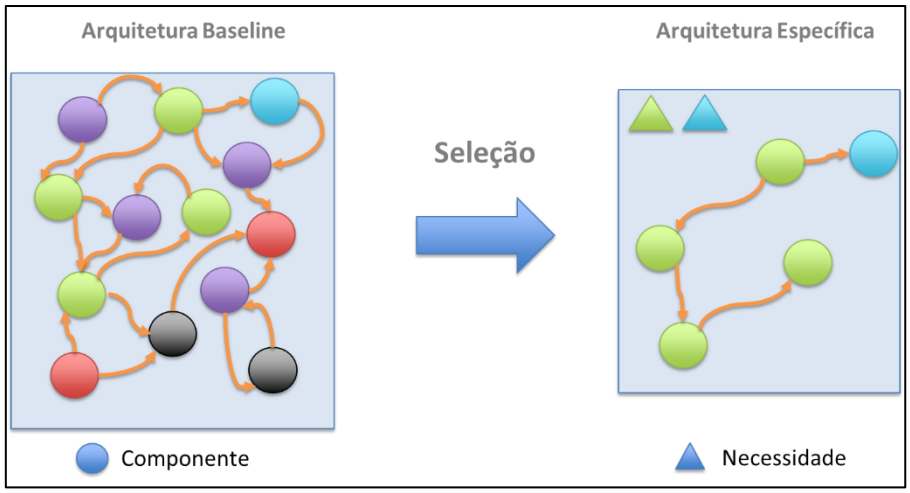

Figura 9 Ilustração do processo de derivação de arquiteturas específicas a partir da arquitetura *baseline*

Esta abordagem implica que os conceitos, que compõem a arquitetura *baseline*, não estejam diretamente modelados na estrutura conceptual do modelo, mas que sejam inicializados através da criação de instâncias dos mesmos. Esta foi uma das razões que levou ao surgimento do conceito de *component-instance*.

Component-Instance

Um *component-instance*, corresponde a uma instância de um determinado componente, mas representada como conceito, ao nível da arquitetura *baseline*, sendo que as instâncias deste conceito constituem a arquitetura específica. Por exemplo, se uma instância de um componente, *message-broker*, endereça uma determinada necessidade, então é criada uma instância de *component-instance*, *na* nova arquitetura, associada à esta instância de *message-broker*, na arquitetura *baseline.* Uma instância de um componente pode ser instanciada por um ou mais *component-instances*, mas um *component-instance* só pode ser instância de uma instância de componente.

À semelhança do modelo de alto nível, o *entrypoint-instance* é um subconceito de *component-instance*, representando uma instância de um *entrypoint*, contendo igualmente um *South Port* e um *North Port*. No entanto, um *entrypoint-instance* usa apenas um protocolo de comunicação no *South Port*, de forma a modularizar e possibilitar a escalabilidade da comunicação com o exterior da plataforma, sendo que deverá existir um *entrypoint-instance* por cada protocolo diferente que seja necessário para comunicar com o exterior da plataforma.

A tecnologia que implementa determinado *component-instance* depende da tecnologia associada ao *componente*. Por exemplo, uma instância do componente *message-broker* pode ser implementada por duas tecnologias, Apache Kafka e RabbitMQ. Se existir um *component-instance* que seja instância de *message-broker*, então este também pode ser implementado por

estas duas tecnologias. A forma como é realizada a escolha da tecnologia que irá implementar um *component-instance* será abordada na subsecção 3.3.4.

As relações entre *component-instances* ocorrem através de um protocolo *p* e restringem-se a situações em que pertencem a conjuntos *disjoint*, ou seja, componentes diferentes e, apenas se existirem duas tecnologias (t_1 e t_2) implementando os componentes c_1 e c_2 respetivamente, de modo que t_1 requer (*requires*) o protocolo *p* e t_2 disponibiliza (*provides*) o protocolo *p*, ou vice-versa. Esta relação de conexão não implica a direccionalidade no fluxo de dados, implica apenas que é possível a comunicação entre os *component-instances*.

A extensão do modelo de alto-nível, com o conceito de *component-instance*, define o pilar conceptual da ontologia, em que a sua estrutura conceptual é brevemente descrita na Figura 10. A interpretação desta figura é igual à da Figura 5 acrescentando as linhas diretas com o nome "*sub*", que são um tipo especial de relação que significa que uma entidade é subordinada de uma outra entidade, herdando todas as características da entidade pai.

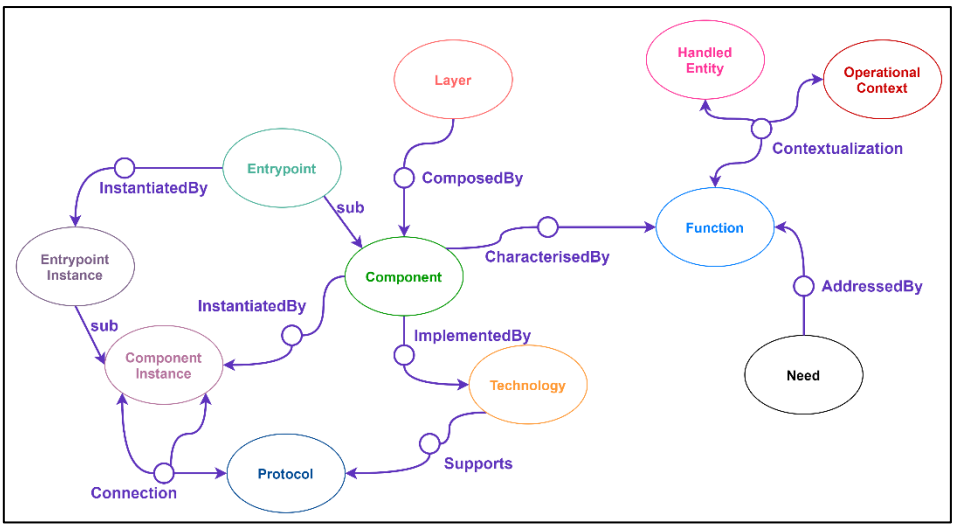

Figura 10 Excerto da estrutura conceptual da ontologia

Uma outra abordagem para a especificação das arquiteturas IIoT, sem a necessidade de introduzir do conceito de *component-instance*, seria modelar os vários tipos de conceitos da arquitetura *baseline* diretamente na estrutura conceptual. Por exemplo, em vez de modelar o conceito de componente e depois instanciar um componente do tipo *message-broker*, poderíamos modelar os vários tipos de componentes da arquitetura *baseline* diretamente na estrutura conceptual, e assim sucessivamente para os outros conceitos. Esta abordagem seria baseada em categorias ou tipos de conceitos, em vez de instâncias, e tornaria o modelo mais simples, em que não seria necessário o conceito de *component-instance*. Isto reduziria o número de instâncias presentes no modelo, simplificando também o processo de inferência. No entanto, o modelo seria mais rígido, em que se fosse necessário adicionar, por exemplo, um novo tipo de

componente seria necessário alterar a estrutura conceptual do modelo. A abordagem utilizada torna o modelo mais flexível, através da possibilidade de fazer alterações à arquitetura *baseline* e aos seus componentes, sem ter que realizar alterações à estrutura do modelo. Por exemplo, no caso de se adicionar um novo componente à arquitetura *baseline*, bastava inserir uma nova instância de componente. Outra das vantagens é que aumenta a possibilidade de reutilização do modelo no domínio do IIoT. Isto porque permitiria a outros especialistas do domínio, ou, organizações especializadas em arquiteturas IIoT, introduzirem as suas arquiteturas *baseline*, sem ter que realizar alterações à estrutura do modelo, para automatizar o processo de especificação do qual fazem negócio.

Na Figura 11 é apresentada a arquitetura *baseline* utilizada na solução, que é composta pelos componentes e as respetivas ligações lógicas entre os mesmos.

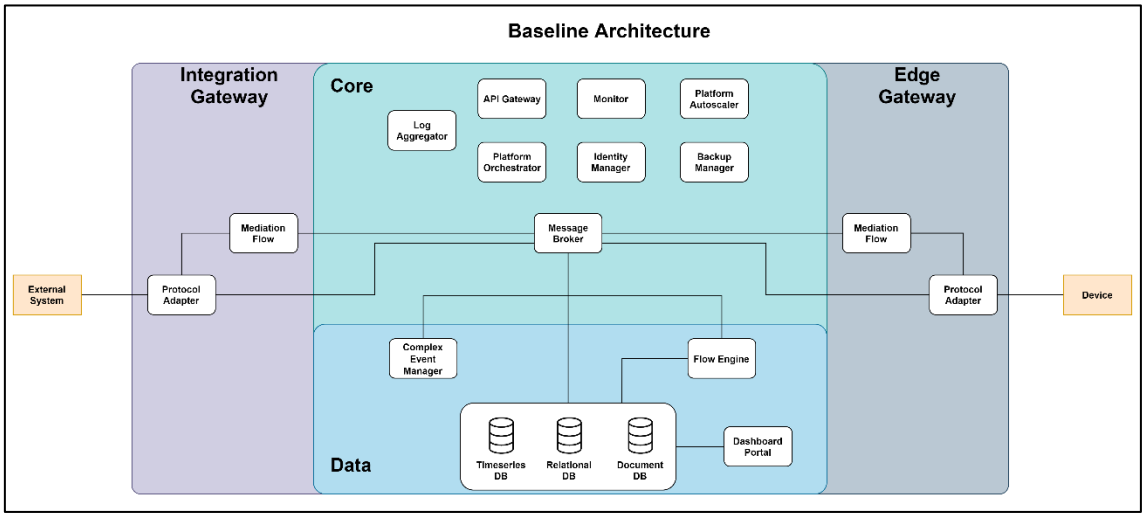

Figura 11 Arquitetura *baseline* instanciada no modelo semântico

As ligações lógicas são agnósticas às implementações dos componentes e aos protocolos utilizados, restringindo apenas quais os componentes que podem comunicar entre si. No caso de os componentes interagirem com outros componentes através da infraestrutura da plataforma, como por exemplo, o *platform-orchestrator* ou o *monitor*, não são representadas ligações lógicas. Estas ligações são depois utilizadas para inferir as conexões entre *component-instances*, já incluindo o protocolo de comunicação a ser utilizado.

As instâncias dos conceitos que compõem esta arquitetura *baseline*, nomeadamente, os componentes e as suas funções e ligações lógicas, as necessidades que esta arquitetura colmata, e as tecnologias e protocolos subjacentes, foram sistematizadas segundo a experiência do autor com assistência de outros especialistas do domínio, e da literatura. Em anexo a este documento segue um conjunto de tabelas que contêm em detalhe esta sistematização, nomeadamente, nos anexos:

- Anexo A: apresenta as definições das funções disponibilizadas pela arquitetura *baseline.*
- Anexo B: lista as funções compostas que são disponibilizadas pela arquitetura *baseline* e as necessidades que são endereçadas por essas funções.
- Anexo C: apresenta os protocolos de comunicação, e respetivos tipos, disponíveis na arquitetura *baseline*.
- Anexo D: lista as necessidades colmatadas pela arquitetura *baseline* e as suas respetivas definições.
- Anexo E: expõe as tecnologias que implementam os componentes da arquitetura *baseline* e os protocolos de comunicação suportados pelas mesmas.
- Anexo F: contém os componentes e as funções que caracterizam o seu papel na arquitetura *baseline*.

3.2.2.2. Validação das estruturas conceptuais

Partindo da estrutura conceptual desenvolvida, foi definida uma narrativa, com base em *templates* semânticos (Figura 12) para validar a completude e utilidade das estruturas conceptuais desenvolvidas. Estes *templates* são inspirados em *user stories*, cuja descrição é feita de acordo com as preposições formadas pelas estruturas conceptuais. Deste modo, o *template* articula as necessidades com a arquitetura, composta pelos componentes e as suas funções, e as tecnologias concretizadoras da mesma, evidenciando a mediação da arquitetura entre as tecnologias e as necessidades de negócio.

Template:

<technology> implements ***<component>*** to ***<function> <operational_context>*** 's ***<handled_thing>*** so that I can ***<need>***

Examples:

Promotheus implements ***Monitor*** to ***Collect Platform***'s ***Metrics*** so that I can ***Monitor the Platform***

Keycloack implements ***Security Component*** to ***Authenticate Platform***'s ***Users*** so that I can ***Control User Access***

Nifi implements ***Edge Gateway*** to ***Collect Asset***'s ***Datasets*** so that I can ***Manage the Assets***

Figura 12 Template para validação de conceitos semânticos

Os *templates* semânticos utilizados para validar os conceitos do modelo estão presentes no Anexo G.

3.3. Implementação

Para a implementação de uma solução que permita automatizar a especificação de plataformas IIoT, segundo as necessidades específicas de uma organização, é proposta a seguinte arquitetura, apresentada na Figura 13.

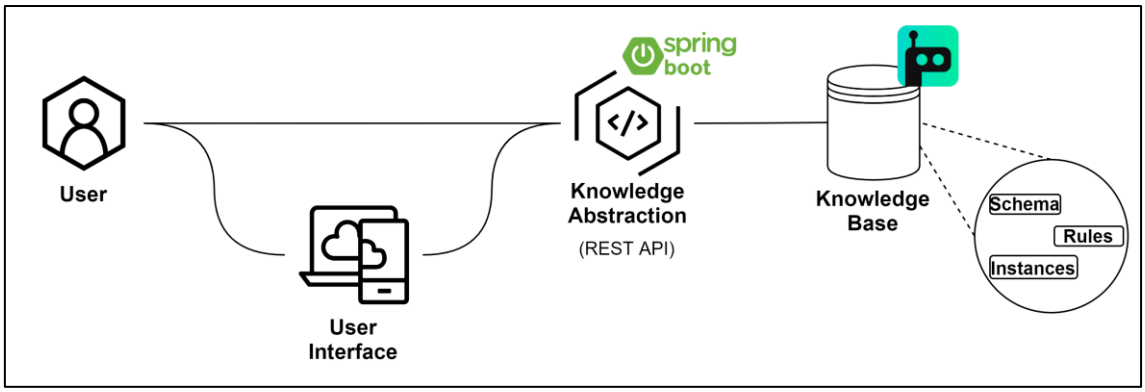

Figura 13 Arquitetura da Solução

Esta arquitetura é composta por 2 componentes principais, uma *knowledge base* onde está formalizado o conhecimento do domínio, e uma *knowledge abstraction*, que simplifica a interação com a *knowledge base*. Um terceiro componente que figura nesta arquitetura é a *user interface*, cujo intuito seria facilitar a interação com os utilizadores. No entanto, esta não foi desenvolvida no âmbito da dissertação, estando planeada como um trabalho futuro. Apesar disso, a ausência desta user interface não tem qualquer impacto na utilidade da solução, dado que é possível interagir com a mesma através de uma REST API.

O contexto de uso desta solução, prevê dois tipos de utilizador: o especialista do domínio do IIoT e/ou o representante da organização. Este último terá interesse em adotar uma plataforma IIoT para colmatar determinadas necessidades de negócio. Assume-se que terá um conhecimento do processo organizacional em geral, dos requisitos de negócio inerentes ao processo de transformação digital e dos dispositivos e sistemas da organização. Por outro lado, o especialista do domínio, terá interesse em criar uma especificação de uma arquitetura de plataformas IIoT, de forma bastante rápida, com o melhor alinhamento com as necessidades de negócio. O intuito poderá passar pela criação de uma especificação para um determinado caso de uso em que não seja possível eleger um representante, como por exemplo, no caso de uma plataforma IIoT que envolve múltiplas organizações. Por outro lado, poderá ser utilizada para criar um protótipo de uma arquitetura IIoT, para posterior modificação, pelo especialista do domínio, conforme determinados requisitos que não estão contemplados no âmbito do modelo, como por exemplo, os interesses dos *stakeholders* no uso de determinadas tecnologias.

De um modo geral, o utilizador irá interagir com a solução, fornecendo os *inputs*, ou seja, as necessidades específicas da sua organização, e a solução irá retornar uma especificação da arquitetura da plataforma IIoT customizada ao caso de uso daquela organização. Esta interação irá ocorrer através de uma REST API disponibilizada pela *knowledge abstraction*, ou por uma interface com o utilizador, alimentada pela REST API. O diagrama de casos de uso do utilizador é apresentado na Figura 14.

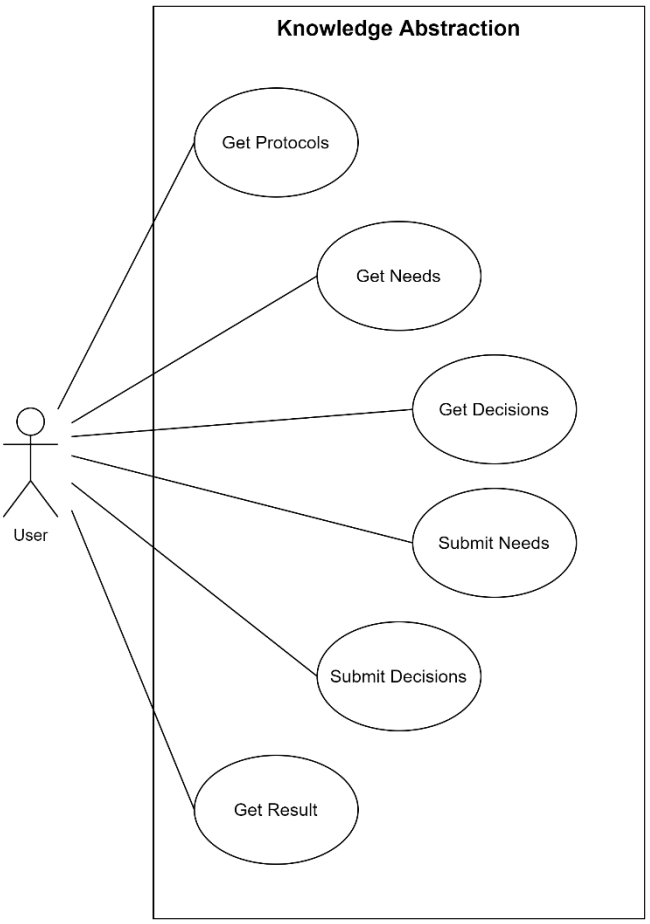

Figura 14 Diagrama de casos de uso do utilizador

Detalhando os componentes principais da arquitetura. A *knowledge abstraction* é uma aplicação, implementada em *Spring Boot*, que funciona como uma abstração sobre a *knowledge base*, facilitando a interação com a mesma através de uma REST API.

Este componente da solução tem como funções:

- Inicializar a *knowledge base*;
- Orquestrar as *querys* à *knowledge base*;
- Inserir as instâncias (*component-instances*) resultantes do processo de especificação;
- Gerir e delegar as decisões que são precisas tomar pelo utilizador.

A *knowledge base* é estruturada segundo a ontologia desenvolvida, contendo a representação formal do conhecimento do domínio do IIoT, as suas regras e restrições.

3.3.1. Estruturação da base de conhecimento

A *knowledge base* foi estruturada seguindo uma abordagem baseada em grafos, em particular híper-grafos. Híper-grafos são uma generalização de grafos, sendo que a principal diferença é

que, num grafo, uma aresta apenas liga dois vértices, enquanto num híper-grafo, uma aresta pode ligar *n* vértices [73]–[75].

O uso de híper-grafos para descrição de domínios em ontologias traz vantagens em relação às abordagens tradicionais, permitindo a modelação de relações mais complexas. Numa abordagem tradicional, como por exemplo, modelação em RDF, as relações são binárias, onde uma relação relaciona apenas duas entidades. Numa abordagem baseada em híper-grafos, a mesma relação pode relacionar *n* entidades, tornando mais eficiente e natural a modelação de relações de muitos-para-muitos [73], [75]–[77]. Esta abordagem beneficia a solução proposta, dada a existência deste tipo de relações em grande parte das entidades do pilar conceptual da ontologia.

Nesta solução foi usado o Grakn na versão 1.8.3 para a implementação da *knowledge base*. A razão pela qual não foi utilizada a versão mais recente é que, até à data de escrita deste documento, a esta versão não suporta, temporariamente, o uso de negação em regras. Este tipo de operador é necessário para o correto funcionamento de 3 das regras presentes no modelo desenvolvido nesta dissertação.

3.3.2. Grakn

Grakn foi a ferramenta utilizada para acomodar a base de conhecimento (*knowledge base*). Grakn é um *knowledge graph*, que permite modelar entidades e relações, e cujo foco é a modelação de relações de alto-nível através do uso de papéis (*roles*) [78]. Esta ferramenta permite não só armazenar o conhecimento, mas também inferir, consultar e modular a ontologia e as restrições subjacentes, com base em regras.

As vantagens da utilização de Grakn passam por uma modelação de conhecimento mais intuitiva e com uma aprendizagem mais rápida, do que comparado com as abordagens tradicionais como RDF ou OWL [79], por ser baseada no modelo de entidade-relação. Para além disso, esta ferramenta traz benefícios a nível da arquitetura da solução, dado ser uma única ferramenta que permite questionar, inferir, estruturar e armazenar o conhecimento, dado que já tem embebidos um motor de inferência e de *querys*. Desta forma, dispensa o uso de mais do que uma ferramenta para o mesmo efeito, o que seria necessário caso se seguisse por uma abordagem tradicional [79], simplificando a arquitetura.

Outra vantagem é facto de simplificar e abstrair a necessidade de conhecer os vários formalismos de modelação de RDF ou OWL. Em Grakn são utilizados híper-grafos como forma de representar o conhecimento.

Através do Grakn, a base de conhecimento é composta por: i) *schema*, que estrutura e valida os conceitos do modelo, representado a estrutura conceptual do modelo; ii) regras, que permitem inferir novo conhecimento; iii) instâncias dos conceitos, que representam a arquitetura *baseline*.

O *schema* materializa a estrutura conceptual do modelo, e define que conceitos podem ser instanciados, que atributos possuem e que papeis podem desempenhar nas relações.

De seguida são apresentados os conceitos do pilar conceptual do modelo, com o respetivo *schema* e representação em Grakn. Por uma questão de brevidade, alguns dos papeis que os conceitos podem desempenhar foram omitidos, estando estes presentes no *schema* do modelo anexado a este documento.

Componente

Um componente é caracterizado pelos atributos:

- *component-type*: O tipo do componente, como por exemplo, *message-broker*, *flow-engine* ou *protocol-adapter*, e que tem que ser único entre componentes.
- *domain*: O domínio do componente, segundo a sua categorização funcional, como por exemplo, *analytics*, *data storage*, ou *data visualisations*.
- *is-default-component*: booleano que determina se um componente faz parte, por defeito, de todas as arquiteturas IIoT geradas.

Um *entrypoint* é caracterizado pelos mesmos atributos que um componente. O *North Port* e *South Port* foram implementados através de relações, a *northport-protocol-support* e *southport-protocol-support*, respetivamente. Estas relacionam uma tecnologia, que implementa um destes *entrypoints*, com os respetivos protocolos de comunicação que são suportados por esta no *North Port* e no *South Port*.

Na Figura 15 apresentado o *schema* de um componente e de um *entrypoint* e a sua representação em híper-grafo no Grakn, sendo que na Figura 16 são expostos alguns exemplos de instâncias destes dois conceitos.

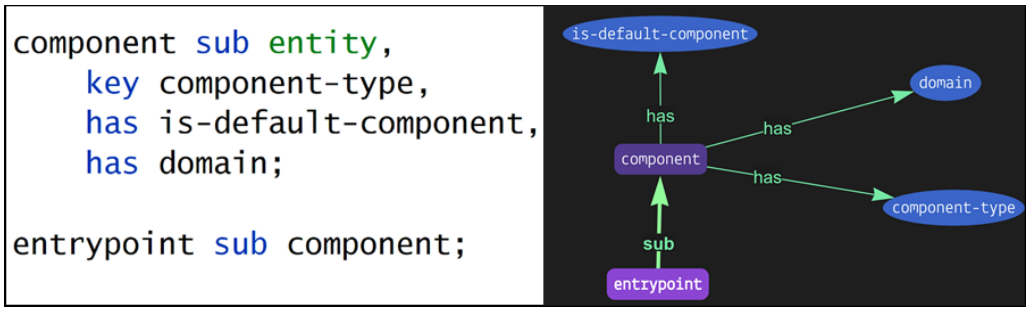

Figura 15 *Schema* de um componente e *entrypoint* e as suas representações em híper-grafo

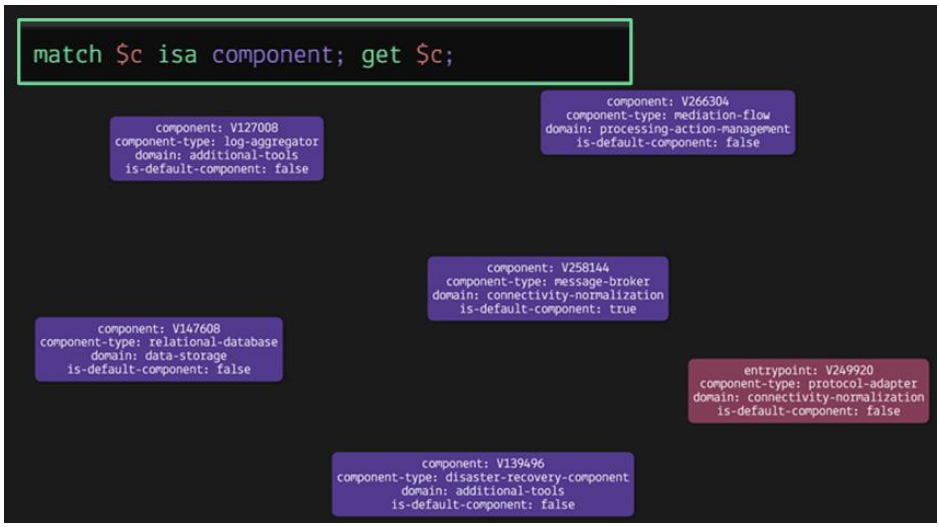

Figura 16 Exemplos de instâncias de componente no modelo semântico e respetiva *query*

Função

Uma função é caracterizada pelo atributo *function-name*, que representa o verbo da função, como por exemplo, *collect*, *present* ou *query*. Os tipos de entidades manipuladas foram implementados diretamente no *schema* como subconceitos de *handled-thing-types*, como é apresentado na Figura 17.

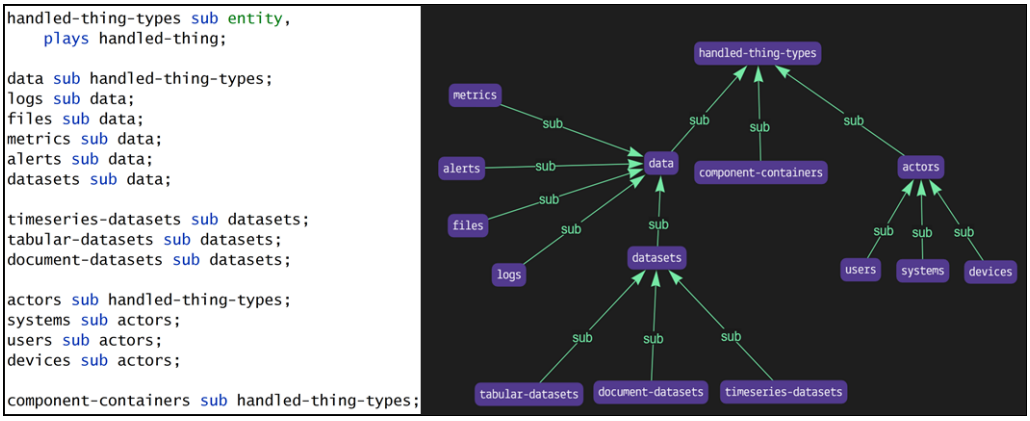

Figura 17 *Schema* das entidades manipuladas e a sua representação em híper-grafo

A implementação dos contextos operacionais seguiu a mesma abordagem, demonstrada na Figura 18, em que cada contexto é um subconceito de *operational-context.*

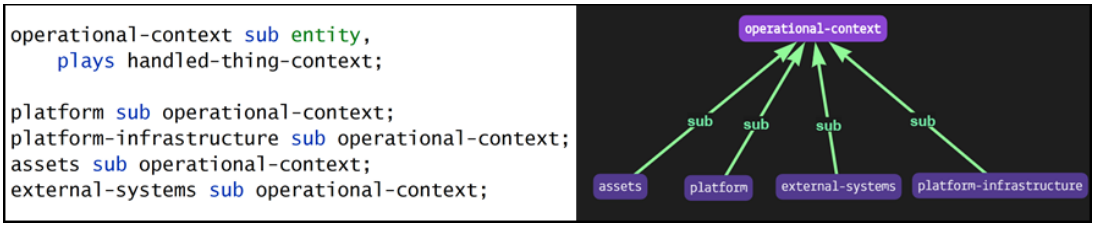

Figura 18 *Schema* dos contextos operacionais e a sua representação em híper-grafo

A contextualização das funções, ou seja, a definição das funções compostas, como apresentando na secção 3.2.2 em Funções, foi representada numa relação, *function-contextualization*. Esta relação é composta por 3 papeis. O *function-handler* que é desempenhado por uma função. O *handled-thing* que corresponde ao um tipo de entidade manipulada. E por fim, o *handled-thing-context* que corresponde ao contexto operacional, relativo à entidade manipulada. Para além disso, esta função é caracterizada pelo atributo *contextualized-function-name* que é composto pela concatenação do nome da função, o tipo de entidade manipulada e o contexto operacional. Alguns exemplos deste atributo são *Present Platform-Infrastructure's Metrics*, *Collect Platform's Logs*, ou, *Connect Assets's Devices*. O *schema* desta relação, assim como a sua representação no Grakn são apresentados na Figura 19. Já a Figura 20, contém alguns exemplos de instâncias desta relação.

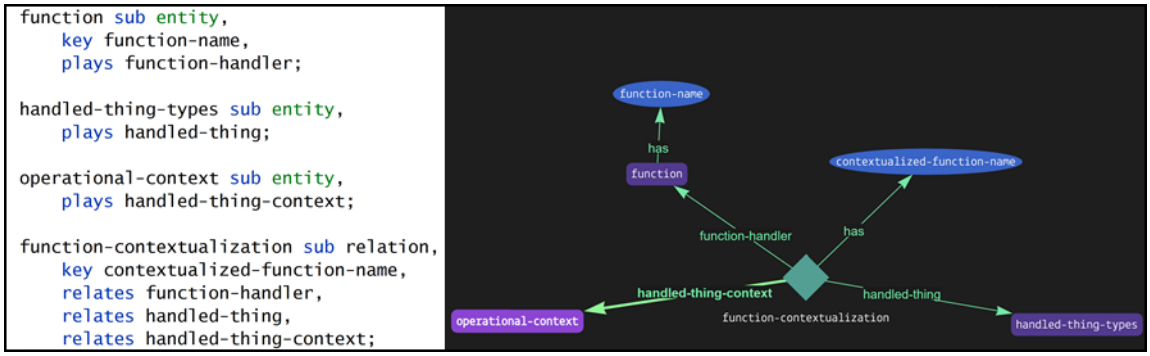

Figura 19 *Schema* da contextualização de funções e a sua representação em híper-grafo

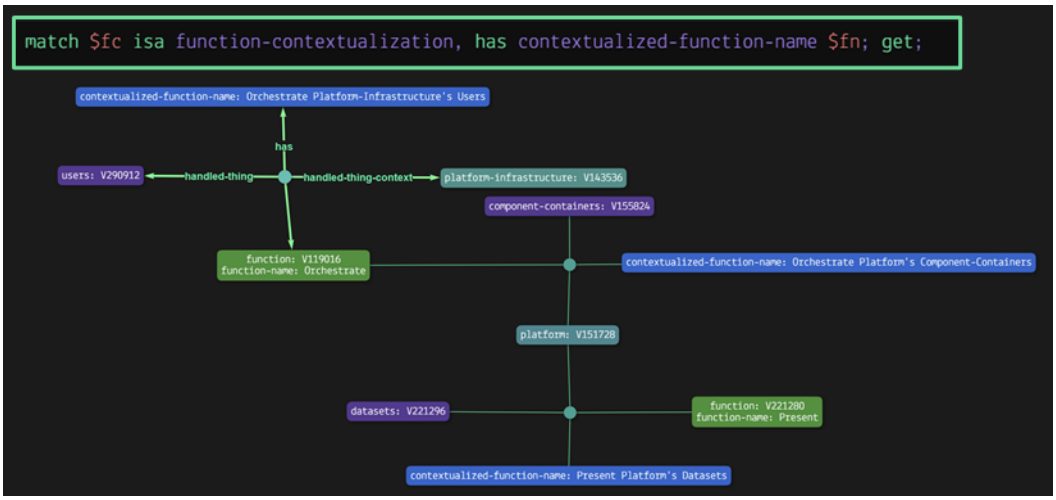

Figura 20 Exemplos de instâncias de funções contextualizadas no modelo semântico e respetiva *query*

Necessidade

Uma necessidade é caracterizada pelos atributos:

- *need-id*: identificador único da necessidade;
- *need-name*: o nome da necessidade, como por exemplo, *trigger platform events;*

- *need-type*: o tipo da necessidade, como por exemplo, *Edge Integration* ou *System Functionality;*
- *need-description*: descrição da necessidade;
- *is-related-to-entrypoint*: booleano que determina se a necessidade está relacionada com um *entrypoint*;
- *is-default-need*: booleano que determina se a necessidade está relacionada com um componente que faz por defeito parte das arquiteturas IIoT.

A Figura 21 apresenta o *schema* de uma necessidade e a sua representação no Grakn, enquanto a Figura 22 contém alguns exemplos de instâncias deste conceito.

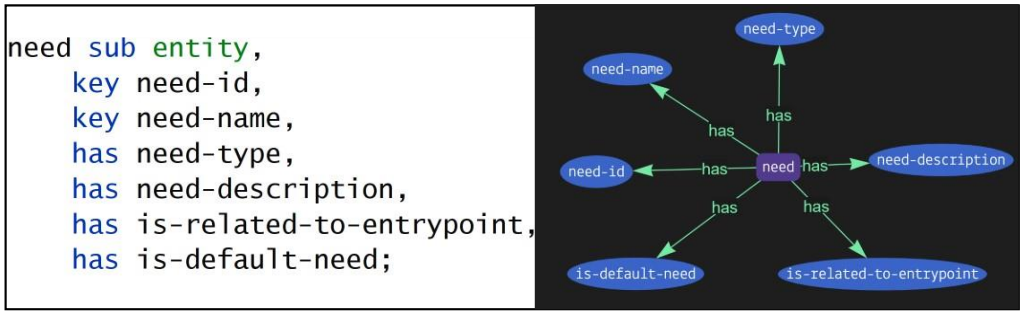

Figura 21 *Schema* de uma necessidade e a sua representação em híper-grafo

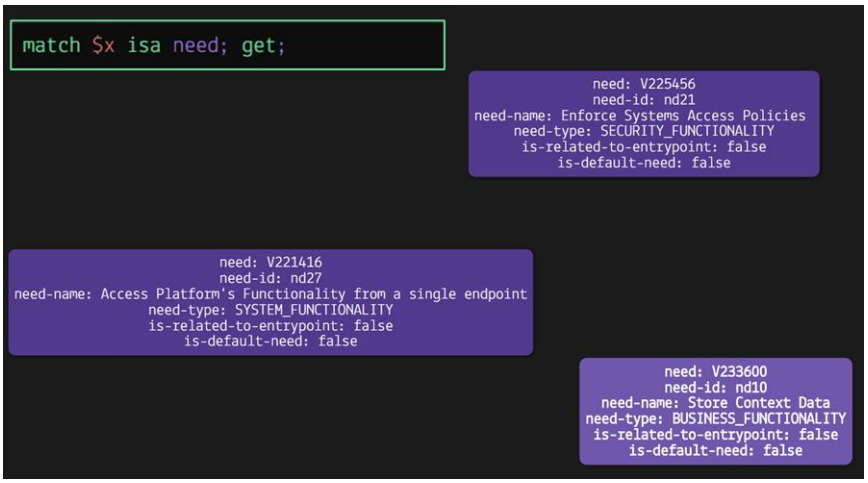

Figura 22 Exemplos de instâncias de necessidades no modelo semântico e respetiva *query*

Tecnologia

Uma tecnologia é caracterizada pelo atributo *technology-name*, que representa o nome da tecnologia, como por exemplo, Apache Kafka, Node-Red e Grafana.

Protocolo

Um protocolo de comunicação e é caracterizado pelos atributos:

- *protocol-name*: o nome do protocolo de comunicação, como por exemplo, MQTT;

- *protocol-type*: o tipo do protocolo, como por exemplo, *Edge.*

Camada

Uma camada da plataforma IIoT é caracterizada pelo atributo, *layer-name*, que representa o nome das camadas, como por exemplo, *Core*, *Edge Gateway* ou *Data*.

A Figura 23 apresenta o *schema* de uma tecnologia, de um protocolo e camada, assim como as suas representações no Grakn. Na Figura 24, Figura 25 e Figura 26 contém alguns exemplos de instâncias de tecnologia, protocolo e camada, respetivamente.

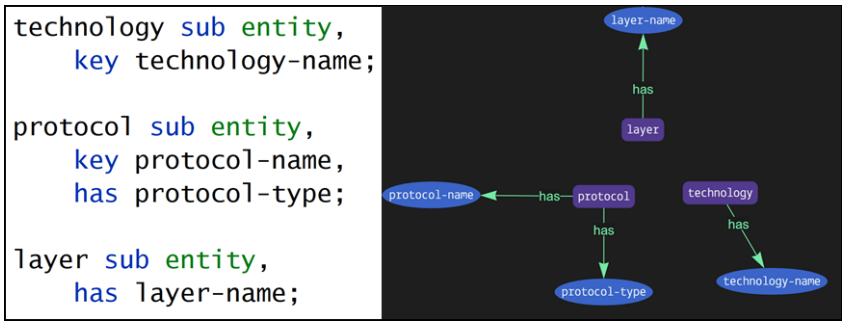

Figura 23 *Schema* de uma tecnologia, protocolo e camada e as suas representações em híper-grafo

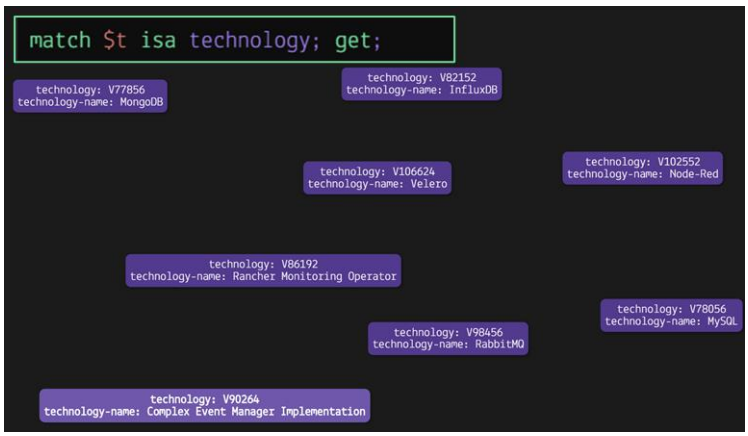

Figura 24 Exemplos de instâncias de tecnologias no modelo semântico e respetiva *query*

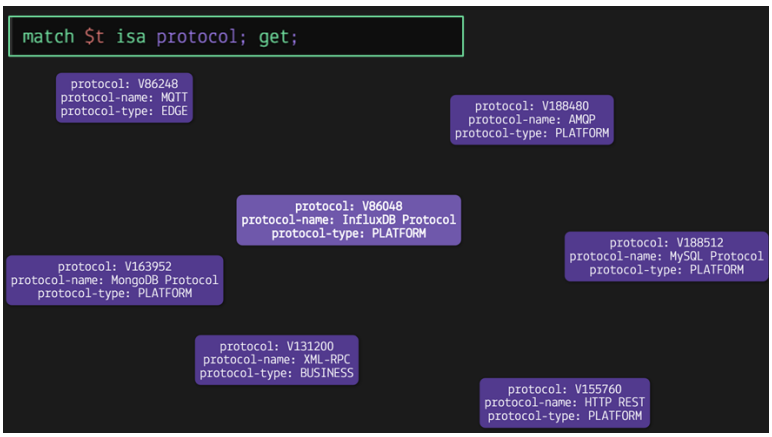

Figura 25 Exemplos de instâncias de protocolos de comunicação no modelo semântico e respetiva *query*

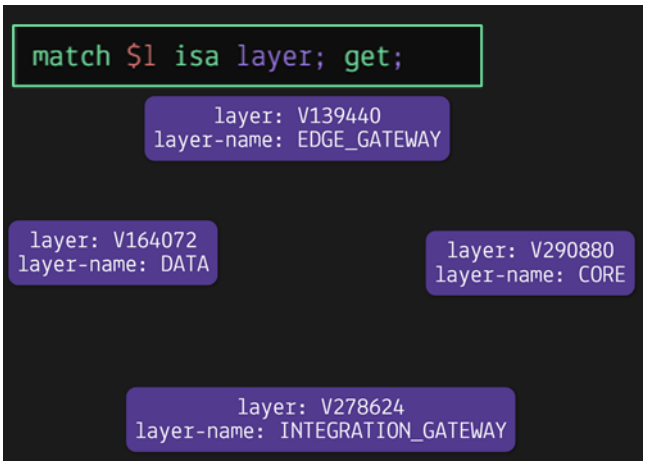

Figura 26 Exemplos de instâncias das camadas de plataformas IIoT no modelo semântico e respetiva *query*

Component-Instance

Um *component-instance* é caracterizado pelos atributos:

- *instance-id*: identificador único da instância;
- *component-type*: o tipo de componente que é instanciado por esta instância, sendo um atributo partilhado com componente;
- *layer-name*: o nome da camada onde a instância é colocada, sendo um atributo partilhado com a camada;
- *unique*: booleano que identifica se a instância é única ou não;
- *is-default-instance*: booleano que identifica se é uma instância de um componente que faz por defeito parte das arquiteturas IIoT.

Um *entrypoint-instance*, para além dos atributos de *component-instance*, é também caracterizado pelo atributo *southport-protocol-name*, que representa o nome do protocolo de comunicação a ser usado no *South Port*. Este atributo é usado para inferir a tecnologia que irá implementar este *entrypoint-instance*, sendo que, no momento em que este é instanciado, já se sabe, à partida, qual será o protocolo de comunicação a utilizar. O *North Port* é representado através da relação *northport-protocol-support* que tenha associada a tecnologia que implementa este *entrypoint-instance*.

Na Figura 27 é apresentado o *schema* de um *component-instance* e de um *entrypoint-instance* e a sua representação em híper-grafo no Grakn, sendo que na Figura 28 são apresentados alguns exemplos de instâncias destes dois conceitos.

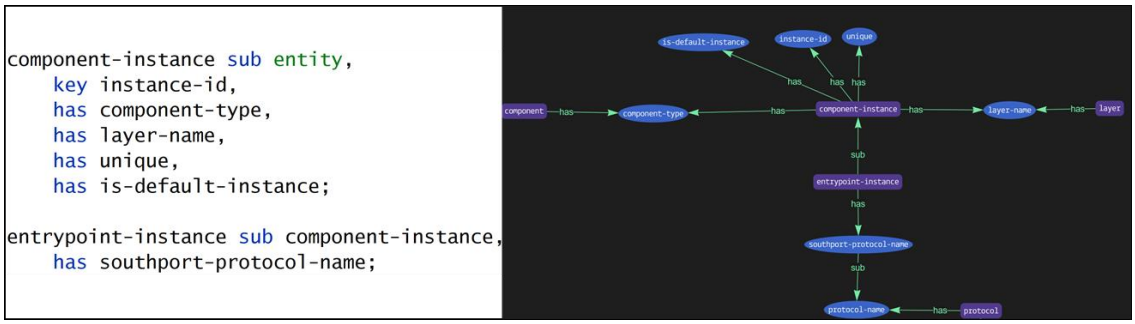

Figura 27 *Schema* de um *component-instance* e *entrypoint-instance* e as suas representações em híper-grafo

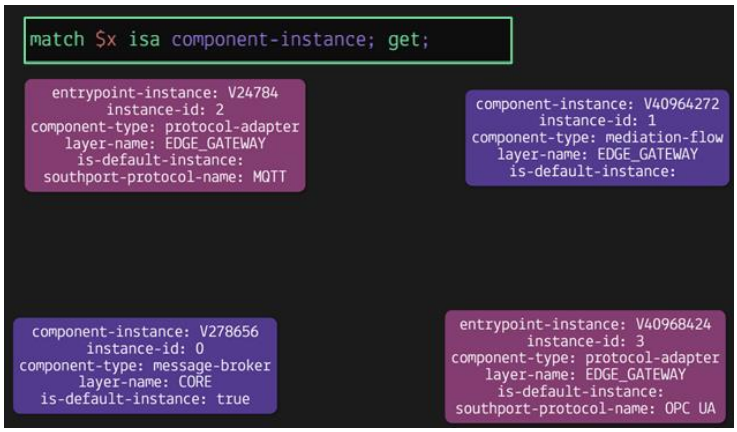

Figura 28 Exemplos de instâncias de *component-instance* no modelo semântico e respetiva *query*

3.3.3. Regras

O processo de inferência depende da definição de regras que formalizem a forma como relacionar os conceitos do modelo semântico de forma a gerar novo conhecimento. Em Grakn, este processo é acionado pela execução de *querys* que questionem conceitos inferidos, ou seja, as regras que inferem um novo conceito só são executadas quando existe uma *query* a esse conceito.

No contexto do domínio do IIoT, foram elencadas um conjunto de restrições de domínio, apresentadas na Tabela 3. Para além disso, foram também definidos um conjunto de mapeamentos entre conceitos, nomeadamente, entre os tipos de necessidades, e, os tipos de protocolos de comunicação e as camadas da plataforma IIoT, estando estes sistematizados na Tabela 4.

Objetivo	Lógica Associada
Identificação dos componentes que colmatam uma necessidade	Dada uma necessidade *n* e um componente *c*, então, *addresses(c,n)* se e só se todas as funções *f* associadas à necessidade *n* são funções do componente *c*.
Identificação de comunicação entre *component-instance*s	A comunicação entre *component-instance*s deverá ocorrer entre *component-instance*s que estejam na mesma camada, ou com instâncias da camada Core.
Identificação de *component-instance*s por defeito	Os componentes marcados como *default* deverão ser instanciados em todas as arquiteturas a especificar pela solução.
Identificação de dependências entre componentes	Um *component-instance* de um componente (*depender*) que dependa de outro componente (*dependee*), do qual não exista um component-instnace, então o mesmo deverá ser criado.
Identificação de dependências entre implementações	Um componente (*depender*), que tenha a implementação *y*, e que dependa da implementação *x* de um outro componente (*dependee*), se existir um *component-instance* do componente *depender* e do *dependee*, e se o *component-instance* do *depender* tem a implementação *y*, então o *component-instance* do *dependee* deverá ter a implementação *x*.

Tabela 3 Restrições do domínio do IIoT

Objetivo	Lógica de Mapeamento	
Mapear o tipo de necessidade com o tipo de protocolo a utilizar por um *component-instance*	**Tipo de Necessidade**	**Tipo de Protocolo**
	Edge Integration	*Edge*
	Business_Integration	*Business*
	Business_Functionality	*Platform*
	System_Functionality	*Platform*
	Security Functionality	*Platform*
Mapear o tipo de necessidade com a camada em que é colocado o *component-instance*	**Tipo de Necessidade**	**Camada**
	Edge Integration	*Edge Gateway*
	Business_Integration	*Integration Gateway*
	Business_Functionality	*Core* ou *Data* (determinado pela camada do componente associado ao *component-instance*)
	System_Functionality	*Core*
	Security Functionality	*Core*

Tabela 4 Mapeamento entre as necessidades e os protocolos e camadas da plataforma IIoT

Por fim, na Tabela 5, são apresentadas as regras de inferência formalizadas no modelo semântico e os seus respetivos objetivos.

Objetivo	Regras

Inferir as ligações lógicas entre *component-instances* que estejam na mesma camada	`create-instances-logical-connections-same-layer sub rule,`
	`  when {`
	`  $i1 isa component-instance, has layer-name $ln1;`
	`  $i2 isa component-instance, has layer-name $ln2;`
	`  (instanced-component: $c1, instance: $i1) isa component-instanciation;`
	`  (instanced-component: $c2, instance: $i2) isa component-instanciation;`
	`  $ln1 == $ln2;`
	`  (connected-component: $c1, connected-component: $c2) isa logical-connection;`
	`  }, then {`
	`  (connected-instance: $i1, connected-instance: $i2) isa instance-logical-connection;`
	`  };`

Inferir as ligações lógicas entre *component-instances* que estejam em camadas diferentes, sendo uma destas o Core	`create-instances-logical-connections-core sub rule,`
	`  when {`
	`  $i1 isa component-instance, has layer-name $ln1;`
	`  $i2 isa component-instance, has layer-name "CORE";`
	`  (instanced-component: $c1, instance: $i1) isa component-instanciation;`
	`  (instanced-component: $c2, instance: $i2) isa component-instanciation;`
	`  (connected-component: $c1, connected-component: $c2) isa logical-connection;`
	`  }, then {`
	`  (connected-instance: $i1, connected-instance: $i2) isa instance-logical-connection;`
	`  };`

Inferir a implementação das ligações entre *component-instances*	`create-instances-connection-implementation sub rule,`
	`  when {`
	`  $i1 isa component-instance;`
	`  $i2 isa component-instance;`
	`  (connected-instance: $i1, connected-instance: $i2) isa instance-logical-connection;`
	`  (technology-instance-implementer: $t1,implemented-instance: $i1) isa instance-implementation;`
	`  (technology-instance-implementer: $t2,implemented-instance: $i2) isa instance-implementation;`
	`  (supporter: $t1, supported-protocol: $sp) isa protocol-support, has supported-integration $sdf1 ;`

	(supporter: $t2, supported-protocol: $sp) isa protocol-support, has supported-integration $sdf2 ;
	$sdf1 == "request";
	$sdf2 == "provide";
	}, then {
	(instance-provider: $i2, instance-requester: $i1, requested-protocol: $sp) isa instance-connection-implementation;
	};

Associar instâncias de componentes as respetivas *component-instances*	create-component-instantiation sub rule,
	when{
	$c isa component, has component-type $ct1;
	$i isa component-instance, has component-type $ct2;
	$ct1 == $ct2;
	}, then {
	(instanced-component: $c, instance: $i) isa component-instantiation;
	};

Determinar que *component-instances* são únicas	mark-unique-instances sub rule,
	when{
	$c isa! component;
	$i1 isa! component-instance, has layer-name $ln1;
	(instanced-component: $c, instance: $i1) isa component-instantiation;
	not { $i2 isa! component-instance;
	$i2 has layer-name $ln2;
	$i1 != $i2;
	$ln1 == $ln2;
	(instanced-component: $c, instance: $i2) isa component-instantiation;
	};
	}, then {
	$i1 has unique true;
	};

Determinar que *component-instances* são equivalentes	check-for-equivalent-inserted-instances sub rule,
	when{
	$c isa! component;
	$i1 isa! component-instance, has layer-name $ln1;
	$i2 isa! component-instance, has layer-name $ln2;
	$i1 != $i2;
	$ln1 == $ln2;
	(instanced-component: $c, instance: $i1) isa component-instantiation;
	(instanced-component: $c, instance: $i2) isa component-instantiation;
	}, then {
	($i1, $i2) isa equivalent-instances;
	};

Determinar que *component-instances* tem mais que uma implementação	select-instances-that-have-more-than-one-implementation sub rule,
	when {
	$i isa! component-instance;
	(instanced-component: $c, instance: $i) isa component-instantiation;
	$t1 isa technology, has technology-name $x;
	$t2 isa technology, has technology-name $y;
	$x != $y;
	(technology-implementer: $t1, implemented-component: $c) isa component-implementation;
	(technology-implementer: $t2, implemented-component: $c) isa component-implementation;
	}, then{
	(technology-instance-implementer: $t1, implemented-instance: $i) isa instance-multi-implementation;
	};

Determinar que *component-instances* tem uma única implementação	select-single-implementation-for-instances sub rule,
	when {
	$i isa! component-instance;
	$c isa! component;
	not{(implemented-instance: $i) isa instance-multi-implementation;};
	(instanced-component: $c, instance: $i) isa component-instantiation;
	(technology-implementer: $t, implemented-component: $c) isa component-implementation;
	}, then {
	(technology-instance-implementer: $t, implemented-instance: $i) isa instance-implementation;
	};

Determinar implementação de um *entrypoint-instance*	`create-entrypoint-instance-implementation sub rule,`
	`  when {`
	`  $ei isa entrypoint-instance, has southport-protocol-name $spn;`
	`  (instance: $ei, instanced-component: $c) isa component-instantiation;`
	`  (technology-implementer: $t, implemented-component:$c) isa component-implementation;`
	`  (supporter: $t, southport-supported-protocol: $p) isa southport-protocol-support;`
	`  $p isa protocol, has protocol-name $pn;`
	`  $spn == $pn;`
	`  }, then {`
	`  (implemented-instance: $ei, technology-instance-implementer: $t) isa instance-implementation;`
	`  };`

Determinar dependências entre *component-instances*	`create-instance-dependency sub rule,`
	`  when{`
	`  (component-dependee: $c1,  component-depender: $c2) isa component-dependency;`
	`  (instanced-component: $c1, instance: $i1) isa component-instantiation;`
	`  (instanced-component: $c2, instance: $i2) isa component-instantiation;`
	`  }, then {`
	`  (instance-dependee: $i1, instance-depender: $i2) isa instance-dependency;`
	`  };`

Criar uma solicitação para a criação de *component-instances*, que participam como *dependee* numa relação de dependência entre componentes, e que não existem	`create-instance-dependee-creation-if-dependee-doesnt-exist sub rule,`
	`  when{`
	`  (component-dependee: $c1,  component-depender: $c2) isa component-dependency;`
	`  $c1 isa component, has component-type $ct1;`
	`  $c2 isa component, has component-type $ct2;`
	`  not { $i1 isa component-instance, has component-type $it1;`
	`  $it1 == $ct1;`
	`  };`
	`  $i2 isa component-instance, has component-type $it2;`
	`  $it2 == $ct2;`
	`  (layer-composer: $l, composed-component: $c1 ) isa layer-component-composition;`
	`  }, then {`
	`  (instanciate-to-layer: $l, component-to-instanciate: $c1) isa instance-dependee-creation;`
	`  };`

Determinar a implementação de um *component-instance* quando este é um *dependee* de uma relação de dependência de implementações	`select-implementation-for-dependee-instances sub rule,`
	`  when {`
	`  $i1 isa! component-instance;`
	`  $i2 isa! component-instance;`
	`  (instanced-component: $c1, instance: $i1) isa component-instantiation;`
	`  (instanced-component: $c2, instance: $i2) isa component-instantiation;`
	`  $impl1 (technology-implementer: $t1, implemented-component: $c1) isa component-implementation;`
	`  $impl2 (technology-implementer: $t2, implemented-component: $c2) isa component-implementation;`
	`  (implementation-dependee: $impl1, implementation-depender: $impl2) isa implementation-dependency;`
	`  (implemented-instance: $i1) isa instance-multi-implementation;`
	`  (implemented-instance: $i2, technology-instance-implementer: $t2) isa instance-implementation;`
	`  }, then {`
	`  (technology-instance-implementer: $t1, implemented-instance: $i1) isa instance-implementation;`
	`  };`

Tabela 5 Regras de Inferência do Modelo Semântico

3.3.4. Processo de especificação semântica de uma arquitetura IIoT

A especificação de uma arquitetura IIoT é um processo iterativo. O processo é despoletado aquando da identificação e seleção das necessidades/requisitos para as quais a solução reúne uma ou mais opções, obtidas por intermédio de regras de inferência. Com base nas opções o utilizador toma a decisão e uma nova iteração inicia. Após a iteração final, é gerado um resultado com base nas informações apuradas anteriormente, sendo este uma arquitetura específica às necessidades de uma determinada organização.

Para além da identificação das necessidades/requisitos, as decisões a tomar pelo utilizador podem ser dos seguintes tipos:

- *Component Decision:* escolher um componente entre dois ou mais componentes que permitem colmatar uma determinada necessidade.
- *Technology Decision:* escolher entre duas ou mais tecnologias que permitem implementar a mesma *component-instance*.
- *Entrypoint Protocol Decision:* escolher 1 ou mais protocolos de comunicação entre os protocolos suportados no *South Port* de um *entrypoint-instance* a instanciar.

A primeira iteração inerente ao processo de especificação de uma arquitetura consiste na identificação/seleção de necessidade/requisitos, conforme representado na Figura 29.

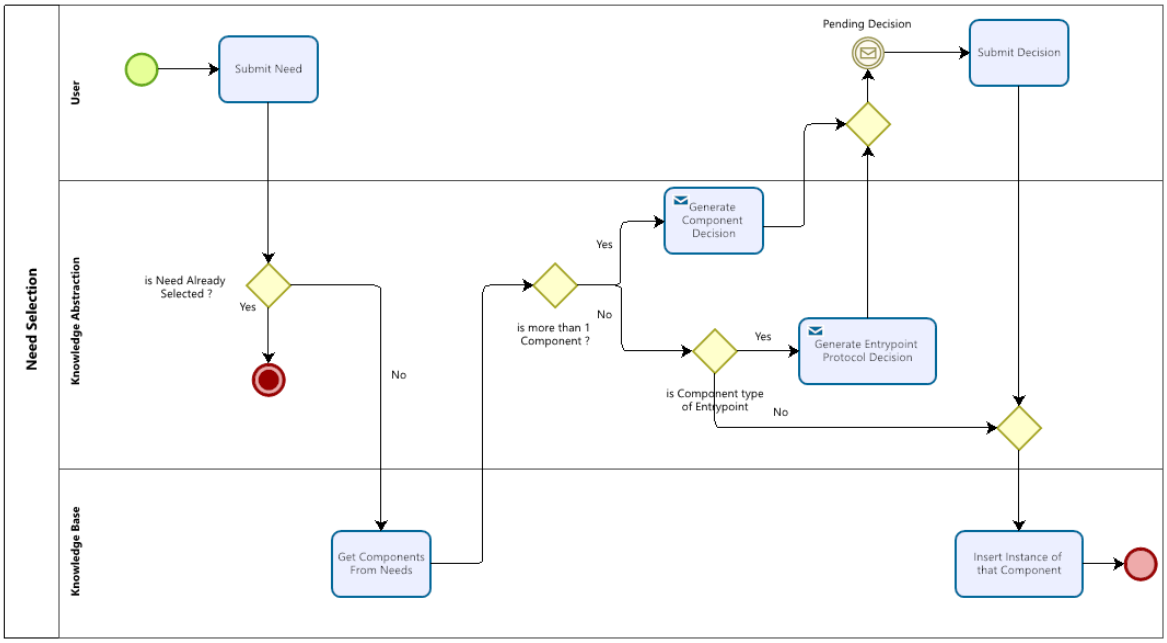

Figura 29 *Workflow* de seleção de necessidades

Esta fase, pressupõe que o utilizador faça um pedido *GET* à *knowledge abstration*, que, por sua vez, executa uma *query* na *knowledge base* para obter todas as necessidades de um determinado tipo. A Figura 30 representa um exemplo desse pedido recorrendo à ferramenta *Postman*[4].

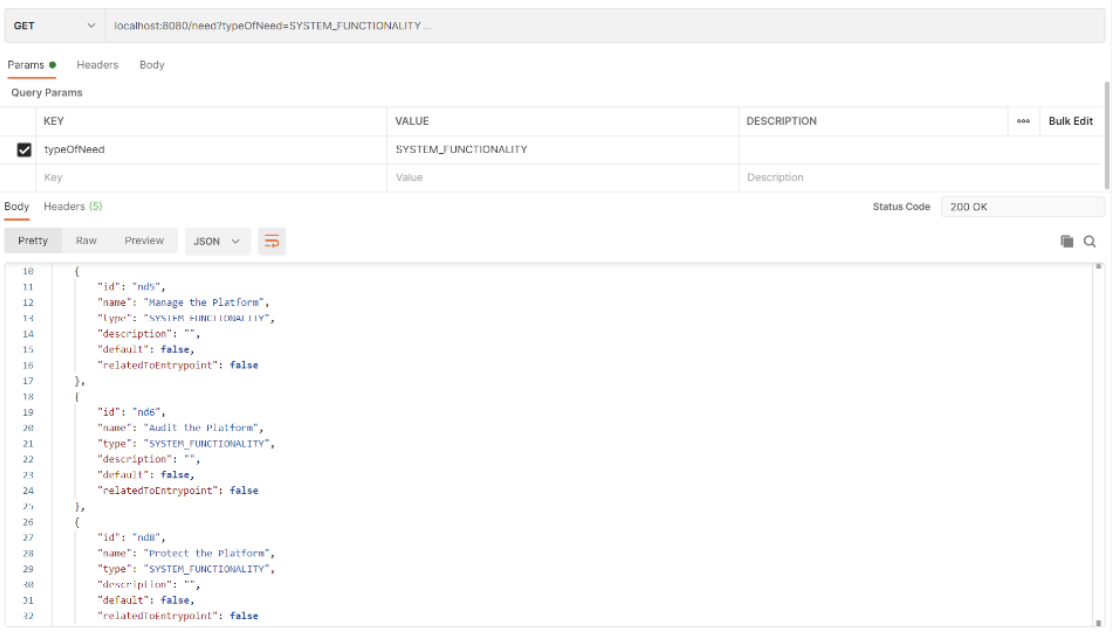

Figura 30 Exemplo de pedido para obtenção das necessidades colmatadas

Posteriormente, o utilizador submete as necessidades que pretende colmatar com a plataforma IIoT. Esta submissão é feita através de um pedido *POST* à *knowledge abstration*, contendo as necessidades selecionadas, cujo um exemplo é exibido na Figura 31.

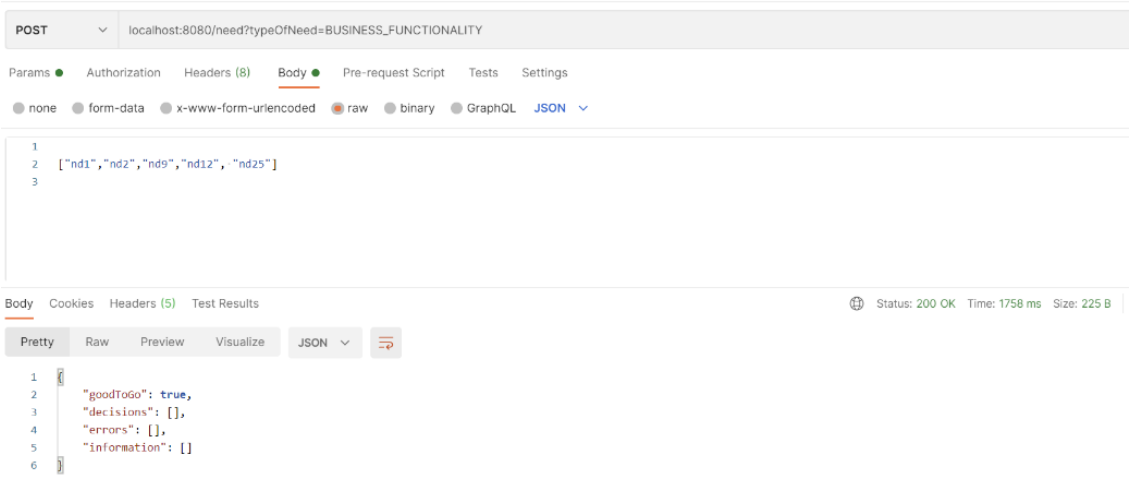

Figura 31 Exemplo de pedido de submissão de necessidades

Na *knowledge abstraction* é verificado, para cada uma das necessidades selecionadas, se esta já foi escolhida previamente. Se sim, não é necessário selecionar outra vez, e termina esta fase,

[4] https://www.postman.com/

caso contrário executa uma *query* na *knowledge base*. A *query* executada permite obter os componentes que são caracterizados por todas funções que enderençam uma determinada necessidade selecionada. A *query*, já tem em conta as funções extras que são adicionadas aos componentes por uma determinada tecnologia que os implemente. Obtido o resultado da *query*, se existir mais do que um componente que possa colmatar aquela necessidade, então é criada uma *Component Decision* para que o utilizador possa escolher entre esses componentes. Se não, é verificado se o componente é do tipo *entrypoint* ou não. No caso de o componente ser do tipo *entrypoint*, é gerada uma *Entrypoint Protocol Decision*, que está relacionada com o protocolo de comunicação a ser usado por este no seu *South Port*. Se não, é criada um *component-instance* daquele componente, e termina o processo.

No caso de terem sido geradas decisões para o utilizador tomar, se for uma *Compent Decision*, após a escolha de um dos componentes, então é criada um *component-instance* desse componente. Se for uma *Entrypoint Protocol Decision*, por cada protocolo diferente selecionado, é instanciado um *entrypoint-instance* em que o seu *South Port* é igual a esse protocolo.

Após a seleção das necessidades, segue-se a segunda fase do processo (Figura 32), ou seja, a obtenção da arquitetura específica. Nesta etapa o utilizador solicita o resultado que se traduz num pedido *GET* à *knowledge abstraction*, conforme representado na Figura 33.

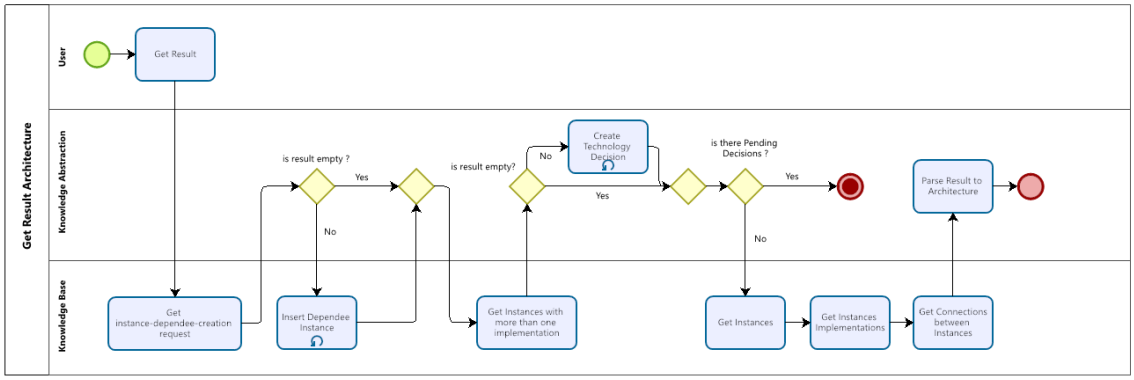

Figura 32 *Workflow* de obtenção da arquitetura IIoT específica

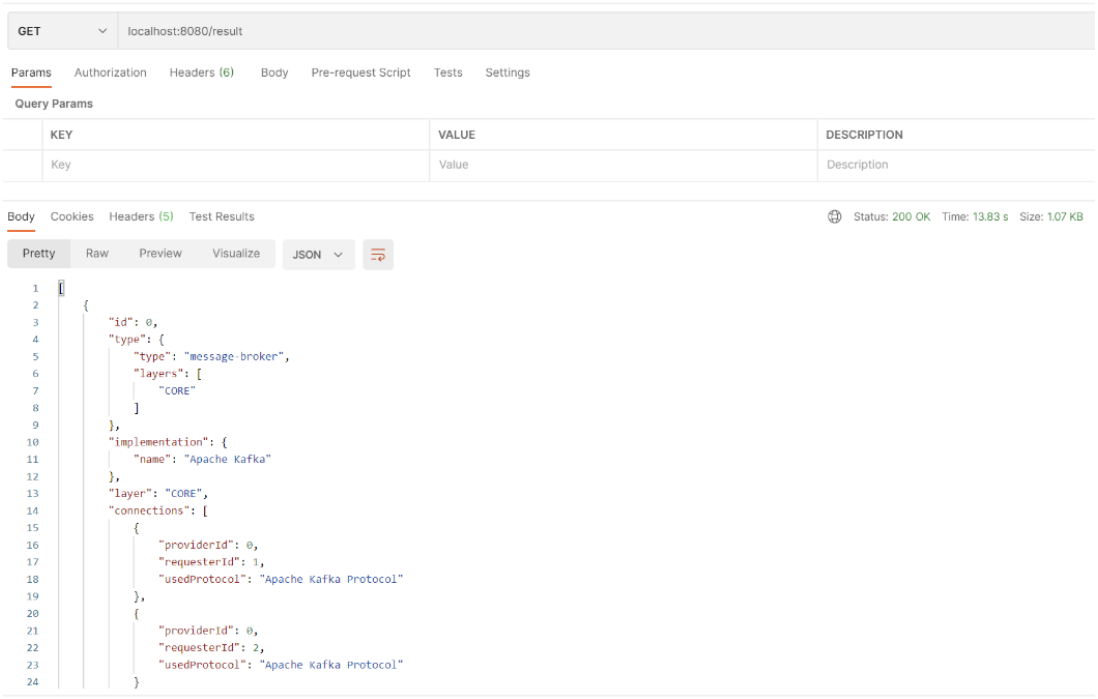

Figura 33 Exemplo de pedido para obter arquitetura específica

Imediatamente após o pedido do utilizador, é verificado pela *knowledge abstraction*, através de uma *query* à *knowledge base*, se existe algum caso de dependência de componentes na arquitetura específica em que seja necessário instanciar um *component-instance* que seja um *dependee*. Esta verificação teve que ser feito do lado da *knowledge abstraction* dada a uma limitação no motor de regras do Grakn, sendo que a mesma será abordada em mais detalhe na secção seguinte. Este processo inicia-se com a *query* realizada que aciona uma regra no Grakn que cria uma solicitação para a criação de *component-instance*s (a relação *instance-dependee-creation*). Se o resultado obtido não estiver vazio, então é inserido, pela *knowledge abstraction*, um *component-instance* segundo o tipo de componente a instanciar e a camada, presentes em cada solicitação.

Para além disso, é também averiguado se existe alguma *Technology Decision*. Isto é verificado através de uma *query* à *knowledge base* que obtém todas os *component-instance*s que podem ser implementados por mais do que uma tecnologia. Nesta *query* já são tidas em conta os casos em haja dependências entre implementações. Se resultado não for vazio, então por cada *component-instance*s é gerada uma *Technology Decision* para escolher a sua implementação. Após a tomada da decisão, é inserida na *knowledge base* a implementação escolhida por parte do utilizador para aquela instância.

Se existirem decisões pendentes, o processo é terminado, e é avisado o utilizador que ainda existem decisões pendentes. Se não, um conjunto de *querys* são executadas na *knowledge base*, que acionam a execução das regras de inferência, nomeadamente, obter todos os *component-*

*instance*s que são únicos e aqueles que são equivalentes. No caso de *component-instance*s equivalentes, é escolhida um desses *component-instance*s aleatoriamente. De seguida, são obtidas as implementações destes *component-instance*s e as conexões entre os mesmos. No final, é obtido a especificação da arquitetura da plataforma IIoT, sendo esta constituída pelos *component-instance*s, as tecnologias que os implementam e as ligações entre os mesmos, através dos respetivos protocolos de comunicação. Na Figura 34, é apresentado um excerto de uma especificação, em JSON, obtida a partir da solução.


```
[
  {
    "id": 0,
    "type": {
      "type": "message-broker",
      "layers": [
        "CORE"
      ]
    },
    "implementation": {
      "name": "Apache Kafka"
    },
    "layer": "CORE",
    "connections": [
      {
        "providerId": 0,
        "requesterId": 1,
        "usedProtocol": "Apache Kafka Protocol"
      },
      {
        "providerId": 0,
        "requesterId": 2,
        "usedProtocol": "Apache Kafka Protocol"
      }
    ]
  },
  {
    "id": 1,
    "type":{
      "type": "protocol-adapter",
      "layers": [
        "EDGE_GATEWAY",
        "INTEGRATION_GATEWAY"
      ]
    },
    "implementation": {
      "name": "OPC UA Protocol Adapter Implementation"
    },
    "layer": "EDGE_GATEWAY",
    "connections": [
      {
        "providerId": 0,
        "requesterId": 1,
        "usedProtocol": "Apache Kafka Protocol"
      }
    ],
    "southPortProtocol": {
      "name": "OPC UA",
      "type": "EDGE"
    }
  }
(…)
]
```


Figura 34 Excerto de uma especificação de uma arquitetura IIoT

3.4. Conclusões

Após a conceptualização e formalização deste modelo semântico já é possível chegar a algumas conclusões sobre o mesmo.

O uso de Grakn permitiu acelerar o processo de formalização do modelo, como era esperado, graças ao facto do seu formato de modelação ser intuitivo e muito próximo dos modelos entidade-relação. No entanto, esta tecnologia também fez com que surgissem entraves que impediram o desenvolvimento de um modelo mais inteligente e autónomo.

Um dos entraves encontrados foi o facto que em Grakn as regras apenas permitem a realização de uma única operação atómica por regra. Neste caso, uma operação atómica é, por exemplo, a criação de uma relação entre instâncias de conceitos, ou, a criação de uma instância de um conceito. O facto de ser apenas possível realizar uma destas operações por regra não é o ideal, mas o problema torna-se mais crítico quando é necessário criar uma instância e preencher os seus atributos. Este problema surgiu quando se tentou automatizar a inserção de um *dependee*, no caso de uma dependência entre componentes, através de uma regra no Grakn, tentando satisfazer a restrição de dependências entre componentes apresentada na subsecção de regras. O que se verificou é que para criar uma instância de um conceito, neste caso *component-instance*, e em que era necessário preencher 3 atributos, então seria necessário criar 3 regras diferentes. Uma das regras permitiria a criação da instância e preenchimento de um dos atributos, e as outras duas regras para o preenchimento dos outros dois atributos. Isto levaria a que o processo de inferência fosse mais demorado, considerando que em vez de executar uma regra onde se criava a instância e preenchia os atributos necessários, é necessário executar, sequencialmente, *n* regras para preencher *n* atributos de uma instância. Neste caso concreto, esta solução era inviável porque não há uma forma de fazer o *tracking* da instância durante a execução das 3 regras. Concretamente, após a criação da instância na primeira regra, não era possível preencher os atributos da instância criada com os valores corretos, de forma sistemática. Este constrangimento poderia ser resolvido se o Grakn permitisse, pelo menos, preencher mais do que um atributo de uma instância, na mesma regra. A situação foi contornada com o "apoio" da *knowledge abstraction*, ou seja, do lado do Grakn existe uma regra que consegue determinar se existem *component-instance* que participam como *dependers* numa relação de dependência entre componentes, e que não existem os *dependees*. Esta regra sinaliza estes casos com uma relação, *instance-dependee-creation*, que funciona como uma espécie de solicitação que contém os atributos necessários para a criação do *component-instance dependee*. A *knowledge abstraction* procura por esta relação, e se existir, cria um *component-instance* com base nos atributos obtidos, instanciando o *dependee*.

Outro dos constrangimentos encontrados foi na linguagem de *querys* do Grakn, o Graql, uma vez que não é possível fazer uma interseção entre conceitos diferentes, cujo um caso hipotético é apresentado na Figura 35.

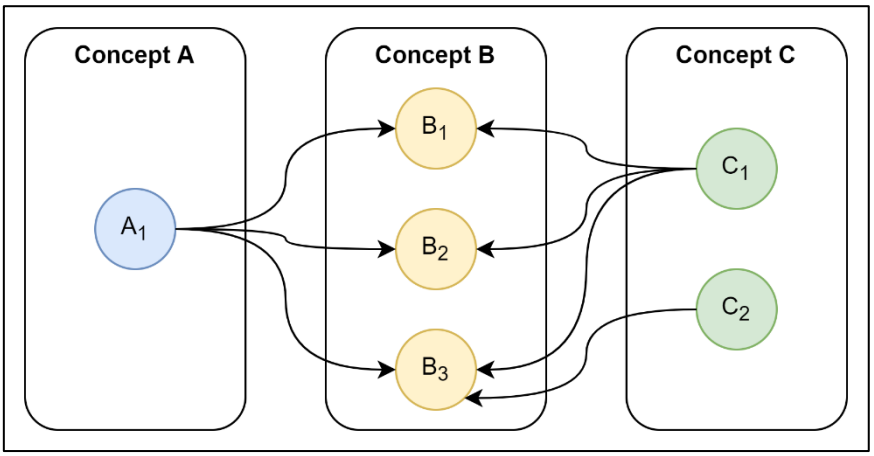

Figura 35 Exemplo da limitação de interseção do Graql

Considerando que existem 3 conceitos, A, B e C e A1 é instância do conceito A; B1, B2 e B3 são instâncias do conceito B, e; C1 e C2 do conceito C. A instância A1 está relacionada com as instâncias B1, B2 e B3; C1 também está relacionada com B1, B2 e B3, e; C2 está apenas relacionada com B3.

Dado A1, pretende-se obter as instâncias do conceito C que estejam relacionadas com todas as instâncias de B comuns a A1. O resultado esperado seria a instância C1, no entanto, o resultado obtido em Grakn é C1 e C2.

Fazendo uma analogia com a teoria dos conjuntos, em que se considera as instâncias de B (B1, B2 e B3) como conjuntos e as instâncias de C são os membros desses conjuntos. O que acontece no Grakn, por defeito, é que este tipo de *querys* são tratadas como uma união, ou seja, obtemos as instâncias de C que sejam membros de B1, B2, B3 ou de todos. O que se apurou é que não existe forma de se fazer uma interseção, ou seja, obter as instâncias de c que sejam membros de B1, B2 e B3. Na prática, todas as instâncias de C que estejam relacionadas com pelo menos uma instância de B, que por sua vez esteja relacionada com A1, já faz parte do resultado.

A forma encontrada para contornar esta desvantagem foi dividir a *query* em duas. Primeiro obtemos todas as instâncias do conceito B que estejam relacionadas com A1. De seguida, é construída uma segunda *query*, em que, por cada instância de B, é gerada uma *subquery* que obtenha uma instância C que esteja relacionada com a tal instância de B. Estas *subquery* são concatenadas com o operador *and*, e o que permite obter as instâncias que estão relacionadas com todas estas instâncias de B obtidas. Um exemplo das duas *querys* encontra-se na Figura 36.

```
## First Query
match
instanceB isa B;
A1 isa A;
(instanceB, A1) isa AB-relation;
get instanceB; ## Returns B1, B2, B3

## Second Query
match
instanceC isa C;
(B1, instanceC) isa BC-relation;
(B2, instanceC) isa BC-relation;
(B3, instanceC) isa BC-relation;
get Cinstance; ## Returns C1

## Obs: ; works like operator and
```

Figura 36 Exemplo das *querys* necessárias para contornar a limitação de interseção do Graql

Este constrangimento impediu a obtenção dos componentes que colmatam as necessidades selecionadas através de uma regra de domínio na base de conhecimento, dado que não é possível fazer uma interseção das funções relacionadas com ambos os conceitos. Apesar de estar definido a nível conceptual que este processo seria realizado desta forma o mesmo não foi possível por causa desta limitação, e teve que ser delegado para a *knowledge abstraction*, utilizando a estratégia da divisão em duas *querys*.

Esta limitação levou a que parte da lógica, que estaria modelado em regras, na *knowledge base*, tivesse que ser delegada *knowledge abstraction*, criando uma maior dependência entre estes dois componentes da solução do que o que seria de esperar. Apesar disto, a utilidade da solução de um modo global não foi afetada, por isso não é previsto que estas limitações/entraves afetem o resultado final na fase de aplicação num caso de estudo.

4. Caso de Estudo

Com o intuito de avaliar o modelo desenvolvido no âmbito desta dissertação, o mesmo foi aplicado num caso de estudo. A avaliação consiste na verificação da similaridade entre uma arquitetura resultante do modelo e a arquitetura IIoT já existente, desenvolvida para o projeto do caso de estudo.

Neste capítulo é descrito em detalhe o caso de estudo, de que forma é que será avaliada a arquitetura IIoT gerada pelo modelo e os resultados obtidos.

4.1. Descrição do Caso de Estudo

O caso de estudo insere-se no projeto FAMEST[5], no âmbito dos instrumentos de financiamento do Portugal 2020. Este projeto tem como objetivo mobilizar e dinamizar a indústria do calçado em Portugal, em que uma das suas tarefas passava pelo desenvolvimento de uma prova de conceito de uma plataforma IIoT para as pequenas e médias empresas do setor. Esta plataforma IIoT, cuja arquitetura se encontra finalizada, será o caso de estudo que visa validar a solução desenvolvida, sendo que esta plataforma é descrita em detalhe em [7]. Na Figura 37 é apresentado o diagrama da arquitetura da plataforma IIoT do caso de estudo.

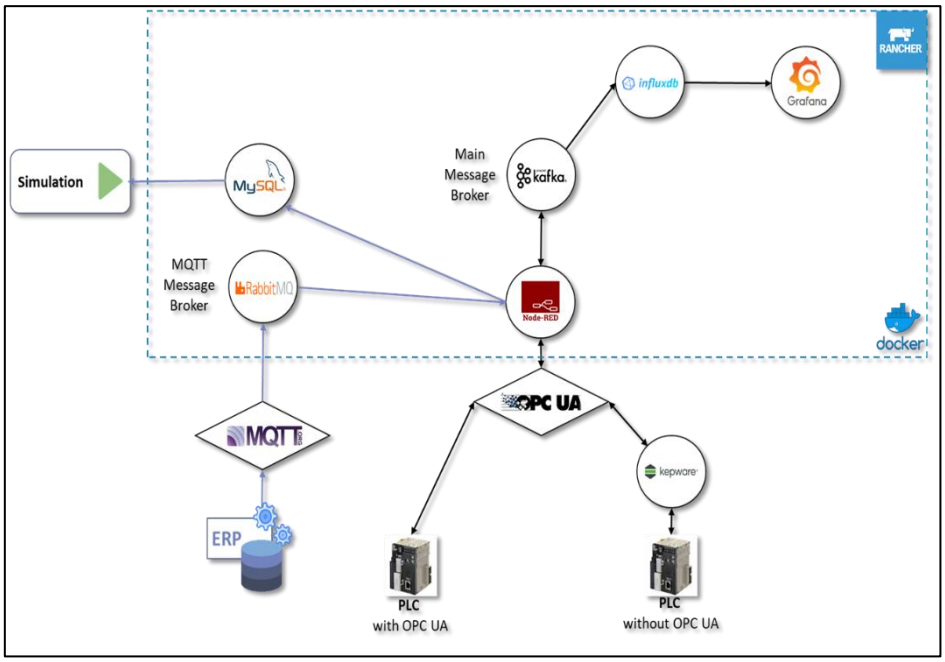

Figura 37 Arquitetura Plataforma IIoT FAMEST

Esta plataforma foi validada em 3 casos de usos. O primeiro e o segundo caso de uso consistia na monitorização, em tempo real, de variáveis de um forno de reativação, que não suporta o

[5] https://famest.ctcp.pt/

protocolo de comunicação OPC UA, e de uma máquina CNC, com suporte OPC UA. No caso do forno de reativação foi utilizado a ferramenta Kepware para criar uma ponte entre o forno e a plataforma. O último caso uso consistiu na integração de um ERP com a plataforma IIoT para alimentar um modelo de simulação.

Na Tabela 6 é apresentada a *stack* tecnológica desta plataforma IIoT e os respetivos papeis que estas tecnologias desempenham na arquitetura.

Tecnologia	Papel
Kepware	Conectar dispositivos que não suportem OPC UA com a plataforma IIoT
Node-Red	Comunicação com os dispositivos, usando OPC UA
	Comunicação com o ERP, usando MQTT
	Reencaminhar dados de dispositivos para o Apache Kafka
	Popular o MySQL com os dados do ERP
RabbitMQ	Conectar o ERP com a plataforma IIoT através de MQTT
Apache Kafka	Popular o InfluxDB com os dados dos dispositivos
MySQL	Armazenar dados de contexto oriundos do ERP
InfluxDB	Armazenar dados de telemetria oriundos dos dispositivos
Grafana	Apresentar *dashboards* dos dados dos dispositivos
Rancher	Orquestração dos serviços da plataforma IIoT

Tabela 6 Caracterização das tecnologias e dos papeis que desempenham na arquitetura do caso de estudo

4.2. Metodologia de Avaliação

A metodologia de avaliação consiste na verificação de quão próxima a arquitetura gerada pela solução é da arquitetura do caso de estudo. Esta é uma avaliação qualitativa, com base na utilidade do resultado, ou seja, será feita uma comparação ao nível das funcionalidades oferecidas por ambas as arquiteturas. Não será feita nenhuma comparação a nível tecnológico entre o resultado e o caso de estudo, porque, o intuito desta avaliação é ser feita a nível arquitetural, sendo este agnóstico às tecnologias usadas.

A avaliação da solução foi efetuada por dois especialistas do domínio que participaram no desenvolvimento da plataforma do caso de estudo, e o autor, que participou como observador. Em conjunto com estes especialistas do domínio, foram elencados os requisitos de alto-nível

desta plataforma, a partir dos casos de uso da mesma, sendo posteriormente derivadas as necessidades específicas que os colmatam. O mapeamento destes requisitos em necessidades é apresentado na Tabela 7.

Requisito de alto-nível	Tipo de Requisito	Necessidade
Monitorização em tempo real de variáveis de dispositivos	Negócio	*Integrate with Assets*
		Distribute Data within the platform
		Store Telemetry Data
		Visualize Status Data from the Assets
Integração com ERP	Negócio	*Integrate with External Systems*
		Distribute Data within the platform
		Store Context Data
Gestão dos serviços contentorizados da plataforma	Sistema	*Manage the Platform*

Tabela 7 Mapeamento dos requisitos de alto-nível em necessidades

Previamente à avaliação, foi definido com os especialistas do domínio que a arquitetura do caso de estudo iria sofrer duas alterações. A primeira é que a simulação e o uso do Kepware não serão tidos em conta, porque estão fora do âmbito da plataforma IIoT. A simulação usa apenas os dados que estão na plataforma, não acrescentando nenhuma funcionalidade à mesma. No caso do Kepware, era um requisito tecnológico de um *stakeholder*, algo que está fora do âmbito do modelo, e por isso será assumido que os dispositivos ligados à plataforma usam todos OPC UA. A segunda alteração é no caso da integração do ERP com a plataforma IIoT, em que é o utilizado o protocolo MQTT, o que não é um protocolo tipicamente utilizado para integração com sistemas externos, por isso será assumido que o ERP em questão suporta o protocolo XML-RPC. Por esta razão deixa de ser necessário o RabbitMQ.

A experiência foi executada de uma forma interativa, em que foram selecionadas as necessidades específicas, apresentadas na Tabela 7, e tomadas as decisões que foram surgindo a partir desta seleção. Na Figura 38 é apresentado um diagrama de sequência da experiência, em que detalha as interações realizadas entre o especialista do domínio e o sistema durante a experiência.

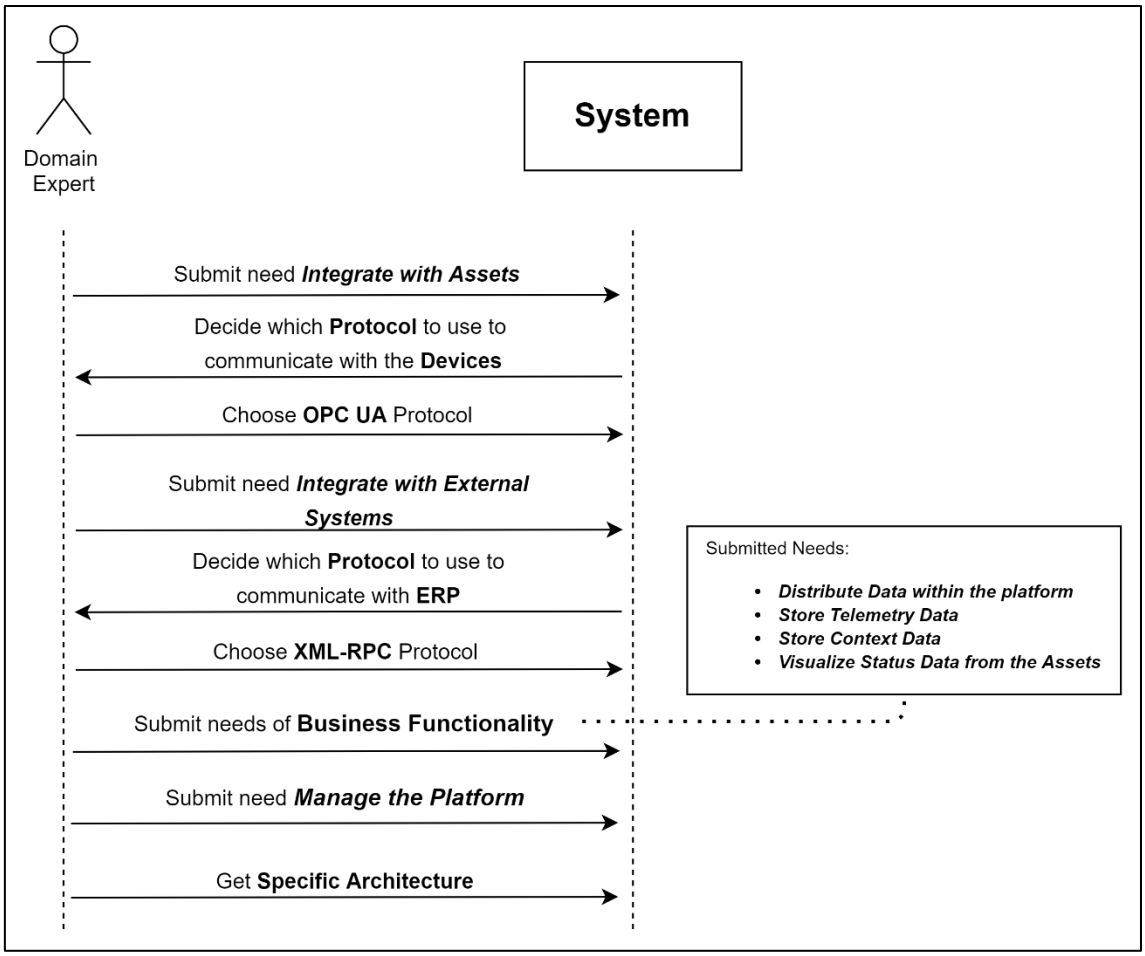

Figura 38 Diagrama de sequência da experiência

4.3. Discussão de Resultados

Uma representação da arquitetura específica obtida para este caso de estudo é apresentada na Figura 39. Um excerto da sua especificação em JSON encontra-se na Figura 40, sendo que a especificação completa se encontra no Anexo H.

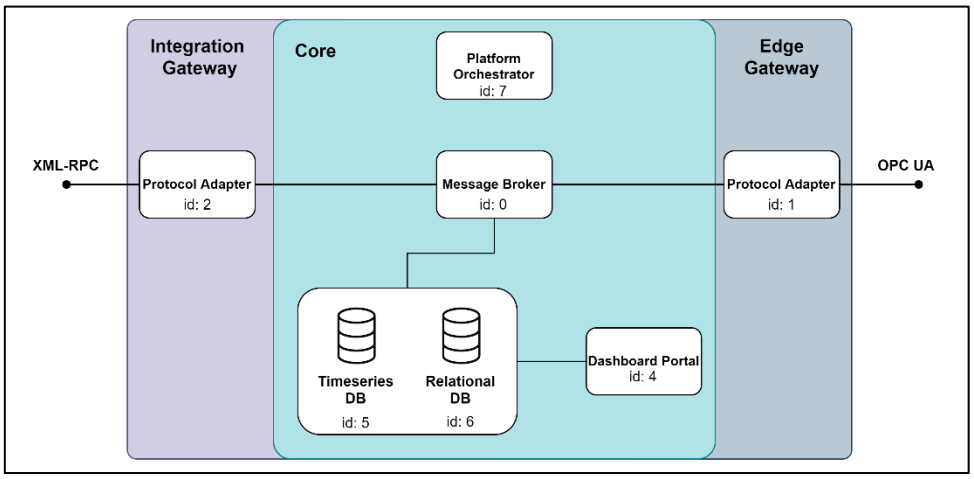

Figura 39 Representação da arquitetura específica gerada pela solução para o caso de estudo

De uma forma geral os resultados obtidos foram positivos. A arquitetura gerada pela solução cumpre os requisitos de alto-nível e oferece as funcionalidades da arquitetura do caso de estudo. Na Tabela 8 é apresentado o mapeamento dos papeis desempenhados pelas tecnologias, na arquitetura do caso de estudo, e os respetivos componentes, da arquitetura gerada. Esta arquitetura específica, gerada pela solução, foi validada pelos especialistas do domínio, considerando que esta oferece as mesmas funcionalidades que a arquitetura do caso de estudo, sendo esta viável para a implementação no mesmo. No entanto, existe uma diferença entre as arquiteturas, em que na arquitetura do caso de estudo, o Node-Red é responsável por recolher e popular a base de dados MySQL com os dados oriundos do ERP. Já na arquitetura gerada, os dados do ERP, recolhidos pelo *protocol-adapter*, são encaminhados para o *message-broker* e este é que é responsável por popular a base de dados (*relational-database*). Apesar disto, esta diferença não afeta em nada a utilidade final da arquitetura gerada, em relação há do caso de estudo.

Uma possível melhoria a efetuar na solução, identificada no caso de estudo, é na redução do tempo para obtenção da arquitetura. A obtenção desta arquitetura demorou cerca de 34 segundos, desde o pedido até obtida a resposta. No entanto, em arquiteturas que sejam compostas por um maior número de *component-instance*, este pedido pode demorar entre 1 e 2 minutos. Apesar de já ser significativamente mais rápida que um humano na especificação da arquitetura, e este tempo de espera ser aceitável do ponto de vista dos especialistas do domínio, esta demora poderá ser minimizada. A demora ocorre pelo facto de que grande parte das regras existentes no modelo serem acionadas ao requisitar a inferência da arquitetura. A utilização da última versão de Grakn poderá trazer benefícios a nível da performance de inferência, no entanto, considera-se que existe espaço para a otimização das regras. Para além disso, é também possível que a utilização de uma abordagem diferente no processo de modelação consiga diminuir este tempo. Uma das possíveis abordagens seria a modelação dos tipos de conceitos da arquitetura *baseline* diretamente na estrutura conceptual. Isto iria simplificar o processo de inferência e, teoricamente, torná-lo mais rápido, mas em contrapartida, perderíamos flexibilidade no modelo. Este é um *trade-off* que vale a pena explorar se o intuito passar por alcançar a performance máxima do modelo.

Caso de Estudo		Arquitetura Gerada	
Tecnologia	**Papel**	**Necessidade**	**Componente**
Node-Red	Comunicação com os dispositivos, usando OPC UA	*Integrate with Assets* (usando OPC UA)	*Protocol Adapter* (**id: 1**)
	Comunicação com o ERP, usando XML-RPC	*Integrate with External Systems* (usando XML-RPC)	*Protocol Adapter* (**id: 2**)
	Reencaminhar dados de dispositivos para o Apache Kafka	*Integrate with Assets*	*Protocol Adapter* (**id: 1**)
	Popular o MySQL com os dados do ERP	*Distribute Data within the platform*	*Message Broker*
Apache Kafka	Popular o InfluxDB com os dados dos dispositivos	*Distribute Data within the platform*	*Message Broker*
MySQL	Armazenar dados de contexto oriundos do ERP	*Store Context Data*	*Relational Database*
InfluxDB	Armazenar dados de telemetria oriundos dos dispositivos	*Store Telemetry Data*	*Timeseries Database*
Grafana	Apresentar *dashboards* dos dados dos dispositivos	*Visualize Status Data from the Assets*	*Dashboard Portal*
Rancher	Orquestração dos serviços da plataforma IIoT	*Manage the Platform*	*Platform Orchetrator*

Tabela 8 Mapeamento das tecnologias da arquitetura do caso de estudo com os componentes da arquitetura específica gerada pela solução

```
[
  {
    "id": 0,
    "type": {
      "type": "message-broker",
      "layers": [
        "CORE"
      ]
    },
    "implementation": {
      "name": "Apache Kafka"
    },
    "layer": "CORE",
    "connections": [
      {
        "providerId": 6,
        "requesterId": 0,
        "usedProtocol": "MySQL Protocol"
      },
      {
        "providerId": 0,
        "requesterId": 1,
        "usedProtocol": "Apache Kafka Protocol"
      },
      {
        "providerId": 5,
        "requesterId": 0,
        "usedProtocol": "InfluxDB Protocol"
      },
      {
        "providerId": 0,
        "requesterId": 2,
        "usedProtocol": "Apache Kafka Protocol"
      }
    ]
  },
  {
    "id": 1,
    "type": {
      "type": "protocol-adapter",
      "layers": [
        "EDGE_GATEWAY",
        "INTEGRATION_GATEWAY"
      ]
    },
    "implementation": {
      "name": "OPC UA Protocol Adapter Implementation"
    },
    "layer": "EDGE_GATEWAY",
    "connections": [
      {
        "providerId": 0,
        "requesterId": 1,
        "usedProtocol": "Apache Kafka Protocol"
      }
    ],
    "southPortProtocol": {
      "name": "OPC UA",
      "type": "EDGE"
    }
  },
(…)
]
```


Figura 40 Excerto da especificação da arquitetura gerada pela solução para o caso de estudo

5. Conclusão

Nesta dissertação desenvolveu-se um protótipo funcional que permite a automatização do processo de especificação de plataformas IIoT, segundo as necessidades de negócio específicas de cada empresa. Este encontra-se assente num modelo semântico, que visa desmistificar e simplificar os conceitos inerentes às arquiteturas IIoT, e assim, fomentando um entendimento partilhado dos mesmos entre os especialistas do domínio e as empresas.

O modelo foi formalizado em híper-grafos, e desenvolvido a partir da conceptualização do domínio do IIoT. Este incluí ainda um conjunto de regras e restrições do domínio, que o habilitam com capacidades de raciocínio através de um motor de inferência. No decorrer deste processo de desenvolvimento, foi seguida uma abordagem socio-semântica que envolveu a construção colaborativa de conhecimento através de interações sociais com especialistas do domínio do IIoT.

A utilidade do protótipo, na especificação de arquiteturas IIoT, foi validada no âmbito da sua aplicação num caso de estudo, sendo esta atestada pelos especialistas deste domínio. Desta forma, entende-se que este protótipo e o modelo subjacente cumprem os objetivos propostos para esta dissertação.

5.1. Contributos

No estado de arte identificou-se uma lacuna na aplicação de semântica ao nível das arquiteturas IIoT, em que existe pouco foco na descrição deste tipo de arquiteturas, e nos casos que existem esta é tipicamente pobre e/ou complexa do ponto de vista das empresas.

O foco desta dissertação foi incrementar a base de conhecimento científica atual, no âmbito da descrição semântica dos conceitos subjacentes às arquiteturas IIoT, abordada da perspetiva das empresas, na forma de articular estes conceitos com as necessidades específicas de negócio a colmatar com estas arquiteturas. Para isso, foi necessário um processo de conceptualização, que envolveu pegar numa parte de um modelo de referência, e incluir o conhecimento extra de especialização do domínio, e assim transformá-lo em algo útil para as empresas, na forma de um modelo semântico.

O modelo foi implementado num instrumento, o Grakn, no entanto pode ser reutilizado noutros instrumentos, como por exemplo, OWL ou RDF, garantindo a sua flexibilidade e escalabilidade. Este abstraí a complexidade na adoção das arquiteturas IIoT, principalmente por parte das PMEs, que não tem o conhecimento e os recursos necessários para, a partir de um modelo de referência e das tecnologias emergentes do domínio, selecionar o que é mais adequado para o seu caso concreto. Esta abstração é alcançada através da identificação automática dos

componentes, das tecnologias e dos protocolos de comunicação mais adequados consoante as necessidades de negócio de cada empresa. Apesar de abstrair toda esta complexidade na especificação da arquitetura, este processo mantém-se transparente, permitindo o entendimento e o envolvimento das empresas e dos especialistas do domínio no mesmo.

5.2. Trabalho Futuro

Como trabalhos futuros, prevê-se a atualização do Grakn para a última versão, assim que esta suporte a utilização de negação nas regras de inferência, e a criação de uma *user interface* para a interação com a solução.

Para além disso, pretende-se investigar um mecanismo para interligar a ontologia desenvolvida com outras ontologias a montante e jusante. Esta é uma ontologia de domínio, mais concretamente do IIoT, e seria benéfico interligar com outras ontologias mais específicas, como por exemplo, ontologias de protocolos de comunicação ou de representação de serviços Cloud. Dado que este tipo de ontologias são tipicamente desenvolvidas utilizando RDF ou OWL, seria necessário determinar qual é que seria a forma mais adequada de interligar estas ontologias com uma ontologia formalizada em híper-grafos.

Um terceiro trabalho a ser conduzido no futuro seria afinar o modelo desenvolvido para performance. O intuito seria a formalização das instâncias do modelo em tipos, sacrificando a flexibilidade do modelo com vista a aumentar performance na inferência de conhecimento. Desta forma, poderia ser conduzida uma análise mais exaustiva das vantagens e desvantagens das duas abordagens.

Por fim, prevê-se a extensão da funcionalidade da solução através da adição de um novo serviço, denominado "*instantiator*", que permita a instanciação automática de uma arquitetura IIoT na Cloud, após a sua especificação. Desta forma, seria possível a criação rápida de protótipos funcionais, que poderiam ser utilizados como bases para o desenvolvimento destas plataformas IIoT, ou, para efeitos de demonstração das funcionalidades destas plataformas para os casos específicos de cada empresa.

6. Referências Bibliográficas

[1] K. Schwab, *The fourth industrial revolution*, First published in Great Britain by Portfolio. [London, UK] u. a: Portfolio Penguin, 2017.

[2] D. T. Matt, V. Modrák, e H. Zsifkovits, Eds., *Industry 4.0 for SMEs: Challenges, Opportunities and Requirements*. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-25425-4.

[3] H. Kagermann, «Change Through Digitization—Value Creation in the Age of Industry 4.0», em *Management of Permanent Change*, H. Albach, H. Meffert, A. Pinkwart, e R. Reichwald, Eds. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, pp. 23–45. doi: 10.1007/978-3-658-05014-6_2.

[4] Bodo Koerber, Heike Freund, Tarek Kasah, e Lea Bolz, *Leveraging industrial software stack advancement for digital transformation*. Digital McKinsey, 2018.

[5] H. Kagermann, W. Wahlster, e J. Helbig, «Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0 – Securing the Future of German Manufacturing Industry», acatech – National Academy of Science and Engineering, München, Final Report of the Industrie 4.0 Working Group, Abr. 2013.

[6] F. Ferreira, J. Faria, A. Azevedo, e A. L. Marques, «Industry 4.0 as Enabler for Effective Manufacturing Virtual Enterprises», em *Collaboration in a Hyperconnected World*, vol. 480, H. Afsarmanesh, L. M. Camarinha-Matos, e A. Lucas Soares, Eds. Cham: Springer International Publishing, 2016, pp. 274–285. doi: 10.1007/978-3-319-45390-3_24.

[7] B. Cunha, E. Hernández, R. Rebelo, C. Sousa, e F. Ferreira, «An IIoT Solution for SME's», em *CONTROLO 2020*, vol. 695, J. A. Gonçalves, M. Braz-César, e J. P. Coelho, Eds. Cham: Springer International Publishing, 2020, pp. 313–321. doi: 10.1007/978-3-030-58653-9_30.

[8] C. Schröder, «The challenges of industry 4.0 for small and medium-sized enterprises», p. 28, 2017.

[9] Deloitte, «Industry 4.0. Challenges and solutions for the digital transformation and use of exponential technologies», 2014. Acedido: Abr. 12, 2021. [Em linha]. Disponível em: https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/manufacturing/ch-en-manufacturing-industry-4-0-24102014.pdf

[10] A. Moeuf, R. Pellerin, S. Lamouri, S. Tamayo-Giraldo, e R. Barbaray, «The industrial management of SMEs in the era of Industry 4.0», *Int. J. Prod. Res.*, vol. 56, n. 3, pp. 1118–1136, Fev. 2018, doi: 10.1080/00207543.2017.1372647.

[11] M. Flores, D. Maklin, M. Golob, A. Al-Ashaab, e C. Tucci, «Awareness Towards Industry 4.0: Key Enablers and Applications for Internet of Things and Big Data», em *Collaborative Networks of Cognitive Systems*, vol. 534, L. M. Camarinha-Matos, H. Afsarmanesh, e Y. Rezgui, Eds. Cham: Springer International Publishing, 2018, pp. 377–386. doi: 10.1007/978-3-319-99127-6_32.

[12] A. G. Frank, L. S. Dalenogare, e N. F. Ayala, «Industry 4.0 technologies: Implementation patterns in manufacturing companies», *Int. J. Prod. Econ.*, vol. 210, pp. 15–26, Abr. 2019, doi: 10.1016/j.ijpe.2019.01.004.

[13] PWC, «Industry 4.0 - Opportunities and Challenges of the Industrial Internet», 2014.

[14] M. Hung, «Gartner Insights on How to Lead in a Connected World», p. 29, 2017.

[15] C. A. Havle e C. Ucler, «Enablers for Industry 4.0», em *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, Out. 2018, pp. 1–6. doi: 10.1109/ISMSIT.2018.8567293.

[16] S. Wang, J. Wan, D. Zhang, D. Li, e C. Zhang, «Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination», *Comput. Netw.*, vol. 101, pp. 158–168, Jun. 2016, doi: 10.1016/j.comnet.2015.12.017.

[17] A. Bröring *et al.*, *Advancing IoT platforms interoperability*. 2018. Acedido: Fev. 13, 2021. [Em linha]. Disponível em: https://www.riverpublishers.com/pdf/ebook/RP_E9788770220057.pdf

[18] M. Noura, M. Atiquzzaman, e M. Gaedke, «Interoperability in Internet of Things: Taxonomies and Open Challenges», *Mob. Netw. Appl.*, vol. 24, n. 3, pp. 796–809, Jun. 2019, doi: 10.1007/s11036-018-1089-9.
[19] Shanzhi Chen, Hui Xu, Dake Liu, Bo Hu, e Hucheng Wang, «A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective», *IEEE Internet Things J.*, vol. 1, n. 4, pp. 349–359, Ago. 2014, doi: 10.1109/JIOT.2014.2337336.
[20] S. S. Arumugam *et al.*, «Accelerating Industrial IoT Application Deployment through Reusable AI Components», em *2019 Global IoT Summit (GIoTS)*, Jun. 2019, pp. 1–4. doi: 10.1109/GIOTS.2019.8766398.
[21] S. S. Kamble, A. Gunasekaran, e R. Sharma, «Analysis of the driving and dependence power of barriers to adopt industry 4.0 in Indian manufacturing industry», *Comput. Ind.*, vol. 101, pp. 107–119, Out. 2018, doi: 10.1016/j.compind.2018.06.004.
[22] J. Stentoft, K. Adsbøll Wickstrøm, K. Philipsen, e A. Haug, «Drivers and barriers for Industry 4.0 readiness and practice: empirical evidence from small and medium-sized manufacturers», *Prod. Plan. Control*, pp. 1–18, Mai. 2020, doi: 10.1080/09537287.2020.1768318.
[23] P. Adolphs *et al.*, «VDI STATUS REPORT Reference Architecture Model Industrie 4.0 (RAMMI 4.0)», VDI/VDE Society Measurement and Automatic Control, Düsseldorf, Jul. 2015.
[24] A. Ismail e W. Kastner, «A middleware architecture for vertical integration», em *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)*, Vienna, Austria, Abr. 2016, pp. 1–4. doi: 10.1109/CPPS.2016.7483915.
[25] UNIFY-IoT, «Report on IoT platform activities», Set. 2016. Acedido: Mai. 12, 2021. [Em linha]. Disponível em: http://internet-of-things-research.eu/pdf/D03_01_WP03_H2020_UNIFY-IoT_Final.pdf
[26] Microsoft, «Microsoft Azure IoT Reference Architecture», Abr. 2021. [Em linha]. Disponível em: https://azure.microsoft.com/pt-pt/resources/microsoft-azure-iot-reference-architecture/
[27] P. Pierleoni, R. Concetti, A. Belli, e L. Palma, «Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison», *IEEE Access*, vol. 8, pp. 5455–5470, 2020, doi: 10.1109/ACCESS.2019.2961511.
[28] F. Baader, I. Horrocks, e U. Sattler, «Description Logics as Ontology Languages for the Semantic Web», em *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, D. Hutter e W. Stephan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 228–248. doi: 10.1007/978-3-540-32254-2_14.
[29] T. Berners-Lee, J. Hendler, e O. Lassila, «The Semantic Web», *Sci. Am.*, vol. 284, n. 5, pp. 34–43, 2001.
[30] T. R. Gruber, «A translation approach to portable ontology specifications», *Knowl. Acquis.*, vol. 5, n. 2, pp. 199–220, Jun. 1993, doi: 10.1006/knac.1993.1008.
[31] B. D. Martino, A. Esposito, e G. Cretella, «Semantic Representation of Cloud Patterns and Services with Automated Reasoning to Support Cloud Application Portability», *IEEE Trans. Cloud Comput.*, vol. 5, n. 4, pp. 765–779, Out. 2017, doi: 10.1109/TCC.2015.2433259.
[32] S. Karadayi-Usta, «An Interpretive Structural Analysis for Industry 4.0 Adoption Challenges», *IEEE Trans. Eng. Manag.*, vol. 67, n. 3, pp. 973–978, Ago. 2020, doi: 10.1109/TEM.2018.2890443.
[33] A. Schroeder, A. Ziaee Bigdeli, C. Galera Zarco, e T. Baines, «Capturing the benefits of industry 4.0: a business network perspective», *Prod. Plan. Control*, vol. 30, n. 16, pp. 1305–1321, Dez. 2019, doi: 10.1080/09537287.2019.1612111.
[34] G. Dalmarco, F. R. Ramalho, A. C. Barros, e A. L. Soares, «Providing industry 4.0 technologies: The case of a production technology cluster», *J. High Technol. Manag. Res.*, vol. 30, n. 2, p. 100355, Nov. 2019, doi: 10.1016/j.hitech.2019.100355.
[35] J. M. Müller, D. Kiel, e K.-I. Voigt, «What Drives the Implementation of Industry 4.0? The Role of Opportunities and Challenges in the Context of Sustainability», *Sustainability*, vol. 10, n. 1, p. 247, Jan. 2018, doi: 10.3390/su10010247.

[36] National Confederation of Industry, «Challenges for Industry 4.0 in Brazil», Brasília, 2016.
[37] Deloitte, «The Industry 4.0 paradox - Overcoming disconnects on the path to digital transformation», 2018. [Em linha]. Disponível em: https://www2.deloitte.com/us/en/insights/focus/industry-4-0/challenges-on-path-to-digital-transformation/summary.html
[38] A. C. Barros, A. C. Simões, C. Toscano, A. Marques, J. C. Rodrigues, e A. Azevedo, «IMPLEMENTING CYBER-PHYSICAL SYSTEMS IN MANUFACTURING», p. 9, 2017.
[39] G. Baxter e I. Sommerville, «Socio-technical systems: From design methods to systems engineering», *Interact. Comput.*, vol. 23, n. 1, pp. 4–17, Jan. 2011, doi: 10.1016/j.intcom.2010.07.003.
[40] D. Shin, «A socio-technical framework for Internet-of-Things design: A human-centered design for the Internet of Things», *Telemat. Inform.*, vol. 31, n. 4, pp. 519–531, Nov. 2014, doi: 10.1016/j.tele.2014.02.003.
[41] H. Haken, *Information and self-organization: a macroscopic approach to complex systems*, 3rd enl. ed. Berlin ; New York: Springer, 2006.
[42] T. R. Gruber, «Towards Principles for the Design of Ontologies Used for Knowledge Sharing», em *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. [Em linha]. Disponível em: citeseer.ist.psu.edu/gruber93toward.html
[43] W3C, «Vocabularies». https://www.w3.org/standards/semanticweb/ontology (acedido Mai. 20, 2021).
[44] N. Guarino, «Formal ontology, conceptual analysis and knowledge representation», *Int. J. Hum.-Comput. Stud.*, vol. 43, n. 5–6, pp. 625–640, Nov. 1995, doi: 10.1006/ijhc.1995.1066.
[45] A. Messina, H. Pribadi, J. Stichbury, M. Bucci, S. Klarman, e A. Urso, «BioGrakn: A Knowledge Graph-Based Semantic Database for Biomedical Sciences», em *Complex, Intelligent, and Software Intensive Systems*, vol. 611, L. Barolli e O. Terzo, Eds. Cham: Springer International Publishing, 2018, pp. 299–309. doi: 10.1007/978-3-319-61566-0_28.
[46] P. Barnaghi, W. Wang, C. Henson, e K. Taylor, «Semantics for the Internet of Things: Early Progress and Back to the Future», *Int. J. Semantic Web Inf. Syst.*, vol. 8, n. 1, pp. 1–21, Jan. 2012, doi: 10.4018/jswis.2012010101.
[47] A. Gyrard, S. K. Datta, C. Bonnet, e K. Boudaoud, «A Semantic Engine for Internet of Things: Cloud, Mobile Devices and Gateways», em *2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Jul. 2015, pp. 336–341. doi: 10.1109/IMIS.2015.83.
[48] A. Bröring, P. Maué, K. Janowicz, D. Nüst, e C. Malewski, «Semantically-Enabled Sensor Plug & Play for the Sensor Web», *Sensors*, vol. 11, n. 8, pp. 7568–7605, Ago. 2011, doi: 10.3390/s110807568.
[49] F. Burzlaff e C. Bartelt, «Knowledge-Driven Architecture Composition: Case-Based Formalization of Integration Knowledge to Enable Automated Component Coupling», em *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Gothenburg, Sweden, Abr. 2017, pp. 108–111. doi: 10.1109/ICSAW.2017.54.
[50] A. Gyrard, M. Serrano, e P. Patel, «Building Interoperable and Cross-Domain Semantic Web of Things Applications», em *Managing the Web of Things*, Elsevier, 2017, pp. 305–324. doi: 10.1016/B978-0-12-809764-9.00014-7.
[51] A. Gyrard, S. K. Datta, C. Bonnet, e K. Boudaoud, «Integrating machine-to-machine measurement framework into oneM2M architecture», em *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Busan, South Korea, Ago. 2015, pp. 364–367. doi: 10.1109/APNOMS.2015.7275364.
[52] C. E. Kaed, A. Ponnouradjane, e D. Shah, «A Semantic Based Multi-Platform IoT Integration Approach from Sensors to Chatbots», em *2018 Global Internet of Things Summit (GIoTS)*, Bilbao, Jun. 2018, pp. 1–6. doi: 10.1109/GIOTS.2018.8534520.
[53] F. Lelli, «Interoperability of the Time of Industry 4.0 and the Internet of Things», *Future Internet*, vol. 11, n. 2, p. 36, Fev. 2019, doi: 10.3390/fi11020036.

[54] Z. Song, A. A. Cardenas, e R. Masuoka, «Semantic middleware for the Internet of Things», em *2010 Internet of Things (IOT)*, Tokyo, Japan, Nov. 2010, pp. 1–8. doi: 10.1109/IOT.2010.5678448.
[55] A. Willner, C. Diedrich, R. Ben Younes, S. Hohmann, e A. Kraft, «Semantic communication between components for smart factories based on oneM2M», em *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Limassol, Set. 2017, pp. 1–8. doi: 10.1109/ETFA.2017.8247690.
[56] L. Sciullo, S. Bhattacharjee, e M. Kovatsch, «Bringing deterministic industrial networking to the W3C web of things with TSN and OPC UA», em *Proceedings of the 10th International Conference on the Internet of Things*, Malmö Sweden, Out. 2020, pp. 1–8. doi: 10.1145/3410992.3410997.
[57] A. S. Thuluva, D. Anicic, S. Rudolph, e M. Adikari, «Semantic Node-RED for rapid development of interoperable industrial IoT applications», *Semantic Web*, vol. 11, n. 6, pp. 949–975, Out. 2020, doi: 10.3233/SW-200405.
[58] A. Bali, M. Al-Osta, e G. Abdelouahed, «An Ontology-Based Approach for IoT Data Processing Using Semantic Rules», em *SDL 2017: Model-Driven Engineering for Future Internet*, vol. 10567, T. Csöndes, G. Kovács, e G. Réthy, Eds. Cham: Springer International Publishing, 2017, pp. 61–79. doi: 10.1007/978-3-319-68015-6_5.
[59] C. El Kaed, I. Khan, H. Hossayni, e P. Nappey, «SQenloT: Semantic query engine for industrial Internet-of-Things gateways», em *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Reston, VA, Dez. 2016, pp. 204–209. doi: 10.1109/WF-IoT.2016.7845468.
[60] H. Hossayni, I. Khan, e C. E. Kaed, «Embedded Semantic Engine for Numerical Time Series Data», em *2018 Global Internet of Things Summit (GIoTS)*, Bilbao, Jun. 2018, pp. 1–6. doi: 10.1109/GIOTS.2018.8534580.
[61] W. T. Lunardi, E. de Matos, R. Tiburski, L. A. Amaral, S. Marczak, e F. Hessel, «Context-based search engine for industrial IoT: Discovery, search, selection, and usage of devices», em *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, Luxembourg, Luxembourg, Set. 2015, pp. 1–8. doi: 10.1109/ETFA.2015.7301477.
[62] R. L. Cagnin, I. R. Guilherme, J. Queiroz, B. Paulo, e M. F. O. Neto, «A Multi-agent System Approach for Management of Industrial IoT Devices in Manufacturing Processes», em *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, Porto, Jul. 2018, pp. 31–36. doi: 10.1109/INDIN.2018.8471926.
[63] M. Tomlein e K. Grønbæk, «A visual programming approach based on domain ontologies for configuring industrial IoT installations», em *Proceedings of the Seventh International Conference on the Internet of Things*, Linz Austria, Out. 2017, pp. 1–9. doi: 10.1145/3131542.3131552.
[64] H. Hossayni, I. Khan, M. Aazam, A. Taleghani-Isfahani, e N. Crespi, «SemKoRe: Improving Machine Maintenance in Industrial IoT with Semantic Knowledge Graphs», *Appl. Sci.*, vol. 10, n. 18, p. 6325, Set. 2020, doi: 10.3390/app10186325.
[65] C. Pereira, C. Sousa, e A. Lucas Soares, «A Socio-semantic Approach to Collaborative Domain Conceptualization», em *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, Berlin, Heidelberg, 2009, pp. 524–533.
[66] W. Shi, J. Cao, Q. Zhang, Y. Li, e L. Xu, «Edge Computing: Vision and Challenges», *IEEE Internet Things J.*, vol. 3, n. 5, pp. 637–646, Out. 2016, doi: 10.1109/JIOT.2016.2579198.
[67] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, e Q. Zhang, «Edge Computing in IoT-Based Manufacturing», *IEEE Commun. Mag.*, vol. 56, n. 9, pp. 103–109, Set. 2018, doi: 10.1109/MCOM.2018.1701231.
[68] A. Gómez-Pérez, «Towards a framework to verify knowledge sharing technology», *Expert Syst. Appl.*, vol. 11, n. 4, pp. 519–529, Jan. 1996, doi: 10.1016/S0957-4174(96)00067-X.
[69] H. Oinas-Kukkonen, «The 7C Model for Organizational Knowledge Creation and Management», p. 12, 2001.
[70] A. Martin, S. Emmenegger, e G. Wilke, «Integrating an enterprise architecture ontology in a case-based reasoning approach for project knowledge», em *Proceedings of the First International Conference on Enterprise Systems: ES 2013*, Cape Town, South Africa, Nov. 2013, pp. 1–12. doi: 10.1109/ES.2013.6690082.

[71] A. Martin, S. Emmenegger, K. Hinkelmann, e B. Thönssen, «A viewpoint-based case-based reasoning approach utilising an enterprise architecture ontology for experience management», *Enterp. Inf. Syst.*, vol. 11, n. 4, pp. 551–575, Abr. 2017, doi: 10.1080/17517575.2016.1161239.
[72] M. R. Genesereth e N. J. Nilsson, *Logical foundations of artificial intelligence*. Los Altos, Calif: Morgan Kaufmann, 1987.
[73] S. Bai, F. Zhang, e P. H. S. Torr, «Hypergraph convolution and hypergraph attention», *Pattern Recognit.*, vol. 110, p. 107637, Fev. 2021, doi: 10.1016/j.patcog.2020.107637.
[74] A. Bretto, *Hypergraph Theory*. Heidelberg: Springer International Publishing, 2013. doi: 10.1007/978-3-319-00080-0.
[75] Z. Wu *et al.*, «Semantic hyper-graph-based knowledge representation architecture for complex product development», *Comput. Ind.*, vol. 100, pp. 43–56, Set. 2018, doi: 10.1016/j.compind.2018.04.008.
[76] M. Masmoudi, S. B. A. B. Lamine, H. B. Zghal, B. Archimede, e M. H. Karray, «Knowledge hypergraph-based approach for data integration and querying: Application to Earth Observation», *Future Gener. Comput. Syst.*, vol. 115, pp. 720–740, Fev. 2021, doi: 10.1016/j.future.2020.09.029.
[77] M. M. Wolf, A. M. Klinvex, e D. M. Dunlavy, «Advantages to modeling relational data using hypergraphs versus graphs», em *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA, Set. 2016, pp. 1–7. doi: 10.1109/HPEC.2016.7761624.
[78] «Grakn». https://grakn.ai/ (acedido Fev. 20, 2021).
[79] A. Berquand *et al.*, «Artificial Intelligence for the Early Design Phases of Space Missions», em *2019 IEEE Aerospace Conference*, Big Sky, MT, USA, Mar. 2019, pp. 1–20. doi: 10.1109/AERO.2019.8742082.

Anexos

Índice de Anexos

Anexo A - Sistematização das funções típicas de uma plataforma IIoT

Function Verb	Definition	Typical Handled Things Type
Broker	Mediate Things	Data
Present	Show Things	
Query	Get Things that match a Criteria	Data
Orchestrate	Arrange Things to achieve a Result	*
Collect	Get Things	Data
Rule-Process	Process Things by rules	Data
Provide	Returns Things	Data
Backup	Backup Things	Data
Restore	Restore Things	Data
Store	Store Things	Data
Transform	Change the Structure of a Thing	Data
Route	Returns Things by condition	Data
Merge	Join things	Data
Connect	Interlink Things to the Platform	
Command	Command Things to do something	
Authenticate	Authenticate Things	Actors
Authorize	Authorize Things	Actors
Scale	Increase or Decrease the number of Things	Component-Containers

Tabela 9 Funções típicas de uma plataforma IIoT

Anexo B - Sistematização das funções típicas de uma plataforma IIoT e respetivos requisitos (Necessidades)

Function Verb	Handled-Thing Context	Handled-Thing	Need	Need ID
Broker	Platform	Datasets	Distribute Data within the platform	1
Broker	Platform	Alerts	Distribute Data within the platform	1
Broker	Platform	Metrics	Distribute Data within the platform	1
Present	Platform	Datasets	Visualize Status Data from the Assets	2
Query	Platform	Datasets	Visualize Status Data from the Assets	2
Present	Platform-Infrastructure	Alerts	Visualize Alerts from Platform	3
Query	Platform-Infrastructure	Alerts	Visualize Alerts from Platform	3
Present	Platform-Infrastructure	Metrics	Visualize Behavioral Data from Platform	4
Query	Platform-Infrastructure	Metrics	Visualize Behavioral Data from Platform	4
Present	Platform	Alerts	Visualize Alerts from Platform	3
Query	Platform	Alerts	Visualize Alerts from Platform	3
Present	Platform	Metrics	Visualize Behavioral Data from Platform	4
Query	Platform	Metrics	Visualize Behavioral Data from Platform	4
Orchestrate	Platform	Component-Containers	Manage the Platform	5
Collect	Platform	Logs	Audit the Platform	6
Query	Platform	Logs	Audit the Platform	6
Provide	Platform	Logs	Audit the Platform	6
Rule-Process	Platform	Logs	Audit the Platform	6
Collect	Platform-Infrastructure	Logs	Audit the Platform	6
Query	Platform-Infrastructure	Logs	Audit the Platform	6
Provide	Platform-Infrastructure	Logs	Audit the Platform	6
Rule-Process	Platform-Infrastructure	Logs	Audit the Platform	6
Collect	Platform	Metrics	Monitor the Platform	7
Query	Platform	Metrics	Monitor the Platform	7
Provide	Platform	Metrics	Monitor the Platform	7
Rule-Process	Platform	Metrics	Monitor the Platform	7
Collect	Platform-Infrastructure	Metrics	Monitor the Platform	7
Query	Platform-Infrastructure	Metrics	Monitor the Platform	7
Provide	Platform-Infrastructure	Metrics	Monitor the Platform	7
Rule-Process	Platform-Infrastructure	Metrics	Monitor the Platform	7
Backup	Platform	Data	Protect the Platform	8
Restore	Platform	Data	Protect the Platform	8
Backup	Platform	Component-Containers	Protect the Platform	8
Restore	Platform	Component-Containers	Protect the Platform	8
Backup	Platform-Infrastructure	Data	Protect the Platform	8

Restore	Platform-Infrastructure	Data	Protect the Platform	8
Store	Platform	Timeseries-Datasets	Store Telemetry Data	9
Store	Platform	Tabular-Datasets	Store Context Data	10
Store	Platform	Document-Datasets	Store Event Data	11
Transform	Platform	Datasets	Manage Data Processing	12
Route	Platform	Datasets	Manage Data Processing	12
Query	Platform	Datasets	Manage Data Processing	12
Merge	Platform	Datasets	Manage Data Processing	12
Connect	Assets	Devices	Integrate with Assets	13
Collect	Assets	Datasets	Integrate with Assets	13
Provide	Platform	Datasets	Integrate with Assets	13
Command	Assets	Devices	Control the Assets	14
Transform	Assets	Datasets	Manage Assets's Data Processing	15
Merge	Assets	Datasets	Manage Assets's Data Processing	15
Connect	External-Systems	Systems	Integrate with External Systems	16
Collect	External-Systems	Datasets	Integrate with External Systems	16
Provide	Platform	Datasets	Integrate with External Systems	16
Transform	External-Systems	Datasets	Manage External-Systems's Data Processing	17
Merge	External-Systems	Datasets	Manage External-Systems's Data Processing	17
Authenticate	Assets	Devices	Control Devices Access	18
Authorize	Assets	Devices	Enforce Devices Access Policies	19
Authenticate	External-Systems	Systems	Control Systems Access	20
Authorize	External-Systems	Systems	Enforce Systems Access Policies	21
Authenticate	Platform	Users	Control Users Access	22
Authorize	Platform	Users	Enforce Users Access Policies	23
Authenticate	Platform-Infrastructure	Users	Control Users Access	22
Authorize	Platform-Infrastructure	Users	Enforce Users Access Policies	23
Orchestrate	Platform	Users	Manage Users Credentials and Policies	24
Orchestrate	Platform-Infrastructure	Users	Manage Users Credentials and Policies	24
Rule-Process	Platform	Datasets	Manage Business Events	25
Provide	Platform	Datasets	Access Platform's Data from a single endpoint	26
Provide	Platform	Component-Containers	Access Platform's Functionality from a single endpoint	27
Rule-Process	Platform	Metrics	Scale Platform on-demand	28
Scale	Platform	Component-Containers	Scale Platform on-demand	28

Tabela 10 Funções típicas de uma plataforma IIoT e respetivas necessidades

Anexo C - Sistematização dos Protocolos de comunicação tipicamente associados a uma plataforma IIoT

Protocols	Tier
Kafka Specific	Platform
OPC UA	Edge
MQTT	Edge
AMQP	Platform
HTTP REST	Platform Business
InfluxDB Specific	Platform
MongoDB Specific	Platform
MySQL Specific	Platform
XML-RPC	Business

Tabela 11 Mapeamento dos protocolos típicos do IIoT com os diferentes níveis da plataforma

Anexo D - Necessidades

ID	Needs	Category	Description
1	Distribute Data within the platform	Business Functionality	Ensure consistent data delivery and mediation across components in the platform
2	Visualize Status Data from the Assets	Business Functionality	Present the data gathered from the connected Assets in interactive dashboards
3	Visualize Alerts from Platform	Business Functionality	Present the alerts generated by the platform in interactive dashboards
4	Visualize Behavioral Data from Platform	System Functionality	Present the metrics from the platform's components and its infrastructure in interactive dashboards
5	Manage the Platform	System Functionality	Manage the platform's components
6	Audit the Platform	System Functionality	Allow detailed examinations of the platform events and operations through the use of the components and infrastructure logs
7	Monitor the Platform	System Functionality	Gathering of information of platform's components and its infrastructure allowing performance and behavior analysis
8	Protect the Platform	System Functionality	Ensure platform resilience through data backup and restore, protecting from eventual disasters
9	Store Telemetry Data	Business Functionality	Store rapidly changing data gathered from the things connected to the platform
10	Store Context Data	Business Functionality	Store context data gathered from the things connected to the platform (Context data examples: measurement units, production orders, origin and destination)
11	Store Event Data	Business Functionality	Store event data generated by the platform or gathered from the connected things
12	Manage Data Processing	Business Functionality	Transform and combine data streams gathered from different sources
13	Integrate with Assets	Edge Integration	Connect heterogeneous devices with the platform allowing data exchange
14	Control the Assets	Edge Integration	Enable the sending of commands and control of the assets connected to the platform
15	Manage Assets's Data Processing	Edge Integration	Structure and/or normalize and enrich data ingested from the devices connected to the platform
16	Integrate with External Systems	Business Integration	Connect with external systems or applications with the platform allowing data exchange
17	Manage External-Systems's Data Processing	Business Integration	Structure and/or normalize and enrich data ingested from the systems connected to the platform

18	Control Devices Access	Security Functionality	Control access to the platform functionality and data by authentication of the connected devices
19	Enforce Devices Access Policies	Security Functionality	Enforce security policies by authorization of the connected devices
20	Control Systems Access	Security Functionality	Control access to the platform functionality and data by authentication of the connected systems
21	Enforce Systems Access Policies	Security Functionality	Enforce security policies by authorization of the connected systems
22	Control Users Access	Security Functionality	Control access to the platform functionality and data by authentication of the users
23	Enforce Users Access Policies	Security Functionality	Enforce security policies by authorization of the users
24	Manage Users Credentials and Policies	Security Functionality	Manage user access to the platform functionality and data
25	Manage Business Events	Business Functionality	Allow users to model business specific restrictions to trigger events, based on the incoming data
26	Access Platform's Data from a single endpoint	System Functionality	Use a single endpoint to access all data stored in the platform
27	Access Platform's Functionality from a single endpoint	System Functionality	Use a single endpoint to access the functionality provided by the platform's components
28	Scale Platform on-demand	System Functionality	Automatic scale up or down the platform's components according to the demand/load

Tabela 12 Necessidades de negócio típicas de uma plataforma IIoT

Anexo E - Tecnologias

Components	Technologies	Tecnhology Specific Functions	South Port Protocol	Requested Protocols	Provided Protocols	Depends On Component Implementation
Message Broker	RabbitMQ				AMQP MQTT	
	Apache Kafka			InfluxDB Specific MongoDB Specific MySQL Specific	Kafka Specific	
Dashboard Portal	Grafana			InfluxDB Specific MongoDB Specific MySQL Specific		
Platform Orchestrator	Rancher	Authenticate Platform-Infrastructure's Users Authorize Platform-Infrastructure's Users				
Log Aggregator	Banzai Cloud Logging Operator					
Monitor	Rancher Monitoring Operator					
Disaster Recovery Component	Velero					
Timeseries DB	InfluxDB				InfluxDB Specific	Message Broker implementedBy Apache Kafka
Document DB	MongoDB				MongoDB Specific	Message Broker implementedBy Apache Kafka
Relational DB	MySQL				MySQL Specific	Message Broker implementedBy Apache Kafka

Flow Engine	Apache Nifi			MQTT AMQP Kafka Specific InfluxDB Specific MongoDB Specific MySQL Specific		
Protocol Adapter	OPCUA - PA Custom Implementation		OPC UA	Kafka Specific AMQP		
Protocol Adapter	MQTT - PA Custom Implementation		MQTT	Kafka Specific AMQP		
Protocol Adapter	REST - PA Custom Implementation		REST	Kafka Specific AMQP		
Protocol Adapter	XML-RPC - PA Custom Implementation		XML-RPC	Kafka Specific AMQP		
Mediation Flow	Node-red			Kafka Specific	AMQP	
Identity Manager	Keycloack					
Complex Event Manager	CEP - Custom Implementation			Kafka Specific		
API Gateway	Kong	Authenticate Platform's Users Authorize Platform's Users Authenticate Platform-Infrastructure's Users Authorize Platform-Infrastructure's Users				
Platform Autoscaler	Horizontal Pod Autoscaler					

Tabela 13 Tabela de Tecnologias

Anexo F - Componentes

Component	Depends On Component	Default	Domain	Layer	Component Characterization
Message Broker		yes	Connectivity and Normalisation	Core	Broker Platform's Datasets Broker Platform's Alerts Broker Platform's Metrics
Dashboard Portal		no	Data Visualisations	Data	Present Platform's Datasets Query Platform's Datasets Present Platform-Infrastructure's Alerts Query Platform-Infrastructure's Alerts Present Platform's Alerts Query Platform's Alerts Present Platform-Infrastructure's Metrics Query Platform-Infrastructure's Metrics Present Platform's Metrics Query Platform's Metrics
Platform Orchestrator		no	Additional Tools	Core	Orchestrate Platform's Component-Containers
Log Aggregator		no	Additional Tools	Core	Collect Platform's Logs Query Platform's Logs Provide Platform's Logs Rule-Process Platform's Logs Collect Platform-Infrastructure's Logs Query Platform-Infrastructure's Logs Provide Platform-Infrastructure's Logs Rule-Process Platform-Infrastructure's Logs
Monitor	Platform Orchestrator	no	Additional Tools	Core	Collect Platform's Metrics Query Platform's Metrics Provide Platform's Metrics Rule-Process Platform's Metrics Collect Platform-Infrastructure's Metrics Query Platform-Infrastructure's Metrics Provide Platform-Infrastructure's Metrics Rule-Process Platform-Infrastructure's Metrics
Disaster Recovery Component		no	Additional Tools	Core	Backup Platform's Data Restore Platform's Data Backup Platform's Component-Containers

					Restore Platform's Component-Containers Backup Platform-Infrastructure's Data Restore Platform-Infrastructure's Data
Timeseries Database		no	Data Storage	Data	Store Platform's Timeseries-Datasets
Relational Database		no	Data Storage	Data	Store Platform's Tabular-Datasets
Document Database		no	Data Storage	Data	Store Platform's Document-Datasets
Flow Engine		no	Processing and Action Management	Data	Transform Platform's Datasets Route Platform's Datasets Merge Platform's Datasets Query Platform's Datasets
Protocol Adapter		no	Connectivity and Normalisation	Edge Gateway Integration Gateway	Connect Assets's Devices Collect Assets's Datasets Provide Platform's Datasets Command Assets's Devices Connect External-Systems's Systems Collect External-Systems's Datasets
Mediation Flow		no	Processing and Action Management	Edge Gateway Integration Gateway	Transform Assets's Datasets Merge Assets's Datasets Transform External-Systems's Datasets Merge External-Systems's Datasets
Identity Manager		no	Security and Privacy	Core	Authenticate Assets's Devices Authorize Assets's Devices Authenticate External-Systems's Systems Authorize External-Systems's Systems Authenticate Platform's Users Authorize Platform's Users Orchestrate Platform's Users Authenticate Platform-Infrastructure's Users Authorize Platform-Infrastructure's Users Orchestrate Platform-Infrastructure's Users
Complex Event Manager		no	Processing and Action Management	Data	Rule-Process Platform's Datasets
API Gateway		no	Additional Tools	Core	Provide Platform's Datasets Provide Platform's Component-Containers
Platform Autoscaler		no	Additional Tools	Core	Rule-Process Platform's Metrics Scale Platform's Component-Containers

Tabela 14 Tabela de Componentes

Anexo G - Validação dos Conceitos

Component	What kind of Thing is Handled by the Function?	What is operational Context of the Handled Thing?	What is Type of the Function?	What can I do?	Validation Template *<technology>* implements *<component>* to *<function>* *<operational_context>*'s *<handled_thing>* so that I can *<need>*
Message Broker	Datasets Alerts Metrics	Platform External-Systems Assets	Broker	Distribute Data within the platform	***Apache Kafka*** implements ***Message Broker*** to ***Broker Platform***'s ***Datasets*** so that I can ***Distribute Data within the platform***
Dashboard Portal	Datasets Alerts Metrics	Platform Platform-Infrastructure	Present Query	Visualize Status Data from the Assets Visualize Alerts from Platform Visualize Behavioral Data from Platform	***Grafana*** implements ***Dashboard-Portal*** to ***Present Platform***'s ***Datasets*** so that I can ***Visualize Status Data from the Assets*** ***Grafana*** implements ***Dashboard-Portal*** to ***Present Platform-Infrastructure***'s ***Alerts*** so that I can ***Visualize Alerts from Platform*** ***Grafana*** implements ***Dashboard-Portal*** to ***Present Platform-Infrastructure***'s ***Metrics*** so that I can ***Visualize Behavioral Data from Platform*** ***Grafana*** implements ***Dashboard-Portal*** to ***Query Platform***'s ***Datasets*** so that I can ***Visualize Status Data from the Assets***
Platform Orchestrator	Component-Containers	Platform	Orchestrate	Manage the Platform	***Rancher*** implements ***Platform Orchestrator*** to ***Orchestrate Platform***'s ***Component-Containers*** so that I can ***Manage the Platform***
Log Aggregator	Logs	Platform Platform-Infrastructure	Collect Query Rule-Process Provide	Audit the Platform	***Banzai Cloud Logging Operator*** implements ***Log-Aggregator*** to ***Collect Platform***'s ***Logs*** so that I can ***Audit the Platform*** ***Banzai Cloud Logging Operator*** implements ***Log-Aggregator*** to ***Query Platform***'s ***Logs*** so that I can ***Audit the Platform*** ***Banzai Cloud Logging Operator*** implements ***Log-Aggregator*** to ***Provide Platform***'s ***Logs*** so that I can ***Audit the Platform***
Monitor	Metrics	Platform Platform-Infrastructure	Collect Query Rule-Process Provide	Monitor the Platform	***Rancher Monitoring Operator*** implements ***Monitor*** to ***Collect Platform***'s ***Metrics*** so that I can ***Monitor the Platform*** ***Rancher Monitoring Operator*** implements ***Monitor*** to ***Collect Platform-Infrastructure***'s ***Metrics*** so that I can ***Monitor the Platform*** ***Rancher Monitoring Operator*** implements ***Monitor*** to ***Query Platform***'s ***Metrics*** so that I can ***Monitor the Platform*** ***Rancher Monitoring Operator*** implements ***Monitor*** to ***Provide Platform***'s ***Metrics*** so that I can ***Monitor the Platform***
Disaster Recovery Component	Data Component-Containers	Platform Platform-Infrastructure	Backup Restore	Protect the Platform	***Velero*** implements ***Disaster Recovery Component*** to ***Backup Platform***'s ***Data*** so that I can ***Protect the Platform*** ***Velero*** implements ***Disaster Recovery Component*** to ***Restore Platform***'s ***Data*** so that I can ***Protect the Platform***
Timeseries Database	Timeseries-Datasets	Platform	Store	Store Telemetry Data	***InfluxDB*** implements ***Timeseries Database*** to ***Store Platform***'s ***Timeseries-Datasets*** so that I can ***Store Telemetry Data***
Relational	Tabular-Datasets	Platform	Store	Store Context Data	***MySQL*** implements ***Relational Database*** to ***Store Platform***'s ***Tabular-Datasets*** so

Database					that I can ***Store Context Data***
Document Database	Document-Datasets	Platform	Store	Store Event Data	***MongoDB*** implements ***Non-Relational Database*** to ***Store Platform***'s ***Document-Datasets*** so that I can ***Store Event Data***
Flow Engine	Datasets	Platform	Transform Route Query Merge	Manage Data Processing	***Nifi*** implements ***Flow Engine*** to ***Transform Platform***'s ***Datasets*** so that I can ***Manage Data Processing*** ***Nifi*** implements ***Flow Engine*** to ***Route Platform***'s ***Datasets*** so that I can ***Manage Data Processing*** ***Nifi*** implements ***Flow Engine*** to ***Query Platform***'s ***Datasets*** so that I can ***Manage Data Processing*** ***Nifi*** implements ***Flow Engine*** to ***Merge Platform***'s ***Datasets*** so that I can ***Manage Data Processing***
Protocol Adapter	Datasets Devices Systems	Platform	Connect Collect Control Provide	Integrate with Assets Control the Assets Integrate with External Systems	***Custom Protocol Adapter*** implements ***Protocol Adapter*** to ***Connect Assets's Devices*** so that I can ***Integrate with Assets*** ***Custom Protocol Adapter*** implements ***Protocol Adapter*** to ***Collect Assets's Datasets*** so that I can ***Integrate with Assets*** ***Custom Protocol Adapter*** implements ***Protocol Adapter*** to ***Provide Platform's Datasets*** so that I can ***Integrate with Assets*** ***Custom Protocol Adapter*** implements ***Protocol Adapter*** to ***Command Assets's Devices*** so that I can ***Control the Assets*** ***Custom Protocol Adapter*** implements ***Protocol Adapter*** to ***Connect External-Systems's Systems*** so that I can ***Integrate with External Systems*** ***Custom Protocol Adapter*** implements ***Protocol Adapter*** to ***Collect External-Systems's Datasets*** so that I can ***Integrate with External Systems*** ***Custom Protocol Adapter*** implements ***Protocol Adapter*** to ***Provide Platform's Datasets*** so that I can ***Integrate with External Systems***
Mediation Flow	Datasets	Assets External Systems	Transform Route	Manage Assets's Data Processing Manage External-Systems's Data Processing	***Node-Red*** implements ***Mediation Flow*** to ***Transform Assets***'s ***Datasets*** so that I can ***Manage Assets's Data Processing*** ***Node-Red*** implements ***Mediation Flow*** to ***Merge Assets***'s ***Datasets*** so that I can ***Manage Assets's Data Processing*** ***Node-Red*** implements ***Mediation Flow*** to ***Transform External-Systems's Datasets*** so that I can ***Manage External-Systems's Data Processing*** ***Node-Red*** implements ***Mediation Flow*** to ***Merge External-Systems's Datasets*** so that I can ***Manage External-Systems's Data Processing***
Identity Manager	Actors	Assets External Systems Platform Platform-Infrastructure	Authenticate Authorize Orchestrate	Control Actors Access Enforce Actors Access Policies Manage Users	***Keycloak*** implements ***Identity Manager*** to ***Authenticate Platform***'s ***<Actor>*** so that I can ***Control <Actors> Access*** ***Keycloak*** implements ***Identity Manager*** to ***Authorize Platform***'s ***<Actor>*** so that I can ***Enforce <Actors> Access Policies*** ***Keycloak*** implements ***Identity Manager*** to ***Orchestrate Platform***'s ***User*** so that I can ***Manage Users Credentials and Policies***
Complex Event Manager	Datasets	Platform	Rule-Process	Manage Business Events	***CEP Custom Implementation*** implements ***Complex Event Manager*** for ***Rule-Process Platform's Datasets*** so that I can ***Manage Business Events***

API Gateway	Datasets Component-Containers	Platform	Provide	Access Platform's Data from a single endpoint Access Platform's Functionality from a single endpoint	***Kong*** implements ***API Gateway*** to ***Provide Platform***'s ***Datasets*** so that I can ***Access Platform's Data from a single endpoint*** ***Kong*** implements ***API Gateway*** to ***Provide Platform***'s ***Component-Containers*** so that I can ***Access Platform's Functionality from a single endpoint***
Platform Autoscaler	Metrics Component-Containers	Platform	Rule-Process Scale	Scale Platform on-demand	***Horizontal Pod Autoscaler*** implements ***Platform-Autoscaler*** to ***Rule-Process Platform***'s ***Metrics*** so that I can ***Scale Platform on-demand*** ***Horizontal Pod Autoscaler*** implements ***Platform-Autoscaler*** to ***Scale Platform's Component-Containers*** so that I can ***Scale Platform on-demand***

Tabela 15 Tabela de validação de conceitos através de *templates* semânticos

Anexo H - Especificação da Arquitetura do Caso de Estudo


```
[
  {
    "id": 0,
    "type": {
      "type": "message-broker",
      "layers": [
        "CORE"
      ]
    },
    "implementation": {
      "name": "Apache Kafka"
    },
    "layer": "CORE",
    "connections": [
      {
        "providerId": 6,
        "requesterId": 0,
        "usedProtocol": "MySQL Protocol"
      },
      {
        "providerId": 0,
        "requesterId": 1,
        "usedProtocol": "Apache Kafka Protocol"
      },
      {
        "providerId": 5,
        "requesterId": 0,
        "usedProtocol": "InfluxDB Protocol"
      },
      {
        "providerId": 0,
        "requesterId": 2,
        "usedProtocol": "Apache Kafka Protocol"
      }
    ]
  },
  {
    "id": 1,
    "type": {
      "type": "protocol-adapter",
      "layers": [
        "EDGE_GATEWAY",
        "INTEGRATION_GATEWAY"
      ]
    },
    "implementation": {
      "name": "OPC UA Protocol Adapter Implementation"
    },
    "layer": "EDGE_GATEWAY",
    "connections": [
      {
        "providerId": 0,
        "requesterId": 1,
        "usedProtocol": "Apache Kafka Protocol"
      }
    ],
    "southPortProtocol": {
      "name": "OPC UA",
      "type": "EDGE"
    }
  },
  {
    "id": 2,
    "type": {
      "type": "protocol-adapter",
      "layers": [
        "EDGE_GATEWAY",
        "INTEGRATION_GATEWAY"
      ]
    },
```

```
    "implementation": {
      "name": "XML-RPC Protocol Adapter Implementation"
    },
    "layer": "INTEGRATION_GATEWAY",
    "connections": [
      {
        "providerId": 0,
        "requesterId": 2,
        "usedProtocol": "Apache Kafka Protocol"
      }
    ],
    "southPortProtocol": {
      "name": "XML-RPC",
      "type": "BUSINESS"
    }
  },
  {
    "id": 4,
    "type": {
      "type": "dashboard-portal",
      "layers": [
        "DATA"
      ]
    },
    "implementation": {
      "name": "Grafana"
    },
    "layer": "DATA",
    "connections": [
      {
        "providerId": 6,
        "requesterId": 4,
        "usedProtocol": "MySQL Protocol"
      },
      {
        "providerId": 5,
        "requesterId": 4,
        "usedProtocol": "InfluxDB Protocol"
      }
    ]
  },
  {
    "id": 5,
    "type": {
      "type": "timeseries-database",
      "layers": [
        "DATA"
      ]
    },
    "implementation": {
      "name": "InfluxDB"
    },
    "layer": "DATA",
    "connections": [
      {
        "providerId": 5,
        "requesterId": 0,
        "usedProtocol": "InfluxDB Protocol"
      },
      {
        "providerId": 5,
        "requesterId": 4,
        "usedProtocol": "InfluxDB Protocol"
      }
    ]
  },
  {
    "id": 6,
    "type": {
      "type": "relational-database",
      "layers": [
        "DATA"
      ]
    },
```

```
    "implementation": {
      "name": "MySQL"
    },
    "layer": "DATA",
    "connections": [
      {
        "providerId": 6,
        "requesterId": 4,
        "usedProtocol": "MySQL Protocol"
      },
      {
        "providerId": 6,
        "requesterId": 0,
        "usedProtocol": "MySQL Protocol"
      }
    ]
  },
  {
    "id": 7,
    "type": {
      "type": "platform-orchestrator",
      "layers": [
        "CORE"
      ]
    },
    "implementation": {
      "name": "Rancher"
    },
    "layer": "CORE",
    "connections": []
  }
]
```