Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação
Mestrado Acadêmico em Sistemas e Computação

Um Serviço de Descoberta Ciente de Contexto para Internet das Coisas

Porfírio Dantas Gomes

Natal-RN
Abril, 2016

Porfírio Dantas Gomes

Um Serviço de Descoberta Ciente de Contexto para Internet das Coisas

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Mestre em Sistemas e Computação.

Linha de pesquisa:
Engenharia de *Software*

Orientador

Prof.ª Dra. Thais Vasconcelos Batista

PPgSC – Programa de Pós-Graduação em Sistemas e Computação
DIMAp – Departamento de Informática e Matemática Aplicada
CCET – Centro de Ciências Exatas e da Terra
UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN

Abril, 2016

Catalogação da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial
Centro de Ciências Exatas e da Terra – CCET.

Gomes, Porfírio Dantas.
Um serviço de descoberta ciente de contexto para internet das coisas / Porfírio Dantas Gomes. - Natal, 2016.
115 f. : il.

Orientadora: Profa. Dra. Thais Vasconcelos Batista.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Programa de Pós-Graduação em Sistemas e Computação.

1. Engenharia de software – Dissertação. 2. Internet das coisas – Dissertação. 3. Serviço de descoberta – Dissertação. 4. Descrição semântica – Dissertação. 5. Ciência de contexto – Dissertação. 6. Informação de contexto – Dissertação. 7. Qualidade de contexto – Dissertação. I. Batista, Thais Vasconcelos. II. Título.

RN/UF/BSE-CCET CDU: 004.41

PORFÍRIO DANTAS GOMES

Um Serviço de Descoberta Ciente de Contexto para Internet das Coisas

Esta Dissertação foi julgada adequada para a obtenção do título de mestre em Sistemas e Computação e aprovado em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

Dr[a]. Thais Vasconcelos Batista – UFRN
(Presidente)

Dr[a]. Elizabeth Ferreira Gouvêa Goldbarg – UFRN
(Vice-coordenadora do Programa)

Banca Examinadora

Dr. Gibeon Soares de Aquino Junior – UFRN
(Examinador)

Dr[a]. Flavia Coimbra Delicato – UFRJ
(Examinador)

Dr. Paulo de Figueiredo Pires – UFRJ
(Examinador)

Abril, 2016

À minha família, por sempre me proporcionar totais condições para a realização dos meus estudos.

Agradecimentos

Primeiramente gostaria de agradecer a Deus por me proporcionar todas as condições para a realização deste trabalho e, principalmente, por ter colocado em meu caminho pessoas muito especiais que me ajudaram nesta aventura.

Existem coisas na vida que não conseguimos expressar com palavras, apenas o coração sabe explicar o que sentimos. É desta forma que me sinto todas as vezes que tento agradecer aos meus pais, Sônia Dantas e Francisco Gomes. Sou constantemente grato ao suporte que, sem dúvida, foi fundamental para me proporcionar uma boa formação, tanto como homem quanto profissional. Agradeço ainda a meus outros familiares, em especial minha irmã, Suellen Dantas, e minha tia, Soriene Dantas, que estavam ao meu lado, me apoiando e incentivando desde sempre.

Também gostaria de expressar minha sincera gratidão por minha orientadora e amiga, Thais Batista. Obrigado pela orientação, pelo espírito acolhedor, pelo incentivo e cobranças, totalmente necessárias, para me manter o tempo todo focado no objetivo final. Não poderia deixar de agradecer a liberdade e abertura para discutir novas ideias e definir novas metas, sem deixar de lado a qualidade do trabalho. Obrigado, Thais, por tudo.

Agradeço aos amigos Everton Cavalcante, Taniro Rodrigues e Pedro Petrovitch pela contribuição na realização desse trabalho, pelos conselhos, pelas reuniões e pelos debates que possibilitaram o desenvolvimento e a conclusão de mais essa etapa na minha vida.

Aos professores Flavia Delicato e Paulo Pires pela contribuição inestimável neste trabalho, pela parceria nos trabalhos que desempenhamos juntos, e por terem concordado em participar da banca de defesa desta dissertação. Além deles, agradeço também ao professor Gibeon Aquino, por ter feito parte da minha banca de qualificação do mestrado, fornecendo dicas e contribuições para a melhoria do meu projeto, e também por ter concordado em fazer parte da banca de defesa desta dissertação.

Aos colegas de Pós-Graduação que tive o prazer de conhecer nas disciplinas cursadas. Em especial, agradeço aos amigos Thomaz Filipe, Valdigleis Costa e Ranyer Lopes, companheiros dessa mesma trajetória; muito obrigado pela amizade, apoio e incentivo.

Finalmente, gostaria de agradecer a todos que de alguma forma, direta ou indiretamente, contribuíram para a realização deste trabalho.

Luck is what happens when preparation meets opportunity.

Seneca

Um Serviço de Descoberta Ciente de Contexto para Internet das Coisas

Autor: B.Sc. Porfírio Dantas Gomes
Orientador(a): Prof.ª Dra. Thais Vasconcelos Batista

Resumo

A *Internet das Coisas* (do inglês *Internet of Things* - IoT) é um paradigma emergente caracterizado por uma miríade de dispositivos heterogêneos conectados à Internet. Porém, a alta heterogeneidade e a larga distribuição dos dispositivos disponíveis em IoT dificultam a implantação desse paradigma, fazendo com que máquinas e usuários enfrentem desafios para encontrar, selecionar, e usar recursos de forma rápida, confiável, e amigável. Nesse contexto, serviços de descoberta desempenham um papel chave, permitindo que clientes (e.g., plataformas de *middleware*, usuários finais, e aplicações) recuperem recursos através da especificação de critérios de busca contendo uma série de atributos, tais como o tipo do recurso, capacidades, localização, parâmetros de qualidade de contexto (*Quality of Context* - QoC), etc. Esta dissertação introduz o *QoDisco*, um serviço de descoberta distribuído que permite buscas por múltiplos atributos, buscas em intervalos, e operações de busca síncrona e assíncrona. Além disso, o *QoDisco* inclui um modelo baseado em ontologias para a descrição semântica de recursos (i.e., sensores e atuadores), serviços, e dados capturados por sensores. Esta dissertação apresenta, em detalhes, (i) a arquitetura do *QoDisco*, (ii) seu modelo de informação, (iii) a implementação de um protótipo, (iv) e a integração do *QoDisco* com uma plataforma de *middleware* para IoT, a EcoDiF. Por fim, este trabalho apresenta uma prova de conceito em um cenário de poluição urbana e uma avaliação qualitativa do desempenho do procedimento de busca do *QoDisco*.

Palavras-chave: Internet das Coisas, IoT, Serviço de descoberta, Descrição semântica, Ciência de contexto, Informação de contexto, Qualidade de contexto, QoC.

A Context-Aware Discovery Service for Internet of Things

Author: Porfírio Dantas Gomes, B.Sc.
Supervisor: Prof. Thais Vasconcelos Batista, Ph.D.

Abstract

The Internet of Things (IoT) is an emergent paradigm characterized by a plethora of heterogeneous devices connected to the Internet. However, the high heterogeneity and the wide distribution of the available devices in IoT make difficult the adoption of this paradigm, with users facing challenges to find, select, and use resources in a fast, reliable, and user-friendly way. In this context, discovery services perform a key role, allowing clients (e.g., middleware platforms, end-users, and applications) to retrieve resources through the specification of search criteria containing attributes, such as the resource type, capabilities, location, quality of context (QoC) parameters, and etc. This dissertation introduces the *QoDisco*, a distributed discovery service to search resources by multiple attributes, range queries, and synchronous and asynchronous operations. Furthermore, *QoDisco* includes an ontology-based model for semantic description of resources (i.e., sensors and actuators), services, and data captured by sensors. This dissertation presents, in details, (i) the *QoDisco* architecture, (ii) its information model, (iii) a propotype implementation, (iv) and the integration of *QoDisco* with a middleware platform for IoT, EcoDiF. Lastly, this work presents a proof of concept in an urban pollution scenario and a qualitative avaliation of the QoDisco's search procedure performance.

Keywords: Internet of Things, IoT, Discovery service, Semantic description, Context-aware, Context information, Quality of context, QoC.

Lista de figuras

1 Relação entre os termos: entidade de interesse, dispositivo, recurso, e serviço. . . . . p. 26

2 Arquitetura da Ecodif. . . . . p. 31

3 Meta-modelo do QoCIM. . . . . p. 37

4 Exemplo de uso do QoCIM. . . . . p. 37

5 Exemplo de modelagem RDF. . . . . p. 40

6 Indivíduos na OWL. . . . . p. 41

7 Propriedades na OWL. . . . . p. 41

8 Classes na OWL. . . . . p. 42

9 Padrão Estímulo-Sensor-Observação. . . . . p. 43

10 Perspectiva de sensores. . . . . p. 44

11 Perspectiva de sistemas. . . . . p. 45

12 Principais classes e propriedades da SAN e a sua relação com a SSN. . p. 47

13 Ontologia OWL-S. . . . . p. 52

14 Formato de consulta SPARQL. Fonte: Arenas *et al.*, página 4 (ARENAS; GUTIERREZ; PÉREZ, 2010). . . . . p. 55

15 Interação entre o *QoDisco* e os seus clientes. . . . . p. 62

16 Arquitetura do *QoDisco*. . . . . p. 63

17 Relação entre repositório e documentos RDO. . . . . p. 64

18 Processo de busca síncrona de registros. . . . . p. 69

19 Processo de busca assíncrona de registros. . . . . p. 69

20 Processo de inserção de documento RDO. . . . . p. 70

21 Processo de inserção de registro. . . . . p. 71
22 Modelo de informação do *QoDisco*. . . . . p. 71
23 Componentes do *framework Jena*. Fonte: http://jena.apache.org . . . . . p. 74
24 Servidor Fuseki executando. . . . . p. 75
25 Página Web para consulta de registros. . . . . p. 76
26 Página Web de registro de novos recursos. . . . . p. 78
27 Página Web de busca de registros. . . . . p. 79
28 Página Web de busca de registros por localização. . . . . p. 79
29 Página Web de busca de registros usando a API do YASQE. . . . . p. 80
30 Página Web contendo resultados de uma busca por registros. . . . . p. 81
31 Página Web de registro de novos critérios de QoC. . . . . p. 81
32 Integração entre a EcoDiF e o *QoDisco*. . . . . p. 84
33 Página Web da EcoDiF. . . . . p. 89
34 Buscando registros usando a interface Web do *QoDisco*. . . . . p. 95
35 Aplicações do cenário de poluição. . . . . p. 95

Lista de tabelas

1 Exemplos de aplicações de IoT no domínio de cuidados de saúde. . . . p. 27

2 Requisitos para um serviço de descoberta em IoT. . . . . p. 60

3 API dos repositórios. . . . . p. 64

4 API do *QoDisco*. . . . . p. 67

5 Tempo gasto pelo *QoDisco* (em milissegundos) para responder a uma *query* de busca à medida que o número de observações aumenta. . . . . p. 99

6 Tempo gasto pelo *QoDisco* (em milissegundos) para responder a uma *query* de busca à medida que o número de repositórios, cada um com 1000 observações registradas, aumenta. . . . . p. 100

7 Tempo gasto pelo *QoDisco* (em milissegundos) para notificar clientes assincronamente. . . . . p. 101

8 Tempo gasto pelo *QoDisco* (em milissegundos) para responder a uma *query* de busca em uma rede simulada à medida que o número de observações aumenta. . . . . p. 101

9 Comparação do *QoDisco* com outras propostas de serviços de descoberta para IoT. . . . . p. 103

Lista de Trechos de Código

2.1 Exemplo de especificação usando a SSN . . . . . p. 45
2.2 Descrição de um atuador através da ontologia SAN. . . . . p. 49
2.3 Descrição da classe *Alarm Sound Property*. . . . . p. 49
2.4 Descrição do atuador através da *perspectiva de operação*. . . . . p. 50
2.5 Descrição da característica de interesse *AlarmFeature*. . . . . p. 50
2.6 Especificação da entrada do atuador. . . . . p. 51
2.7 Trecho de descrição OWL-S do serviço *ReservarVoo*. . . . . p. 52
2.8 Exemplo de consulta SPARQL. . . . . p. 55
2.9 Exemplo da operação *INSERT DATA* em SPARQL. . . . . p. 56
2.10 Exemplo de modificação de dados usando a SPARQL. . . . . p. 56
2.11 Exemplo de remoção de dados usando a SPARQL. . . . . p. 57
3.1 Exemplo de requisição síncrona de busca de registros. . . . . p. 66
3.2 Exemplo de requisição assíncrona de busca de registros. . . . . p. 68
3.3 Classe criada para representar critérios de QoC no modelo de informação do *QoDisco*. . . . . p. 72
3.4 Classe criada para representar indicadores de QoC no modelo de informação do *QoDisco*. . . . . p. 73
3.5 Assinatura da API do *QoDisco*. . . . . p. 81
3.6 Classe *RepositoryService*. . . . . p. 84
3.7 Classe *RepositoryDao*. . . . . p. 85
3.8 Método utilizado para a inserção de observações. . . . . p. 86
3.9 Método responsável pela inserção de descrições de saída de sensores. . . p. 87
3.10 Método responsável pela inserção de valores de observações. . . . . p. 87
3.11 Método responsável pela inserção de indicadores de QoC. . . . . p. 88
4.1 Descrição do recurso *Sensor01*. . . . . p. 92
4.2 Observação feita pelo *Sensor01*. . . . . p. 92
4.3 Saída do sensor *Sensor01*. . . . . p. 92
4.4 Indicador de QoC para a observação *Observation01*. . . . . p. 93
4.5 Critério de QoC *Precision*. . . . . p. 93
4.6 *Query* SPARQL. . . . . p. 94

4.7 *Query* SPARQL usada nos experimentos. . . . . p. 98

Lista de abreviaturas e siglas

IoT – *Internet of Things*

QoC – *Quality of Context*

SOUPA – *Standard Ontology for Ubiquitous and Pervasive Applications*

M2M – Machine-to-Machine

EcoDiF – Ecossistema Web de Dispositivos Físicos

UML – *Unified Modeling Language*

QoCIM – *Quality of Context Information Meta-model*

RDF – *Resource Description Framework*

URI – *Uniform Resource Identifier*

OWL – *Web Ontology Language*

UNA – *Unique Name Assumption*

SSN – *Semantic Sensor Networks*

SAN – *Semantic Actuator Network*

OWL-S – *Ontology Web Language for Services*

IOPE – *Input, Ouput, Precondition, Effect*

XML – *eXtensible Markup Language*

SQL – *Structured Query Language*

URL – *Unified Resource Locator*

Sumário

1 Introdução p. 19

1.1 Motivação . . . . . p. 21

1.2 Objetivos . . . . . p. 21

1.3 Metodologia . . . . . p. 22

1.4 Estrutura do trabalho . . . . . p. 23

2 Fundamentação Teórica p. 24

2.1 Internet das Coisas . . . . . p. 24

2.1.1 *Middleware* para Internet das Coisas . . . . . p. 28

2.1.1.1 EcoDiF . . . . . p. 30

2.2 Sensibilidade ao Contexto . . . . . p. 33

2.2.1 QoCIM . . . . . p. 35

2.3 Ontologias . . . . . p. 38

2.3.1 RDF . . . . . p. 39

2.3.2 OWL . . . . . p. 40

2.3.3 Ontologias para IoT . . . . . p. 41

2.3.3.1 SSN . . . . . p. 42

2.3.3.2 SAN . . . . . p. 46

2.3.3.3 OWL-S . . . . . p. 51

2.4 Busca Semântica . . . . . p. 53

2.4.1 SPARQL . . . . . p. 54

3 QoDisco p. 58

3.1 Requisitos . . . p. 59

3.2 Arquitetura . . . p. 62

3.2.1 Módulo de Ontologias . . . p. 63

3.2.2 Módulo de Repositórios . . . p. 63

3.2.3 Módulo de Busca . . . p. 65

3.2.4 Módulo de Operações . . . p. 65

3.3 Funcionamento . . . p. 68

3.4 Modelo de informação . . . p. 70

3.5 Implementação . . . p. 73

3.5.1 *QoDisco* Web . . . p. 76

3.5.2 Integração com a EcoDiF . . . p. 83

4 Avaliação p. 90

4.1 Cenário . . . p. 90

4.1.1 Modelo de informação do *QoDisco* no cenário de poluição de ar p. 91

4.1.2 *Queries* de busca . . . p. 94

4.1.3 Aplicações . . . p. 95

4.2 Configuração dos experimentos . . . p. 97

4.3 Resultados . . . p. 98

5 Trabalhos Relacionados p. 102

5.1 Modelagem de recursos e informação de contexto em IoT . . . p. 102

5.2 Aspectos Arquiteturais . . . p. 105

6 Considerações Finais p. 107

6.1 Contribuições . . . p. 107

6.2 Limitações . . . p. 110

6.3 Trabalhos em andamento e futuros . . . . . p. 111

Referências p. 112

1 Introdução

A *Internet das Coisas* (do inglês *Internet of Things* - IoT) (ATZORI; IERA; MORABITO, 2010) é um paradigma emergente que visa a integração do mundo físico com o mundo digital através da comunicação e interação com *entidades físicas* perfeitamente interconectadas (também chamadas de *coisas*) que provêm informação com valor agregado e funcionalidades para usuários finais e/ou aplicações. Para monitorar ou interagir com essas entidades físicas, *dispositivos* podem ser anexados ou incorporados nelas, ou até mesmo implantados no ambiente onde essas entidades estão inseridas. Dois tipos de dispositivos são de particular relevância em IoT, *sensores* e *atuadores* (HOLLER et al., 2014). *Sensores* tipicamente obtém dados de entidades ou do ambientes em que elas estão inseridas, enquanto *atuadores* são capazes de atuar em uma entidade ou em seu ambiente. Ambos, sensores e atuadores, podem incluir *recursos*, componentes de *software* que permitem que clientes (i.e., plataformas de *middleware*, usuários finais, e aplicações) interajam com os dispositivos através da Internet. Como implementações de recursos podem ser altamente dependentes do *hardware* do dispositivo, *serviços* provêm interfaces de alto-nível que expõem as funcionalidades dos recursos. Por fim, vale ressaltar que *recursos* podem ser descritos em termos de metadados, tais como tipo do recurso, capacidades, localização, etc. (BASSI et al., 2013), e também através das informações requeridas para interagir com os serviços expostos pelos mesmos.

O número de recursos de IoT implantados ao redor do mundo está crescendo rapidamente. A Cisco IBSG (*Internet Business Solutions Group*) estima que existem cerca de 15 bilhões de dispositivos conectados a Internet e prediz que cerca de 50 bilhões estarão conectados até 2020 (MACAULAY; BUCKALEW; CHUNG, 2015). Entretanto, comunicar-se com esses recursos não cria nenhum valor adicional a menos que o cliente seja capaz de escolher os recursos apropriados para a realização de suas atividades. Porém, a alta heterogeneidade inerente dos recursos disponíveis em IoT, uma consequência das diferentes capacidades/funcionalidades e protocolos de rede dos dispositivos, dificulta a implantação desse paradigma. Dificuldade essa que, quando a visão de IoT for implementada em sua

extensão, tende a ser ainda maior, com máquinas e usuários enfrentando desafios para encontrar, selecionar, e usar recursos de forma rápida, confiável, e amigável (MAYER; GUINARD, 2011). Nesse contexto, serviços de descoberta desempenham um papel importante, permitindo que clientes recuperem recursos disponíveis baseados em um dado critério de busca, até mesmo se sua localização é desconhecida (PAGANELLI; PARLANTI, 2012).

Em IoT, a tarefa de descoberta pode significantemente beneficiar-se da *ciência de contexto*, i.e., considerando *contexto* como qualquer informação de ambiente que pode ser usada para caracterizar a situação de entidades físicas que é considerada relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e aplicação (ABOWD et al., 1999). Contexto é tipicamente a localização, identidade, e estado de pessoas, grupos, e objetos físicos computacionais em associação com os dados providos pelos dispositivos. De fato, o objetivo final da IoT é construir um ecossistema que provenha serviços orientados ao usuário e cientes de contexto (WEI; JIN, 2012), por isso é importante aumentar a riqueza do processo de descoberta e suportar ações de tomada de decisão com base em informações de contexto. Um serviço de descoberta ciente de contexto em IoT permite a provisão de recursos de acordo com a situação atual do usuário e/ou da aplicação e com intervenção humana mínima, além de facilitar a interpretação de dados e a comunicação *Machine-to-Machine* (M2M).

Entretanto, abordagens tradicionais de descoberta de serviços Web cientes de contexto não são apropriadas para a descoberta de serviços em IoT devido as diferenças existentes entre serviços providos por dispositivos físicos e serviços Web tradicionais. Primeiramente, serviços providos por dispositivos provêm o estado de entidades do mundo físico quase que em tempo real. Enquanto isso, serviços Web tradicionais são entidades inteiramente virtuais que encapsulam lógicas de negócio. Além disso, dispositivos que expõem serviços possuem tipicamente capacidade computacional, banda de rede, bateria, e armazenamento limitado, enquanto serviços Web tradicionais são normalmente implantados em computadores sem tantas restrições. Ademais, serviços providos por dispositivos são encontrados em ambientes altamente dinâmicos, onde os recursos e seus serviços constantemente degradam, desaparecerem e possivelmente reaparecem devido a desconexão intermitente de redes sem-fio, mobilidade e as capacidades altamente limitadas dos dispositivos, enquanto serviços Web tradicionais são, em geral, estáticos. Por fim, vale ressaltar que informações de contexto em IoT são inerentemente imperfeitas e inconsistentes, assim requerendo que seja considerado conhecimento adicional em termos de Qualidade de Contexto (QoC) (MARIE et al., 2014), e.g., informação caracterizando a acurácia, precisão, completude, e atualidade de dados.

Pelos motivos discutidos acima, é fundamental que seja desenvolvido um serviço de descoberta ciente de contexto que seja compatível com o ambiente altamente dinâmico de IoT, e que possa, de forma autônoma, prover serviços centrados no usuário e cientes de contexto com ou sem a requisição de usuários, e com mínima intervenção humana.

1.1 Motivação

Apesar de já existirem serviços de descoberta de recursos para IoT (PERERA et al., 2013b) (WEI; JIN, 2012) (CIRANI et al., 2014), a maioria é baseada em arquiteturas centralizadas e/ou não definem um modelo de informação para a descrição de recursos, como mostra o *survey* realizado por Issarny *et al* (ISSARNY et al., 2011). Além disso, a maioria dos modelos de informação propostos por serviços de descoberta para IoT não suportam o gerenciamento de informação de contexto, sendo que nenhum deles suporta a adição de parâmetros de QoC (LI; ZAMAN; LI, 2015) (PERERA et al., 2013b) (JAYARAMAN; CALBIMONTE; QUOC, 2015). Ademais, um mapeamento sistemático (MAIA et al., 2014), realizado pelo grupo de pesquisa do qual o trabalho apresentado nesta dissertação faz parte, auxiliou no levantamento de uma série de requisitos definidos na literatura que não são plenamente atendidos por serviços de descoberta para IoT (DEY et al., 2014) (KHODADADI; DASTJERDI; BUYYA, 2015) (CIRANI et al., 2014), como: (i) o suporte a definição de critérios de busca contendo mais de um atributo, (ii) a provisão de uma API de gerenciamento que permita a adição, atualização, e remoção de metadados associados a recursos, (iii) o suporte a múltiplos publicadores, (iv) buscas por intervalos de valores e não somente pelo casamento exato de valores de metadados de recursos, e (v) esquema de identificação flexível (PAGANELLI; PARLANTI, 2012). Por fim, devido ao contexto dinâmico em que recursos de IoT operam, há a necessidade que o serviço de descoberta possibilite requisições assíncronas, que não é uma facilidade comumente oferecido por serviços existentes. Por tais motivos, a definição de uma arquitetura adequada para serviços de descoberta no domínio de IoT, com um modelo de informação associado, ainda são desafios em aberto na literatura, o que motiva a realização deste trabalho.

1.2 Objetivos

Este trabalho tem como objetivo apresentar o *QoDisco*, um serviço de descoberta ciente de contexto para IoT. O *QoDisco* é estruturado sobre uma arquitetura distribuída composta por repositórios autônomos que armazenam descrições de recursos, incluindo

os dados capturados pelos recursos (nesse caso, *sensores*). O *QoDisco* provê uma API para realizar tarefas de descoberta em tais repositórios e está de acordo com requisitos importantes para descoberta de recursos no contexto de IoT, e.g., consultas em intervalo e com múltiplos-atributos, e operações síncronas e assíncronas. Além disso, o *QoDisco* envolve um modelo de informação baseado em ontologias com um vocabulário consistente de conceitos relacionados a recursos e informações de contexto, incluindo o relacionamento entre eles. Esse modelo de informação utiliza ontologias bem definidas e padronizadas para a descrição semântica de recursos e seus serviços, são elas: as ontologias SAN (*Semantic Actuator Network*) (SPALAZZI; TACCARI; BERNARDINI, 2014), SSN (*Semantic Sensor Network*) (BARNAGHI et al., 2011), SOUPA (*Standard Ontology for Ubiquitous and Pervasive Applications*) (CHEN; FININ; JOSHI, 2005), e a OWL-S (*Ontology Web Language for Services*) (MARTIN et al., 2007). Adicionalmente, parte do meta-modelo QoCIM (*Quality of Context Information Meta-model*) (MARIE et al., 2014)(MARIE et al., 2015)(CHABRIDON et al., 2016) foi adicionado ao modelo de informação do *QoDisco* visando lidar com critérios de QoC no procedimento de busca.

Como objetivo secundário, o *QoDisco* deve ser integrado a uma plataforma de *middleware* para IoT. A plataforma de *middleware* escolhida foi a EcoDiF (Ecossistema Web de Dispositivos Físicos) (PIRES et al., 2014), devido este trabalho fazer parte de um projeto mais amplo do grupo de pesquisa onde a EcoDiF foi desenvolvida. Porém, o *QoDisco* é independente da EcoDiF, podendo ser usado para a busca de recursos e dados capturados por sensores por outras plataformas de *middleware*, aplicações cliente e usuários finais.

1.3 Metodologia

O processo de desenvolvimento do *QoDisco* foi dividido em três níveis de abstração: (i) *o nível de requisitos*, onde requisitos presentes na literatura e alguns novos são definidos para um serviço de descoberta de recursos no contexto de IoT; (ii) *o nível arquitetural*, onde foi definido, baseado nos requisitos, a estrutura do serviço de descoberta em termos de componentes, bem como sua API, e; (iii) *o nível de implementação*, onde foi realizada a implementação do *QoDisco* de acordo com a arquitetura proposta. Visando atender os objetivos apresentados na Seção 1.2, e seguindo o processo de desenvolvimento supracitado, as seguintes atividades foram realizadas:

1. Uma revisão da literatura para identificar o estado da arte atual em se tratando de serviços de descoberta para IoT, bem como para elicitar os requisitos do serviço.

2. A definição de um conjunto de requisitos que deverão ser atendidos por um serviço de descoberta de recursos no contexto de IoT.

3. A definição de um modelo de informação para a especificação de recursos (i.e., sensores e atuadores) e dados capturados por sensores.

4. A definição de uma arquitetura e a implementação do serviço de descoberta baseado em uma arquitetura distribuída.

5. A integração do serviço de descoberta com a plataforma EcoDiF.

6. A validação do serviço de descoberta por meio da sua utilização em um cenário relacionado ao monitoramento da poluição do ar em uma área urbana.

1.4 Estrutura do trabalho

Este trabalho está estruturado da seguinte forma: o Capítulo 2 apresenta os conceitos básicos necessários para a compreensão do restante deste trabalho, o que inclui a contextualização de Internet das Coisas, de ontologias para a descrição de recursos e serviços, de informações de contexto e de critérios e indicadores de QoC, e de serviços de descoberta, além da apresentação da plataforma de *middleware* EcoDiF. O Capítulo 3 apresenta o *QoDisco*, incluindo seus requisitos, arquitetura, funcionamento, modelo de informação, protótipo de implementação, e integração com a EcoDiF. O Capítulo 4 apresenta a aplicação da solução proposta em um cenário relacionado a poluição urbana do ar e a validação do *QoDisco* por meio de uma avaliação quantitativa do desempenho do seu procedimento de busca. Ademais, o Capítulo 5 discute os trabalhos relacionados ao trabalho proposto nesta dissertação. Por fim, o Capítulo 6 apresenta as considerações finais e discute possíveis trabalhos futuros.

2 Fundamentação Teórica

Este capítulo introduz os conceitos e as definições necessárias para o entendimento deste trabalho. A Seção 2.1 apresenta uma breve descrição sobre Internet das Coisas, incluindo definições, exemplos de domínios de aplicação, e soluções de *middleware* encontradas nessa área. A Seção 2.2 define o conceito de informação de contexto e apresenta a importância de parâmetros de qualidade de contexto na definição desse tipo de informação. A Seção 2.3 expõe conceitos e exemplos de ontologias relevantes para a modelagem de informações em diversas áreas, especialmente em IoT. Ademais, a Seção 2.4 expõe conceitos relacionados à busca semântica e apresenta a linguagem de consulta SPARQL[1], comumente usada para permitir a busca por informações expressas através de ontologias.

2.1 Internet das Coisas

A *Internet das Coisas* (*Internet of Things* - IoT), também conhecida como *Internet dos Objetos*, é um paradigma que vem crescendo no cenário moderno de telecomunicações sem fio. A ideia básica deste conceito é conectar entidades físicas à Internet através de dispositivos físicos - como *Radio-Frequency IDentification* (RFID), sensores, atuadores, e telefones celulares - que, por meio de mecanismos de endereçamento único, são capazes de interagir e cooperar uns com os outros para alcançar objetivos em comum. Com isso, é esperado que a Internet passe a comunicar não somente pessoas com pessoas, mas também dispositivos com pessoas e dispositivos com dispositivos, do inglês *Machine-to-Machine* (M2M).

Além de entidades físicas e dispositivos, outros dois termos são tipicamente associados com a Internet das Coisas e são fundamentais no entendimento do trabalho apresentado nesta dissertação, são eles: *recursos* e *serviços*. A Figura 1, baseada em De *et al.* (DE et al., 2011), apresenta a relação entre esses quatro termos, cujas definições são expostas a seguir.

[1] *SPARQL Query Language for RDF*: `https://www.w3.org/TR/rdf-sparql-query/`

- *Entidades físicas*: também chamadas de *coisas* ou *entidades de interesse*, são objetos físicos que alguém deseja monitorar ou interagir. Exemplos incluem objetos inanimados - como carros, geladeiras, casas e prédios - e objetos animados - como pessoas e animais. Basicamente, qualquer objeto que é relevante para um usuário ou aplicação pode ser considerado uma entidade de interesse.

- *Dispositivos*: componentes de *hardware* que permitem a comunicação e interação com as entidades físicas. Dispositivos podem ser anexados a essas entidades ou instalados no ambiente em que elas estão inseridas. Dois tipos de dispositivos são fundamentais no contexto de IoT: *sensores* e *atuadores* (HOLLER et al., 2014). Sensores são dispositivos capazes de fornecer informações acerca da situação de um entidade de interesse e/ou do ambiente em que a mesma está inserida. Enquanto isso, atuadores são dispositivos capazes de atuar em entidades de interesse ou ambientes.

- *Recursos*: componentes de *software* que abstraem a comunicação com os dispositivos. Recursos são tipicamente hospedados em um dispositivo e interagem com as entidades de interesse, provendo informações e/ou permitindo controle sobre elas através da Internet.

- *Serviços*: componentes de *software* que permitem o acesso de usuários aos recursos. Plataformas de *middleware*, aplicações e usuários interagem via internet com os serviços expostos pelos recursos, e não com o recurso em si. Serviços expõem as funcionalidades dos recursos através de interfaces abertas e padronizadas, abstraindo assim os detalhes de implementação de baixo-nível dos recursos. Por exemplo, um sensor de temperatura implantado em uma sala pode hospedar um componente de *software* (i.e., recurso) que recebe e responde consultas acerca da temperatura da sala através dos seus serviços.

Dentro do domínio de IoT, um dispositivo não gera nenhum valor ao menos que esteja associado a um recurso, necessário para que o dispositivo interaja via Internet, que, por sua vez, necessita de serviços para prover qualquer tipo de interação. Por isso, apesar das diferentes definições, o termo *recurso* é bastante usado como um termo genérico que engloba os conceitos de dispositivo, do próprio recurso, e de serviços de IoT, inclusive ao longo deste trabalho.

A Internet das Coisas tem e terá cada vez mais impacto sobre vários domínios, influenciando aspectos do cotidiano e do comportamento das pessoas (ATZORI; IERA; MORABITO, 2010). Para se ter uma ideia, a IoT é considerada pelo Conselho Nacional de Inteligência

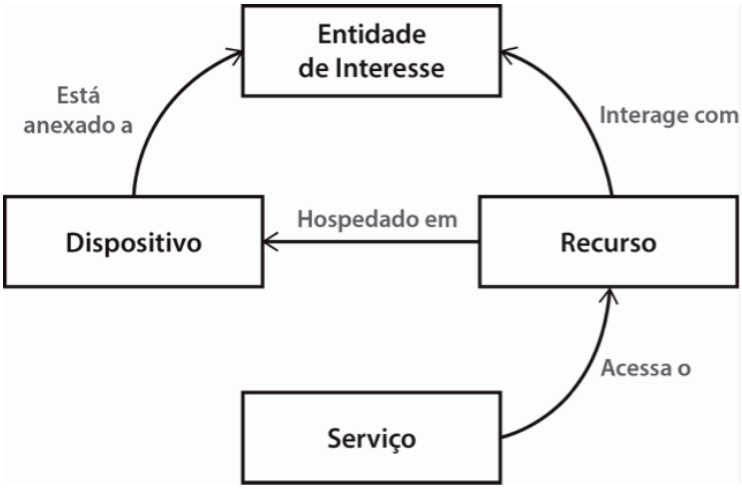


Figura 1: Relação entre os termos: entidade de interesse, dispositivo, recurso, e serviço.

dos EUA (NIC) como uma das seis tecnologias civis com impactos potenciais sobre o poder nacional dos EUA, podendo ser encontradas aplicações de IoT em domínios como: cidades inteligentes, cuidados de saúde (do inglês *healthcare*), agricultura, logística e segurança pública. Para ilustrar a aplicabilidade de IoT em um desses domínios, a Tabela 1 apresenta alguns cenários do domínio de cuidados de saúde descritos por Pereira (PEREIRA, 2014) onde o emprego da infraestrutura tecnológica provida pela IoT é benéfico.

Porém, o ambiente em que as aplicações de IoT estão inseridas é tipicamente caracterizado por um alto nível de heterogeneidade de *software* e de *hardware*, sendo composto por vários dispositivos com diferentes capacidades, funcionalidades, e protocolos de rede (PIRES et al., 2014). Para abstrair essa diversidade e prover acesso transparente a dispositivos, aplicações de IoT tipicamente utilizam plataformas de *middleware*. Tais plataformas são inseridas entre as aplicações e a infraestrutura (de comunicação, processamento e sensoriamento) subjacente, provendo um meio padronizado para o acesso aos dados e serviços fornecidos pelos recursos através de uma interface de alto nível. A adoção de uma plataforma de *middleware* também pode contribuir para facilitar a construção de aplicações para IoT. Nesse contexto, o desafio reside no fato de que, a fim de permitir a criação de aplicações que combinem recursos do mundo físico disponibilizados via Web, são necessários modelos de alto nível que abstraiam os serviços e dispositivos físicos subjacentes. Com isso, usuários e aplicações consumidores dos dados originados dos dispositivos conectados não precisarão lidar com funcionalidades de baixo nível para a manipulação de tais objetos.

Entretanto, o grande volume de informações geradas por esses objetos tem sido cada vez mais difíceis de serem gerenciadas por essas plataformas de *middleware*, que preci-

Tabela 1: Exemplos de aplicações de IoT no domínio de cuidados de saúde.

Aplicação	**Motivação**	**Contribuição da IoT**
Monitoramento de pacientes	Em hospitais ou em residências, pacientes podem necessitar de monitoramento e cuidados de saúde constantes.	Uma aplicação composta por uma rede de sensores pode permitir o monitoramento de informações acerca do estado de saúde de um paciente, fornecendo-as à equipe médica responsável pelo mesmo. Essas informações são obtidas a fim de auxiliar os médicos a regular a dosagem de remédios do paciente. Além da provisão dessas informações, a aplicação pode alertar a equipe médica acerca da ocorrência de alguma emergência com o paciente.
Suprimentos médicos	Companhias farmacêuticas precisam monitorar suprimentos médicos do momento que eles são produzidos até o momento da sua entrega, incluindo a destruição desses suprimentos em caso de não utilização	Etiquetas RFID podem ser anexadas a esses suprimentos, permitindo que companhias farmacêuticas identifiquem, gerenciem e monitorem suprimentos médicos.
Cuidados no esporte	O monitoramento e registro de atividades físicas e de informações acerca do estado de saúde de um indivíduo pode incentivar a prática de hábitos mais saudáveis.	Sensores e softwares especializados podem permitir o monitoramento das atividades físicas realizadas por usuários e fornecer dicas que ajudem a melhorar o desempenho do mesmo nessas atividades.
Radiação ultravioleta	O monitoramento dos níveis de radiação ultravioleta pode ajudar a evitar o câncer de pele.	Informações em tempo real da localização do usuário, hora, e posição do sol podem ser providas por uma aplicação para indicar em quais momentos do dia a exposição ao sol pode ser benéfica ou não a saúde do usuário .

sam empregar estratégias que identifiquem e determinem os recursos mais adequados para cada aplicação, dentre a miríade de recursos disponíveis. Para isso, é necessário a descrição

das características físicas e das funcionalidades desses dispositivos, que devem ser especificadas seguindo um modelo único padrão. Esse modelo deve permitir a definição, sem ambiguidade, das características e funcionalidades dos dispositivos, além da especificação das informações necessárias de acesso a esses recursos via Internet. Para permitir a automatização da descoberta e a invocação dos serviços providos pelos recursos, esses modelos são tipicamente definidos seguindo uma abordagem de descrição semântica de recursos. Uma vez que recursos estejam descritos semanticamente, torna-se possível, para máquinas, a interpretação das funcionalidades desses recursos. Essa descrição é feita normalmente através de modelagem semântica por ontologias (BARNAGHI et al., 2011)(SPALAZZI; TACCARI; BERNARDINI, 2014), que permitem a descrição das características e funcionalidades dos recursos de IoT, fornecendo interoperabilidade entre dispositivos, provedores de dados e consumidores, e facilitando o acesso de dados efetivo, integração, descoberta de recursos, interpretação semântica, e extração de conhecimento.

A Seção 2.1.1 descreve o que é uma plataforma de *middleware* e qual é o seu papel no contexto de IoT.

2.1.1 *Middleware* para Internet das Coisas

Uma plataforma de *middleware* para IoT é um artefato de *software* que reside entre a camada de aplicação e a camada de *hardware*, fornecendo acesso padronizado aos dados e serviços providos por recursos de IoT (ZARGHAMI, 2013). O desenvolvimento de plataformas de *middleware* tem atraído cada vez mais atenção da comunidade acadêmica e da indústria. Evidência disso é que diversas pesquisas já foram reportadas na literatura visando a concepção e a implementação de plataformas de *middleware* para IoT, tais como a OpenIoT (SOLDATOS; SERRANO; HAUSWIRTH, 2012) e a Xively[2].

No contexto de IoT, plataformas de *middleware* devem satisfazer a um conjunto de requisitos a fim de atender às necessidades de usuários e aplicações. Apesar desses requisitos variarem de domínio para domínio, alguns são frequentemente mencionados na literatura, como: (i) interoperabilidade; (ii) descoberta e gerenciamento de dispositivos; (iii) interfaces de alto nível; (iv) ciência de contexto; (v) escalabilidade; (vi) gerenciamento de grandes volumes de dados; (vii) segurança, e; (viii) adaptação dinâmica. A definição e os cenários em que cada um desses requisitos são necessários no contexto de plataformas de *middleware* em IoT fogem do escopo deste trabalho. Entretanto, dentre os oito requisitos supracitados, alguns podem ser atendidos total ou parcialmente por serviços de

[2] *Xively – Business solutions for the Internet of Things*: `http://xively.com`

descoberta, como o requisito de descoberta e gerenciamento de dispositivos, o de adaptação dinâmica, e o de ciência de contexto. Esses três requisitos são descritos em mais detalhes a seguir.

- *Descoberta e gerenciamento de dispositivos*: As infraestruturas de comunicação em ambientes de IoT possuem topologia frequentemente dinâmica e desconhecida, visto que dispositivos podem ser integrados ao ambiente e utilizados de maneira oportunista e não previamente planejada (DELICATO; PIRES; BATISTA, 2013). Por isso, é importante que uma plataforma de *middleware* ofereça mecanismos que permitam que recursos sejam dinamicamente descobertos a fim de atender os requisitos das aplicações. Além disso, é necessário prover mecanismos para o gerenciamento de recursos, que diz respeito à capacidade de fornecer informações de localização e estado do dispositivo.

- *Adaptação dinâmica*: Considerando a alta dinamicidade dos ambientes de IoT, nos quais recursos podem se tornar indisponíveis pelos mais diversos motivos (e.g., falha, capacidade energética, indisponibilidade de conexão à rede, mobilidade de usuário, etc.), plataformas de *middleware* devem prover estratégias para adaptação dinâmica, garantindo assim a disponibilidade e qualidade das aplicações durante a sua execução. Dessa forma, plataformas de *middleware* em IoT devem manter-se disponíveis e funcionando adequadamente em ambientes dinâmicos, coletando, analisando e reagindo a mudanças no contexto em que elas e objetos a ela conectados estão inseridos.

- *Ciência de contexto*: De acordo com Dey (DEY, 2001), contexto é qualquer informação que pode ser utilizada para caracterizar uma pessoa, lugar ou objeto considerado relevante ao ambiente em questão. Dessa forma, informações de contexto, tais como o estado do objeto, seus vizinhos e sua localização, por exemplo, necessitam ser coletadas e processadas com o objetivo de efetuar ações ou reagir a estímulos com base nos dados extraídos (PERERA et al., 2013b). Plataformas de *middleware* em IoT devem possibilitar a coleta, gerenciamento e processamento de informações de contexto providas por múltiplas fontes, liberando as aplicações e usuários da tarefa de manipulá-las e tornando transparente tal manipulação. Mais detalhes sobre a definição e gerenciamento de informações de contexto são dadas na Seção 2.2.

Serviços de descoberta, em conjunto com um modelo de informação para a descrição de recursos, são essenciais para plataformas de *middleware*, responsabilizando-se pela

descoberta de recursos e permitindo o gerenciamento de metadados. Quanto a adaptação dinâmica, serviços de descoberta dão suporte através da busca por recursos a fim de substituir recursos indisponíveis. Finalmente, apesar da coleta e gerenciamento de informações de contexto não ser um papel a ser desempenhado por um serviço de descoberta, este pode permitir a anotação de dados capturados por sensores com informações de contexto e informações de qualidade de contexto (QoC). De fato, o serviço de descoberta apresentado neste trabalho faz parte de um projeto mais amplo que visa estender uma plataforma de *middleware* para IoT, a EcoDiF (PIRES et al., 2014), com o intuito de incluir funcionalidades de descoberta de recursos. A Seção 2.1.1.1 apresenta em detalhes a EcoDiF.

2.1.1.1 EcoDiF

A EcoDiF (Ecossistema Web de Dispositivos Físicos) é uma plataforma de *middleware* que integra dispositivos e os conecta à Internet, fornecendo funcionalidades de controle, visualização, processamento e armazenamento de dados em tempo real. A EcoDiF foi projetada tendo como base princípios REST (FIELDING; TAYLOR, 2002), utilizando assim o HTTP como mecanismo padrão de suporte à interações, por meio das quatro operações básicas fornecidas pelo HTTP (GET, POST, PUT, DELETE), entre dispositivos endereçáveis. Dessa forma, a plataforma provê uma interface bem definida para expor as funcionalidades dos dispositivos na Web, provendo padronização e simplificação do processo de desenvolvimento de aplicações, além de minimizar barreiras no tocante à incompatibilidade entre diferentes fabricantes, protocolos proprietários e formatos de dados. Desse modo, a plataforma soluciona parte das questões de interoperabilidade de dispositivos, requisito essencial para plataformas de *middleware* no contexto de IoT, e consolida a integração de dispositivos físicos heterogêneos. Assim, a EcoDiF é capaz de oferecer um acesso unificado a dados e serviços providos por dispositivos integrados através de interfaces de alto nível, bem como contribuir para tornar mais fácil o desenvolvimento de aplicações em ambientes de IoT altamente distribuídos e heterogêneos. A arquitetura lógica da plataforma é composta de sete módulos, ilustrados na Figura 2 e descritos a seguir.

1. *Módulo de Conexão de Dispositivos* - visa facilitar a conexão de dispositivos físicos à IoT através do uso de *drivers customizados*, desenvolvidos para cada tipo específico de plataforma de dispositivo. Os *drivers* da EcoDiF são construídos com base em princípios REST e padrões e protocolos da Web, desempenhando o papel de abs-

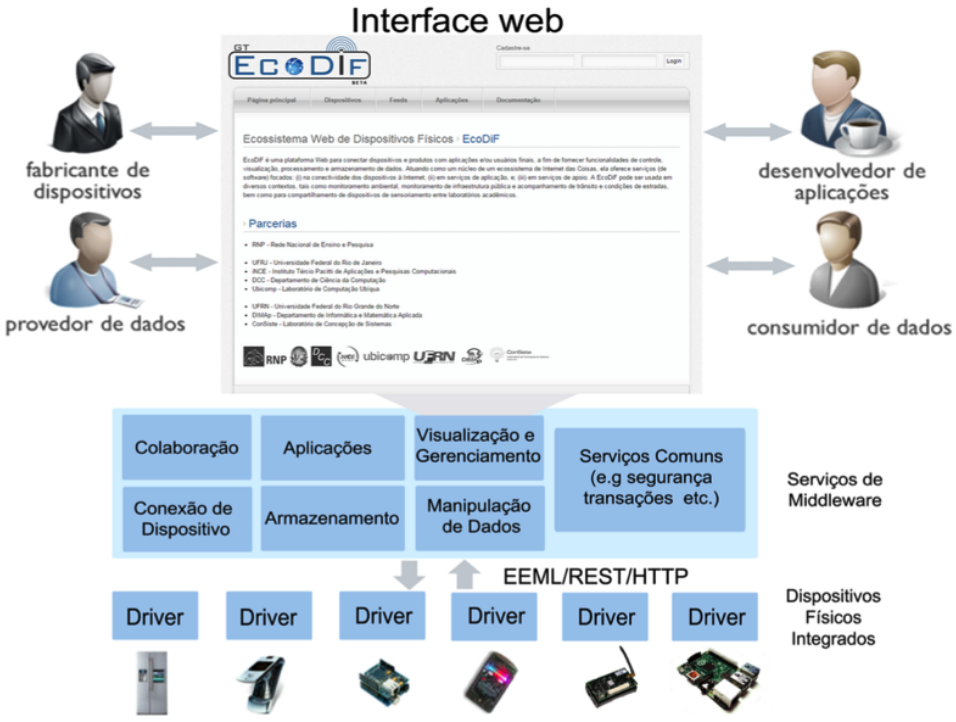


Figura 2: Arquitetura da Ecodif.

trair dos usuários e aplicações a heterogeneidade dos dispositivos. Resumidamente, os *drivers* obtém dados dos dispositivos (os chamados *feeds*[3]) e os estruturam no formado EEML (*Extended Environments Markup Language*[4]) .

2. *Módulo de Manipulação de Dados* - responsável por receber requisições HTTP PUT dos *drivers*, contendo representações EMML (*Enterprise Mashup Markup Language*[5]) dos dados dos dispositivos. A EMML é uma linguagem de marcação para a criação de *mashups*. *Mashups* são aplicações Web que consomem e realizam operações em dados de diversas fontes (no caso do EcoDIF, *feeds*) para, então, exibi-los através de uma interface gráfica.

3. *Módulo de Visualização e Gerenciamento* - provê uma interface Web que permite que usuários gerenciem os dispositivos conectados à EcoDiF. Através dessa interface, usuários podem monitorar o estado e localização de seus dispositivos, bem como visualizar dados históricos armazenados na plataforma.

4. *Módulo de Colaboração* - visa facilitar a colaboração entre usuários da EcoDiF,

[3] *Feed*: formato de dados comum na Web em cenários de comunicação onde os dados são atualizados frequentemente.

[4] *Environments Markup Language* (EEML): `http://www.eeml.org/`.

[5] *Enterprise Mashup Markup Language* (EMML) – `http://en.wikipedia.org/wiki/Enterprise_Mashup_Markup_Language`

permitindo realizar buscas por dispositivos e aplicações usando a interface Web a partir de seus respectivos metadados (e.g., tipo, usuário, localização). Vale ressaltar, porém, que esse módulo não suporta buscas mais complexas, como buscas por múltiplos atributos e em intervalos.

5. *Módulo de Armazenamento* - consiste de dois repositórios básicos, um para armazenamento de dados utilizando uma base de dados relacional, e outro para armazenamento de *scripts* de aplicações em um sistema de arquivos. É importante destacar que esses repositórios podem fazer uso de uma infraestrutura de computação em nuvem para armazenar dados e arquivos, provendo assim atributos de qualidade como robustez, confiabilidade, disponibilidade e escalabilidade.

6. *Módulo de Serviços Comuns* - envolve serviços de infraestrutura providos pela plataforma, tais como segurança (autenticidade, confidencialidade, integridade), gerenciamento de ciclo de vida de aplicações, transações, etc.

7. *Módulo de Aplicações* - provê um modelo e um ambiente para programação e execução de aplicações que: (i) faz uso dos dados (*feeds*) disponíveis na EcoDiF e, (ii) gera novas informações que podem ser disponibilizadas pela plataforma para serem usadas por outros *feeds* e aplicações mais complexas. Essas aplicações são construídas como *mashups*. O modelo de programação e execução adotado pela EcoDiF é baseado no uso da linguagem EMML. Aplicações *mashup* são implementadas como *scripts* escritos em EMML e executadas em um motor (*engine*) de execução que processa tais *scripts*.

Na EcoDiF, desenvolvedores de aplicações podem construir aplicações *mashup* fazendo uso dos dados providos pelos dispositivos (i.e., *feeds*) que são integrados à EcoDiF através do Módulo de Aplicações. Aplicações são implementadas usando a linguagem EMML, de modo que quando um *feed* é vinculado a uma aplicação, o *Módulo de Aplicações* automaticamente o inclui como uma variável de entrada em um *script* EMML para a aplicação, e os usuários podem, então, codificar a lógica de programação. Por outro lado, usuários que possuem conhecimento acerca da linguagem EMML podem escrever seus próprios *scripts* (fora do ambiente da EcoDiF, em um editor de sua escolha) e então importá-los para a plataforma. As aplicações registradas na EcoDiF são executadas utilizando o motor de execução EMML e seus respectivos resultados são exibidos para os usuários através da interface Web da EcoDiF.

2.2 Sensibilidade ao Contexto

A *computação ubíqua* pode ser descrita como um método para melhorar o uso de dispositivos computacionais, deixando-os disponíveis e dispersos no ambiente de forma *invisível* aos usuários (WEISER, 1993). Conforme descrito por Weiser (WEISER, 1999), as boas ferramentas são aquelas em que seu uso é tão natural que torna-se imperceptível no dia a dia, daí a noção de invisibilidade.

Computação ubíqua e *computação pervasiva* são termos muitas vezes utilizados de forma intercambiável (WEERASINGHE; WARREN, 2010). A computação pervasiva tem como objetivo capturar o contexto do ambiente e construir modelos computacionais que alteram seu comportamento de acordo com o *contexto*, ajustando-se dinamicamente às necessidades do usuário e sem a interferência deste. Contexto é qualquer informação que possa ser usada para caracterizar a situação de uma entidade (i.e., pessoa, lugar ou objeto) considerada relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação (ABOWD et al., 1999). Já a computação ubíqua combina a computação pervasiva com os avanços da computação móvel para prover ao usuário um contexto computacional global e integrado (ZHAO; WANG, 2010). Assim, entende-se que a sensibilidade ao contexto é a habilidade de um sistema ser sensível ao que está a sua volta, reagindo conforme o contexto evolui de um estado para outro. Também podemos observar que a relevância das informações e serviços depende das tarefas dos usuários, as quais são descobertas utilizando informações de contexto. Alguns tipos de informação são comumente utilizadas para determinar o contexto, como localização, temperatura, velocidade, distância até os recursos disponíveis, e etc. (KRUMM, 2009).

Em um sistema contendo um *middleware* e várias aplicações, o ciclo de vida das informações de contexto segue basicamente os seguintes passos:

1. *Obtenção da informação*: refere-se aos meios pelos quais a informação de contexto é capturada pelo sistema;

2. *Representação da informação*: busca organizar a informação de acordo com os requisitos das aplicações

3. *Processamento da informação*: utiliza o conjunto de informações obtido e representado nas etapas anteriores para extrair informações não explícitas que possam ser úteis às aplicações;

4. *Entrega da informação*: é responsável por fornecer as informações de contexto rela-

tivas a cada aplicação em execução no *middleware*.

Para organizar e armazenar as informações de contexto é preciso definir uma forma de representação dessas informações. Segundo Strang e Linnhoff-Popien (STRANG; LINNHOFF-POPIEN, 2004), são identificados na literatura basicamente seis tipos de modelos utilizados para representar informações de contexto:

- *Valor-chave*: é o modelo mais simples para estruturação dos dados. Neste modelo cada informação é associada a uma chave (e.g., Temperatura) e um valor (e.g., 24 graus);
- *Esquema de marcação*: utiliza uma estrutura de dados hierárquica composta por *tags* de marcação para atributos e conteúdo.
- *Gráfico*: utiliza uma modelagem de contexto gráfica que pode ser implementada utilizando linguagens como a *Unified Modeling Language* (UML);
- *Orientação a objeto*: nessa abordagem vários objetos são utilizados para representar o contexto. Ao utilizar a orientação a objeto, os detalhes de processamento são encapsulados e o acesso ao contexto é feito via interfaces bem definidas;
- *Baseado em lógica*: este tipo de modelo possui um alto grau de formalismo. Fatos, expressões e regras são utilizados para modelar o contexto. O processo de inferência permite a derivação de novos fatos com base em regras existentes;
- *Baseado em ontologia*: nessa abordagem, os conceitos, os subconceitos e os fatos relevantes são descritos de maneira uniforme, geralmente seguindo a sintaxe de uma linguagem para ontologias.

Dentre os modelos citados acima, as ontologias destacam-se por oferecerem características como *simplicidade*, *flexibilidade*, *generalidade* e *expressividade semântica* (KORPIPää; MäNTYJäRVI, 2003). A simplicidade é um requisito que define que as expressões e relacionamentos devem ser tão simples quanto possível, de modo a facilitar o trabalho dos desenvolvedores. A flexibilidade refere-se à capacidade da ontologia suportar a inclusão de novos elementos do contexto, assim como seus relacionamentos. A terceira característica, generalidade, indica que o uso de ontologias não é limitado a um certo domínio, podendo suportar diferentes tipos de contexto. Por fim, a expressividade diz respeito a capacidade de descrever contextos com níveis de detalhes arbitrários (BALDAUF; DUSTDAR; ROSENBERG, 2007). Devido a essas características, vários trabalhos encontrados na literatura

utilizam ontologias para representar informações de contexto, como em (KIM; CHO; OH, 2005) (ZHOU et al., 2009) (GU; PUNG; ZHANG, 2004).

A expansão da IoT, da computação em nuvem, da *big data* e das tecnologias móveis alimentou o desenvolvimento de novas aplicações ubíquas e cientes de contexto. Entretanto, a correta definição do contexto de um usuário ou aplicação depende da confiabilidade das informações capturadas por sensores ou providas pelo usuário. Por exemplo, um sistema de localização que auxilia na refrigeração de um ambiente deve fornecer a localização precisa de um usuário que esteja próximo a parede e não na sala vizinha, de modo que o sistema refrigerador acionado seja o do ambiente em que o usuário se encontra. Como não se pode garantir que todas as informações de contexto tenham sido obtidas corretamente, se faz necessário o gerenciamento eficiente da *qualidade da informação de contexto* (do inglês *Quality of Context* - QoC) utilizada pelas aplicações. QoC é relacionado com qualquer informação que descreve a qualidade de dados de contexto, como especificado pela definição proposta por Buchholz *et al.* (BUCHHOLZ; SCHIFFERS, 2003).

De acordo os objetivos de negócio das aplicações, alguns critérios de QoC podem ser mais importantes do que outros. Algumas vezes o critério de atualidade (*freshness*) é suficiente, algumas vezes é o critério de precisão, e outras vezes ambos são necessários. Já que uma qualidade de contexto ruim pode levar aplicações a decisões erradas e reações irrelevantes, é necessário a proposta de soluções de *middleware* que levem em conta QoC em cada passo do ciclo de vida da informação de contexto. Como forma de prover plataformas de *middleware* com uma forma genérica, computável e expressiva para a manipulação e exploração de QoC de forma simples e eficiente, foi proposto o meta-modelo QoCIM, que é descrito a seguir na Seção 2.2.1.

2.2.1 QoCIM

O QoCIM (*Quality of Context Information Meta-model*) (MARIE et al., 2014)(MARIE et al., 2015) é um meta-modelo para o desenvolvimento e representação de QoC. Esse meta-modelo não é dependente de nenhum critério de QoC, oferecendo uma solução unificada para modelar, em tempo de *design*, critérios de QoC heterogêneos. Antes de apresentar o meta-modelo do QoCIM em si, é importante destacar a diferença entre um *critério* e um *indicador de QoC*. Um critério de QoC descreve um atributo usado para qualificar informações de contexto (e.g., atualidade, precisão, e acurácia). Por outro lado, um indicador de QoC associa um critério a um valor, sendo usado diretamente para qualificar uma informação de contexto (e.g., a informação de contexto X possui precisão de 100%).

A Figura 3 apresenta o meta-modelo do QoCIM. O QoCIM representa informação de contexto através da classe *ContextInformation* enquanto um indicador de QoC é representado pela classe *QoCIndicator*, que contém a qualidade da informação de contexto e é descrita através de um critério, definido através da classe *QoCCriterion*. Indicadores e critérios são identificados unicamente com o atributo *id*. Por sua vez, a valoração de um indicador de QoC é realizada através da classe *QoCMetricValue*, também identificada por um atributo *id*, e com um valor literal inteiro especificado através do atributo *value*. Ademais, a data de criação de um *QoCMetricValue* também é definida, através do atributo *creationDate*. Por fim, para guiar a definição de instâncias da classe *QoCMetricValue*, o QoCIM propõe o conceito de *QoCMetricDefinition*, que define uma série de atributos necessários para a especificação de critérios de QoC. Os atributos pertencentes a classe *QoCMetricDefinition* são expostos a seguir:

- *isInvariant*: indica quando o valor produzido é constante, não editável, e calculado de forma não dinâmica.
- *unit*: apresenta a unidade do valor produzido. Ela pode ser, por exemplo, uma das unidades do *International System*.
- *direction*: compara diferentes *QoCMetricValues* do ponto de vista do consumidor da informação de contexto e com base nos seus respectivos atributos *value*: Os valores possíveis desse atributo são: *INF*, *SUP* e *UNDEF*. *INF* indica que quanto maior o *value*, pior o nível de QoC. *SUP* indica que quanto maior o *value*, melhor o nível de QoC. Por fim, *UNDEF* é usado quando os valores *INF* e *SUP* não podem ser expressos.
- *providerUri*: identifica o recurso que provêm o *QoCMetricValue*: esse atributo traz uma forma de filtrar o QoC baseado na entidade que o computa em tempo de execução.
- *minValue* e *maxValue*: respectivamente definem o valor mínimo e máximo permitidos para o atributo *value*.

O QoCIM apresenta ainda mais uma classe, a classe *Description*, necessária para trazer semântica para a classe *QoCMetricDefinition*. Essa classe possui três atributos: (i) *name*, (ii) *keywords*, e (iii) *informalDefinition*. O atributo *name* contêm um nome qualquer que identifica o critério de QoC sendo descrito. O atributo *keywords* é uma lista de palavras-chave relacionadas ao critério. Finalmente, o atributo *informalDefinition* é um texto que informalmente descreve o critério.

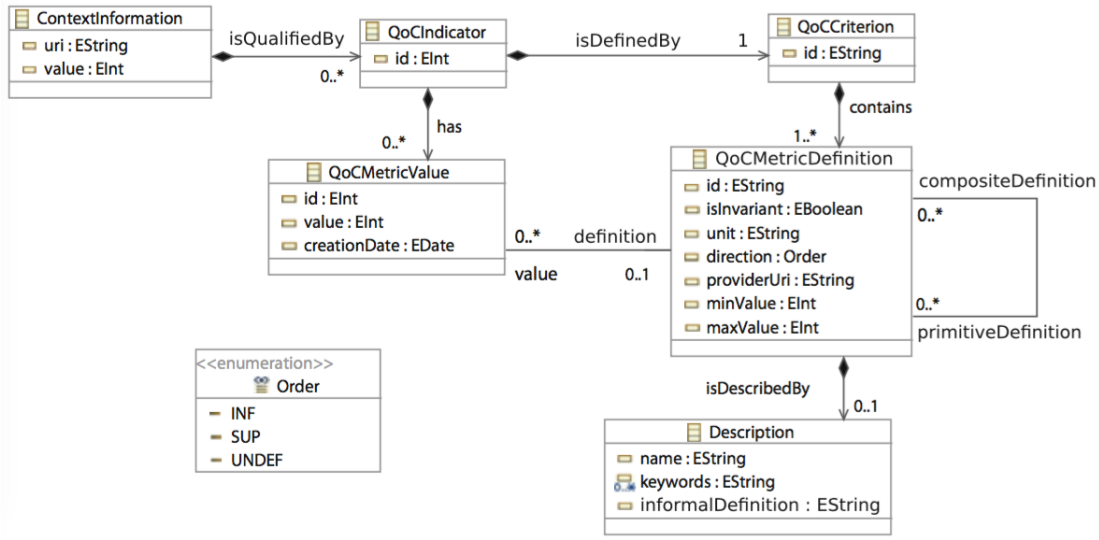


Figura 3: Meta-modelo do QoCIM.

Para ilustrar a modelagem de critérios e indicadores de QoC através do meta-modelo do QoCIM, a Figura 4 apresenta a modelagem de dois indicadores (*FreshnessIndicator* e *PrecisionIndicator*) e seus respectivos critérios de QoC (*FreshnessCriterion* e *PrecisionCriterion*).

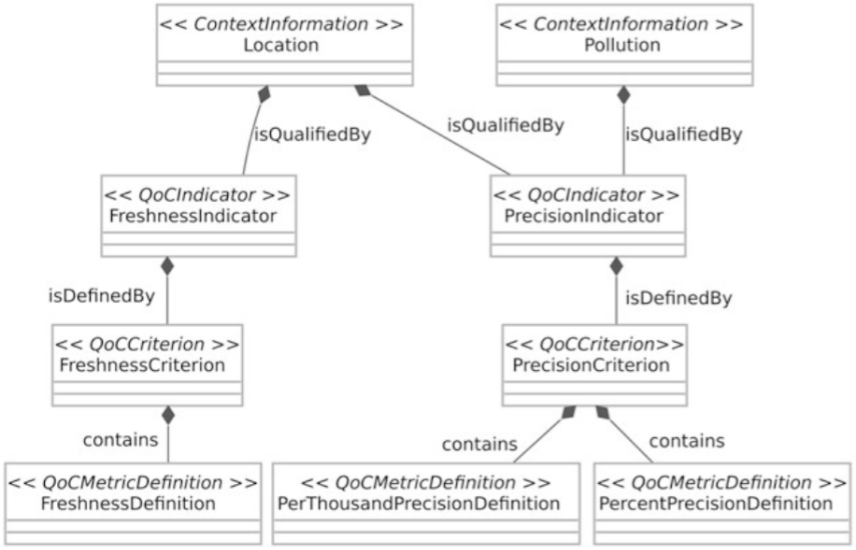


Figura 4: Exemplo de uso do QoCIM.

Nesse exemplo, a informação de localização é qualificada com dois indicadores de QoC, a atualidade (i.e., *freshness*) e a precisão (i.e., *precision*), enquanto a informação de poluição é qualificada através de um indicador de precisão. A atualidade é o tempo gasto desde a determinação da informação de contexto até a sua entrega ao cliente. A *precisão* é definida como o quão perto ou o qual repetível os resultados de medição de uma informação de contexto são. Qualificar uma informação de contexto por meio de

diferentes indicadores de QoC permite a análise da informação de diferentes pontos de vista. Assim, é possível conseguir uma descrição mais completa da qualidade real da informação de contexto, provendo aplicações ciente de QoC com tudo que elas precisam para entregar serviços relevantes aos usuários finais.

Vale ressaltar que a Figura 4 também mostra que o indicador de precisão pode qualificar duas informações de contexto ao mesmo tempo, a medição de poluição e a localização. Com uma coleção de critérios de QoC, desenvolvedores precisam somente escolher quais critérios eles precisam para suas aplicações. De fato, oferecer uma forma de reusar critérios de QoC por desenvolvedores de aplicações cientes de QoC é uma das principais características do QoCIM. Por fim, outra característica importante do QoCIM também é ilustrada nessa figura, a possibilidade de múltiplas definições para um mesmo critério de QoC. Essa pluralidade é ilustrada pelas duas definições associadas com o mesmo critério de precisão. Uma definição expressa a precisão em porcentagem enquanto a outra expressa a precisão por partes de mil. Prover múltiplas definições para um mesmo critério permite que diferentes sensores com diferentes capacidades escolham qual definição é a mais apropriada para qualificar suas medições.

2.3 Ontologias

Uma ontologia (GRUBER, 1993) é um modelo de dados que representa um conjunto de conceitos e os seus relacionamentos dentro de um domínio, fornecendo expressividade formal e prevenindo ambiguidade na interpretação dessas informações. Do ponto de vista da representação do conhecimento, as ontologias levam em consideração a necessidade de codificar o conhecimento para torná-lo acessível a quem precisa, categorizando-o, descrevendo-o, modelando-o, e inserindo regras e padrões definidos. Também cabe ressaltar que o propósito de uma ontologia não é o de conceituar tudo o que existe, mas sim informações consideradas relevantes no contexto de um problema específico. Desse modo, uma ontologia pode ser vista como uma forma de representar o conhecimento referente a um domínio. Atualmente, ontologias são empregadas na representação formal do conhecimento em diversas áreas da computação, como as áreas de Inteligência Artificial e Web Semântica. Além disso, o uso de ontologias também permite o compartilhamento e o reúso do conhecimento em um ambiente aberto e distribuído (PETERS; SHROBE, 2003).

Para representar conhecimento através de uma ontologia, é necessário o uso de linguagens capazes de expressar satisfatoriamente o conhecimento desejado e de inferir sobre

o mesmo. As relações e restrições expressas em uma ontologia podem permitir inferir conhecimento que é explicitamente definido e podem ajudar a determinar inconsistências lógicas nas informações descritas na ontologia. As duas linguagens mais relevantes para a representação formal de ontologias são a RDF[6] e a OWL[7]. A seguir essas duas linguagens são apresentadas.

2.3.1 RDF

A *Resource Description Framework* (RDF) é uma linguagem de representação formal de metadados sobre recursos na Web. Até certo ponto, a RDF pode ser considerada um padrão simplificado para a definição de ontologias que apoia a interoperabilidade entre aplicações que trocam informações interpretáveis por máquinas (BREITMAN; CASANOVA; TRUSZKOWSKI, 2006). Um documento RDF, comumente chamado de *triplestore*, é sintaticamente estruturado através de declarações (*statements*). Cada declaração, chamada de tripla na terminologia RDF, é, por sua vez, composta por três elementos: *sujeito* (*subject*), *predicado* (*predicate*) e *objeto* (*object*). O sujeito é um recurso identificado por uma URI (*Uniform Resource Identifier*); este é o objeto que se pretende descrever semanticamente. Predicado é uma propriedade do sujeito também identificada por uma URI. Por fim, o objeto é o valor atribuído à propriedade do sujeito. Diferentemente dos outros dois elementos, o objeto pode ser tanto um recurso identificado por uma URI como um valor literal simples (e.g., datas e nomes). Ou seja, sujeitos, predicados e objetos, são identificados normalmente por uma URI. Basicamente, uma URI é uma sequência se caracteres utilizada para identificar um recurso na Internet, tal como: <`http://www.w3.org`> ou <`http://purl.org/dc/elements/1.1/title`>.

Descrever dados RDF utilizando URIs pode ser algo pouco prático e ilegível, devido aos tamanhos das URIs. Por conveniência, podemos usar abreviações (QName - *XML qualified names*) para as URIs quando criamos arquivos RDF. Uma abreviação é formada por um prefixo e o nome local (*localname*) da URI. Por exemplo, *rdfs:class* pode ser uma abreviação para a URI: `http://www.w3.org/2000/01/rdf-schema#class`. O prefixo é sempre associado a um *namespace*, sendo usado para facilitar o gerenciamento e o referenciamento a classes e propriedades de URIs. No nosso exemplo, *rdfs* estaria associado ao espaço de nome: `http://www.w3.org/2000/01/rdf-schema#`. O nome local da URI, no nosso exemplo, seria a *string* "class".

[6] *Resource Description Framework* (RDF): `https://www.w3.org/RDF/`
[7] *Web Ontology Language* (OWL): `http://www.w3.org/2001/sw/wiki/OWL`

A Figura 5 apresenta um exemplo conceitual de modelagem usando a linguagem RDF. Nesse exemplo, o sujeito é definido como "João", o predicado como "tem irmã", e o objeto como "Maria".

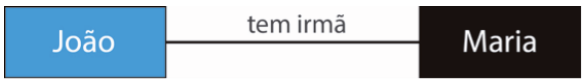


Figura 5: Exemplo de modelagem RDF.

A linguagem RDF oferece enorme flexibilidade, mas não provê meios para a definição de classes, propriedades e hierarquias específicas de um domínio específico (BREITMAN; CASANOVA; TRUSZKOWSKI, 2006). Por isso, um padrão conhecido como RDFS surgiu para permitir a definição personalizada de modelos de classes e propriedades. Uma das principais características do padrão RDFS é a capacidade de estabelecer hierarquias de classes, assim como no paradigma da orientação a objetos existente no contexto da Engenharia de *Software*. Essa característica é realizada através da propriedade *rdfs:subClassOf*. Adicionalmente, o padrão RDFS ainda permite o estabelecimento de hierarquias de propriedades através da aplicação do elemento *rdfs:subPropertyOf*.

As classes e propriedades definidas em um documento RDFS são instanciadas em documentos RDF. Desta forma, dois efeitos muito comuns na Web Semântica são observados. Primeiramente, nota-se o reuso de vocabulário, pois o mesmo modelo de classes é consumido por qualquer documento RDF interessado. Em segundo lugar, observa-se a geração de declarações sensíveis ao domínio, o que é fundamental na definição de ontologias.

2.3.2 OWL

O padrão mais recente de linguagens para a descrição de ontologias é a OWL (*Web Ontology Language*), publicado pela W3C. Derivada da RDF, a OWL visa tornar a descrição de ontologias compreensíveis tanto por humanos quanto por computadores, podendo ser considerada uma evolução da RDF. Dentre os diversos elementos que as ontologias OWL possuem, destacam-se:

- *Indivíduos*: representam objetos em um domínio de interesse. A OWL não usa a UNA (*Unique Name Assumption*), isto significa que dois nomes diferentes podem remeter ao mesmo indivíduo. Por exemplo, *Queen Elizabeth* (Rainha Elizabeth) e *Elizabeth Windsor* podem ser referências ao mesmo indivíduo. Em OWL, deve-se declarar explicitamente que os indivíduos são os mesmos, ou diferentes uns dos outros. A Figura 6 apresenta exemplos de indivíduos.

- *Propriedades*: são relações binárias entre indivíduos, podendo ser uma relação com um atributo de dados, denominada propriedade primitiva (*Datatype Property*), ou outro indivíduo, denominada propriedade objeto (*Object Property*). Por exemplo, a propriedade *temPrimo* pode ligar o indivíduo José ao indivíduo João; ou a propriedade *viveEm* pode ligar o indivíduo José ao indivíduo Brasil. A Figura 7 mostra exemplos de propriedades que conectam indivíduos.

- *Classes*: são conjuntos que contêm indivíduos. Classes são descritas formalmente definindo os requisitos para a participação nas mesmas. Por exemplo, a classe *Pessoa* pode conter todos os indivíduos que são seres humanos em um domínio de interesse; e a classe *País* pode ser composta pelos indivíduos caracterizados como países. As classes também podem ser organizadas em hierarquias *superclasse-subclasse*, conhecidas como taxonomias. Subclasses são especializações de suas superclasses, por exemplo, considerando-se as classes *Pessoa* e *Criança*, *Criança* pode ser subclasse de *Pessoa*, e assim *Pessoa* é superclasse de *Criança*. Isso quer dizer que todos os indivíduos da classe *Criança* são *Pessoas*, ou seja, ser uma *Criança* implica ser uma *Pessoa*. A Figura 8 apresenta um exemplo de divisão de indivíduos em classes.

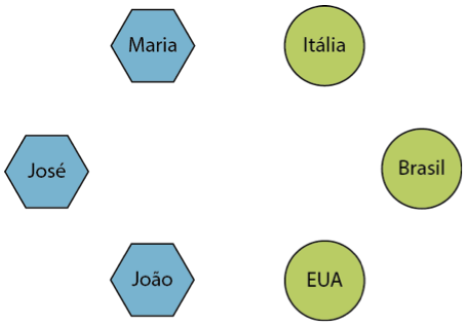


Figura 6: Indivíduos na OWL.

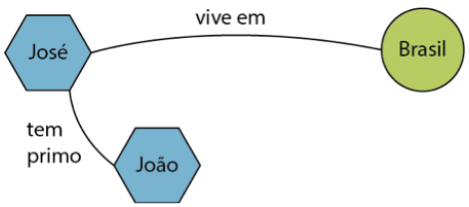


Figura 7: Propriedades na OWL.

2.3.3 Ontologias para IoT

Neste capítulo são apresentadas algumas ontologias encontradas na literatura para a modelagem de conhecimento em ambientes de IoT.

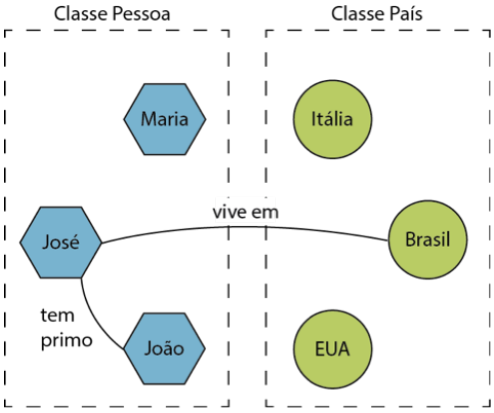


Figura 8: Classes na OWL.

2.3.3.1 SSN

A ontologia SSN (*Semantic Sensor Networks*) (COMPTON et al., 2012) foi criada pelo grupo W3C como um modelo para descrição de sensores, suas capacidades, plataformas e outros atributos relacionados à redes de sensores e à aplicações Web de sensores. A ontologia SSN define o padrão *Estímulo-Sensor-Observação* (do inglês *Stimulus-Sensor-Observation* - SSO) como ponto de partida para a descrição de sensores. De acordo com esse padrão, um estímulo (*ssn:Stimulus*) é uma mudança em um estado ou em um ambiente que um sensor pode detectar e usar para medir uma propriedade (*ssn:Property*) de uma característica de interesse (*ssn:FeatureOfInterest*). Na SSN, propriedades são aspectos observáveis de coisas do mundo real, enquanto características de interesse são as próprias coisas: por exemplo, a temperatura ou profundidade (propriedades) de um lago (característica de interesse).

Um sensor é qualquer coisa que pode realizar atividades de sensoriamento e, usualmente, consiste de um componente de *hardware* que captura e transforma estímulos de entrada em outras representações. Por fim, uma observação (i.e., dado capturado por um sensor) é o resultado de uma operação de medição que conecta um estímulo ao sensor que a captura. A Figura 9 ilustra o padrão SSO, destacando o papel central dos conceitos de estímulo, sensores e observações, e como eles se relacionam com características, propriedades e outros conceitos chaves de sensoriamento. Vale ressaltar que a nomeação das classes e propriedades da ontologia SSN seguem um padrão onde definições de conceitos começando com letra maiúscula representam uma classe (e.g., *ssn:Sensor*) e definições de conceitos começando com letra minúscula representam propriedades (e.g., *san:observes*).

Na SSN, esse padrão é empregado sob o ponto de vista de quatro perspectivas, que guiam a forma com que os sensores e suas características são descritos. Essas quatros

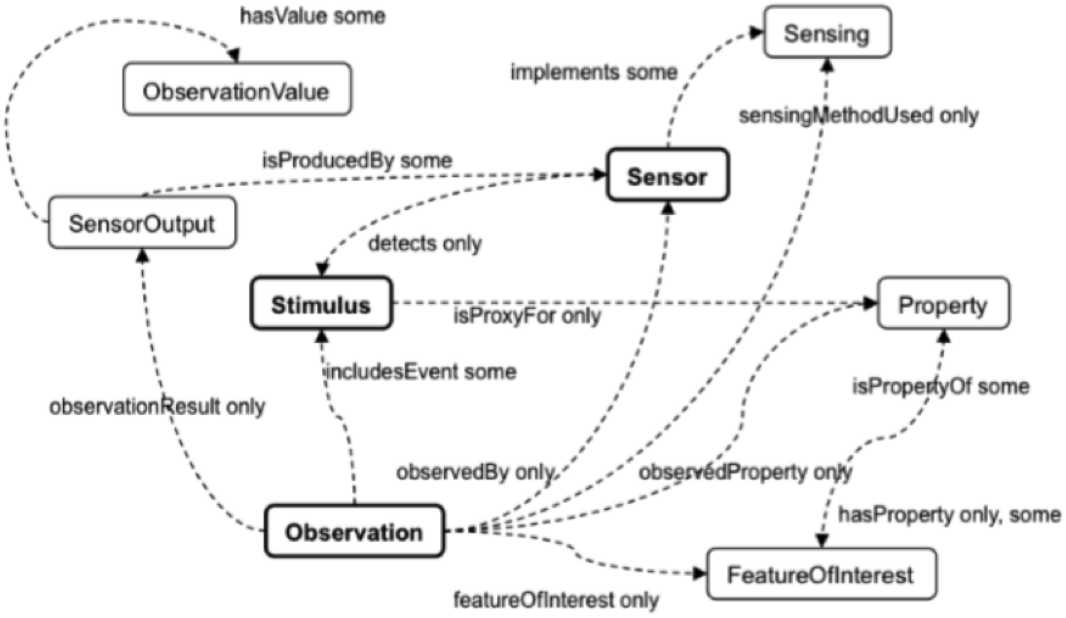


Figura 9: Padrão Estímulo-Sensor-Observação.

perspectivas são descritas em detalhes a seguir:

1. *Perspectiva de sensores*: A ontologia SSN tem uma visão liberal do que pode ser um sensor, considerando qualquer coisa que detecta uma propriedade do mundo real como um sensor. Consequentemente, dispositivos, sistemas inteiros, configurações de laboratórios, e até sistemas biológicos podem todos serem descritos como sensores. A ontologia pode ser usada para descrever capacidades de sensores (*ssn:MeasurementCapability*) que, por sua vez, especificam propriedades de medição (*ssn:MeasurementProperty*) de um sensor como, por exemplo:

    - ssn:*Accuracy* - A proximidade entre o valor de uma observação e o valor real da propriedade observada.
    - ssn:*Frequency* - O menor tempo possível entre observações.
    - ssn:*Latency* - O tempo entre uma requisição de dados de observação e a resposta do sensor.
    - *ssn:MeasurementRange* - O conjunto de valores que um sensor pode retornar como resultado de uma observação.
    - *ssn:Resolution* - A menor diferença no valor de uma propriedade (*ssn:Property*) observada por um sensor que irá resultar em diferenças de valores perceptíveis nas observações feitas por esse sensor.

- *ssn:Precision* - A habilidade de um sensor consistentemente reproduzir uma observação.

Essas propriedades são aspectos observáveis de um sensor, dado algumas condições do ambiente em que o sensor está inserido. Por exemplo, uma especificação pode mostrar que um sensor tem acurácia de 2% em uma condição, mas de 5% em uma outra. A Figura 10 resume os conceitos e relações existentes na *perspectiva de sensores*.

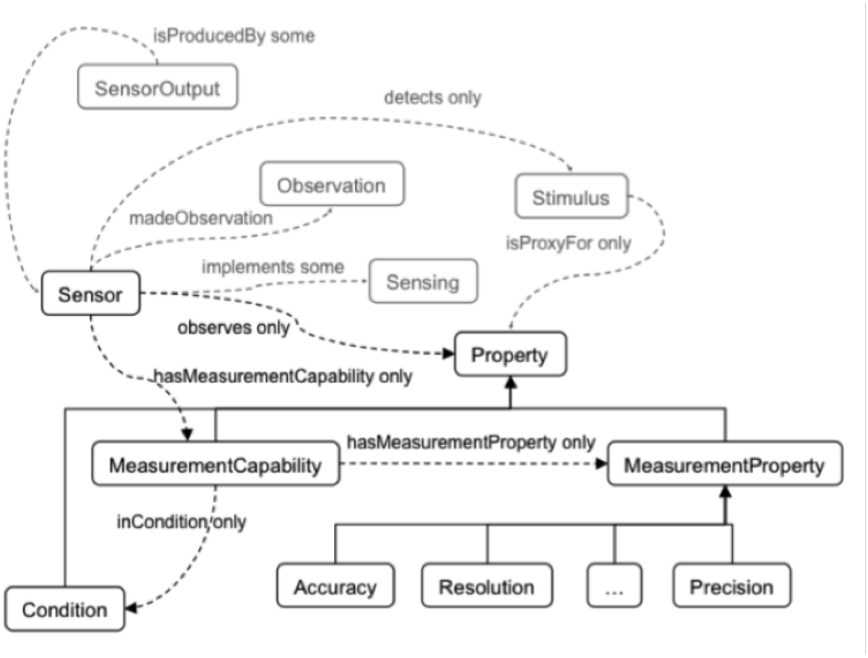


Figura 10: Perspectiva de sensores.

2. *Perspectiva de sistemas*: Sistemas (*ssn:System*) são unidades de organização que podem ser compostas por subsistemas (*ssn:SubSystem*) e implantadas (*ssn:Deployment*) em plataformas (*ssn:Platform*). Um sistema tem intervalos de operação (*ssn:OperatingRange*) e sobrevivência (*ssn:SurvivalRange*) que descrevem, respectivamente, as condições esperadas de operação do sistema e as condições nas quais eles são considerados danificados. Como acontece com a *ssn:MeasurementCapability* na descrição de sensores, a *ssn:OperatingRange* e a *ssn:SurvivalRange* são propriedades observáveis de um sistema. Além disso, cada dispositivo de sensoriamento (*ssn:SensingDevice* que é subclasse de *ssn:Device*) também é considerado como uma subclasse de um sistema. Isso permite a descrição de plataformas complexas, nas quais diversos sensores são agrupados como um único dispositivo. A Figura 11 ilustra os principais componentes da perspectiva de sistemas.

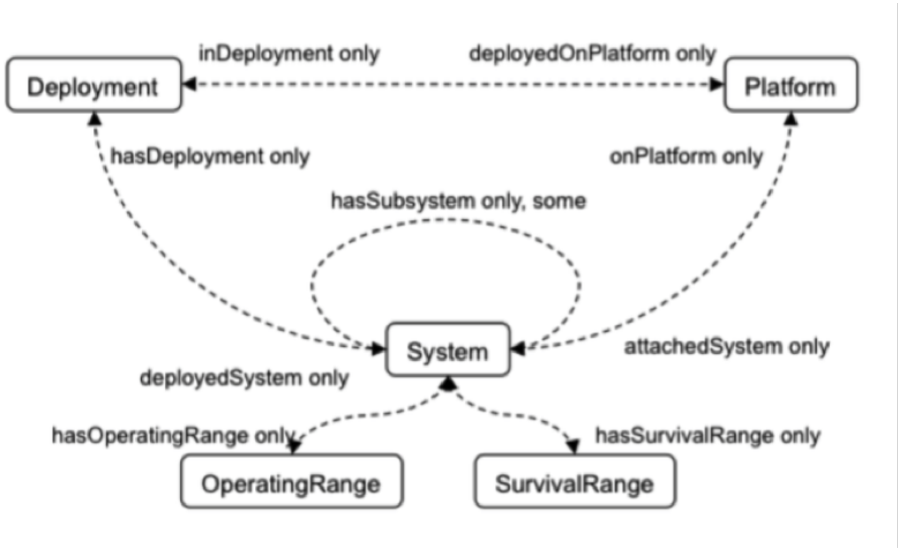


Figura 11: Perspectiva de sistemas.

3. *Perspectiva de características e propriedades*: foca nos dados que são capturados pelos sensores ou nas observações que podem ser feitas acerca desses dados. Essa perspectiva descreve as propriedades (*ssn:Property*) observadas pelos sensores que, por sua vez, são agrupadas por características de interesse (*ssn:FeatureOfInterest*). Ver um mundo de uma perspectiva de características e propriedades permite, por exemplo, buscar em uma base de conhecimento qual dispositivo observa uma propriedade qualquer p, quais dispositivos tem observações afetadas por p, e quais observações tem sido feitas sobre p.

4. *Perspectiva de dados ou observação*: na ontologia SSN, observações (*ssn:Observation*) são situações que descrevem o estímulo e o resultado de um sensoriamento. Observações ligam o ato de sensoriamento, o evento de estímulo, o sensor, a propriedade e a característica de interesse sensoriada, e o resultado, colocando esses em um contexto interpretativo. Vale ressaltar que a observação é a explicação do ato e o seu resultado - não o evento em si. O resultado da observação corresponde a saída de um sensor (*ssn:SensorOutput*).

O Trecho de Código 2.1 apresenta um exemplo de especificação usando a SSN através da perspectiva de sensores. Nesse exemplo, um sensor do tipo *Vaisala_ WM30* é descrito. Esse sensor disponibiliza dois valores, um do tipo *WindSpeed* (definido através da ontologia *WM30_ WindSpeed*, linhas 5 a 10) e outro do tipo *WindDirection* (definido através da ontologia *WM30_ WindDirection*, linhas 11 a 16) que são monitorados pelo próprio sensor (respectivamente, linhas 20 a 25 e linhas 26 a 31).

Listing 2.1: Exemplo de especificação usando a SSN

```
<owl:Class rdf:about="&wm30;Vaisala_WM30">

   <!-- Propriedades disponibilizadas pelo recurso -->
   <rdfs:subClassOf rdf:resource="&ssn;SensingDevice"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="&ssn;hasSubSystem"/>
         <owl:someValuesFrom rdf:resource="&wm30;WM30_WindSpeed"/>
      </owl:Restriction>
   <rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="&ssn;hasSubSystem"/>
         <owl:someValuesFrom rdf:resource="&wm30;WM30_WindDirection"/>
      </owl:Restriction>
   <rdfs:subClassOf>

   <!-- Propriedades monitoradas pelo recurso -->
   <rdfs:subClassOf rdf:resource="&ssn;SensingDevice"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="&ssn;observes"/>
         <owl:someValuesFrom rdf:resource="&wm30;WM30_WindSpeed"/>
      </owl:Restriction>
   <rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="&ssn;observes"/>
         <owl:someValuesFrom rdf:resource="&wm30;WM30_WindDirection"/>
      </owl:Restriction>
   <rdfs:subClassOf>
</owl:Class>
```

2.3.3.2 SAN

A SAN (*Semantic Actuator Network*) (SPALAZZI; TACCARI; BERNARDINI, 2014) é uma ontologia para descrição semântica de atuadores e redes de atuadores baseada no trabalho da *W3C Semantic Sensor Network Incubator Group*, que propôs a SSN. Assim, através dessas duas ontologias (SAN e SSN), é possível modelar qualquer dispositivo complexo composto por sensores e atuadores.

A SAN propõe o padrão *Atuador-Estímulo-Operação* para a descrição de atuadores. De acordo com esse padrão, um *atuador* é um objeto capaz de modificar uma propriedade no ambiente em que ele opera através de um *estimulo*. Finalmente, uma *operação* representa a atividade que é realizada pelo atuador. A ontologia SAN define duas novas perspectivas para a descrição de atuadores, a *perspectiva do atuador* e a *perspectiva de*

operação. Essas duas perspectivas são detalhadas a seguir e a relação entre as ontologias SAN e SSN, incluindo as principais classes e propriedades definidas pela SAN, são apresentadas na Figura 12. Vale ressaltar que, semelhante ao que acontece com a ontologia SSN, a nomeação das classes e propriedades da ontologia SAN seguem um padrão onde definições de conceitos começando com letra maiúscula representam uma classe (e.g., *san:Actuator*) e definições de conceitos começando com letra minúscula representam propriedades (e.g., *san:modify*).

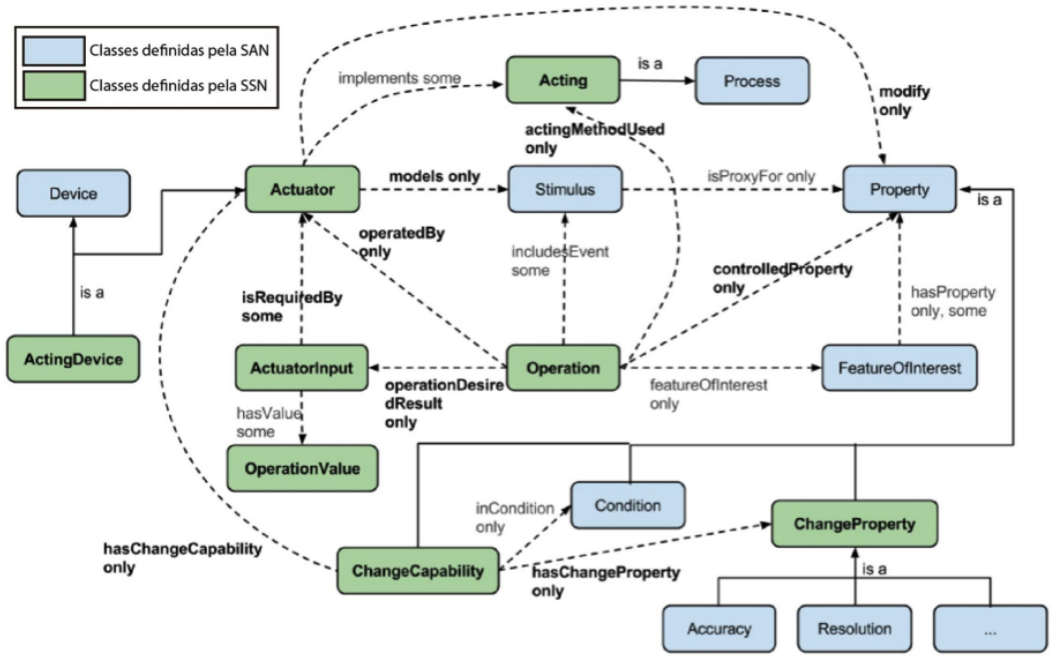


Figura 12: Principais classes e propriedades da SAN e a sua relação com a SSN.

- *Perspectiva do atuador*: descreve as propriedades que um atuador pode gerenciar e/ou modificar. Essa perspectiva também permite a especificação do funcionamento de um atuador. Segundo essa perspectiva, um atuador (*san:Actuator*) é um objeto físico que, através da geração de um estímulo (*ssn:Stimulus*), é capaz de modificar uma propriedade (*ssn:Property*). Nessa ontologia, um estímulo é a mudança realizada por um atuador (*san:models*) com o intuito de modificar (*san:modify*) uma propriedade qualquer do ambiente. Vale ressaltar que um estímulo não se refere necessariamente a propriedade que um atuador modifica. Por exemplo, considerando-se um motor elétrico como um atuador, a propriedade que é gerenciada por ele é a velocidade rotacional do motor. A fim de modificá-la, uma variação no estado atual do rotor do motor é necessária; esse é o estímulo no qual o atuador age.

  Além disso, essa perspectiva define que um atuador implementa (*san:implements*) um processo de atuação (*san:Acting* é uma subclasse de *ssn:Process*), que descreve

o processo implementado por um atuador para gerenciar uma propriedade. Para descrever as especificações de funcionamento de um atuador, é definido o conceito de mudança de propriedade (*san:ChangeProperty*), que é qualificada de acordo com uma série de conceitos (tais como *ssn:Accuracy*, *ssn:Resolution*, etc.) definidos na ontologia SSN e importadas pela ontologia SAN. Além disso, o conceito *san:ChangeCapability* é definido como uma forma de conectar (*ssn:inCondition*) um conjunto de especificações de funcionamento de um atuador com uma condição específica (*ssn:Condition*).

- *Perspectiva de Operação*: permite a descrição das operações (*san:Operation*) realizadas por um atuador em um dado instante (*san:operationResultTime*). Operações são realizadas (*san:operatedBy*) por atuadores com o intuito de modificar (*san:controlledProperty*) uma propriedade presente em uma característica de interesse (*san:featureOfInterest*). Assim, essa perspectiva é capaz de definir o estímulo que um atuador recebe e o método usado por esse atuador (*san:actingMethodUsed*). Obviamente, a operação realizada por um atuador está relacionada com o resultado esperado do processo de atuação. Tal resultado é a entrada do atuador (*san:ActuatorInput*). Nesse caso, a entrada do atuador é a modificação esperada pelo uso do atuador, ou seja, além da entrada do atuador, a classe *san:ActuatorInput* define a saída do atuador. Voltando para o exemplo do motor elétrico, o novo valor da velocidade rotacional do motor é dado pela própria entrada do atuador.

A *perspectiva de sistema* e a *perspectiva de características e propriedades* definidas pela SSN também podem ser especificadas através da ontologia SAN. Porém, tais perspectivas já foram definidas pela própria SSN. Por isso, quanto a *perspectiva de sistemas*, a SAN simplesmente introduz um conceito para modelar um dispositivo de atuação (*san:ActingDevice* que é subclasse de *san:Actuator* e de *ssn:Device*) e define suas relações com a classe *ssn:Device*. Quanto a *perspectiva de características e propriedades*, nenhum novo conceito foi adicionada pela SAN, já que as características de interesse e propriedades são as mesmas descritas pela SSN. A única diferença é que características e propriedades são alteradas por atuadores ao invés de serem capturadas por sensores.

O grupo que definiu a SAN ilustra o seu uso em um cenário envolvendo o gerenciamento de emergências de terremoto. Esse cenário envolve uma série de sensores para identificação e avaliação de terremotos, como fissurômetros e acelerômetros. Além disso, alguns atuadores foram identificados nesse cenário, como por exemplo:

- *Alarme sonoro*: instalados em edifícios com o objetivo de alertar as pessoas sobre a necessidade de evacuar o edifício;
- *Luz de sinalização da saída*: também instaladas em edifícios, sinaliza o caminho para a saída do edifício durante uma evacuação de emergência;
- *Notificador de mensagens*: envia mensagens para os dispositivos móveis dos cidadãos das áreas afetadas por um terremoto informando sobre locais seguros.

O Trecho de Código 2.2 apresenta a descrição OWL de um alarme sonoro através da perspectiva do atuador proposta pela SAN.

Listing 2.2: Descrição de um atuador através da ontologia SAN.

```
1 <Class rdf:about="&eem;AlarmSiren">
2    <rdfs:label>Alarme Sonoro</rdfs:label>
3    <rdfs:subClassOf rdf:resource="&san;ActingDevice"/>
4    <rdfs:subClassOf>
5       <Restriction>
6          <onProperty rdf:resource="&san;modify"/>
7          <allValuesFrom rdf:resource="&eem;AlarmSoundProperty"/>
8       </Restriction>
9    </rdfs:subClassOf>
10 </Class>
```

Esse Trecho de Código apresenta a definição de um alarme sonoro como sendo um dispositivo de atuação (linha 3). Para especificar a atuação realizada por esse alarme no ambiente, é especificado que um alarme sonoro modifica somente propriedades do ambiente do tipo *AlarmSoundProperty* (linha 7). A definição dessa propriedade é apresentada no Trecho de Código 2.3 .

Listing 2.3: Descrição da classe *Alarm Sound Property*.

```
1 <Class rdf:about="&eem;AlarmSoundProperty">
2    <rdfs:label>Propriedade Sonora do Alarme</rdfs:label>
3    <rdfs:subClassOf rdf:resource="&qu;DimensionLess"/>
4    <rdfs:subClassOf rdf:resource="&ssn;Property"/>
5 </Class>
```

Nesse segundo Trecho de Código é especificado que a classe *AlarmSoundProperty* lida com uma propriedade adimensional (linha 3), no caso, o som (linha 4).

Também é possível definir esse mesmo atuador através de outras perspectivas. Por exemplo, o Trecho de Código 2.4 apresenta a descrição do mesmo atuador do ponto de vista da *perspectiva de operação*.

Listing 2.4: Descrição do atuador através da *perspectiva de operação*.

```
1  <Class rdf:about="&eem;AlarmSoundOperation">
2     <rdfs:label>OperaÃ§Ã£o Sonora do Alarme</rdfs:label>
3     <rdfs:subClassOf rdf:resource="&san;Operation"/>
4     <rdfs:subClassOf>
5        <Restriction>
6           <onProperty rdf:resource="&ssn;operatedBy"/>
7           <someValuesFrom rdf:resource="&eem;AlarmSiren"/>
8        </Restriction>
9     </rdfs:subClassOf>
10       <rdfs:subClassOf>
11       <Restriction>
12          <onProperty rdf:resource="&ssn;featureOfInterest"/>
13          <someValuesFrom rdf:resource="&eem;AlarmFeature"/>
14       </Restriction>
15    </rdfs:subClassOf>
16       <rdfs:subClassOf>
17       <Restriction>
18          <onProperty rdf:resource="&ssn;controlledProperty"/>
19          <someValuesFrom rdf:resource="&eem;AlarmSoundProperty"/>
20       </Restriction>
21    </rdfs:subClassOf>
22       <rdfs:subClassOf>
23       <Restriction>
24          <onProperty rdf:resource="&ssn;operationDesiredResult"/>
25          <someValuesFrom rdf:resource="&eem;AlarmSirenActuatorInput"/>
26       </Restriction>
27    </rdfs:subClassOf>
28 </Class>
```

Nessa perspectiva, a operação realizada pelo atuador é descrita. Nota-se que essa operação é realizada por um atuador, o alarme sonoro (linha 7), e envolve a característica de interesse *AlarmFeature* (linha 13). A descrição dessa característica de interesse é apresentada no Trecho de Código 2.5.

Listing 2.5: Descrição da característica de interesse *AlarmFeature*.

```
1  <Class rdf:about="&eem;AlarmFeature">
2     <rdfs:label>CaracterÃstica de Interesse do Alarme</rdfs:label>
3     <rdfs:subClassOf rdf:resource="&ssn;FeatureOfInterest"/>
4     <rdfs:subClassOf>
5        <Restriction>
6           <onProperty rdf:resource="&ssn;hasProperty"/>
7           <someValuesFrom rdf:resource="&eem;AlarmSoundProperty"/>
8        </Restriction>
9     </rdfs:subClassOf>
10 </Class>
```

Por fim, através da perspectiva de operação também é possível especificar a propriedade do ambiente que o sensor atua, a *AlarmSoundProperty* (apresentada no Trecho

de Código 2.3) e a entrada do atuador (*AlarmSirenActuatorInput*), descrita a seguir no Trecho de Código 2.6.

Listing 2.6: Especificação da entrada do atuador.

```
1 <Class rdf:about="&eem;AlarmSirenActuatorInput">
2    <rdfs:label>Entrada do Atuador de Alarme Sonoro</rdfs:label>
3    <rdfs:subClassOf rdf:resource="&san;ActuatorInput"/>
4 </Class>
```

2.3.3.3 OWL-S

A OWL-S (*Ontology Web Language for Services*) (MARTIN et al., 2007) é uma linguagem de descrição para serviços Web de propósito geral. De acordo com essa linguagem, um serviço (*Service*) deve ser descrito por três sub-ontologias, *ServiceProfile*, *ServiceModel* e *ServiceGrouding*. A justificativa para tal estruturação baseia-se no fato que qualquer consumidor de informações descritas através dessa ontologia deve ter acesso a três tipos de conhecimento sobre o serviço, como listado na proposta ao W3C. A relação entre essas quatro ontologias são ilustrados na Figura 13 e as suas definições são apresentadas a seguir.

- *ServiceProfile*: descreve o que o serviço provê para os seus clientes. A OWL-S permite que o provedor de um serviço descreva o que seu serviço oferece criando um subclasse da classe *ServiceProfile*. A especificação OWL-S define uma classe *Profile*, que é uma possível representação de um serviço, mas não é obrigatória nem restritiva e os provedores de serviços podem adaptar a classe *Profile* ou criar outras classes completamente diferentes para representar um *Profile*. Considerando a utilização de repositórios ou mecanismos de busca semântica, a ontologia *Profile* permite aos agentes de *software* descobrir automaticamente o serviço em questão.

- *ServiceModel*: responsável por descrever todos os passos necessários para a correta execução de uma determinada funcionalidade do serviço. Essa especificação é feita expressando transformações de dados, com entradas e saídas, e o estado de transformação, com precondições e efeitos. Apesar do *ServiceProfile* e do *ServiceModel* terem diferentes papéis durante o ciclo de vida do serviço, eles representam o mesmo serviço, por isso é natural esperar que as entradas, saídas, precondições, e efeitos (também conhecido como IOPE) de um sejam refletidas nos IOPE do outro. Um serviço pode ser descrito por no máximo uma instância de *ServiceModel*. Por fim,

essa ontologia permite ainda a composição de serviços para a geração de um serviço complexo.

- *ServiceGrounding*: especifica os detalhes de como acessar o serviço. Essa ontologia discrimina, inclusive, os protocolos de transporte e comunicação, os formatos de mensagens, serialização e endereçamento suportados pelo serviço. O objetivo do *ServiceGrouding* é mapear noções abstratas de semântica em elementos concretos que são necessários para interagir com o serviço.

- *Service*: principal integrante do padrão OWL-S, atua simplesmente como ponto de ligação entre as três ontologias listadas acima. Mais especificamente, ela define as classes abstratas que são posteriormente especializadas pelas demais ontologias.

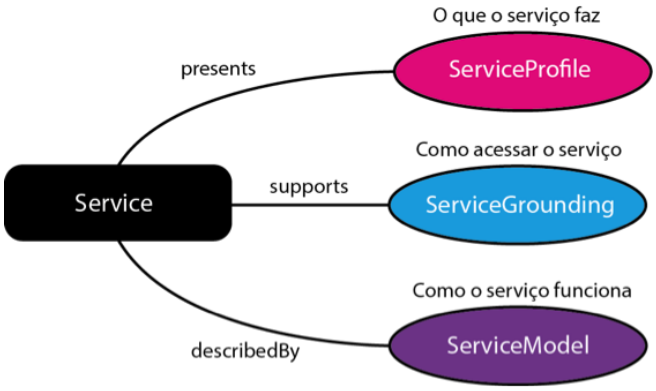


Figura 13: Ontologia OWL-S.

O Trecho de Código 2.7, adaptada de Júnior (JÚNIOR, 2008), apresenta parte da ontologia de um serviço de reserva de voos chamado de *ReservarVooService* (linhas 2 a 6). Esse serviço é apresentado por um *ReservarVooProfile* (linhas 9 a 16), descrito por um *ReservarVooProcess* (linha 19 a 27) e suportado por um *ReservarVooGrounding* (linha 30 a 43). O primeiro elemento descreve o serviço como uma composição de três ontologias, o segundo descreve as funcionalidades que o serviço oferece, o terceiro descreve o funcionamento do serviço como um processo de um único passo, que exige as entradas *Origem*, *Destino*, *Data*, *Hora* e *Voo*, e produz a saída *Localizador*. Por fim, o quarto elemento descreve a forma de acessar esse serviço, especificando a URI da operação (*reservarVoo*) que deverá ser invocada.

Listing 2.7: Trecho de descrição OWL-S do serviço *ReservarVoo*.

```
1 <!-- Service -->
2 <service:Service rdf:ID="ReservarVooService">
3    <service:presents rdf:resource="#ReservarVooProfile"/>
4    <service:describedBy rdf:resource="#ReservarVooProcess"/>
```

```
5     <service:supports rdf:resource="#ReservarVooGrounding"/>
6  </service:Service>
7
8  <!-- ServiceProfile -->
9  <service:Profile rdf:ID="ReservarVooProfile">
10    <service:presentedBy rdf:resource="#ReservarVooService"/>
11    <profile:serviceName>Reserva de Voos</profile:serviceName>
12    <profile:textDescription>
13       Este servico realiza a reserva de voo de uma origem para
14       um destino em uma data e hora especÃfica.
15    </profile:textDescription>
16 </service:Profile>
17
18 <!-- ServiceModel -->
19 <process:AtomicProcess rdf:ID="ReservarVooProcess">
20    <service:describes rdf:resource="#ReservarVooService"/>
21    <service:hasInput rdf:resource="#Origem"/>
22    <service:hasInput rdf:resource="#Destino"/>
23    <service:hasInput rdf:resource="#Data"/>
24    <service:hasInput rdf:resource="#Hora"/>
25    <service:hasInput rdf:resource="#Voo"/>
26    <service:hasInput rdf:resource="#Localizador"/>
27 </process:AtomicProcess>
28
29 <!-- ServiceGrounding -->
30 <grounding:WsdlGrounding rdf:ID="ReservarVooGrounding">
31    <service:supportedBy rdf:resource="#ReservarVooService"/>
32    <grounding:hasAtomicProcessGrounding rdf:resource="#ReservarVooWsdl"/>
33 </grounding:WsdlGrounding>
34 <grounding:WsdlAtomicProcessGrounding rdf:ID="ReservarVooWsdl">
35    <grounding:owlsProcess rdf:resource="#ReservarVooProcess"/>
36    <grounding:WsdlDocument>CompanhiaArea?wsdl</grounding:WsdlDocument>
37    <grounding:WsdlOperation>
38       <grounding:WsdlOperationRef>
39          <grounding:portType>CompanhiaAreaPortType</grounding:portType>
40          <grounding:operation>reservarVoo</grounding:operation>
41       </grounding:WsdlOperationRef>
42    </grounding:WsdlOperation>
43 </grounding:WsdlAtomicProcessGrounding>
```

2.4 Busca Semântica

A maneira mais comum que a busca semântica tem sido entendida e endereçada na literatura consiste na construção de um engenho de busca que: (i) recebe requisições no formato de uma *linguagem de consulta em ontologias*, (ii) executa a requisição em uma base de dados contendo a(s) ontologia(s), e (iii) retorna triplas de valores contidos na(s) ontologia(s) que satisfazem as condições especificadas na requisição (FERNÁNDEZ MENDEZ et al., 2011). Linguagens de consulta em ontologias são usadas para expressar

buscas complexas através de uma sintaxe simples. Na Web semântica, a linguagem de consulta em ontologias comumente adotada para recuperar informação em uma base de dados RDF tem sido a SPARQL, que é um padrão da W3C desde janeiro de 2008. Haja vista que a linguagem OWL é uma extensão da RDF, a linguagem de consulta SPARQL também pode ser aplicada a OWL. A Seção 2.4.1 apresenta uma descrição detalhada da linguagem SPARQL.

2.4.1 SPARQL

A linguagem SPARQL [8] é uma linguagem de consulta que permite a recuperação e a manipulação de dados armazenados no formato RDF, podendo ser inclusive utilizada em qualquer derivação de RDF como, por exemplo, a OWL. O padrão SPARQL é composto por uma linguagem de consulta, um protocolo de consulta e um documento XML para reproduzir os resultados. A grande vantagem do uso de SPARQL é a sintaxe da linguagem de consulta, bastante similar a SQL , com a exceção do uso de triplas RDF.

A SPARQL permite a consulta em *triplestores* RDF, de forma a obter resultados precisos a partir da passagem de parâmetros que podem ser associados livremente, possibilitando consultas por informações que atendam às necessidades de forma mais apropriada. Com isso, a consulta torna-se mais inteligente, e os resultados mais expressivos para quem busca. Consultas SPARQL consistem em três partes fundamentais, ilustradas na Figura 14 e apresentadas a seguir.

- *Cláusulas que indicam o tipo de retorno*, tais como: (i) a cláusula *SELECT* (utilizada para especificar que os resultados serão retornados para o usuário em forma de tabela), (ii) a cláusula *CONSTRUCT* (utilizada para retornar um novo grafo RDF construído através de *templates*, onde as suas variáveis são substituídas pelos seus valores), (iii) a cláusula *ASK* (verifica se há pelo menos uma resposta à consulta e retorna resultados booleanos como "true"ou "false", indicando se uma consulta está associada à alguma tripla), e (iv) a cláusula *DESCRIBE* (retorna um novo grafo RDF contendo recursos associados);

- *Cláusulas de DATASET*, representados pela cláusula *FROM*, que especifica qual(is) bases de conhecimento devem ser consultadas;

- *Cláusula WHERE*, utilizada para especificar o padrão de grafo a ser buscado na base de conhecimento especificada através das cláusulas de *DATASET*. Apesar da

[8] *SPARQL Query Language for RDF*: `https://www.w3.org/TR/rdf-sparql-query/`

cláusula *WHERE* possuir uma sintaxe inicialmente simples, este padrão pode conter alguns operadores extras, como o *FILTER* (utilizado para adicionar restrições aos padrões de grafos), o *DISTINCT* (utilizado para remover resultados duplicados), o *PROJECTION* (utilizado para reduzir a resposta ao subconjunto de variáveis definidos na cláusula *SELECT*), o *REDUCED* (utilizado para permitir que resultados duplicados sejam removidos), o *OFFSET* (indica que o resultado deve ser exibido a partir de um certo número determinado de soluções), o *OPTIONAL* (utilizado para tornar uma parte do padrão opcional), o *LIMIT* (utilizado para limitar o número de resultados retornados) e o *ORDER BY* (ordena a saída em ordem alfabética).

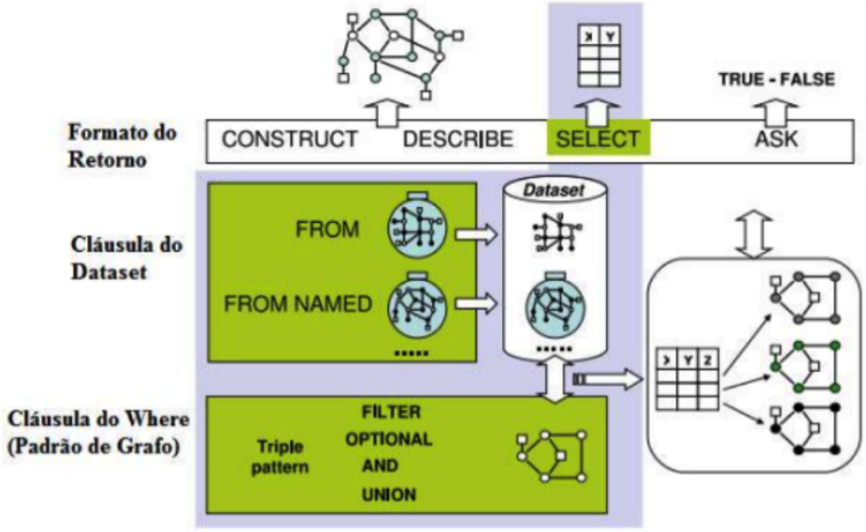


Figura 14: Formato de consulta SPARQL.
Fonte: Arenas *et al.*, página 4 (ARENAS; GUTIERREZ; PÉREZ, 2010).

O Trecho de Código 2.8 apresenta um exemplo de consulta SPARQL realizada sobre a ontologia SSN (descrita na Seção 2.3.3.1). Nesse exemplo, as linhas 1 e 2 apresentam os prefixos referentes as ontologias SSN e RDFS. Já a linha 4 indica que a *query* de busca SPARQL especificada irá retornar todas as informações contidas na variável *?superClass*. Essa variável, como define a linha 5, irá conter todos as classes das quais a classe *ssn:SensingDevice* herda (*rdfs:subClassOf*). O resultado dessa consulta será a descrição completa das classes *ssn:Device* e *ssn:Sensor*.

Listing 2.8: Exemplo de consulta SPARQL.

```
1 PREFIX ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3
4 SELECT ?superClass WHERE {
5      ssn:SensingDevice rdfs:subClassOf ?superClass .
6 }
```

Outra importante característica da SPARQL é a possibilidade de atualizar *triplestores* RDF, e inserir e remover triplas. Existem três tipos de operações: *inserção de dados*, *modificação de dados*, e *remoção de dados*.

Na primeira dessas três operações, a operação de *inserção de dados*, a cláusula *INSERT DATA* é usada para inserir triplas seguindo uma requisição. O Trecho de Código 2.9 ilustra a inserção de um indivíduo da classe *ssn:SensingDevice* que realiza o monitoramento de dados de aceleração. Nesse exemplo, a linha 1 define o prefixo para a ontologia SSN, enquanto a linha 4 define que o indivíduo que será inserido (`(http://consiste.dimap.ufrn.br/ontologies/qodisco#accelerometer)`) é um dispositivo de sensoriamento *ssn:SensingDevice*. Por fim, a linha 5 define que esse mesmo dispositivo monitora dados de aceleração (`http://purl.oclc.org/NET/muo/ucum/physical-quality/acceleration`). Como podemos notar nesse exemplo, a utilização de prefixos, como o empregado na linha 1, é opcional, haja vista a URI utilizada para identificação do indivíduo e a URI associada ao conceito de aceleração. Porém, a utilização dos mesmos ajuda a evitar erros na definição de uma *query* SPARQL e a torna mais simples e amigável.

Listing 2.9: Exemplo da operação *INSERT DATA* em SPARQL.

```
1 PREFIX ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
2
3 INSERT DATA{
4   <http://consiste.dimap.ufrn.br/ontologies/qodisco#accelerometer> rdf:type
        ssn:SensingDevice ;
5    ssn:observes <http://purl.oclc.org/NET/muo/ucum/physical-quality/acceleration> .
6 }
```

A operação de *modificação de dados* pode ser usada para remover ou adicionar triplas com base em um padrão de *query* especificado em uma cláusula *WHERE*. Um exemplo desse tipo de operação é exibida no Trecho de Código 2.10, onde o conceito de aceleração presente em todos os dispositivos de sensoriamento de uma base de conhecimento é modificado de `http://purl.oclc.org/NET/muo/ucum/physical-quality/acceleration` para `http://dbpedia.org/ontology/acceleration`.

Listing 2.10: Exemplo de modificação de dados usando a SPARQL.

```
1 PREFIX ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
2
3 DELETE { ?sensingDevice ssn:observes <http://purl.oclc.org/NET/muo/ucum/physical-
      quality/acceleration> }
4 INSERT { ?sensingDevice ssn:observes <http://dbpedia.org/ontology/acceleration> }
5 WHERE { ?sensingDevice rdf:type ssn:SensingDevice . }
```

Finalmente, a *operação de remoção* é nada mais que uma simplificação do processo

de remoção utilizado pela *operação de modificação*. Na *operação de remoção*, a cláusula *WHERE* é usada para definir as triplas que serão removidas. O Trecho de Código apresenta um exemplo de uso da SPARQL para a remoção de todas as descrições de dispositivos de sensoriamento que monitoram valores de aceleração.

Listing 2.11: Exemplo de remoção de dados usando a SPARQL.

```
1 PREFIX ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
2
3 DELETE WHERE { ?sensingDevice ssn:observes <http://purl.oclc.org/NET/muo/ucum/
      physical-quality/acceleration> .
4             ?sensingDevice ?property ?value }
```

3 QoDisco

O *QoDisco* (um acrônimo para *QoC-aware Discovery Service*) é um serviço distribuído de descoberta de recursos e informações de contexto para IoT que estende o *ForwarDS-IoT* (GOMES et al., 2015) para permitir a anotação de dados capturados por sensores com informações de contexto. Ele segue um conjunto de importantes requisitos para a construção de serviços de descoberta para IoT, incluindo o suporte a consultas com múltiplos atributos, informações de contexto, e operações síncronas e assíncronas de descoberta. Além disso, o *QoDisco* é estruturado sobre uma arquitetura distribuída, permitindo a descoberta de recursos registrados em um ou mais repositórios.

Para que uma entidade (e.g., aplicação, usuário final ou plataforma de *middleware*) permita a busca dos seus registros (descrições de recursos e dados capturados por sensores) pelo *QoDisco*, é necessário que ela disponibilize uma API que permita o acesso do *QoDisco* a seu(s) repositório(s) de registros. Essas descrições deverão seguir o modelo de informação definido pelo *QoDisco*, que contém um vocabulário base de conceitos relacionados a recursos de IoT e informações de contexto e os relacionamentos entre eles. O modelo de informação pode ser estendido pelos clientes para incluir conceitos específicos de um domínio de aplicação, permitindo assim uma descrição mais detalhada e precisa de um registro. Vale destacar que as APIs fornecidas pelos repositórios das entidades que desejam fazer parte da infraestrutura do *QoDisco* devem seguir a API padrão definida pelo próprio *QoDisco* e apresentada na Seção 3.2.2.

O restante deste capítulo apresenta detalhes do serviço de descoberta proposto neste trabalho. A Seção 3.1 apresenta os requisitos que este trabalho busca atender para a criação de um serviço de descoberta no contexto de IoT. A Seção 3.2 apresenta a arquitetura do *QoDisco* enquanto a Seção 3.3 expõe o seu funcionamento, incluindo o processo de busca síncrona e assíncrona de registros. Ademais, a Seção 3.4 descreve o modelo de informação proposto e, por fim, a Seção 3.5 apresenta um protótipo de implementação do *QoDisco*.

3.1 Requisitos

Para propor um conjunto de requisitos para serviços de descoberta em IoT, este trabalho baseou-se em um estudo acerca dos mecanismos já existentes na literatura. Nesse estudo foi adotada uma técnica baseada na *Evidence Based Software Engineering* (KITCHENHAM; DYBA; JORGENSEN, 2004), o mapeamento sistemático (KITCHENHAM; BUDGEN; BRERETON, 2011). O mapeamento sistemático realizado (MAIA et al., 2014) visava apresentar uma visão geral sobre os cenários atuais e abordagens para o desenvolvimento de Sistema-de-Sistemas (JAMSHIDI, 2009) baseados em IoT, além de identificar e discutir alguns desafios e oportunidades de pesquisa nesse contexto. Apesar do escopo desse mapeamento ser mais abrangente que o escopo do trabalho apresentado nesta dissertação, esse mapeamento serviu como ponto de partida para encontrar propostas e requisitos de serviços de descoberta em IoT. Dentre os trabalhos analisados, Paganelli e Parlanti (2012) destacam os requisitos que são fundamentais para qualquer serviço de descoberta no contexto de IoT, em particular a identificação de recursos, a habilidade de realizar consultas em intervalos e com múltiplos atributos, o suporte a múltiplos provedores de informações sobre recursos, e APIs para gerenciamento de recursos. Em adição a esses requisitos, foram considerados cinco outras importantes preocupações nesta dissertação, são elas: (i) a descrição semântica de recursos; (ii) o suporte a busca e anotação de dados capturados por sensores; (iii) a modelagem de informações de contexto e critérios de QoC; (iv) o suporte a busca por localização, e; (v) o suporte a requisições síncronas e assíncronas. Detalhes desses dez requisitos são apresentados na Tabela 2, enquanto o restante dessa seção apresenta a motivação para a escolha dos cinco novos requisitos propostos nesta dissertação.

É esperado que os dados oriundos dos dispositivos em IoT sejam altamente requisitados por aplicações e usuários finais, sendo necessário o armazenamento desses dados em sistemas altamente distribuídos e heterogêneos a fim de se garantir disponibilidade. Entretanto, recuperar dados dessas bases é uma tarefa não trivial sem que haja um modelo de informação comum para representação de dados interpretáveis por máquinas. Chun *et al.* (CHUN et al., 2015) argumentam que um modelo de informação para IoT deve possibilitar a modelagem semântica de dados, tipicamente através de ontologias, permitindo assim a definição de termos comuns usados em IoT para descrição de recursos. Adicionar semântica a um modelo de informação promove interoperabilidade entre recursos, modelos de informação, provedores de dados, e consumidores de recursos, e também facilitar o acesso a dados, integração de serviços, a interpretação semântica, e a extração de conhecimento.

Tabela 2: Requisitos para um serviço de descoberta em IoT.

Requisito	**Descrição**
Esquema de identificação flexível	Transparência e flexibilidade em relação aos esquemas de identificação adotados (e.g., URIs, endereços IP, etc.) no endereçamento de recursos.
Consultas com múltiplos atributos	Capacidade de encontrar recursos através de consultas baseadas no casamento de um ou mais atributos (e.g., um cliente pode buscar por recursos que possuam um certo tipo e uma certa localização).
Consultas em intervalos	Capacidade de encontrar recursos a partir da especificação de limites inferiores e superiores para um ou mais atributos.
Múltiplos publicadores	Várias entidades, além do proprietário do recurso, podem produzir e publicar informações sobre um mesmo recurso.
APIs de gerenciamento	Capacidade de adicionar, atualizar, e remover informações associadas com um dado recurso.
Descrição semântica de recursos	Uso de ontologias para modelar semanticamente recursos e os serviços expostos pelos mesmos.
Suporte a busca e anotação de dados capturados por sensores	Suporte a modelagem semântica dos dados produzidos por sensores, e não somente os recursos que os produzem. Consequentemente, a busca desses dados também deve ser suportada por serviços de descoberta para IoT.
Modelagem de informações de contexto e critérios de QoC	Modelagem de critérios e indicadores de QoC associados com informação de contexto.
Suporte a busca por localização	Capacidade de modelar e buscar informações de localização de recursos.
Requisições síncronas e assíncronas	Capacidade de lidar com requisições síncronas e assíncronas realizadas por clientes.

Além da preocupação com a descrição e descoberta de recursos em IoT, é importante considerar os dados gerados por esses recursos (i.e., sensores). Os fluxos de dados oriundos dos sensores irão alavancar a criação de aplicações novas e inovadoras na área de IoT e contribuirão para a concretização do paradigma emergente da *Big Data*. Entretanto, lidar com o grande volume de dados gerados por esses sensores impõe um desafio à plataformas de gerenciamento de dados em IoT. Nesse contexto, a anotação semântica dos fluxos de dados facilita o processo de gerenciamento de informações, permitindo que usuários e aplicações possam buscar diretamente pelos dados relevantes para a realização de suas atividades, e não somente pelos recursos que fornecem esses dados.

Ademais, vale ressaltar que dados por si só possuem pouco valor à menos que sejam analisados juntamente ao contexto do usuário ou aplicação que deseja usufruir dessas informações. De fato, o objetivo final da IoT é construir um ecossistema que provenha serviços orientados ao usuário e cientes de contexto (WEI; JIN, 2012), por isso é importante aumentar a riqueza do processo de descoberta e suportar ações de tomada de decisão com base em informações de contexto. Porém, a correta definição do contexto de um usuário ou aplicação depende da confiabilidade das informações capturadas por sensores ou providas pelo usuário ou aplicação. Como não se pode garantir que todas as informações de contexto tenham sido obtidas corretamente, se faz necessário o gerenciamento eficiente das informações de QoC utilizadas pelas aplicações. Por consequência, em um modelo de informação ciente de contexto, é necessário o uso de uma abordagem para a descrição do contexto e das informações de QoC associadas.

Entre as principais informações de contexto associados a IoT, a localização é a mais importante, sendo um critério altamente seletivo para a descoberta de informação sobre o mundo físico (CONSORTIUM, 2013). Tipicamente, nós não estamos interessados em encontrar informações ou recursos em qualquer lugar, mas em uma localização bastante específica, geralmente perto do usuário (LOPEZ et al., 2013). Existem várias formas de se modelar localização, i.e., através de coordenadas geográficas multi-dimensionais ou como localizações simbólicas com ou sem uma estrutura hierárquica subjacente. Para eficientemente descobrir recursos e seus serviços associados de acordo com suas localizações, o serviço de descoberta precisa prover o suporte necessário.

Por fim, vale ressaltar que um serviço de descoberta em IoT deve ser capaz de lidar com requisições síncronas e assíncronas. Quando receber uma requisição síncrona, o serviço de descoberta deve processar a requisição e responder a ela imediatamente após a requisição do cliente. Por outro lado, clientes também podem se beneficiar de operações

assíncronas. Como exemplo, o serviços de descoberta pode notificar um cliente sempre que um recurso se encontra indisponível. Desse modo, não é necessário que um cliente consulte periodicamente a plataforma em busca de atualizações.

3.2 Arquitetura

O *QoDisco* baseia-se em uma arquitetura distribuída composta por uma coleção de repositórios autônomos que armazenam descrições de recursos e dados capturados por sensores (i.e., registros). Essa abordagem foi escolhida porque arquiteturas distribuídas são mais adequadas para o desenvolvimento de serviços de descoberta em IoT, já que elas podem prover várias vantagens em relação a abordagens centralizadas, em particular alta escalabilidade, tolerância a falhas, controle independente de repositórios, e distribuição de dados.

No contexto deste trabalho, nós visionamos *clientes* com dois tipos de papéis: (i) *proprietários de repositórios*, clientes que permitem a descoberta de seus registros por provê-los através de um repositório registrado no *QoDisco*, e; (ii) *clientes comuns*, clientes que desejam buscar e gerenciar recursos na plataforma. Entretanto, um mesmo cliente pode desempenhar ambos os papéis, disponibilizando assim repositórios com os seus registros e realizando buscas e gerenciamento de registros através da plataforma do *QoDisco*. Além disso, o *QoDisco* utiliza o estilo arquitetural RESTful para suportar a interação entre clientes (ambos os tipos) e a plataforma. A Figura 15 ilustra a interação entre o *QoDisco* e os seus clientes.

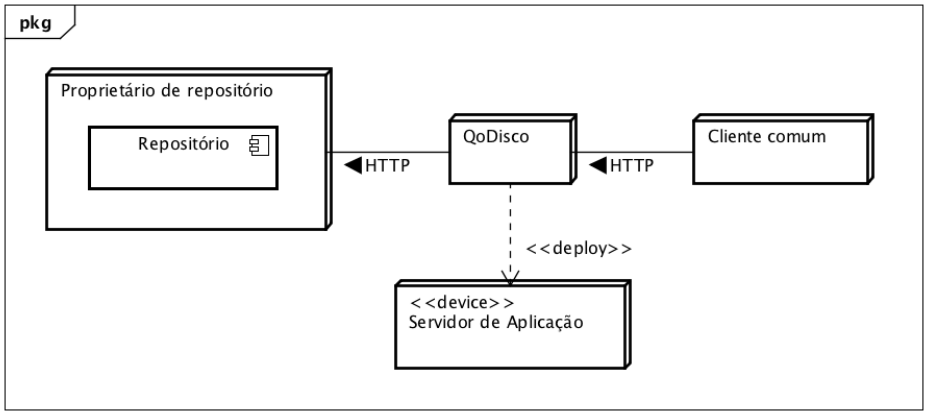


Figura 15: Interação entre o *QoDisco* e os seus clientes.

A arquitetura do *QoDisco* compreende quatro módulos, cada um responsável por um conjunto de funcionalidades específicas providas pela plataforma. A Figura 16 ilustra esses

módulos, descritos nas subseções a seguir.

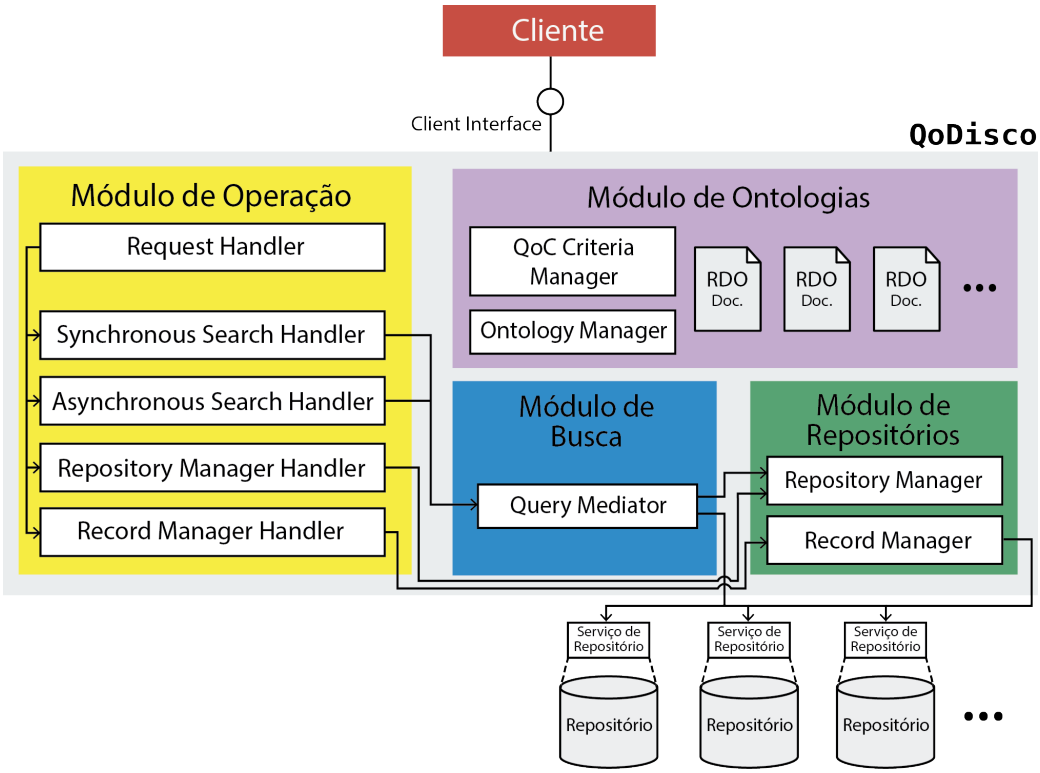


Figura 16: Arquitetura do *QoDisco*.

3.2.1 Módulo de Ontologias

O *Módulo de Ontologias* consiste de documentos RDO (*Repository Domain Ontology*), que descrevem os conceitos que compõem o modelo de informação do *QoDisco*. Nesse módulo, o componente *Ontology Manager* oferece operações para adicionar, remover, e modificar documentos RDO. Já o componente *QoC Criteria Manager* é responsável por adicionar, remover, e modificar critérios de QoC (também definidos em documentos RDO). Novos documentos RDO podem ser definidos estendendo os já existentes, permitindo assim a adição de conceitos utilizados em um domínio em particular. Vale ressaltar que os critérios de QoC definidos nesses documentos também devem ser definidos seguindo o modelo de informação do *QoDisco*.

3.2.2 Módulo de Repositórios

A principal funcionalidade do *Módulo de Repositórios* é gerenciar repositórios, cada um tendo suas funcionalidades expostas por uma API, a chamada *Serviço de Repositório*. Nesse módulo, o componente *Repository Manager* é responsável por gerenciar repositórios

Tabela 3: API dos repositórios.

Operação	**Descrição**	**URL**
Query	Busca por registros no repositório. Essa operação recebe como entrada uma *query* SPARQL e retorna os registros que casam com a *query* especificada.	/dataset/query?query
Update	Insere, remove, e atualiza registros no repositório. Essas ações são realizadas através das operações de inserção, remoção, e atualização oferecidas pela linguagem SPARQL (ver Seção 2.4.1)	/dataset/update?data

e mapeá-los para um (ou mais) documentos RDO presentes no *Módulo de Ontologias*. Isso implica que um repositório, juntando-se a plataforma do *QoDisco*, tem que ser mapeado para ao menos um domínio especificado por um documento RDO. Esse agrupamento de repositórios em domínios permite que buscas por registros possam ser dinamicamente construídas com somente os repositórios mapeados para um certo domínio. A Figura 17 ilustra a relação entre repositórios e documentos RDO.

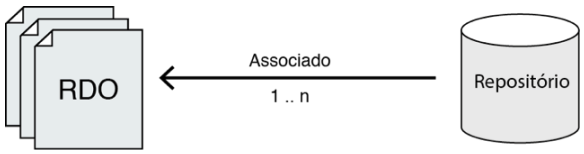


Figura 17: Relação entre repositório e documentos RDO.

Por sua vez, o componente *Record Manager* é responsável por adicionar, remover, e modificar registros nos repositórios. Registros contendo dados capturados por sensores podem incluir ainda indicadores de QoC. Porém, é necessário que os critérios de QoC associados a esses indicadores estejam presentes nos documentos RDO que compõem o *Módulo de Ontologias*. Vale ressaltar ainda que os repositórios que compõem o *QoDisco* mantém sua autonomia gerencial e operacional, o que permite que os donos desses repositórios gerenciem os seus recursos (i.e., adicionem, removam, e os atualizem) de forma independente do *QoDisco*.

Um cliente que deseja provê um repositório próprio para os seus registros e registrá-lo no *QoDisco* deve seguir um conjunto de padrões definidos na API dos repositórios, que é apresentada na Tabela 3.

3.2.3 Módulo de Busca

O *Módulo de Busca* compreende o componente *Query Mediator*, que encaminha *queries* para os repositórios registrados no *QoDisco*. Para realizar a busca, esse componente fornece o nome de um documento RDO (i.e., nome de domínio) ao componente *Repository Manager*, que provê a URL dos repositórios mapeados para tal domínio. Por exemplo, se o *Query Mediator* realizar uma requisição para os repositórios pertencentes ao domínio de cuidados de saúde, o componente *Repository Manager* irá retornar as URL de todos os repositórios mapeados para esse domínio.

3.2.4 Módulo de Operações

O Módulo de Operações provê alguns serviços de alto-nível, tais como a descoberta síncrona e assíncrona de registros, e é composto por cinco componentes:

1. *Request Handler*: É o ponto de entrada do *Módulo de Operação*, recebendo requisições síncronas e assíncronas para a busca, além de requisições para o gerenciamento de repositórios e registros. As requisições feitas a esse componente são encaminhadas para os outros componentes que compõem esse módulo, sendo que cada um deles é responsável por um tipo de requisição.

2. *Synchronous Search Handler*: Realiza as consultas síncronas por registros usando o *Query Mediator*;

3. *Asynchronous Search Handler*: Lida com consultas assíncronas de registros baseadas em dois tipos de notificação: (i) notificações acerca da descoberta de um *novo recurso* que case com uma *query* de busca previamente fornecida pelo cliente e (ii) notificações acerca da modificação ou remoção de um registro;

4. *Repository Manager Handler*: Interage com o componente *Repository Manager* para realizar as operações de adição e remoção de repositórios.

5. *Record Manager Handler*: Interage com o componente *Record Manager* para a adição, remoção e modificação de registros.

A *Client Interface* é o ponto de entrada para o *QoDisco*, fornecendo as funcionalidades do *Módulo de Operação* e do *Módulo de Ontologias* aos clientes. A *Client Interface* obedece o estilo arquitetural REST (*Representational State Transfer*), confiando, assim,

no protocolo HTTP para a realização das métodos sumariados na Tabela 4. As operações da *Client Interface* são relacionadas a busca síncrona e assíncrona de registros, e também ao gerenciamento (inserção, remoção, atualização) de registros, repositórios, documentos RDO, e critérios de QoC. As *queries* de busca e as operações de gerenciamento de registros e critérios de QoC da *Cliente Interface* são especificadas usando a linguagem SPARQL. Visando auxiliar o usuário no processo de busca e gerenciamento de registros, o *QoDisco* oferece um editor textual para criação de *queries* e uma interface Web de fácil uso para a especificação de descrições de recursos, abstraindo assim a necessidade do usuário dominar a linguagem SPARQL.

O Trecho de Código 3.1 apresenta um exemplo de requisição síncrona feita à *Client Interface* para recuperar todos os dados capturados por sensores que estão relacionados com medições de níveis de ozônio. Nesse exemplo, o cliente faz uma requisição HTTP GET provendo o método (*sync-search*), o domínio (*pollution*), e a *query* de busca na URL da mensagem.

Listing 3.1: Exemplo de requisição síncrona de busca de registros.

```
1 <!-- URL da mensagem -->
2 GET http://consiste.dimap.ufrn.br/api/qodisco/sync-search?
3 domain=pollution&query=SELECT * WHERE { ?observation a ssn:Observation ;
      ssn:ObservedProperty pol:Ozone }
```

Já para implementar as notificações da busca assíncrona, o *QoDisco* utiliza o paradigma *publish/subscribe* (pub/sub). O paradigma pub/sub implementa um *middleware* chamado de *broker*. O *broker* é responsável por receber, enfileirar e disparar as mensagens recebidas dos *publishers* (i.e., *QoDisco*) para os *subscribers* (i.e., clientes que realizaram a busca assíncrona) (LIU; PLALE, 2003). O *publisher* é responsável por se conectar ao *broker* e publicar mensagens. Já o *subscriber* é responsável por se conectar ao *broker* e receber as mensagens que ele tiver interesse.

O paradigma pub/sub utiliza o conceito de tópicos para processar as mensagens, em que cada mensagem é enviada para um determinado tópico. Diferente de outros protocolos de mensagem, o *publisher* não envia a mensagem diretamente ao *subscriber*, mas sim ao *broker*. O *publisher* envia a mensagem para o *broker* em um determinado tópico. O *broker* é responsável por receber a mensagem do *publisher* e enviá-las para os *subscribers* que estivem registrados em um determinado tópico.

O Trecho de Código 3.2 apresenta o mesmo exemplo de *query* apresentado no Trecho de Código 3.1, porém, dessa vez, o cliente deseja receber todos os novos registros relacionados a observações de níveis de ozônio a medida que eles são registrados na plataforma.

Tabela 4: API do *QoDisco*.

Operação	**Descrição**	**Método HTTP**	**URL**
Busca síncrona	Busca por registros nos repositórios. Essa operação recebe como entrada o domínio e a *query* de busca e retorna os registros que casam com a *query* especificada. O cliente pode, inclusive, incluir critérios de QoC nessa *query* de busca.	GET	/qodisco/api/sync-search?domain &query
Busca assíncrona	Busca por registros nos repositórios, recebendo como entrada o domínio, a *query* de busca, e o tipo de notificação de descoberta. Como resposta desse tipo de requisição, o *QoDisco* retorna o endereço de um *broker* e o identificador de um tópico no qual o cliente deve se subscrever. Sempre que a condição especificada pelo tipo de notificação de descoberta acontece, o cliente é notificado através do *broker*. O *QoDisco* suporta dois tipos de notificações de descoberta, são elas: (i) *Tipo 1* - notificações sobre a descoberta de novos recursos casando com a *query* de busca, e (ii) *Tipo 2* - notificações indicando que uma descrição de recurso foi modificada ou removida da infraestrutura do *QoDisco*.	GET	/qodisco/api/async-search?domain &query&type
Inserção de registro	Adiciona registros e/ou critérios de QoC em um repositório recebendo como entrada o domínio e uma especificação em SPARQL dos dados a serem inseridos.	POST	/qodisco/api/add ?domain&data
Remoção de registro	Remove um registro e/ou critério de QoC de um repositório recebendo o domínio do registro e uma especificação em SPARQL do conceito a ser removido.	DELETE	/qodisco/api/remove ?domain&data
Atualização de registro	Modifica um registro e/ou critério de QoC de um repositório recebendo o domínio do registro e uma especificação em SPARQL do conceito a ser modificado.	PUT	/qodisco/api/update ?domain&data
Inserção de repositório	Insere um repositório recebendo a URL do repositório, os domínios nos quais o repositório irá ser mapeado, e as operações suportadas pelo repositório (ver Seção 3.2.2). Se a inserção for bem sucedida, um identificador para o repositório é gerado pelo *QoDisco* e retornado para o cliente.	PUT	/qodisco/api/ repository ?domains&url &operations
Remoção de repositório	Remove um repositório recebendo a URL do repositório como entrada.	DELETE	/qodisco/api/ repository?url
Inserção de documento RDO	Insere um documento RDO recebendo o nome do domínio (i.e., nome do documento) e a URL de acesso a esse documento.	POST	/qodisco/api/rdo ?rdo
Remoção de documento RDO	Remove um documento RDO recebendo o nome do documento.	DELETE	/qodisco/api/rdo ?name
Atualização de documento RDO	Modifica um documento RDO recebendo o nome do documento e a nova URL do documento.	PUT	/qodisco/api/rdo?rdo

Nesse exemplo, o cliente faz uma requisição HTTP GET especificando o método (*async-search*), o domínio (*pollution*), e o tipo de notificação que o cliente deseja receber (tipo *1*). O *QoDisco* então gera um identificador para essa requisição e o retorna ao cliente juntamente com a URL de um *broker*. Em seguida, o cliente deve se subscrever em um tópico no *broker* passando o identificador retornado pelo *QoDisco*, cabendo ao *QoDisco* publicar atualizações nesse tópico do *broker* a medida que novos recursos relacionados a níveis de ozônio são registrados, e cabendo ao *broker* repassar essas atualizações ao(s) cliente(s) subscrito(s) nesse tópico.

Listing 3.2: Exemplo de requisição assíncrona de busca de registros.

```
1 <!-- URL da mensagem -->
2 GET http://consiste.dimap.ufrn.br/api/qodisco/async-search?
3 domain=pollution&
4 query=SELECT * WHERE { ?observation a ssn:Observation ; ssn:ObservedProperty
      pol:Ozone }&type=1
```

3.3 Funcionamento

As operações fornecidas pela API do *QoDisco*, a *Client Interface*, podem ser divididas em três grupos: (i) operação de descoberta síncrona de registros; (ii) operação de descoberta assíncrona de registros, e; (iii) operação de gerenciamento de registros, repositórios, e documentos RDO.

No processo de busca síncrona de registros ilustrado na Figura 18, o cliente requisita a descoberta de registros enviando como parâmetro uma *query* de busca e o nome do domínio em que o registro deve ser buscado. A *Client Interface* encaminha a requisição para o componente *Request Handler*, que identifica que se trata de uma requisição de descoberta de recursos síncrona, encaminhando para o componente responsável por tratar esse tipo de requisição, o *Synchronous Search Handler*, que, por sua vez, encaminha a requisição para o componente *Query Mediator*. O *Query Mediator* verifica todos os repositórios mapeados para o domínio especificado pelo cliente através de uma consulta ao módulo *Repository Manager*. Após ter acesso as URLs desses repositórios, o componente *Query Mediator* consulta cada um deles através dos *Serviços de Repositório* pertencentes a cada um deles. Cada *Serviço de Repositório* irá retornar um conjunto de registros que atendam a *query* especificada pelo cliente. Os conjuntos de registros retornados pelos repositórios serão agrupados em um conjunto único pelo módulo *Query Mediator* e retornados ao cliente como resposta da requisição de busca síncrona de registros.

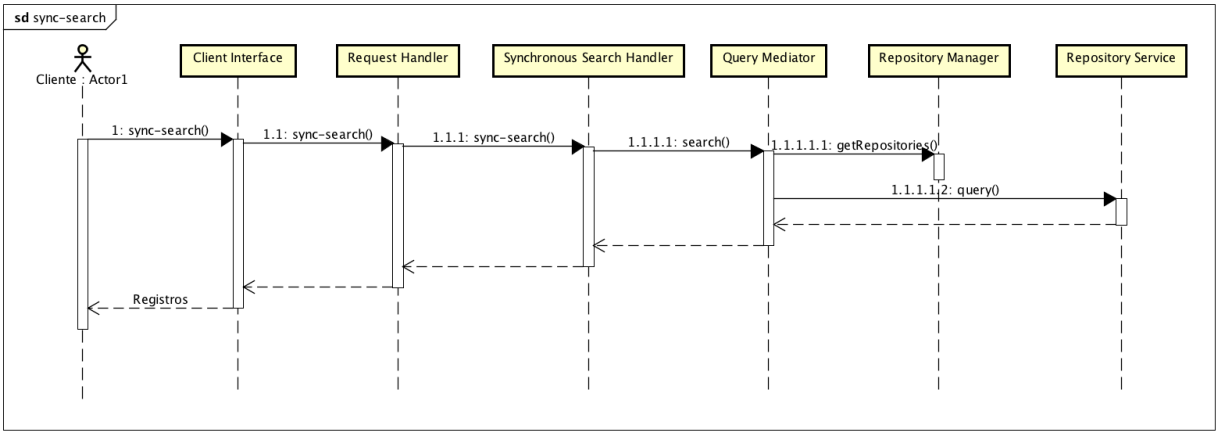


Figura 18: Processo de busca síncrona de registros.

Na busca assíncrona de registros, o cliente passa como parâmetro para a *Client Interface* uma *query* de busca, o domínio dos repositórios, e o tipo de ocorrência que ele deseja monitorar. A *Client Interface* encaminha a requisição para o componente *Request Handler*, que identifica o tipo da requisição e a repassa ao componente *Asynchronous Search Handler*. O componente *Asynchronous Search Handler* gera um tópico e o armazena juntamente com a *query* de busca passada pelo cliente. O identificador desse tópico e o endereço de um *broker* são retornados ao cliente, que deve se subscrever nesse tópico do *broker* para receber as notificações de busca assíncrona. Vale ressaltar que o componente *Record Manager Handler* é responsável por verificar se os novos registros inseridos e atualizados na infraestrutura atendem as *queries* de busca associadas ao tópicos. Caso uma *query* seja atendida, o componente *Record Manager Handler* envia o resultado da *query* para o *broker* no tópico associado com essa *query*.

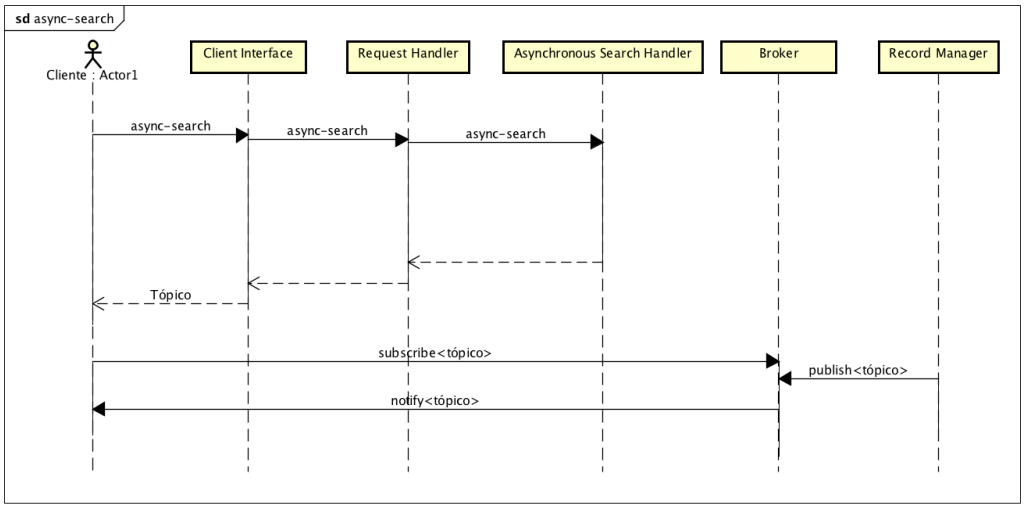


Figura 19: Processo de busca assíncrona de registros.

A Figura 20 ilustra como as operações de inserção, remoção e atualização de documentos RDO podem ser requisitadas através da *Client Interface*. Para realizar essas operações, a *Client Interface* encaminha a requisição do cliente para o componente *Ontology Manager*. Esse componente disponibiliza, através de uma URL, o documento RDO passado como parâmetro pelo cliente. Já que o documento é publicamente acessível, outros clientes do *QoDisco* podem acessá-lo e definir registros como instâncias desse novo documento RDO.

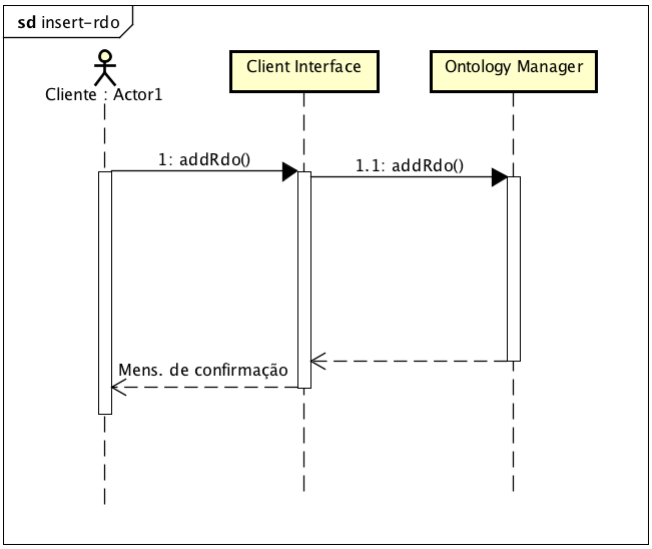


Figura 20: Processo de inserção de documento RDO.

Por fim, as operações de inserção, remoção e atualização de registros e as operações de inserção e remoção de repositórios foram agrupadas devido a similaridade entre esses processos. A Figura 21 ilustra um desses processos, a inserção de registros. Nesse processo, o cliente requisita a busca de registros através da *Client Interface*, que encaminha a requisição para o componente *Request Handler*. Esse componente identifica que se trata de uma requisição de gerenciamento de registros, encaminhando-a para o componente *Record Manager*. O componente *Record Manager* comunica-se com um dos *Serviço de Repositório* mapeados para o domínio especificado pelo usuário de forma randômica, solicitando a inserção do registro no repositório gerenciado por esse *Serviço de Repositório*.

3.4 Modelo de informação

A Figura 22 destaca o modelo de informação adotado pelo *QoDisco*. Como previamente mencionado, esse modelo de informação compreende um vocabulário baseado em ontologias de conceitos relacionados a recursos de IoT e serviços, os relacionamentos entre

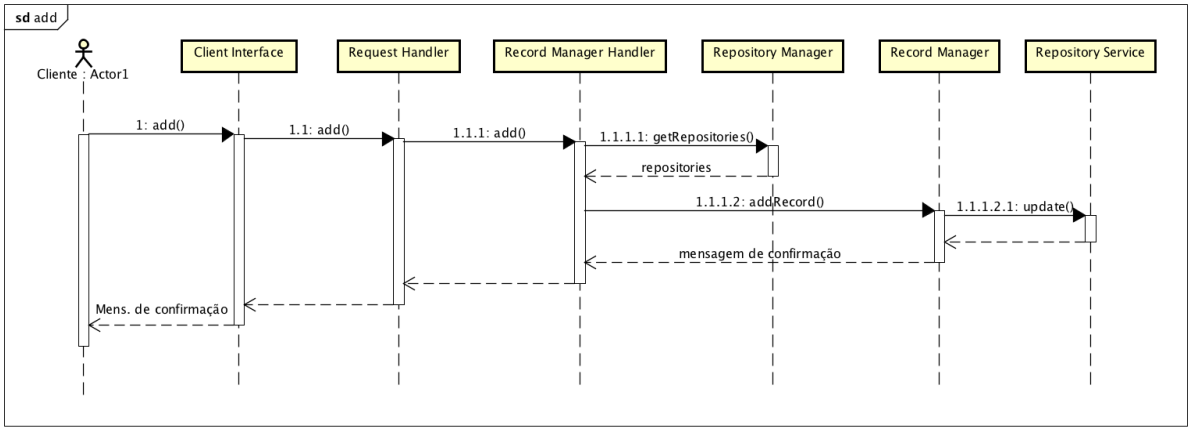


Figura 21: Processo de inserção de registro.

eles, e informações relacionadas a QoC. Para semanticamente descrever recursos, o modelo de informação do *QoDisco* utiliza a ontologia SAN, uma extensão da ontologia SSN que provê conceitos, atributos, e propriedades para modelar sensores e atuadores. Adicionalmente, nós incorporamos parte da ontologia SOUPA (CHEN; FININ; JOSHI, 2005), visando incluir conceitos para descrição de localizações espaciais de entidades (chamadas na ontologia de *PhysicalStructures*) em termos de latitude, longitude, altitude, distancia, e superfície, e também representações simbólicas de relações espaciais e de espaço (i.e., representações de alto-nível, ou seja, sem a necessidade da especificação de dados absolutos como latitude e longitude), e.g., um *cômodo* situado em um dado *piso* de um *prédio*. No nosso modelo de informação, informações relacionadas a localização podem ser associadas com recursos e com os dados capturados por sensores.

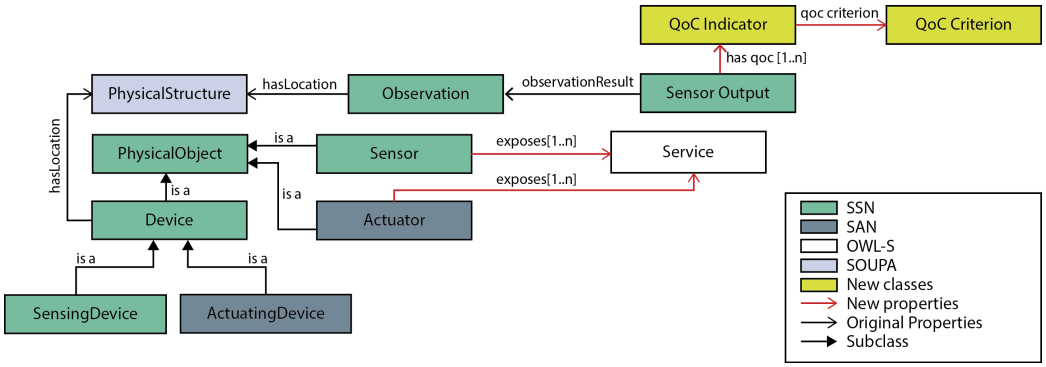


Figura 22: Modelo de informação do *QoDisco*.

Apesar de abrangerem a descrição de sensores e atuadores, as ontologias SAN e SSN não provêm elementos para modelar os serviços expostos pelos recursos. Por essa razão, o modelo de informação do *QoDisco* também compreende a ontologia OWL-S, que tem sido largamente utilizada para a modelagem semântica de serviços Web em aplicações na academia e na industria. Para representar a relação entre recursos e serviços, nós criamos

a propriedade *exposes*, que liga os conceitos de recursos (i.e., sensores e atuadores) ao conceito de *serviço*, como mostra a Figura 22.

Para cobrir preocupações relacionadas a QoC, nós incorporamos parte do meta-modelo do QoCIM (MARIE et al., 2014), descrevendo ele através de uma ontologia. No modelo de informação do *QoDisco*, a propriedade *has_ qoc* associa um indicador de QoC a uma dado capturado por um sensor (*SensorOutput*), enquanto a propriedade *qoc_ critetion* representa a associação entre indicadores de QoC e critérios de QoC. Nosso modelo de informação segue a mesma ideologia flexível do QoCIM, i.e., definir uma base para o desenvolvimento e representação de qualquer critério de QoC ao invés de prover uma lista pré-definida de critérios de QoC suportados. Além disso, um dado critério de QoC pode ser construído sobre outros critérios primitivos ou compostos. O Trecho de Código 3.3 apresenta, no formato *Turtle*[1], a classe definida para representar critérios de QoC no modelo de informação do *QoDisco*.

Listing 3.3: Classe criada para representar critérios de QoC no modelo de informação do *QoDisco*.

```
@prefix rdfs: <https://www.w3.org/TR/rdf-schema#>
@prefix owl: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>
@prefix qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>

qodisco:QoC
    a                owl:Class ;
    rdfs:comment     qodisco;description ;
    rdfs:subClassOf  [ a                    owl:Restriction ;
                       owl:onProperty       qodisco:composed_by ;
                       owl:someValuesFrom   qodisco:QoC ;
                     ] ;
    rdfs:subClassOf  [ a                           owl:Restriction ;
                       owl:onDataRange             xsd:string ;
                       owl:onProperty              qodisco:description> ;
                       owl:qualifiedCardinality    1
                     ] ;
    rdfs:subClassOf  [ a                           owl:Restriction ;
                       owl:onDataRange             xsd:string ;
                       owl:onProperty              qodisco:direction> ;
                       owl:qualifiedCardinality    1
                     ] ;
    rdfs:subClassOf  [ a                           owl:Restriction ;
                       owl:onDataRange             xsd:boolean ;
                       owl:onProperty              qodisco:invariant> ;
                       owl:qualifiedCardinality    1
                     ] ;
    rdfs:subClassOf  [ a                           owl:Restriction ;
                       owl:onDataRange             xsd:int ;
                       owl:onProperty              qodisco:maximum_value> ;
```

[1]RDF 1.1 Turtle – https://www.w3.org/TR/turtle/

```
30                        owl:qualifiedCardinality  1
31                      ] ;
32      rdfs:subClassOf   [ a                       owl:Restriction ;
33                        owl:onDataRange           xsd:int ;
34                        owl:onProperty            qodisco:minimum_value> ;
35                        owl:qualifiedCardinality  1
36                      ] ;
37      rdfs:subClassOf   [ a                       owl:Restriction ;
38                        owl:onDataRange           xsd:string ;
39                        owl:onProperty            qodisco:name> ;
40                        owl:qualifiedCardinality  1
41                      ] ;
42      rdfs:subClassOf   [ a                       owl:Restriction ;
43                        owl:onDataRange           xsd:string ;
44                        owl:onProperty            qodisco:unit ;
45                        owl:qualifiedCardinality  1
46                            ] .
```

Finalmente, o Trecho de Código 4.4 apresenta a descrição OWL no formato Turtle de um indicador de QoC. Um indicador de QoC é composto por um valor inteiro e uma referência para um critério de QoC. O valor de um indicador de QoC deve estar entre o valor mínimo e máximo definido no critério de QoC associado a esse indicador.

Listing 3.4: Classe criada para representar indicadores de QoC no modelo de informação do *QoDisco*.

```
1  @prefix rdfs: <https://www.w3.org/TR/rdf-schema#>
2  @prefix owl: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>
3  @prefix qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5
6  qodisco:QoCIndicator
7      a                 owl:Class ;
8      rdfs:subClassOf   [ a                    owl:Restriction ;
9                          owl:onProperty       qodisco:has_qoc_criterion ;
10                         owl:someValuesFrom   qodisco:#QoC>
11                       ] ;
12     rdfs:subClassOf   [ a                    owl:Restriction ;
13                         owl:onProperty       qodisco:has_qoc_value ;
14                         owl:someValuesFrom   xsd:integer
15                       ] .
```

3.5 Implementação

O *QoDisco* foi desenvolvido como um projeto do Eclipse[2] utilizando a linguagem de programação Java, o *QoDisco Web*. O *QoDisco* Web foi implementado usando a pla-

[2]Eclipse IDE: `http://www.eclipse.org/`

taforma Spring IO [3], que constitui um conjunto de API's para o desenvolvimento de aplicações, sendo desenvolvido como um único projeto Web que executa em um servidor Web de aplicação. No entanto, os repositórios que compõem a arquitetura *QoDisco* devem ser providos e gerenciados de forma isolada pelos clientes do *QoDisco* que desejem permitir a busca de seus registros. Para isso, o *QoDisco* define uma API que deve ser implementada pelos clientes que desejem prover seus recursos através de um repositório (ver Seção 3.2.2).

Tanto na implementação do *QoDisco* Web quanto na implementação dos repositórios, é necessário o uso de ferramentas que auxiliem no armazenamento, manipulação e recuperação de informações a partir de arquivos OWL. Para isso, nós utilizamos o *framework Jena*[4]. O Jena é um *framework* Java *open-source* mantido pela *Apache Software Foundation* para a construção de aplicações que seguem o paradigma da Web Semântica. A Figura 23 apresenta os principais componentes do *framework Jena*.

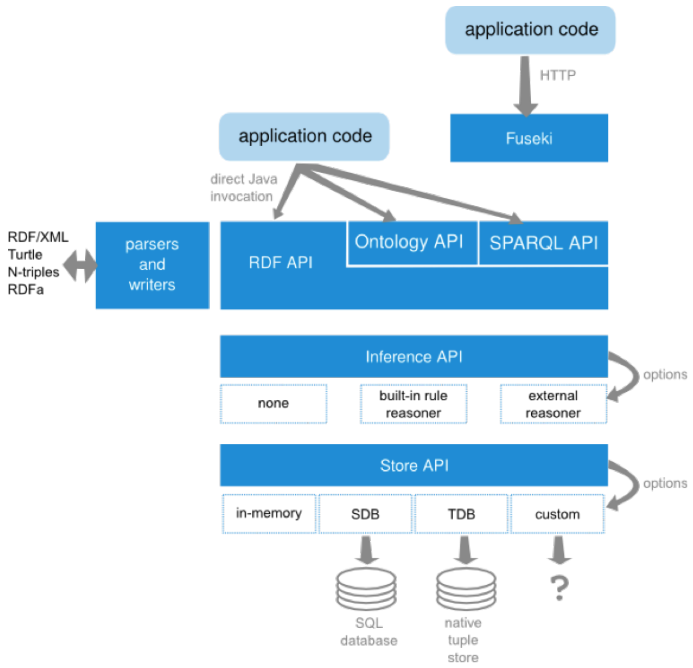


Figura 23: Componentes do *framework Jena*.
Fonte: http://jena.apache.org

Os componentes do *Jena* abordam diversas tecnologias relacionadas à Web semântica, como RDF (e OWL), SPARQL, e ontologias. Na Figura 23, todos os componentes que têm o nome terminado com "API"representam bibliotecas Java, já os outros componentes são ferramentas nativas ou de terceiros. Esse *framework* oferece três formas padrões de

[3] *Spring IO*: `http://spring.io/`
[4] *Jena – A free and open source Java framework for building Semantic Web and Linked Data applications*: `http://jena.apache.org`

armazenamento de dados RDF:

- *In-memory*: oferece a carga e a leitura direta dos dados na memória principal do computador (forma transiente);
- *Simple Database System* (SDB): um sistema de acesso a bancos relacionais como se fossem grafos RDF virtuais (forma persistente);
- *Triplestore Database System* (TDB): uma *triplestore* nativa para armazenamento de grafos RDF (forma persistente).

Esses padrões oferecem um forma de armazenamento de dados RDF, mas eles sozinhos não possibilitam que uma aplicação ou usuário tenha acesso a esses dados via rede. Para oferecer esse tipo de serviço é necessário um *endpoint* SPARQL, uma aplicação que atende consultas SPARQL de aplicações clientes em uma determinada porta, a maioria dos *endpoints* ainda oferecem alguma página Web com um terminal interativo para receber consultas SPARQL feitas por humanos. O *endpoint* do *framework Jena* é o *Fuseki*, o qual consegue oferecer esse serviço para qualquer uma das tecnologias de armazenamento do *framework*. O *Fuseki* oferece, além de uma API para o gerenciamento e consulta de triplas RDF, uma interface gráfica Web que facilita a utilização da plataforma. A Figura 24 apresenta uma captura de tela da página Web principal do Fuseki enquanto em execução.

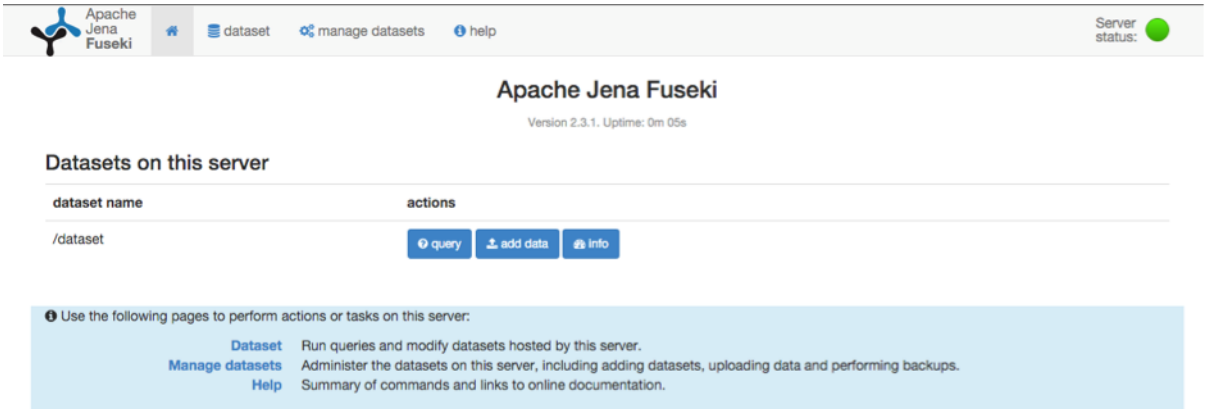


Figura 24: Servidor Fuseki executando.

A interface gráfica do Fuseki oferece um conjunto de operações, como a consulta através de *queries* SPARQL e a adição e edição de triplas RDF. A Figura 25 apresenta a página Web do Fuseki para consulta de registros.

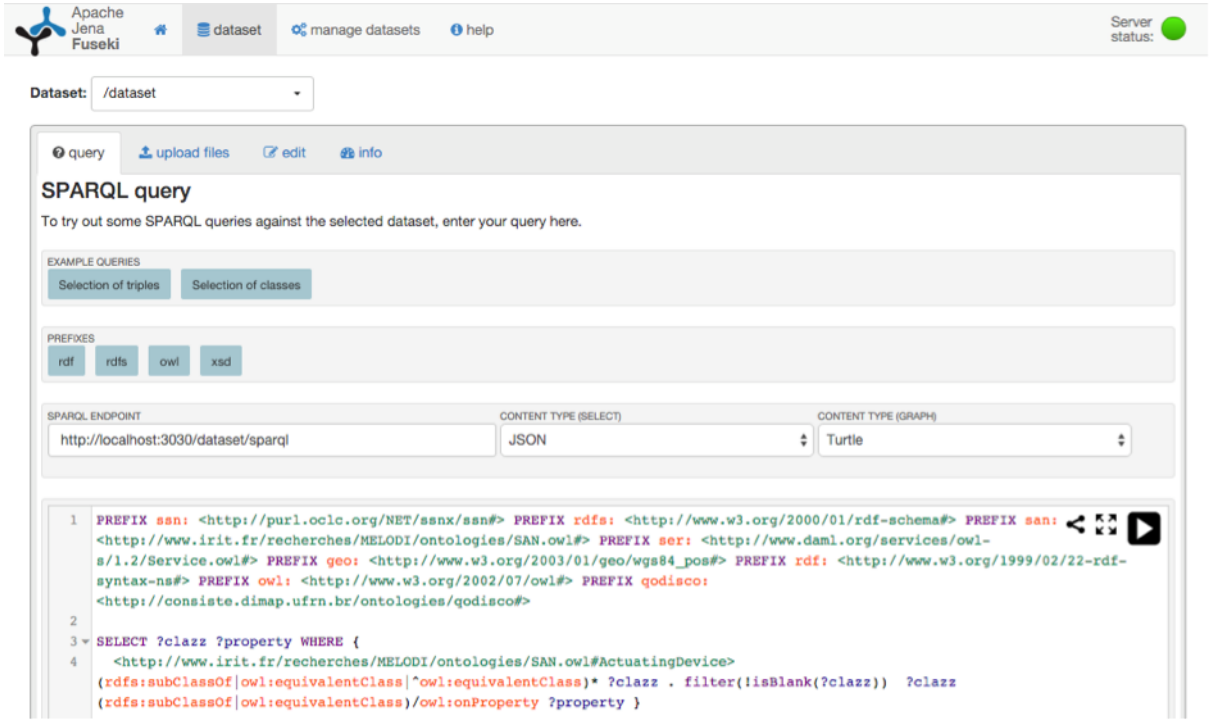


Figura 25: Página Web para consulta de registros.

Devido ao Fuseki atender as especificações definidas pela API do repositório do *QoDisco*, ele é recomendado como implementação padrão para esse API. De fato, ele foi usado como base para a definição da API de repositórios do *QoDisco*. Porém, vale ressaltar que qualquer outra ferramenta pode ser usada para realizar o armazenamento e manipulação de dados no formato RDF/OWL e para a realização de consultas SPARQL, desde que atenda a API definida pelo *QoDisco* para os repositórios (ver Seção 3.2.2). Para a persistência de dados, o padrão de armazenamento escolhido foi o TDB, por fornecer uma forma nativa de banco de dados para RDF.

A seguir, a Seção 3.5.1 apresenta a plataforma *QoDisco* Web.

3.5.1 *QoDisco* Web

O *QoDisco* Web [5] é um projeto Java Web que contém a implementação dos componentes restantes da arquitetura (ver Seção 3.2), a saber: (i) *Client Interface*; (ii) *Request Handler*; (iii) *Synchronous Search Handler*; (iv) *Asynchronous Search Handler*; (v) *Repository Manager Handler*; (vi) *Record Manager Handler*; (vii) *QoC Criteria Manager*; (viii) *Ontology Manager*, e; (ix) *Query Mediator*; (x) *Repository Manager*, e; (xi) *Record Manager*. Além de fornecer suas operações através de uma interface RESTful, esse projeto

[5] O código fonte desse projeto pode ser encontrado em: `http://consiste.dimap.ufrn.br/projects/qodisco/`

contém uma interface gráfica Web que permite a interação de usuários com a plataforma. Para indicar a importância de prover uma interface gráfica para a manipulação de ontologias, o estudo realizado por Broring *et al.* (BRÖRING et al., 2011) descreve um experimento onde foi pedido que 20 participantes descrevessem, através de ontologias, um sensor de estação climática usando uma editor de texto simples. Essas 20 pessoas cometeram 45 erros ao todo. Nesse caso, uma interface gráfica poderia facilitar a descrição de sensores, eliminando a necessidade da inserção manual de metadados. Além disso, tais erros podem ser evitados ao se permitir que os fabricantes de *hardware* produzam e disponibilizem ontologias de domínio contendo conceitos específicos para a modelagem de registros, permitindo assim que desenvolvedores de soluções de IoT possam recuperar e incorporar essas ontologias de domínio em seus próprios sistemas de *software*.

Para realizar o gerenciamento de usuários no *QoDisco* foi utilizado o MySQL [6], um sistema de código aberto de gerenciamento de banco de dados (SGBD) relacional que utiliza a linguagem SQL (*Structured Query Language*) como interface. Algumas de suas principais características são a portabilidade, a compatibilidade com várias linguagens através de *drivers*, a facilidade de uso, a exigência de poucos recursos de *hardware*, e etc. Contudo, o MySQL apenas armazena dados, sejam eles numéricos ou nominais, em tabelas, enquanto os dados persistidos pelo *QoDisco* estão em objetos Java, de modo que rotinas que mapeiem as informações dos objetos Java em informações das tabelas MySQL são necessárias. Para isso foi utilizado a *Java Persistence API* (JPA), uma interface padrão da linguagem Java para persistência de dados que define um modelo de mapeamento objeto-relacional para gerenciamento de dados persistidos. No entanto, uma interface é um tipo abstrato que não contém dados ou código, mas define o comportamento das classes através das assinaturas dos método, sendo necessário uma implementação para poder executar algo. Sendo assim, o presente trabalho utiliza o *framework Hibernate* [7], um *framework* de código aberto para mapeamento objeto-relacional escrito na linguagem Java que basicamente cria as chamadas SQL que permitem a conversão de objetos Java para tabelas do banco de dados e vice-versa. A autenticação do usuário no sistema é necessária para permitir que o *QoDisco* Web gerencie os recursos pertencentes a um usuário, permitindo que somente os usuários que registraram documentos RDO e repositórios sejam capazes de removê-los e editá-los.

Para descrever um novo recurso, o usuário deve primeiramente selecionar através de qual ontologia de domínio ele irá descrever o seu recurso. Neste trabalho, além do domínio,

[6] *MySQL*: `https://www.mysql.com/`
[7] *Hibernate*: `http://www.hibernate.org/.`

o usuário deve especificar um nome único para o recurso, as propriedades observadas, a localização, e os serviços oferecidos pelo recurso. O tipo do recurso está associado ao dispositivo no qual o recurso está hospedado. Inicialmente, o *QoDisco* só possui dois tipos de recursos, *SensingDevice* e *ActuatingDevice* (ver Seção 3.4). Porém, esses tipos podem ser estendidos por ontologias de domínio cadastradas na plataforma. A Figura 26 apresenta uma captura da tela de cadastro de novos recursos do *QoDisco* Web.

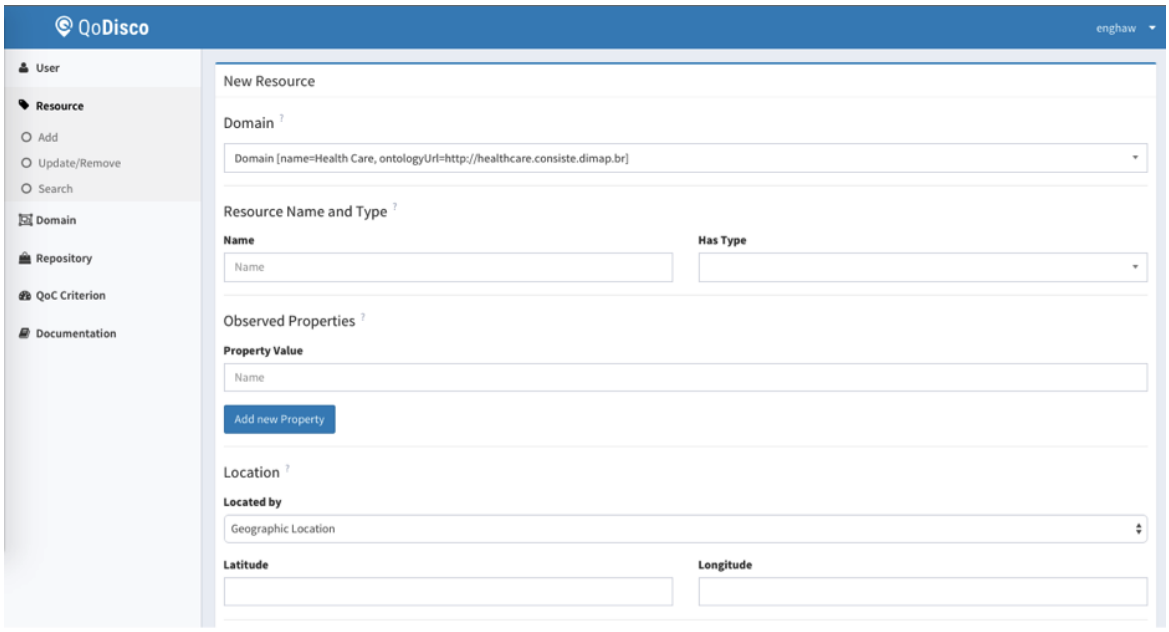


Figura 26: Página Web de registro de novos recursos.

Para buscar registros na plataforma, o usuário deve especificar o domínio dos recursos onde ele quer buscar. Além disso, o usuário deve selecionar valores para atributos definidos na ontologia de domínio especificada. Ao especificar esses atributos, o usuário estará filtrando o resultado da busca. Por exemplo, o usuário pode buscar no domínio de *poluição* todos os dispositivos do tipo *SensingDevice* que monitoram valores relacionados a níveis de ozônio. A Figura 27 apresenta a tela inicial de busca de recursos.

Além disso, o *QoDisco* Web permite a busca de recursos através de uma mapa, implementado usando a API do *Google Maps*[8]. Através desse mapa é possível selecionar e buscar recursos por meio de informações de localização. A busca através do mapa deve ser feita especificando um valor de latitude e longitude. Além disso, o raio em quilômetros deve ser especificado, permitindo a realização de uma busca mais flexível. A busca de recursos através do mapa não permite a adição de nenhum atributo adicional, ou seja, não é possível filtrar a busca através de outros atributos, como tipo de dispositivo e domínio. Vale ressaltar ainda que, caso dados de localização não tenham sido definidos na

[8] *Google Maps*: `https://maps.google.com/`

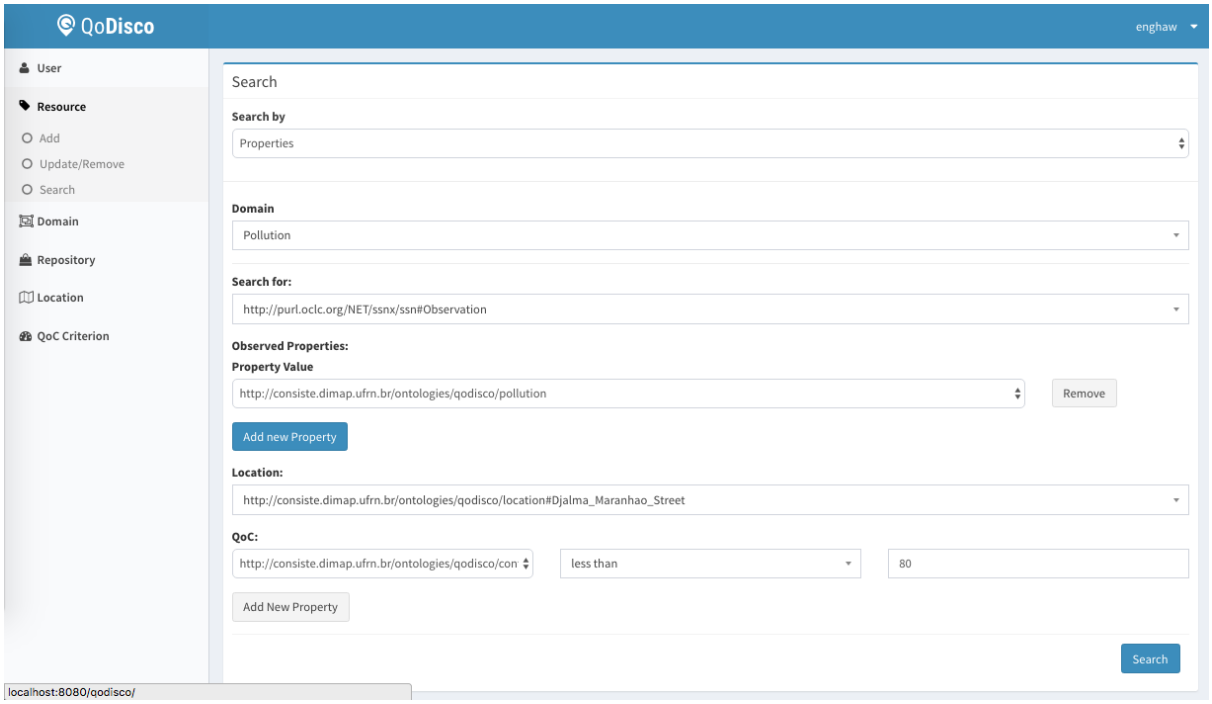


Figura 27: Página Web de busca de registros.

descrição de um registro, ele não será indexado através da busca por localização. Uma captura da tela de busca de registros através de informações de localização é apresentada na Figura 28.

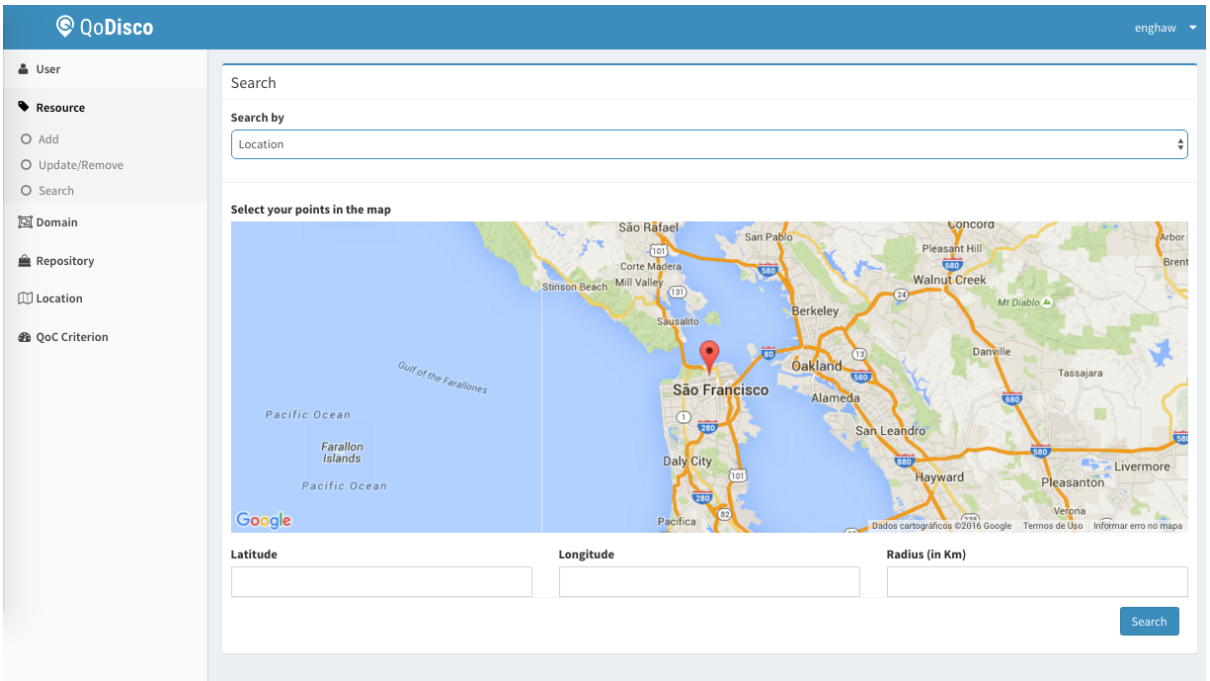


Figura 28: Página Web de busca de registros por localização.

A tela de busca de registros, apresentada na Figura 29, permite que o usuário versado em SPARQL especifique manualmente suas *queries* de busca. Para auxiliar essa

especificação, o *QoDisco* Web utiliza a API YASQE[9], implementada na linguagem de programação Javascript e que permite que os termos reservados da linguagem SPARQL fiquem em destaque e possam ser, em alguns, casos, automaticamente auto-completados pelos editor.

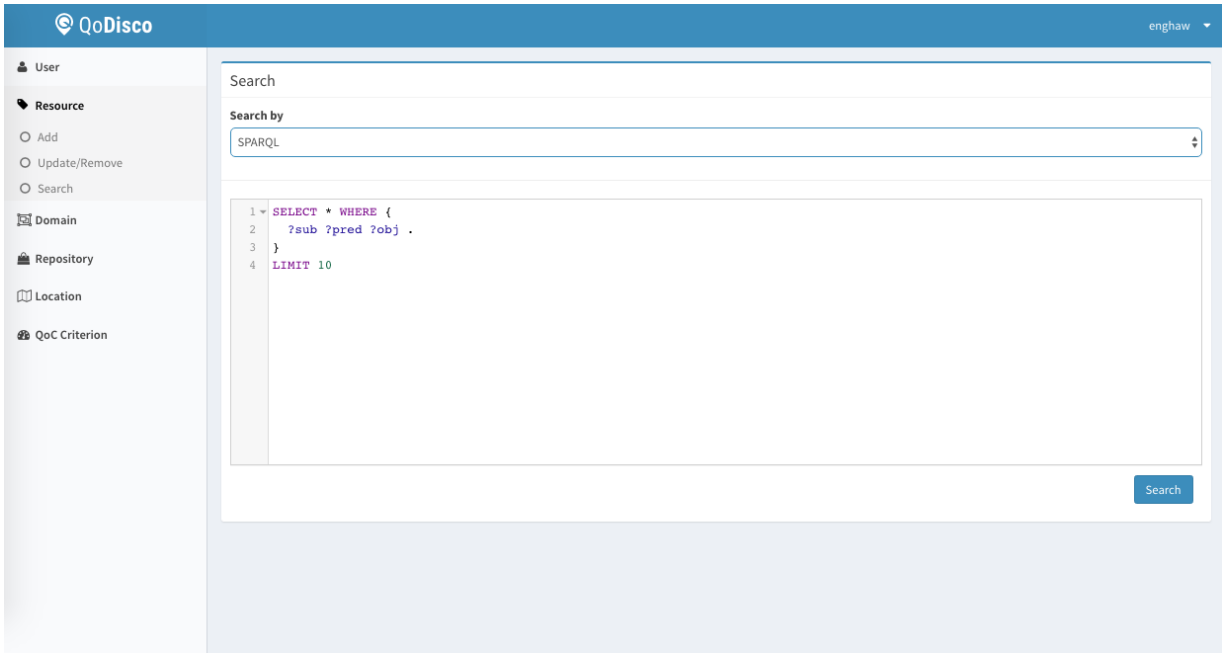


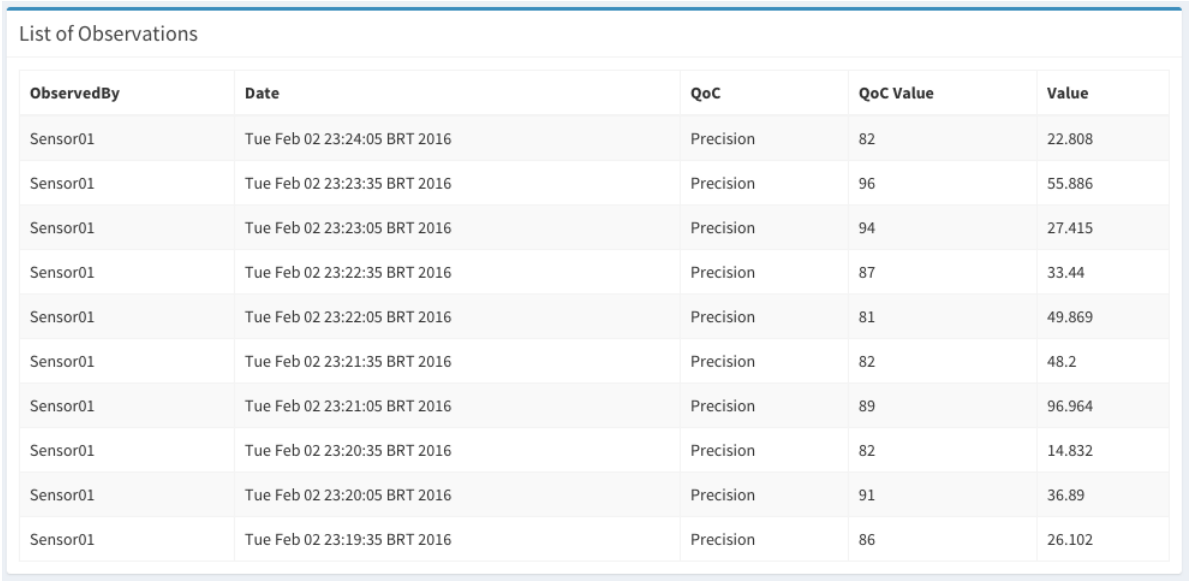
Figura 29: Página Web de busca de registros usando a API do YASQE.

Quanto ao resultado da busca, o *QoDisco* Web exibe uma lista de resultados através de uma tabela, ilustrado na captura de tela apresentada na Figura 30. A tabela contém o nome e o tipo do recurso. Além disso, no lado direito de cada linha da tabela, há dois botões, o primeiro exibe a descrição completa do recurso e o segundo apresenta a posição do recurso em um mapa (também implementado usando a API do *Google Maps*), caso a posição do recurso tenha sido especificada na descrição.

Vale ressaltar que também é possível adicionar novos critérios de QoC através da interface, como mostra a Figura 31. Essa página Web de cadastro de critérios de QoC permite a definição de critérios seguindo o o modelo de informação definido pelo *QoDisco*, derivado do *QoCIM* (ver Seção 3.4). Após a adição de critérios de QoC usando a interface Web do QoC, é possível especificar novos dados capturados por sensores, ou atualizar os já existentes, com indicadores de QoC associados ao novo critério.

O *QoDisco* Web também oferece todas as suas funcionalidade através de um API, que permite a interação via métodos HTTP com usuários, aplicações, e plataformas de *middleware*. A assinatura dos métodos dessa API são apresentadas a seguir no Trecho de

[9] *YASQE*: `http://yasqe.yasgui.org/`



List of Observations

ObservedBy	Date	QoC	QoC Value	Value
Sensor01	Tue Feb 02 23:24:05 BRT 2016	Precision	82	22.808
Sensor01	Tue Feb 02 23:23:35 BRT 2016	Precision	96	55.886
Sensor01	Tue Feb 02 23:23:05 BRT 2016	Precision	94	27.415
Sensor01	Tue Feb 02 23:22:35 BRT 2016	Precision	87	33.44
Sensor01	Tue Feb 02 23:22:05 BRT 2016	Precision	81	49.869
Sensor01	Tue Feb 02 23:21:35 BRT 2016	Precision	82	48.2
Sensor01	Tue Feb 02 23:21:05 BRT 2016	Precision	89	96.964
Sensor01	Tue Feb 02 23:20:35 BRT 2016	Precision	82	14.832
Sensor01	Tue Feb 02 23:20:05 BRT 2016	Precision	91	36.89
Sensor01	Tue Feb 02 23:19:35 BRT 2016	Precision	86	26.102

Figura 30: Página Web contendo resultados de uma busca por registros.

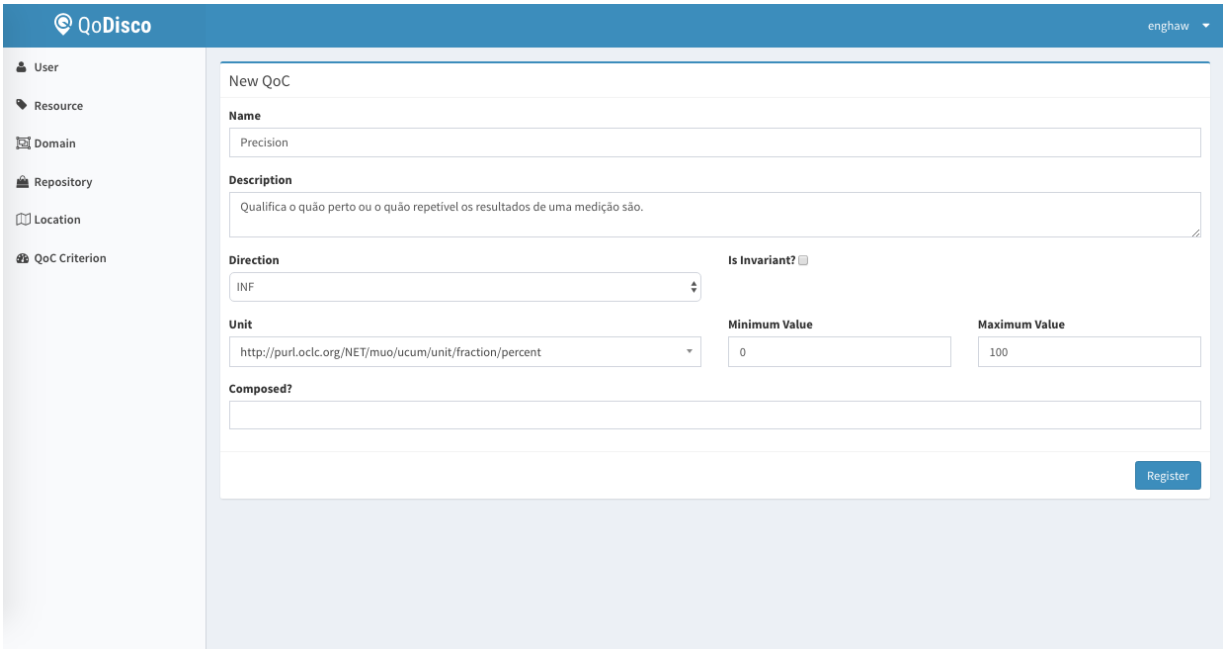


Figura 31: Página Web de registro de novos critérios de QoC.

Código 3.5, e são descritos em detalhes na Seção 3.2. Através da plataforma Spring, é possível passar parâmetros para os métodos via HTTP através das anotações *@RequestParam*, que especifica que o parâmetro será passado via URL, e *@ResponseBody*, que especifica que o parâmetro será passado via corpo da mensagem HTTP.

Listing 3.5: Assinatura da API do *QoDisco*.

```
1 public class QoDiscoApi {
2
3     @RequestMapping(value="/sync-search", method=RequestMethod.GET, produces="
          application/xml")
```

```
4   public @ResponseBody String syncSearch(@RequestParam("domain") String domain,
        @RequestParam("query") String query){
5         // Implementacao ...
6   }
7
8   @RequestMapping(value="/async-search", method=RequestMethod.GET, produces="text/
        html")
9   public @ResponseBody String asyncSearch(@RequestParam("query") String query,
        @RequestParam("domain") String domain, @RequestParam("type") int type){
10        // Implementacao ...
11  }
12
13  @RequestMapping(value="/add", method=RequestMethod.POST)
14  @ResponseStatus(HttpStatus.OK)
15  public void add(@RequestParam("data") String data, @RequestParam("domain") String
        domain){
16        // Implementacao ...
17  }
18
19  @RequestMapping(value="/update", method=RequestMethod.PUT)
20  public void update(@RequestParam("data") String data, @RequestParam("domain")
        String domain){
21        // Implementacao ...
22  }
23
24  @RequestMapping(value="/remove", method=RequestMethod.DELETE)
25  public void remove(@RequestParam("data") String data, @RequestParam("domain")
        String domain){
26        // Implementacao ...
27  }
28
29  @RequestMapping(value="/repository", method=RequestMethod.POST)
30  public void addRepository(@RequestParam("domains") List<String> domains,
        @RequestParam("url") String repositoryUrl, @RequestParam("operations") List<
        String> operations){
31        // Implementacao ...
32  }
33
34  @RequestMapping(value="/repository", method=RequestMethod.DELETE)
35  public ResponseEntity<Void> removeRepository(@RequestParam("url") String url){
36        // Implementacao ...
37  }
38
39  @RequestMapping(value="/rdo", method=RequestMethod.POST)
40  public ResponseEntity<Void> addRdo(@RequestBody Domain domain){
41        // Implementacao ...
42  }
43
44  /* Removes an RDO document by receiving its identifier as input. */
45  @RequestMapping(value="/rdo", method=RequestMethod.DELETE)
46  public ResponseEntity<Void> removeRdo(@RequestParam("name") String name){
47        // Implementacao ...
48  }
49
```

```
50     @RequestMapping(value="/rdo", method=RequestMethod.PUT)
51     public ResponseEntity<Void> updateRdo(@RequestBody Domain domain){
52           // Implementacao ...
53     }
54 }
```

Por fim, o *broker* utilizado não foi desenvolvido neste projeto, pois não é o foco do mesmo. Assim sendo foi escolhido o *broker* Mosquitto[10]. Este *broker* foi escolhido por ser *open-source* e por possibilitar a interação entre o *publishers* e *subscribers* independentemente da linguagem de programação na qual eles são desenvolvidos.

3.5.2 Integração com a EcoDiF

Para detalhar o processo de integração do *QoDisco* com a plataforma de *middleware* EcoDiF, é necessário esmiuçar detalhes de implementação da própria EcoDiF, incluindo as tecnologias de *software* usadas em sua implementação e o modo com que alguns módulos de sua arquitetura foram desenvolvidos (ver Seção 2.1.1.1). A EcoDiF foi implementada utilizando a linguagem de programação Java e implantada em um servidor de aplicações JBoss [11], que permite o gerenciamento de componentes distribuídos, promove a confiabilidade de aplicações baseadas em um conjunto de serviços acessíveis via Web e em grandes volumes de dados, como é típico em ambientes de IoT. Usuários podem acessar as funcionalidades da EcoDiF através de uma interface Web provida pelo *Módulo de Gerenciamento e Visualização* e implementada com as tecnologias abertas JavaServer Faces (JSF) [12] e Primefaces [13].

Na EcoDiF, os dispositivos são integrados a plataforma através dos seus respectivos *drivers* (via o *Módulo de Conexão de Dispositivos*). Os dados obtidos dos dispositivos são enviados à plataforma através de requisições HTTP PUT, provendo assim uma interface RESTful para os clientes (sejam humanos ou aplicações). Para dar suporte a tal abordagem RESTful, a EcoDiF adota a implementação RESTEasy [14]. Os dados (*feeds*) produzidos pelos dispositivos são estruturados utilizando o protocolo EEML e são tratados pelo *Módulo de Manipulação de Dados* para serem efetivamente registrados em uma base de dados relacional MySQL, utilizando as especificações da JPA [15] implementadas no

[10] *Mosquitto*: `http://mosquitto.org`
[11] *Jboss*: `http://www.jboss.org/jbossas`
[12] *Java Server Faces*: `http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html/`
[13] *Primefaces*: `http://primefaces.org/`
[14] *RESTEasy*: `http://www.jboss.org/resteasy/`
[15] *Java Persistence API* (JPA): `http://www.oracle.com/technetwork/java/javaee/tech/`

framework Hibernate. Os *feeds* são publicamente disponibilizados através de URLs e os desenvolvedores de aplicações podem construir então aplicações *mashup* fazendo uso dos dados providos pelos *feeds* que são integrados à EcoDiF através do Módulo de Aplicações e da linguagem para construções de aplicações EMML.

Entretanto, o *QoDisco* necessita que os valores dos *feeds* sejam descritos em OWL e armazenados em um repositório no formato RDF que, por sua vez, deve ser capaz de responder a *queries* SPARQL. Por isso, foi necessário a implementação de um módulo que realize a conversão de EEML para RDF e que permita que esses valores em RDF sejam armazenados em um repositório, implementado usando a tecnologia Fuseki e gerenciado pela própria EcoDiF.

Para realizar a conversão, toda vez que um HTTP PUT é realizado por um dispositivo para atualizar um *feed*, a EcoDiF estrutura esse dado em EEML e utiliza esse módulo para fazer a conversão e o armazenamento de dados EEML também no formato RDF. A Figura 32 apresenta a interação entre a EcoDiF, o módulo de conversão, e o *QoDisco*.

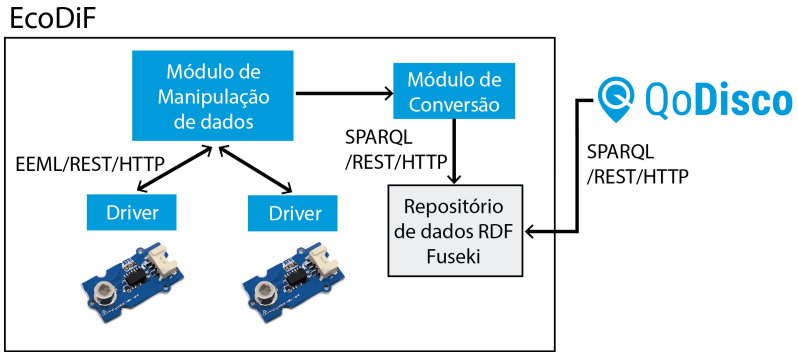


Figura 32: Integração entre a EcoDiF e o *QoDisco*.

O ponto de entrada do módulo de conversão é um classe *RepositoryService*, apresentada no Trecho de Código 3.6 e responsável por descrever os dados dos *feeds* seguindo o modelo de informação do *QoDisco* (ver Seção 3.4). Esses dados são especificados em RDF e enviados para o servidor Fuseki via HTTP.

Listing 3.6: Classe *RepositoryService*.

```
1 public class RepositoryService {
2
3    private RepositoryDao repositoryDao;
4
5    public void insertObservationWithQoC(String repositoryUrl, String sensorName,
         String data, String observedProperty, String unit, String date, String
         qoCCriterion, int qoCValue, String observationName, String feed){
6       repositoryDao = new RepositoryDao(repositoryUrl);
```

persistence-jsp-140049.html

```
7         repositoryDao.insertObservationResultValue(observationName, data, unit);
8         repositoryDao.insertQoCIndicator(observationName, qoCCriterion, String.valueOf(
              qoCValue));
9         repositoryDao.insertSensorOutputWithQoC(observationName, sensorName);
10        repositoryDao.insertObservation(observationName, date, sensorName,
              observedProperty, feed);
11    }
12
13    public void insertObservationWithoutQoC(String repositoryUrl, String sensorName,
          String data, String observedProperty, String unit, String date, String
          observationName, String feed){
14        repositoryDao = new RepositoryDao(repositoryUrl);
15        repositoryDao.insertObservationResultValue(observationName, data, unit);
16        repositoryDao.insertSensorOutputWithoutQoC(observationName, sensorName);
17        repositoryDao.insertObservation(observationName, date, sensorName,
              observedProperty, feed);
18    }
19 }
```

A classe *RepositoryService* possui dois métodos, um responsável por descrever os dados dos *feeds* sem a adição de indicadores de QoC (linha 5) e outro responsável por descrever os dados incluindo esse tipo de indicador (linha 13). Devido a EcoDiF não lidar com valores de QoC, o método *insertObservationWithQoC* só é usado via a geração de valores aleatórios para a especificação de indicadores de QoC. Excluindo os valores de QoC, os valores passados como parâmetros na chamada de ambas as funções da classe *RepositoryService* são gerenciados pela EcoDiF e, por isso, podem ser facilmente especificados na chamada a esses métodos. Nota-se que, para fazer essa descrição, os dois métodos se utilizam de uma outra classe, a *RepositoryDao*. Essa classe, apresentada no Trecho de Código 3.7, é responsável por realizar a inserção de dados na base de dados Fuseki através de chamadas a API Jena.

Listing 3.7: Classe *RepositoryDao*.

```
1  public class RepositoryDao {
2
3     private String repositoryUrl;
4
5     public RepositoryDao(String repositoryUrl){
6        this.repositoryUrl = repositoryUrl;
7     }
8     public void insertObservation(String observationName, String date, String
          sensorName, String observedProperty, String address){
9        // Implementacao
10    }
11    public void insertSensorOutput(String observationName, String sensorName){
12       // Implementacao
13    }
14    public void insertObservationResultValue(String observationName, String data,
          String unit){
```

```
15        // Implementacao
16    }
17    public void insertQoCIndicator(String observationName, String qoCCriterion, String
          qoCValue){
18        // Implementacao
19    }
20 }
```

A classe *RepositoryDao* possui quatro métodos, responsáveis pela anotação de dados em formato RDF e pela inserção desses dados em uma base de dados Fuseki. O primeiro método é o *insertObservation* (linha 9), descrito em detalhes no Trecho de Código 3.8. Esse método constrói uma *query* para a inserção de valores através da linguagem SPARQL (linhas 2 a 13). Essa *query* descreve uma *ssn:Observation* (linha 8), incluindo o momento no tempo (linha 9) na qual essa observação foi captada pelo sensor, o identificador de um sensor que associa essa observação com o sensor que a mediu (linha 10), a propriedade observada pelo sensor (linha 11), o endereço URL do *feed* (linha 12), e o valor medido pelo sensor (linha 13). Por fim, através do *framework* Jena, é realizada a requisição ao banco de dados Fuseki para inclusão desses dados (linhas 15 a 17).

Listing 3.8: Método utilizado para a inserção de observações.

```
1  public void insertObservation(String observationName, String date, String sensorName,
       String observedProperty, String address){
2     StringBuilder sb = new StringBuilder();
3     sb.append("PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>");
4     sb.append(" PREFIX qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>");
5     sb.append(" PREFIX ecodif: <http://consiste.dimap.ufrn.br/ontologies/ecodif#>");
6
7     sb.append(" INSERT DATA { ");
8     sb.append(" <http://consiste.dimap.ufrn.br/ontologies/qodisco#"+observationName+">
          a ssn:Observation ;");
9     sb.append(" ssn:observationResultTime '" + date + "' ; " );
10    sb.append(" ssn:observedBy <http://consiste.dimap.ufrn.br/ontologies/qodisco#"+
          sensorName + "> ; " );
11    sb.append(" ssn:observedProperty <"+ observedProperty + "> ; " );
12    sb.append(" ecodif:from <" + address +"> ;"  );
13    sb.append(" ssn:observationResult <http://consiste.dimap.ufrn.br/ontologies/
          qodisco#"+ observationName+ "_output> . }");
14
15    UpdateRequest request = UpdateFactory.create(sb.toString());
16    UpdateProcessor proc = UpdateExecutionFactory.createRemote(request, repositoryUrl+
          "/update");
17    proc.execute();
18 }
```

O segundo método, descrito no Trecho de Código 3.9, é responsável por especificar a saída do sensor, sendo que o valor em si é representado por uma *ssn:ObservationValue*

(linha 10). Esse método recebe o identificador da observação, necessário para associar a observação à saída do sensor, e o identificador do sensor, necessário para associar a saída do sensor ao sensor que a produziu.

Listing 3.9: Método responsável pela inserção de descrições de saída de sensores.

```
1 public void insertSensorOutput(String observationName, String sensorName){
2    StringBuilder sb = new StringBuilder();
3    sb.append("PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>");
4    sb.append(" PREFIX qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>");
5
6    sb.append(" INSERT DATA { ");
7    sb.append(" <http://consiste.dimap.ufrn.br/ontologies/qodisco#"+observationName+"
         _output> a ssn:SensorOutput ;");
8    sb.append(" ssn:isProducedBy <http://consiste.dimap.ufrn.br/ontologies/qodisco#"+
         sensorName+"> ; ");
9    sb.append(" ssn:hasValue <http://consiste.dimap.ufrn.br/ontologies/qodisco#"+
         observationName+ "_resultvalue> . } ");
10
11   UpdateRequest request = UpdateFactory.create(sb.toString());
12   UpdateProcessor proc = UpdateExecutionFactory.createRemote(request, repositoryUrl+
         "/update");
13   proc.execute();
14 }
```

O terceiro método, descrito no Trecho de Código 3.10, é responsável por especificar o resultado da observação realizada por um sensor, a *ssn:ObservationValue*. Esse valor é a saída de um sensor e representa o valor resultante da observação. Esse método recebe o identificador da observação, utilizado nesse método para nomear o próprio *ssn:ObservationValue*, a unidade na qual o valor da medição está expresso, e o valor da observação. Após isso, novamente o *framework* Jena é usado para inserir esses valores (linhas 10 a 12).

Listing 3.10: Método responsável pela inserção de valores de observações.

```
1 public void insertObservationResultValue(String observationName, String data, String
      unit){
2    StringBuilder sb = new StringBuilder();
3    sb.append("PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>");
4
5    sb.append(" INSERT DATA { ");
6    sb.append(" <http://consiste.dimap.ufrn.br/ontologies/qodisco#" + observationName
         + "_resultvalue> a ssn:ObservationValue ;");
7    sb.append(" <http://purl.oclc.org/NET/ssnx/product/smart-knife#
         hasQuantityUnitOfMeasurement> <" + unit + "> ; ");
8    sb.append(" ssn:hasQuantityValue " + data + " . }");
9
10   UpdateRequest request = UpdateFactory.create(sb.toString());
11   UpdateProcessor proc = UpdateExecutionFactory.createRemote(request, repositoryUrl+
         "/update");
```

```
12     proc.execute();
13 }
```

Visto a necessidade de avaliar a descrição de informações de QoC, o quarto método, descrito no Trecho de Código 3.11, apresenta como a descrição em formato OWL de um valor de QoC pode ser realizada. Devido à EcoDiF não gerenciar dados de QoC, esse método foi desenvolvido e testado através da geração de dados aleatórios. Esse método necessita que a EcoDiF passe um critério de QoC previamente registrado no *QoDisco* e um valor para esse critério. Em posse desses valores, esse método é capaz de descrever um indicador de QoC (linhas 5 a 8) e realizar a inserção em um base de dados Fuseki (linhas 10 a 12).

Listing 3.11: Método responsável pela inserção de indicadores de QoC.

```
1 public void insertQoCIndicator(String observationName, String qoCCriterion, String
      qoCValue){
2     StringBuilder sb = new StringBuilder();
3     sb.append("PREFIX qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>");
4
5     sb.append(" INSERT DATA { ");
6     sb.append(" <http://consiste.dimap.ufrn.br/ontologies/qodisco#"+ observationName+"
          _qocindicator> a qodisco:QoCIndicator ;");
7     sb.append(" qodisco:has_qoc_criterion <" + qoCCriterion + "> ; ");
8     sb.append(" qodisco:has_qoc_value " + qoCValue + " . } ");
9
10    UpdateRequest request = UpdateFactory.create(sb.toString());
11    UpdateProcessor proc = UpdateExecutionFactory.createRemote(request, repositoryUrl+
          "/update");
12    proc.execute();
13 }
```

Por fim, haja vista que tanto a EcoDiF quanto o *QoDisco* fornecem interfaces gráficas que facilitam a interação entre usuários e plataforma, um *link* na página Web da EcoDiF foi inserido para redirecionar o usuário da EcoDiF que deseja realizar a busca de dados de sensores anotados semanticamente para o *QoDisco*. A página Web ilustrada na Figura 33 apresenta em destaque em vermelho o *link* entre *QoDisco* e a EcoDiF exibido na barra de navegações da EcoDiF. Vale ressaltar que o usuário da EcoDiF não precisa estar logado para ter acesso as funcionalidades do *QoDisco*. Através do *link* exibido na barra de navegação da EcoDiF, o usuário é redirecionado para a plataforma do *QoDisco* que, além dos dados gerenciados pela própria EcoDiF, pode conter recursos de outras plataformas.

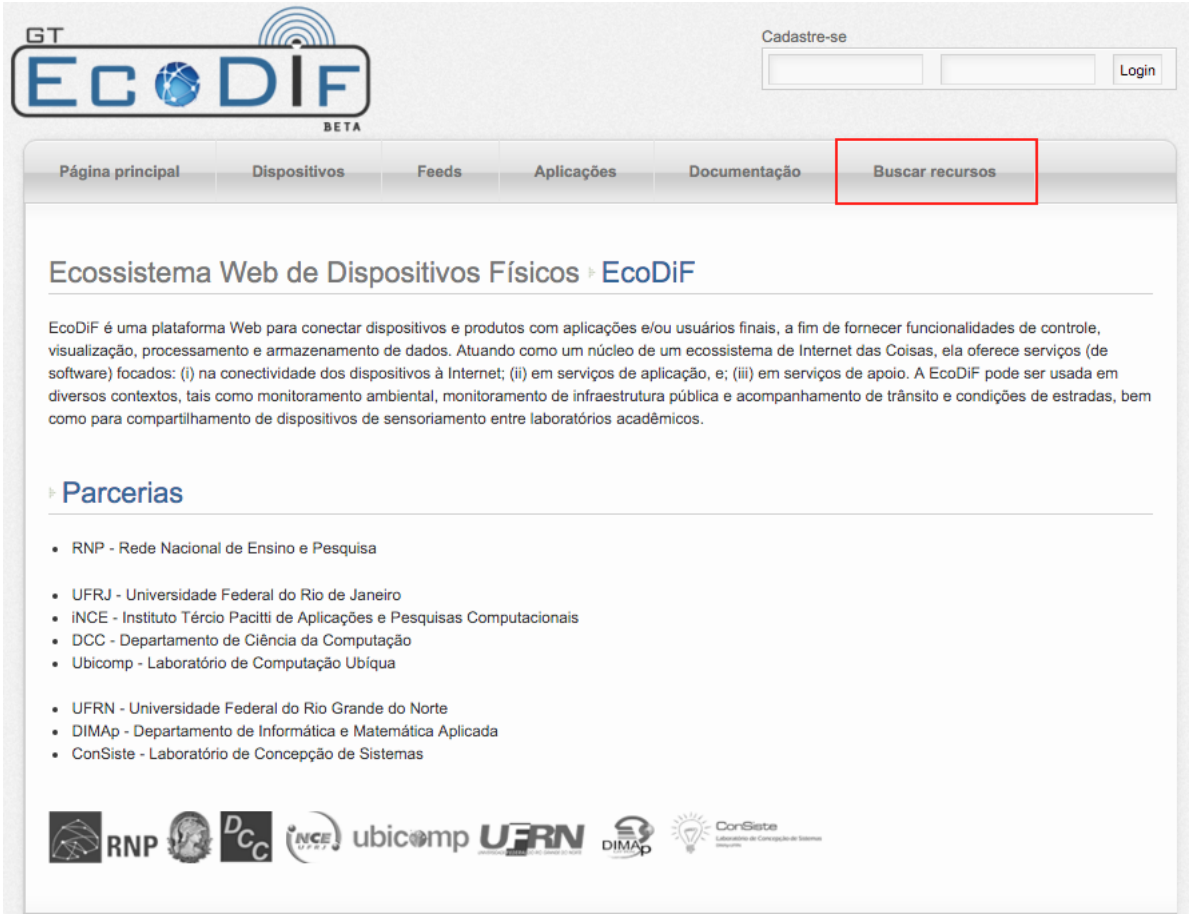


Figura 33: Página Web da EcoDiF.

4 Avaliação

Essa seção apresenta uma avaliação que visa demonstrar a escalabilidade do *QoDisco* através da análise do desempenho da operação de busca por recursos, utilizando um estudo de caso relacionado ao monitoramento de níveis de poluição de ar (Seção 4.1). Tal análise foi motivada por dois objetivos: (i) verificar o quanto a quantidade de recursos registrados em um repositório impacta no desempenho do procedimento de busca, e (ii) verificar como o número de repositórios influencia o comportamento do procedimento de busca. A Seção 4.1 descreve o cenário considerado na nossa avaliação. Já a Seção 4.2 destaca a configuração usada para a realização dos experimentos computacionais. Por fim, a Seção 4.3 mostra os resultados dos experimentos computacionais realizados.

4.1 Cenário

Esta seção descreve um cenário relacionado com o monitoramento de poluição do ar em uma área urbana. Nesse cenário, ônibus públicos são equipados com sensores de monitoramento de poluição do ar e um receptor GPS, responsáveis por prover, respectivamente: (i) medições de poluições, e (ii) a localização dos ônibus durante as medições. A cidade também equipa paradas de ônibus com sensores de monitoramento de ar mais sofisticados (sensores capazes de prover informações de contexto com melhor QoC). Sensores de monitoramento de poluição de ar são responsáveis por medir poluentes tais como ozônio (O_3), monóxido de carbono (CO), dióxido de enxofre (SO_2), e óxido nitroso (N_2O). Quatro critérios de qualidade de contexto são usados para qualificar medições de poluição e também a localização por GPS:

1. *Incerteza* (*Uncertainty*): permite quantificar dúvida sobre o resultado de uma medição. Esse critério (também chamado de *margem de erro*) é um intervalo expresso na mesma unidade da medição.

2. *Precisão* (*Precision*): qualifica o quão perto ou o quão repetível os resultados de

uma medição são. Precisão é tipicamente expressa em porcentagem.

3. *Taxa de atualização* (*Reflesh rate* ou *Information fleshness*): é o tempo despendido desde a determinação da informação de contexto atá a sua entrega ao cliente. Esse critério é expresso usando uma unidade de tempo, e.g., segundo, minuto, etc.

4. *Resolução espacial* (*Spatial Resolution*): mede a precisão com a qual uma área física é expressa. Resolução espacial é um importante indicador de QoC para proteger a privacidade do usuário, já que informações mais específicas (com maior resolução espacial) necessitam de maiores cuidados em relação a privacidade do usuário. Como a resolução espacial mede uma distância, uma unidade de distância (e.g., metros, quilômetros, etc.) é usado para expressar esse critério.

A resolução espacial é usada para qualificar as localizações dos ônibus e das paradas de ônibus. A localização e as medições de poluição são qualificadas com um indicador de *taxa de atualização*. Já as medições de poluição providas pelos ônibus são qualificadas através do critério de *precisão*, enquanto as paradas de ônibus usam a *incerteza* para qualificar suas medições de poluição.

Depois de medir a quantidade de poluentes do ar, os dados são transmitidos para a EcoDiF, que realiza o armazenamento e o gerenciamento desses dados. Entretanto, os clientes do EcoDiF precisam buscar e selecionar os recursos necessários para prover suas funcionalidades. Para fazer isso, os dados e as características dos sensores são descritas através do modelo de informação do *QoDisco*.

O restante dessa seção é estruturado como segue: a Seção 4.1.1 ilustra o uso do modelo de informação do *QoDisco* nesse cenário, apresentando a modelagem de recursos, dados, critérios de QoC, e indicadores de QoC. A Seção 4.1.2 mostra como o processo de busca pode ser realizado usando a API e a interface Web do *QoDisco*. Finalmente, a Seção 4.1.3 foca na definição de duas aplicações de IoT cientes de contexto que podem ser utilizadas nesse cenário.

4.1.1 Modelo de informação do *QoDisco* no cenário de poluição de ar

Para ilustrar o uso do modelo de informação do *QoDisco* nesse cenário, o Trecho de Código 4.1 apresenta uma descrição OWL, escrita no formato Turtle, de um detector de poluição do ar (*Sensor01*) que pode ser anexado a um ônibus. Esse dispositivo é uma instância de um *SensingDevice*, um conceito modelado na ontologia

SSN (aqui indicado usando o prefixo SSN). Além disso, esse dispositivo monitora quatro propriedades, definidas através da ontologia de domínio de *poluição* prefixada pela URL `http://consiste.dimap.ufrn.br/ontologies/qodisco/pollution#`, nomeadamente: *carbon_ monoxide*, *nitrous_ oxide*, *ozone*, e *sulfur_ dioxide*. Enquanto isso, o ônibus associado a esse dispositivo é modelado como uma *estrutura física* usando o modelo de informação do *QoDisco*.

Listing 4.1: Descrição do recurso *Sensor01*.

```
@prefix ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
@prefix DUL: <http://www.loa-cnr.it/ontologies/DUL.owl#>
@prefix pollution: <http://consiste.dimap.ufrn.br/ontologies/qodisco/pollution#>

:Sensor01
      a                    ssn:SensingDevice ;
      ssn:observes         pollution:carbon_monoxide ;
      ssn:observes         pollution:nitrous_oxide ;
      ssn:observes         pollution:ozone ;
      ssn:observes         pollution:sulfur_dioxide ;
      DUL:hasLocation      :Bus01 .
```

Cada dado produzido por um sensor pode ser anotado com metadados de QoC. O Trecho de Código 4.2 detalha uma observação feita pelo *Sensor01*. A observação realizada pelo *Sensor01* (*observedBy*) é descrita como uma saída produzida pelo sensor (*observationResult*), a propriedade observada (*observedProperty*), e o momento no tempo em que a observação foi obtida (*observationResultTime*).

Listing 4.2: Observação feita pelo *Sensor01*.

```
@prefix ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
@prefix pollution: <http://consiste.dimap.ufrn.br/ontologies/qodisco/pollution#>

:Observation01
      a                            ssn:Observation ;
      ssn:observationResult        :observation01_output ;
      ssn:observedProperty         pollution:carbon_monoxide ;
      ssn:observedBy               :Sensor01 ;
      ssn:observationResultTime    <2015-10-22T16:00:24.000-03:00> .
```

O Trecho de Código 4.3 apresenta a descrição da saída do sensor. Além da especificação de uma referência para o valor da observação (*hasValue*), a descrição da saída do sensor também inclui uma referência para o indicador de QoC da observação, como mostra o exemplo apresentado no trecho de código 4.

Listing 4.3: Saída do sensor *Sensor01*.

```
@prefix ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
@prefix qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>
```

```
3
4 :observation01_output
5       a                       ssn:SensorOutput ;
6       qodisco:has_qoc         :observation01_qocindicator ;
7       qodisco:hasValue        :observation01_resultvalue ;
8       ssn:isProducedBy        :Sensor01 .
```

O Trecho de Código 4.4 ilustra um indicador de QoC representado através de uma referência para o critério *Precision* e um valor inteiro, que é expresso como um valor percentual entre 0 e 100 (90% nesse caso).

Listing 4.4: Indicador de QoC para a observação *Observation01*.

```
1 @prefix ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
2 @prefix qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>
3
4 :observation01_qocindicator
5       a                           qodisco:QocIndicator ;
6       qodisco:has_qoc_criterion   :Precision ;
7       qodisco:has_qoc_value       90 .
```

Uma das funcionalidades providas pelo *QoDisco* é permitir a especificação de novos critérios de QoC usando a sua interface Web. O Trecho de Código 4.5 mostra a definição de um novo critério de QoC usando a API do *QoDisco*. Para definir um novo critério de QoC, é necessário especificar os seguintes atributos: (i) *name*, um identificador para o critério; (ii) *description*, uma descrição textual informal do que o critério expressa; (iii) *direction*, compara diferentes valores de QoC pelo ponto de vista do consumidor da informação de contexto; (iv) *unit*, a unidade do valor produzido; (v) *minimum value*, o mínimo valor permitido; (vi) *maximum value*, o máximo valor permitido, e (vii) *composed*, expressa quais os outros critérios de QoC são necessários para a sua especificação (se houver algum). Nota-se ainda nesse Trecho de Código, especificamente na linha 12, a utilização de uma outra ontologia para especificar que o critério de precisão é medido usando valores em porcentagem, a UCUM (*Units of measurement ontology*), ontologia usada para a especificação de unidades de medição.

Listing 4.5: Critério de QoC *Precision*.

```
1 @prefix ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
2 @prefix qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>
3
4 :Precision
5       a                           qodisco:QoC ;
6       qodisco:description         "Qualifica o quao perto ou o quao repetivel os
            resultados de uma medicao sao" ;
7       qodisco:direction           "INF" ;
8       qodisco:invariant           false ;
```

```
9       qodisco:maximum_value       100 ;
10      qodisco:minimum_value       1 ;
11      qodisco:name                "Precision" ;
12      qodisco:unit                <http://purl.oclc.org/NET/muo/ucum/unit/fraction/
            percent> ;
```

4.1.2 *Queries* de busca

Queries de busca podem ser criadas usando a linguagem SPARQL para permitir a descoberta de recursos usando dados de localização, tipo, propriedades observadas, e etc. O Trecho de Código 4.6 apresenta uma *query* SPARQL que permite a busca por todas as medições de ozônio obtidas na Rua Djalma Maranhão com precisão maior ou igual a 90 porcento. Vale ressaltar que o modelo de informação do *QoDisco* permite a definição de localizações usando coordenadas geográficas, tais como latitude e longitude, e usando nomes de *estruturas físicas*, tais como nomes de ruas e prédios. É possível registar novos nomes de localização usando a API e a interface Web do *QoDisco*. Além disso, o usuário pode também definir novas ontologias de domínio contendo um conjunto pré-definido de localizações.

Listing 4.6: *Query* SPARQL.

```
1 SELECT * WHERE {
2    ?observation a ssn:Observation ;
3                   ssn:ObservedProperty pol:Ozone }
4    FILTER ( dul:hasLocation pol:Djalma_Maranhao_Street &&
5             qodisco:has_qoc_criterion qodisco:Precision &&
6             qodisco:has_qoc_value > 80)
7 }
```

Alternativamente, a *query* descrita no Trecho de Código 4.6 pode também ser especificada usando a interface Web do *QoDisco*, como mostra a Figura 34. A interface Web do *QoDisco* permite a busca de observações e recursos por suas propriedades observadas, localização, e critérios de QoC. Cada atributo é especificado usando uma caixa de seleção contendo um conjunto de valores apropriados para ela. Nesse cenário, nós registramos um conjunto de valores para propriedades observadas (*carbon monoxide*, *nitrous oxide*, *ozone*, e *sulfur dioxide*) e critérios de QoC (*precision*, *uncertainty*, *reflesh rate*, *spatial resolution*) na plataforma. Vale ressaltar que a interface Web do *QoDisco* permite ainda a busca através da seleção de pontos em um mapa e através de um editor de texto, como mostra a Seção 3.5.

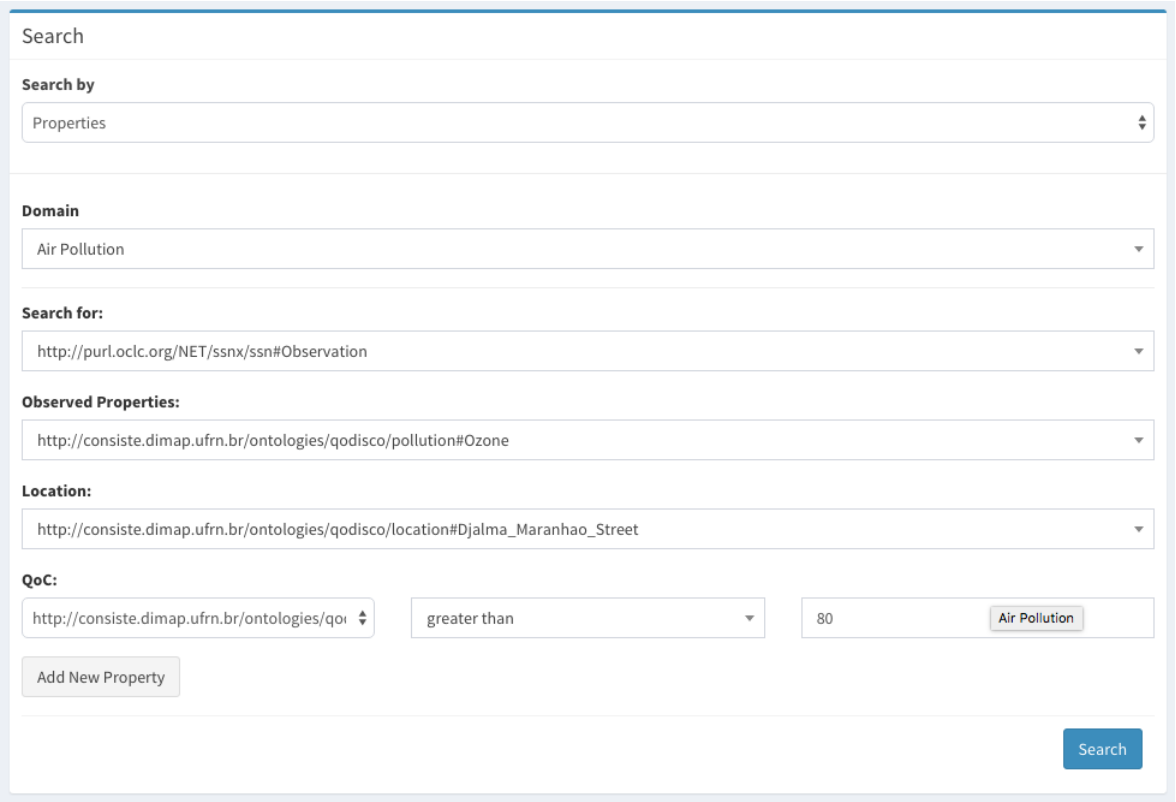


Figura 34: Buscando registros usando a interface Web do *QoDisco*.

4.1.3 Aplicações

Nesse cenário, considere duas diferentes aplicações cientes de contexto para IoT que se registram para receber informação de contexto e metadados de QoC do *QoDisco*. A aplicação A é uma aplicação de cuidados de saúde que visa informar os cidadãos e turistas sobre o nível de poluição atual nas ruas, enquanto a aplicação B é um planejador de rotas. A Figura 35 ilustra o cenário apresentado nesta seção, incluindo essas duas aplicações ciente de contexto.

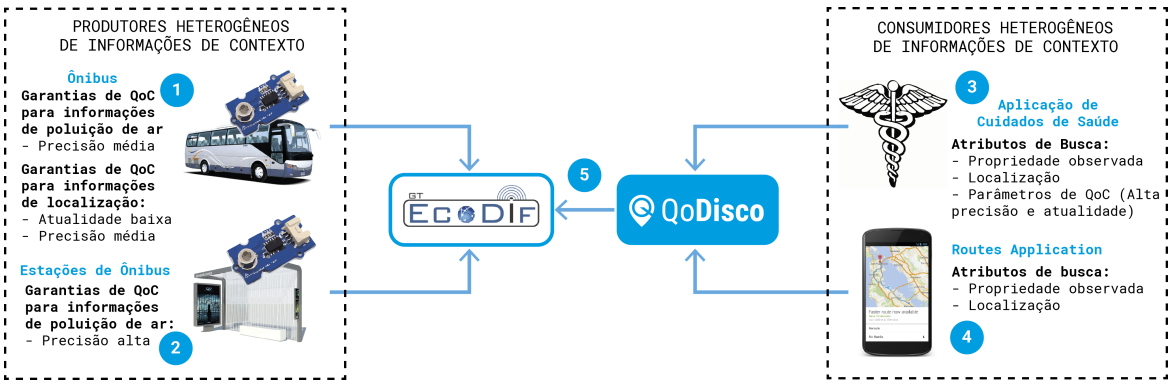


Figura 35: Aplicações do cenário de poluição.

Como mostra a Figura 35, ônibus (1) provêm suas localizações com, no mínimo,

pouca atualidade (freshness) e precisão média, e as medições de poluição com, no mínimo, atualidade média. Já as paradas de ônibus (2) comprometem-se a prover suas medições com precisão alta. Essas medições são enviadas a uma plataforma de *middleware* que gerencia informação de contexto, semanticamente anotando esses dados de acordo com o modelo de informação do *QoDisco*. Depois de anotados, a plataforma armazena os dados em um repositório acessível pelo *QoDisco* através de requisições HTTP contento *queries* SPARQL. Aplicações cientes de contexto irão requerer informação sobre níveis de poluição das ruas da cidade, provendo seus requisitos para o *QoDisco* (5) através de *queries* SPARQL, enquanto o *QoDisco* oferece informação de contexto fácil de entender e de usar.

A aplicação A (3) requer medições de poluição (tais como os níveis de monóxido de carbono e de ozônio) das ruas da cidade com alta atualidade e alta precisão. Essa aplicação usa as medições de poluição para fornecer aos seus usuários informações de alto nível na forma de *Air Quality Index* (AQI), que é dividida em seis níveis simbólicos: *good*, *moderate*, *unhealthy for sensitive groups*, *unhealthy*, *very unhealthy*, e *hazardous*. Os metadados de QoC usados para qualificar o nível de poluição são representados pelo critério de QoC precisão, usado para caracterizar a probabilidade que uma instância de contexto representa fielmente uma situação correspondente do mundo real, como aferida pela fonte de contexto, no momento em que ela foi determinada. Esse critério tem três valores simbólicos, nomeadamente *low*, *medium*, e *high*. O valor *low* implica que o nível de poluição medido não é confiável, o valor *high* implica que a medição é bastante confiável, e o valor medium representa um meio termo entre os valores *low* e *high*. Quando a aplicação de cuidados de saúde não recebe informação de contexto com o QoC esperado (alta precisão e atualidade), a aplicação irá parar seus serviços e exibir uma mensagem para indicar que ela não tem informação suficiente para prover seus serviços, evitando problemas que podem ocorrer ao se fornecer informações sem um nível adequado de qualidade de contexto. Por exemplo, a aplicação de cuidados de saúde pode ser crítica para pessoas asmáticas, por isso é preferível não prover quaisquer serviços ao invés de prover serviços com indicações errôneas. Como tais informações podem surgir em qualquer momento durante a execução da aplicação, essa aplicação se subscreve no *QoDisco* para receber novas informações de contexto com o indicador de QoC esperado à medida que elas cheguem.

A aplicação B (4) busca por informação de contexto com alta atualidade usando o *QoDisco* (5). Essa aplicação ajuda pedestres e ciclistas a traçar uma rota de um ponto a outro da cidade. Ao contrário da aplicação A, essa aplicação irá prover rotas para o usuário mesmo que as informações de poluição não tenham uma precisão alta ou quando

essas informações estejam indisponíveis. Além disso, essa aplicação precisa somente buscar dados de forma síncrona usando a API do *QoDisco*. Isso acontece porque a aplicação precisa somente da busca por dados de poluição imediatamente após a requisição do usuário. Como a aplicação não é crítica do ponto de vista do usuário, medições de poluição com alta precisão não são estritamente necessárias: se a aplicação não receber informação com o QoC esperado, ela irá continuar a prover seus serviços, mas um aviso será exibido indicando aos usuários que a informação está momentaneamente menos confiável.

Como exemplo, um ciclista em Natal, Brasil, que queira ir do Instituto Metrópole Digital (IMD) para o *shopping* Midway Mall irá ter três opções de rotas, (i) pela rua Djalma Maranhão, (ii) pela avenida Rui Barbosa, ou (iii) pela avenida Capitão-Mor Gouveia. A aplicação irá exibir essas rotas, incluindo seus tamanhos (respectivamente, 4.4 km, 5.0 km, 5.6 km) e também as medições de poluição das ruas compondo as rotas. Similarmente a aplicação A, a aplicação B expressa informação sobre poluição usando valores de AQI. Com ambas opções, o usuário será capaz de escolher a melhor rota com base em suas necessidades.

4.2 Configuração dos experimentos

Visando minimizar a influência da rede, a maioria dos experimentos foram conduzidos usando um mesmo computador com processador Intel(R) Core(TM) i5, 8GB de RAM, e sistema operacional Mac OS 10.11.3 de 64-bit. Esse computador foi usando para implantar os repositórios compondo a plataforma do *QoDisco* em uma máquina virtual com 1GB de RAM rodando o sistema operacional Linux Ubuntu Server 14.04.3 de 64-bit. Cada repositório foi implantado em um *container* Linux usando a tecnologia Docker[1]. Ao contrário de uma máquina virtual rodando um sistema operacional completo, um *container* possui mínimas configurações e roda aparte de outros *containers*. Por implementar os repositórios em *containers* Linux, nós fomos capazes de simular um cenário real no qual cada repositório executa de forma independente dos outros e também diminuir o esforço computacional para rodar esses repositórios. Para assegurar a confiabilidade desses repositórios, os experimentos foram executados 20 vezes.

Para descobrir recursos usando a sua localização, tipo, ou qualquer uma de suas propriedades no *QoDisco*, *queries* para busca semântica podem ser criadas usando a linguagem SPARQL ou a interface Web da plataforma. Em todos os experimentos, nós

[1]Docker – `https://www.docker.com/`

calculamos o tempo gasto pelo *QoDisco* para a busca usando uma *query* SPARQL que busca por todas as observações relacionadas com níveis de poluição de ozônio registradas no *QoDisco* com um indicador de QoC de precisão maior ou igual a 95%. A *query* usada nos experimentos é apresentada no Trecho de Código 4.7.

Listing 4.7: *Query* SPARQL usada nos experimentos.

```
1 PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
2 PREFIX qodisco: <http://consiste.dimap.ufrn.br/ontologies/qodisco#>
3 PREFIX dul: <http://www.loa-cnr.it/ontologies/DUL.owl#>
4 PREFIX pol: <http://consiste.dimap.ufrn.br/ontologies/qodisco/ pollution#>
5 PREFIX context: <http://consiste.dimap.ufrn.br/ontologies/qodisco/context#>
6
7 SELECT * WHERE {
8    ?observation a ssn:Observation ;
9          ssn:observedProperty pol:Ozone ;
10         ssn:observationResult ?observationResult .
11
12   ?observationResult qodisco:has_qoc ?qoc .
13
14   ?qoc qodisco:has_qoc_criterion context:Precision ;
15         qodisco:has_qoc_value ?value .
16
17   FILTER (?value >= 95)
18 }
```

4.3 Resultados

Para calcular o impacto da quantidade de dados registrados em um repositório, foi inserido um conjunto de observações simuladas em cinco repositórios que, por sua vez, foram associados ao domínio de poluição de ar. Depois disso, nós medimos o tempo gasto pelo *QoDisco* para responder a *query* SPARQL realizada pelo cliente à medida que nós aumentamos o número de observações. A Tabela 5 mostra o tempo mínimo, médio, máximo, e o desvio padrão (em milissegundos) para responder a *query* de busca à medida que nós aumentamos o número de observações contidas nesses repositórios. É possível observar que o *QoDisco* realiza a busca síncrona de dados anotados semanticamente em menos de um segundo, até mesmo para 100.000 observações. Entretanto, é importante ressaltar que diferentes *queries* SPARQL podem influenciar os resultados obtidos, já que diferentes *queries* podem retornar diferentes resultados.

Já para calcular o impacto da quantidade de repositórios registrados, foi inserido um conjunto de repositórios no *QoDisco*, cada um com 1000 observações simuladas relacionadas com níveis de poluição de ozônio. Depois disso, nós medimos o tempo despendido pelo

Tabela 5: Tempo gasto pelo *QoDisco* (em milissegundos) para responder a uma *query* de busca à medida que o número de observações aumenta.

Observações	**Mínimo**	**Máximo**	**Média**	**Desvio Padrão**
100	80	192	109	28.23
250	76	244	138	54.46
500	85	419	205	68.95
1,000	87	891	216	229.18
2,500	131	694	225	118.79
5,000	225	419	283	43.37
10,000	239	445	305	49.54
25,000	533	768	607	65.40
50,000	881	1184	997	72.26
100,000	1446	1997	1579	134.64

QoDisco para responder a *query* SPARQL do cliente à medida que nós aumentamos o número de repositórios disponíveis. A Tabela 6 mostra o tempo mínimo, médio, máximo e o desvio padrão (em milissegundos) para responder a *query* de busca à medida que nós aumentamos o número de repositórios registrados. Nota-se que, até mesmo com 20 repositórios, o desempenho do *QoDisco* não ultrapassou dois segundos em média. É importante mencionar que o aumento do número de repositórios em outros domínios (nesse caso, repositórios não relacionados a medições de poluição de ar) não impacta o tempo de resposta de uma requisição de busca, já que eles não serão nem ao menos consultados devido a nossa estratégia de dividir repositórios em domínios.

Já para a busca assíncrona, foi calculado o tempo gasto pelo *QoDisco* para notificar o cliente quando um novo registro que atende a *query* de busca é inserido em um repositório através da API do *QoDisco*. Para avaliar a performance do *QoDisco* para a busca assíncrona, nós consideramos cinco repositórios associados com o domínio de poluição de ar, cada um com 1000 observações de níveis de ozônio simulados. Depois, nós medimos o tempo gasto pelo *QoDisco* para responder as buscas realizadas ao aumentar o número de clientes interessados em notificações sobre novas medições de poluição de ozônio maiores ou iguais a 95%. Tais requisições simultâneas foram realizadas através do

Tabela 6: Tempo gasto pelo *QoDisco* (em milissegundos) para responder a uma *query* de busca à medida que o número de repositórios, cada um com 1000 observações registradas, aumenta.

Repositórios	**Mínimo**	**Máximo**	**Média**	**Desvio Padrão**
3	415	886	795	80.71
5	533	1211	875	217.11
8	673	1243	991	91.96
12	799	1489	1244	198.21
20	1052	1944	1311	321.33

uso do *Apache JMeter*[2], uma ferramenta de código aberto baseada em Java para a análise e medição do desempenho de serviços Web. A Tabela 7 mostra o tempo mínimo, médio e máximo (em milissegundos) para notificar o cliente. Vale ressaltar que os clientes irão receber somente os novos recursos registrados que atendam a sua *query* inicial. Devido o resultado retornado pelo processo de busca assíncrona ser tipicamente bem menor do que o da busca síncrona (onde todos os recursos registrados em um domínio que atendam à *inicial* do cliente serão retornados pelo *QoDisco* de uma única vez), o processo de notificação foi realizado, quase que em sua totalidade, em menos de 1 segundo. Além disso, tal tempo de resposta depende altamente do *broker publish-subscribe* responsável pela troca de mensagens assíncrona. No protótipo atual do *QoDisco*, foi usado o Mosquitto, um *broker publish-subscribe* para IoT (ver Seção 3.5).

Por fim, a IoT irá exigir uma rede que seja capaz de lidar com um grande número de requisições de forma rápida. De fato, não há Internet das Coisas sem rede. Por isso, é importante considerar a influência da rede no desempenho de serviços de busca como o *QoDisco*. Nesse sentido, foram realizados experimentos a fim de se avaliar o desempenho do *QoDisco* através de simulações de rede. Para essas simulações foi usado uma ferramenta chamada *Dummynet* (CARBONE; RIZZO, 2010). Dummynet é uma ferramenta de emulação de rede que permite o controle sob diferentes aspectos do tráfico de rede, tais como o gerenciamento da banda de rede, latência e qualidade da conexão. A Tabela 8 apresenta o tempo mínimo, máximo, e a média (em milissegundos) para a busca síncrona à medida que nós aumentamos o número de observações contidas em cinco repositórios. Os resultados apresentados nessa tabela foram obtidos ao se implantar o *QoDisco* em uma rede simulada

[2] *Apache JMeter*: `http://jmeter.apache.org`

Tabela 7: Tempo gasto pelo *QoDisco* (em milissegundos) para notificar clientes assincronamente.

Clientes	**Mínimo**	**Máximo**	**Média**
1	107	280	186.31
5	126	388	198.17
10	156	323	226.8
25	129	405	293.81
50	493	854	652.04
75	617	1025	797.13
100	623	1301	905.98

com o Dummynet que apresenta um *delay* de $50ms$, perda de pacotes de 0.01% e banda de rede de 1Mb/s.

Tabela 8: Tempo gasto pelo *QoDisco* (em milissegundos) para responder a uma *query* de busca em uma rede simulada à medida que o número de observações aumenta.

Observações	**Mínimo**	**Máximo**	**Média**
100	106	152	123
250	410	417	414
500	751	1610	839
1,000	1434	1924	1473
2,500	3510	3566	3524
5,000	6945	7916	7038
10,000	13843	15238	13979
25,000	34618	37296	34802
50,000	69323	73430	69846
100,000	116241	130417	117922

5 Trabalhos Relacionados

Serviços de descoberta representam um tópico bem estudado em sistemas distribuídos e serviços Web, por isso propostas existentes nessas duas áreas podem servir de inspiração para o desenvolvimento de serviços de descoberta para IoT. Entretanto, a maioria dessas propostas não podem ser diretamente aplicadas a IoT, já que elas não atendem alguns dos requisitos presentes nesse contexto (LI; ZAMAN; LI, 2015) (WEI; JIN, 2012), tais como: (i) restrições de recurso relacionadas com capacidades de computação, armazenamento, e comunicação de dispositivos; (ii) heterogeneidade dos dispositivos, o que aumenta a complexidade do processo de descoberta, e (iii) a alta volatilidade das conexões dos dispositivos. Devido a essas limitações, novos serviços de descoberta tem sido recentemente propostos para o contexto de IoT.

A Tabela 9 sumariza algumas propostas de serviços de descoberta existentes na literatura em relação à satisfação dos requisitos vistos na Seção 3.1. Nessa tabela, o símbolo ✓ denota que o requisito é completamente atendido, o símbolo ◯ indica que o requisito é parcialmente atendido, e o símbolo × indica que o requisito não é atendido. O restante deste capítulo fornece uma visão geral dessas propostas e discute aspectos que as diferenciam do *QoDisco* no tocante a: (i) modelagem de recursos e informação de contexto em IoT, e; (ii) aspectos arquiteturais.

5.1 Modelagem de recursos e informação de contexto em IoT

Atualmente, várias propostas de serviços de descoberta tem emergido no campo de IoT. Por exemplo, Khodadadi *et al.* (KHODADADI; DASTJERDI; BUYYA, 2015) propõem um *framework* para a definição, descoberta e composição de coisas e seus respectivos serviços. Esse *framework* define coisas, incluindo suas respectivas propriedades, e serviços através de um meta-modelo chamado *Thing Description Document* (TDD). Um TDD é um arquivo JSON que consiste de duas partes principais: (i) propriedades da entidade, e (ii) serviços

Tabela 9: Comparação do *QoDisco* com outras propostas de serviços de descoberta para IoT.

Serviço de Descoberta	Esquema de Identificação Flexível	Consultas com múltiplos atributos	Consultas em intervalos	Múltiplos publicadores	APIs de gerenciamento	Descrição semântica de recursos	Modelagem de critérios de QoC	Requisições síncronas e assíncronas
QoDisco	○	✓	✓	✓	✓	✓	✓	✓
Khodadadi *et al.*	○	✓	×	✓	✓	×	×	○
Dey *et al.*	○	✓	✓	✓	✓	○	×	×
Chun *et al.*	○	✓	○	×	✓	○	×	○
Jayaraman *et al.*	○	✓	✓	✓	✓	×	×	○
Perera *et al.*	○	✓	✓	✓	✓	✓	×	○
Li *et al.*	○	✓	✓	✓	✓	○	×	○
Cirani *et al.*	○	×	×	×	✓	×	×	○
Paganelli e Parlanti	○	✓	✓	✓	✓	✓	×	○

oferecidos pela entidade. Somente três atributos são obrigatórios para a descrição de uma entidade: um nome, a data da última modificação de uma entidade, e a localização da entidade. O usuário pode definir outros atributos enquanto eles estiverem em conformidade com as regras de validação do JSON. Quanto aos serviços oferecidos pela entidade, é necessário definir a API que é disponibilizada pela entidade sendo descrita. Entretanto, esse *framework* não propõe nenhum meio para definir semanticamente uma entidade e, apesar da simplicidade oferecida por arquivos JSON, a especificação das propriedades da entidade não seguem nenhum tipo de estratégia semântica ou padronização, levantando problemas de ambiguidade e prejudicando a composição de serviços em tempo de execução.

Dey *et al.* (DEY et al., 2014) propõem uma interface de usuário para facilitar o processo de modelagem de sensores. Uma vez que os metadados de um sensor são registrados no *framework*, eles podem ser obtidos usando consultas SPARQL através de uma interface de usuário. Um usuário pode também buscar por sensores ao desenhar regiões limites em um mapa e encontrar recursos baseado nessas regiões. Como a nossa proposta, eles

argumentam que tal interface de usuário desempenha um papel chave no aumento da adoção da abordagem, porque escrever descrições semânticas manualmente é uma tarefa tediosa, propensa a erros e que pode consumir bastante tempo. Entretanto, o *framework* proposto não apresenta uma maneira de modelar dados de sensores e os mecanismos de busca por localização são baseados somente em valores absolutos, tais como latitude e longitude, não oferecendo maneiras de associar conceitos de alto-nível para localização de recursos.

Chun *et al.* (CHUN et al., 2015) apresenta um serviço de descoberta que compreende um modelo de informação baseado em ontologias para suportar a descrição semântica de três componentes: entidades de interesse, recursos, e serviços. Para refletir a dinamicidade desses componentes de IoT, o modelo de informação proposto em Chun *et al.* inclui um atributo que indica se a descrição do componente é estática ou dinâmica. Uma descrição indicada como estática não tem os valores de seus atributos alterados com o tempo, e.g., a localização de uma entidade ou recurso pode ser estática. Uma descrição dinâmica indica que os valores de seus atributos variam frequentemente ou de forma indeterminada. Se um componente é ajustado como dinâmico, uma nova relação entre componentes pode ser criada ou uma relação existente pode ser removida. Por exemplo, no caso de uma entidade dinâmica de IoT, os valores de atributos da descrição mudam ao decorrer do tempo, ao contrário da descrição estática. Além disso, para definir uma interação entre componentes de IoT e o diretório que armazena a descrição desses componentes, é proposto a utilização de dois valores adicionais no modelo de informação: *passivo* e *ativo*. Uma componente *passivo* indica que o diretório deve enviar uma requisição para adquirir informação atualizada. Já um componente ativo indica que um componente de IoT encaminha metadados atualizados para o diretório sempre que seus metadados mudam. Porém, ao contrário do *QoDisco*, o modelo de informações proposto em Chun *et al.* não estende ontologias existentes na literatura. De fato, o reuso de ontologias padronizadas na literatura facilita a evolução, extensão e integração com outras ontologias, além de facilitar a adotação de uma ontologia.

Também é importante mencionar a plataforma *OpenIoT* [1] e suas contribuições para o processo de descoberta de recursos no campo de IoT. A OpenIoT é uma plataforma de código aberto para IoT que provê uma arquitetura flexível baseada em nuvem que ajuda no gerenciamento do ciclo de vida de serviços de IoT e aplicações. A arquitetura da OpenIoT inclui, entre outros, módulos que gerenciam a aquisição de dados de sensores, chamada de X-GSN, a provisão de dados semânticos e a sua consulta (*Linked Sensor Middleware-*

[1] OpenIoT: `http://www.openiot.eu/`

Light, chamada de *LSM-Light*), e também ferramentas de *front-ent* para a descoberta de dados e análise (e.g., *Request Definition*, e *Request Presentation*). A integração de todos esses módulos é possível graças ao uso da ontologia da OpenIoT, que é baseada na ontologia SSN. Considerando que usuários de plataformas de IoT não são usualmente bem versados em ontologias, Jayaraman *et al.* (JAYARAMAN; CALBIMONTE; QUOC, 2015) propõem um editor gráfico de descrição de sensores para a geração e automatização das anotações semânticas e do processo de registro de sensores. O editor prover meios para (i) definir/modificar novos tipos de sensores e (ii) criar novas instâncias de sensores. As extensões na ontologia gerada pela criação de novos tipos de sensores são associadas dinamicamente a ontologia da OpenIoT usando o componente LSM-Light. Entretanto, a ontologia da OpenIoT não dá suporte a modelagem de atuadores e a anotação de dados de sensores com informações de contexto.

Realmente, apesar da existência de várias propostas para serviços de descoberta em IoT, nós notamos que não há muito esforço empregado na descoberta de recursos baseada em informações de contexto. Um dos poucos exemplos de propostas existentes é o CASSARAM (PERERA et al., 2013a), um *framework* que usa informação de contexto (e.g., confiabilidade e acurácia) para automaticamente buscar, selecionar, e classificar os sensores pertencentes a um grande conjunto de dispositivos disponíveis de acordo com critérios e prioridades definidos pelo usuário. Similarmente ao *QoDisco*, o CASSARAM é construído sobre um modelo de informação baseado em ontologias que compreende a ontologia SSN e a linguagem SPARQL. O mecanismo *framework* LOCA (LI; ZAMAN; LI, 2015) também considera informação de contexto em seu modelo de informação, porém a modela através de uma ontologia leve, completamente nova e independente de outras ontologias existentes. Além disso, apesar desses trabalhos levarem em conta informação de contexto, eles não consideram critérios de QoC no processo de descoberta, o que é fundamental para prover resultados mais precisos de acordo com os requisitos de usuários e aplicações. De fato, nós não encontramos nenhuma proposta na literatura endereçando o uso de critérios e indicadores de QoC em serviços de descoberta para IoT, o que constitui uma contribuição importante do nosso trabalho.

5.2 Aspectos Arquiteturais

Um aspecto importante para ser considerado com respeito aos trabalhos mencionados na Seção 5.1 é o uso de abordagens centralizadas e descentralizadas. Uma limitação importante do CASSARAM e da proposta de Chun *et al.* é o uso de um repositório único

centralizado, que não é apropriado para IoT devido aos já conhecidos pontos negativos relacionados a escalabilidade, tolerância a falhas, e ameaças de segurança. Como relatado no estudo comparativo conduzido por Evdokimov *et al.* (EVDOKIMOV et al., 2010), arquiteturas decentralizadas tais como as apresentadas pelo LOCA e pelo *QoDisco* são mais apropriadas para IoT, já que elas cumprem requisitos importantes, tais como tolerância a falhas e escalabilidade.

Cirani *et al.* (CIRANI et al., 2014) apresentam uma arquitetura federada para descoberta de serviços em IoT. Tal arquitetura federada é estruturada sobre redes *peer-to-peer* (P2P), que exibem características fundamentais a sistemas distribuídos, como escalabilidade, eficiência, robustez, e balanço de carga. Informações sobre recursos providas por objetos anexados a uma rede *wireless* local são obtidas por um nó fronteira especial, chamado de *IoT Gateway*, que também é parte da rede P2P e é usado para armazenar e recuperar essas informações dos objetos, resultando em um diretório de recursos escalável e distribuído. Ademais, redes P2P podem ter um valor significante para IoT desde que são capazes de lidar com entrada e saída de *peers*, assim se adequando a dinamicidade inerente de cenários de IoT. Vale ressaltar que a proposta de uma arquitetura para o *QoDisco* baseada em redes P2P será analisada no futuro.

Por fim, Paganelli e Parlanti (PAGANELLI; PARLANTI, 2012) também propõem um serviço de descoberta distribuído baseado em redes P2P para cenários de IoT. Além disso, esse trabalho apresenta um conjunto de requisitos comuns para serviços de descoberta no contexto de IoT, incluindo o suporte a *queries* com múltiplos atributos e de intervalo. Esses requisitos nortearam a proposta inicial do *QoDisco* que, além de satisfazê-los, propõe novos requisitos, que são descritos na Seção 3.1.

6 Considerações Finais

Devido a dinamicidade e heterogeneidade intrínseca dos recursos de IoT, descobrir recursos entre a miríade de entidades existentes conectadas a Internet constitui um desafio importante para o desenvolvimento de sistemas de IoT. Por isso, é fundamental prover a clientes (e.g., plataformas de *middleware*, usuários finais, e aplicações) uma forma de expressar, sem ambiguidade, os seus recursos, além de uma forma de recuperar essas descrições facilmente. Além disso, complementar essas descrições com informações de contexto permite aumentar a riqueza do processo de busca e suportar o processo de tomada de decisão. No entanto, a natureza inerentemente imperfeita das informações de contexto requer a consideração de conhecimento adicional em termos da qualidade dessa informação, i.e., critérios de QoC.

Nessa dissertação, esses aspectos foram endereçados através da proposta do *QoDisco*, um serviço de descoberta ciente de QoC para IoT. O *QoDisco* suporta a descoberta de recursos através de múltiplos atributos, consultas em intervalo, operações síncronas e assíncronas, e critérios de QoC. Ademais, o *QoDisco* envolve um modelo de informação que se beneficia de ontologias bem estabelecidas e padronizadas para descrever semanticamente recursos (sensores e atuadores) e os serviços que eles expõem. Ao contrário de arquiteturas centralizadas comumente encontradas na literatura, a arquitetura distribuída do *QoDisco* é capaz de lidar com alguns requisitos importantes em IoT, como escalabilidade, tolerância a falhas, e alta-disponibilidade. Além disso, a definição de um modelo de informação consistente é altamente relevante, já que anotações semânticas podem ser usadas para descrever recursos e especificar *queries* de busca.

6.1 Contribuições

Entre as contribuições do *QoDisco* apresentadas nesta dissertação destacam-se:

(i) a proposição, especificação, e implementação do *QoDisco*, possibilitando a descrição,

gerenciamento e busca síncrona e assíncrona de recursos providos por diferentes clientes.

(ii) a proposição de uma arquitetura que permita a busca de recursos de forma síncrona e assíncrona.

(iii) o desenvolvido de um modelo de informação para a especificação de recursos de IoT, os serviços que os mesmos expõem e a anotação de informações de contexto. Como descrito na Seção 4.1.1, o *QoDisco* emprega um conjunto de ontologias para a especificação de recursos e informações de contexto em IoT, todas descritas utilizando a linguagem OWL. Vale ressaltar ainda que esse modelo propõe a modelagem de informações de QoC através de uma ontologia, criada a partir do meta-modelo do QoCIM. De fato, a anotação de observações de sensores com informações de QoC é uma das maiores contribuições deste trabalho, já que não foi encontrado nenhuma proposta de serviço de descoberta na literatura que endereçasse esse tipo de informação.

(iv) o desenvolvimento de um ambiente gráfico que facilita a especificação de recursos de IoT, além da busca de recursos e dados capturados por sensores anotadas semanticamente.

(v) conforme apresentado no Capítulo 4 deste trabalho, o processo de descoberta realizado pelo *QoDisco* foi avaliado em um cenário de monitoramento de poluição de ar onde a utilização de um serviço de descoberta ciente de QoC é benéfico. Com base nesse cenário foi realizado dois experimentos a fim de verificar o desempenho do *QoDisco* para a busca de recursos de forma síncrona. Os resultados mostraram que o *QoDisco* é capaz de lidar com requisições de descoberta síncrona de recursos de forma eficiente, até mesmo para um grande número de dados registrados em um repositório ou para múltiplos repositórios distribuídos.

Além das contribuições listadas acima, o trabalho apresentado nesta dissertação apresenta uma série de requisitos para serviços de descoberta para IoT que são total ou parcialmente atendidos pelo *QoDisco* (ver Seção 3.1), são eles:

(i) *Esquema de Identificação Flexível*: O *QoDisco* compreende somente o uso de URIs para a identificação dos recursos e seus serviços associados. Por isso, esse requisito é só parcialmente atendido, já que, apesar do modelo de informação do *QoDisco* ser transparente em relação ao uso de URIs para identificação de recursos, ele não oferece flexibilidade de escolha para endereçamento de recursos.

(ii) *Consultas com múltiplos atributos e consultas em intervalos*: O uso da linguagem de consulta SPARQL, que só é possível graças ao uso das linguagens OWL e a RDF para modelagem de recursos, possibilita a criação de *queries* de busca contendo múltiplos atributos e/ou intervalos de valores.

(iii) *Múltiplos publicadores*: O *QoDisco* permite o gerenciamento de recursos em repositórios através de uma API (e de uma interface Web) que permite que múltiplos publicadores se autentiquem, publiquem e atualizem informações referentes a um mesmo recurso.

(iv) *APIs de gerenciamento*: O *QoDisco* oferece uma API contendo uma série de operações para gerenciamento de recursos, documentos RDO e repositórios.

(v) *Descrição semântica de recursos*: O *QoDisco* compreende um modelo de informações que permite a descrição semântica de recursos através de ontologias.

(vi) *Suporte a busca e anotação de dados capturados por sensores*: O modelo de informação do *QoDisco* compreende também conceitos para modelagem de dados de sensores. Já a API do *QoDisco* permite que esses dados sejam buscados através de *queries* SPARQL.

(vii) *Modelagem de informações de contexto e critérios de QoC*: O modelo de informação do *QoDisco*, através da ontologia SSN, permite a modelagem de informações de contexto. Já quanto aos critérios de QoC, o *QoDisco* implementa o meta-modelo do QoCIM através de ontologias, permitindo que critérios e indicadores de QoC sejam descritos.

(viii) *Suporte a busca por localização*: O modelo de informação do *QoDisco* compreende uma ontologia para modelagem de informações de localização, a SOUPA. Já a API do *QoDisco* permite a busca de recursos com base em dados de localização através da linguagem de consulta SPARQL, que permite a busca por geo-localização e por localização simbólica.

(ix) *Requisições síncronas e assíncronas* - O *QoDisco* oferece operações de busca síncrona e assíncrona de recursos através de sua API.

Por fim, vale ressaltar que pesquisas iniciais e resultados intermediários deste trabalho foram reportados na literatura, como mostrado a seguir:

MAIA, P. et al. On the Development of Systems-of-Systems based on the Internet of Things: A Systematic Mapping. In: *Proceedings of the 2014 European Conference on Software Architecture Workshops*. New York, NY, USA: ACM, 2014. (ECSAW '14), p. 23:1–23:8. ISBN 978-1-4503-2778-7. Disponível em: <http://doi.acm.org/10.1145/2642803.2642828>.

GOMES, P. et al. A Federated Discovery Service for the Internet of Things. In: *Proceedings of the 2Nd Workshop on Middleware for Context-Aware Applications in the IoT*. New York, NY, USA: ACM, 2015. (M4IoT 2015), p. 25–30. ISBN 978-1-4503-3731-1. Disponível em: <http://doi.acm.org/10.1145/2836127.2836129>.

6.2 Limitações

Em seu estado atual de desenvolvimento, o *QoDisco* apresenta algumas limitações, sendo as principais:

(i) A interface do usuário ainda é limitada no sentido de não fornecer mecanismos que auxiliem a criação de novas ontologias de domínio.

(ii) Apenas um estudo de caso foi considerado nesta dissertação, de modo que é necessário aumentar o grau de avaliação do *QoDisco* através da realização de outros experimentos com outras estudos de caso, além de serem feitas comparações quantitativas com outros abordagens apresentadas na literatura principalmente no tocante a serviços de descoberta distribuídos e cientes de contexto, visto que algumas das abordagens apresentadas na literatura são similares em alguns pontos à empregada neste trabalho (ver Seção 5).

(iii) Não consideramos requisitos de segurança e privacidade dos repositórios que compõem o *QoDisco*. De fato, lidar com esses requisitos é fundamental em serviços que manipulem informações de contexto.

(iv) Detecção e tolerância à falhas dos repositórios não foi tratada nesta dissertação. Além disso, não foram propostos mecanismos que garantam alta disponibilidade do *QoDisco*.

(v) O *QoDisco* não lida com consistência de ontologias e nem com consistência entre repositórios.

(vi) Considerando o imenso cenário de IoT, onde bilhões de dispositivos estão conectados a Internet e uma miríade de aplicações e usuários desejam interagir com esses recursos ao mesmo tempo, é importante verificar como o *QoDisco* se comporta quando requisições para a busca ou gerenciamento de recursos são lidadas em paralelo.

6.3 Trabalhos em andamento e futuros

Diante das limitações elencadas na Seção 6.2, diversos trabalhos futuros poderão ser desenvolvidos a partir deste trabalho, e alguns deles já estão em andamento. Por exemplo, já se encontra em andamento o desenvolvimento de um outro estudo de caso pertencente ao domínio de cuidados de saúde, que poderá ser utilizado em avaliações futuras do processo de busca e gerenciamento de recursos realizados pelo *QoDisco*.

Outro trabalho futuro a ser desenvolvido é a implementação de mecanismos de autorização e autenticação de repositórios. A ideia é permitir que cada repositório implemente os seus próprios mecanismos de segurança, mantendo assim a autonomia dos repositórios. Sendo assim, o *QoDisco* terá que empregar uma abordagem que lide com os diferentes mecanismos de segurança utilizados pelos repositórios.

Finalmente, é necessário a realização de novos experimentos para determinar o desempenho do *QoDisco* para requisições de busca assíncrona. A busca assíncrona de recursos é um ponto fundamental do *QoDisco* e, por isso, é fundamental a sua avaliação em estudos de caso futuros.

Referências

ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. London, UK, UK: Springer-Verlag, 1999. p. 304–307.

ARENAS, M.; GUTIERREZ, F.; PÉREZ, J. On the semantics of sparql. In: _____. *Semantic Web Information Management: A Model-Based Perspective*. [S.l.: s.n.], 2010. p. 281–307.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 54, n. 15, p. 2787–2805, out. 2010. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2010.05.010>.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 2, n. 4, p. 263–277, jun. 2007. ISSN 1743-8225.

BARNAGHI, P. et al. *Semantic Sensor Network XG Final Report*. Jun. 2011. `http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/`.

BASSI, A. et al. *Enabling Things to Talk*. [S.l.]: Springer-Verlag Berlin Heidelberg, 2013.

BREITMAN, K.; CASANOVA, M. A.; TRUSZKOWSKI, W. *Semantic Web: Concepts, Technologies and Applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 184628581X.

BRÖRING, A. et al. Advancing geoinformation science for a changing world. In: _____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. cap. The SID Creator: A Visual Approach for Integrating Sensors with the Sensor Web, p. 143–162. ISBN 978-3-642-19789-5.

BUCHHOLZ, T.; SCHIFFERS, M. Quality of context: What it is and why we need it. In: *10th Workshop of the HP OpenView University Association*. Geneva, Switzerland: [s.n.], 2003.

CARBONE, M.; RIZZO, L. Dummynet revisited. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 40, n. 2, p. 12–20, abr. 2010. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1764873.1764876>.

CHABRIDON, S. et al. Collaborative computing: Networking, applications, and worksharing: 11th international conference, collaboratecom 2015, wuhan, november 10-11, 2015, china. In: _____. Cham: Springer International Publishing, 2016. cap. A Framework for Multiscale-, QoC- and Privacy-aware Context Dissemination in the Internet of Things, p. 207–218. ISBN 978-3-319-28910-6.

CHEN, H.; FININ, T.; JOSHI, A. The SOUPA ontology for Pervasive Computing. In: TAMMA, V. et al. (Ed.). *Ontologies for agents: Theory and experiences*. Switzerland: Birkhäuser Basel, 2005, (Whitestein Series in Software Agent Technologies). p. 233–258.

CHUN, S. et al. Semantic description, discovery and integration for the internet of things. In: *Semantic Computing (ICSC), 2015 IEEE International Conference on*. [S.l.: s.n.], 2015. p. 272–275.

CIRANI, S. et al. A scalable and self-configuring architecture for service discovery in the internet of things. *IEEE Internet of Things Journal*, v. 1, n. 5, p. 508–521, Oct 2014. ISSN 2327-4662.

COMPTON, M. et al. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, v. 17, p. 25 – 32, 2012. ISSN 1570-8268. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1570826812000571>.

CONSORTIUM, O. G. *Location matters: Spatial standards for the Internet of Things*. Switzerland, Sept 2013.

DE, S. et al. Service modelling for the internet of things. In: *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*. [S.l.: s.n.], 2011. p. 949–955.

DELICATO, F.; PIRES, P.; BATISTA, T. *Middleware solutions for the Internet of Things*. [S.l.]: Springer London, 2013.

DEY, A. K. Understanding and using context. *Personal Ubiquitous Comput.*, Springer-Verlag, London, UK, UK, v. 5, n. 1, p. 4–7, jan. 2001. ISSN 1617-4909. Disponível em: <http://dx.doi.org/10.1007/s007790170019>.

DEY, S. et al. A semantic sensor network (ssn) ontology based tool for semantic exploration of sensor. In: . [S.l.]: Semantic Web Challenge Competition ISWC, 2014.

EVDOKIMOV, S. et al. Comparison of discovery service architectures for the Internet of Things. In: *Proceedings of the 2010 IEEE International Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC 2010)*. USA: IEEE, 2010. p. 237–244.

FERNÁNDEZ MENDEZ, M. et al. Semantically enhanced information retrieval: An ontology-based approach. *Web Semant.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 9, n. 4, p. 434–452, dez. 2011. ISSN 1570-8268. Disponível em: <http://dx.doi.org/10.1016/j.websem.2010.11.003>.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, ACM, New York, NY, USA, v. 2, n. 2, p. 115–150, maio 2002. ISSN 1533-5399. Disponível em: <http://doi.acm.org/10.1145/514183.514185>.

GOMES, P. et al. A federated discovery service for the internet of things. In: *Proceedings of the 2Nd Workshop on Middleware for Context-Aware Applications in the IoT*. New York, NY, USA: ACM, 2015. (M4IoT 2015), p. 25–30. ISBN 978-1-4503-3731-1. Disponível em: <http://doi.acm.org/10.1145/2836127.2836129>.

GRUBER, T. R. A translation approach to portable ontology specifications. *Knowl. Acquis.*, Academic Press Ltd., London, UK, UK, v. 5, n. 2, p. 199–220, jun 1993. ISSN 1042-8143. Disponível em: <http://dx.doi.org/10.1006/knac.1993.1008>.

GU, T.; PUNG, H. K.; ZHANG, D. Q. A middleware for building context-aware mobile services. In: *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th.* [S.l.: s.n.], 2004. v. 5, p. 2656–2660.

HOLLER, J. et al. *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*. [S.l.]: Elsevier Science, 2014.

ISSARNY, V. et al. Service-oriented middleware for the future internet: state of the art and research directions. *Journal of Internet Services and Applications*, v. 2, n. 1, p. 23–45, 2011. ISSN 1869-0238. Disponível em: <http://dx.doi.org/10.1007/s13174-011-0021-3>.

JAMSHIDI, M. (Ed.). *Systems of Systems Engineering: Innovations for the 21st Century*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2009.

JAYARAMAN, P. P.; CALBIMONTE, J.-P.; QUOC, H. N. M. The schema editor of openiot for semantic sensor networks. In: KYZIRAKOS, K. et al. (Ed.). [S.l.: s.n.], 2015. p. 25–30.

JÚNIOR, J. *WebFlowAH: Um ambiente para especificação e execução ad-hoc de processos de negócio baseados em serviços Web.* Dissertação (Mestrado) — Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte., 2008.

KHODADADI, F.; DASTJERDI, A.; BUYYA, R. Simurgh: A framework for effective discovery, programming, and integration of services exposed in iot. In: *Recent Advances in Internet of Things (RIoT), 2015 International Conference on*. [S.l.: s.n.], 2015. p. 1–6.

KIM, H.; CHO, Y.-J.; OH, S.-R. Camus: a middleware supporting context-aware services for network-based robots. In: *Advanced Robotics and its Social Impacts, 2005. IEEE Workshop on*. [S.l.: s.n.], 2005. p. 237–242.

KITCHENHAM, B. A.; BUDGEN, D.; BRERETON, O. P. Using mapping studies as the basis for further research - a participant-observer case study. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 53, n. 6, p. 638–651, jun. 2011. ISSN 0950-5849.

KITCHENHAM, B. A.; DYBA, T.; JORGENSEN, M. Evidence-based software engineering. In: *Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004. (ICSE '04), p. 273–281. ISBN 0-7695-2163-0.

KORPIPää, P.; MäNTYJäRVI, J. An ontology for mobile device sensor-based context awareness. In: *Proceedings of the 4th International and Interdisciplinary Conference on Modeling and Using Context*. Berlin, Heidelberg: Springer-Verlag, 2003. (CONTEXT'03), p. 451–458. ISBN 3-540-40380-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1763142.1763181>.

KRUMM, J. *Ubiquitous Computing Fundamentals*. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2009. ISBN 1420093606, 9781420093605.

LI, J.; ZAMAN, N.; LI, H. A decentralized locality-preserving context-aware service discovery framework for internet of things. In: *Services Computing (SCC), 2015 IEEE International Conference on*. [S.l.: s.n.], 2015. p. 317–323.

LIU, Y.; PLALE, B. *Survey of publish subscribe event systems*. 2003. Disponível em: <citeseer.ist.psu.edu/liu03survey.html>.

LOPEZ, P. et al. Scalable oriented-service architecture for heterogeneous and ubiquitous iot domains. *CoRR*, abs/1311.4293, 2013. Disponível em: <http://arxiv.org/abs/1311.4293>.

MACAULAY, J.; BUCKALEW, L.; CHUNG, G. *Internet of Things in Logistics*. [S.l.], 2015.

MAIA, P. et al. On the development of systems-of-systems based on the internet of things: A systematic mapping. In: *Proceedings of the 2014 European Conference on Software Architecture Workshops*. New York, NY, USA: ACM, 2014. (ECSAW '14), p. 23:1–23:8. ISBN 978-1-4503-2778-7. Disponível em: <http://doi.acm.org/10.1145/2642803.2642828>.

MARIE, P. et al. Context in computing: A cross-disciplinary approach for modeling the real world. In: _____. New York, NY: Springer New York, 2014. cap. The QoCIM Framework: Concepts and Tools for Quality of Context Management, p. 155–172. ISBN 978-1-4939-1887-4.

MARIE, P. et al. From ambient sensing to iot-based context computing: an open framework for end to end qoc management. *Sensors, Select Papers from UCAmI & IWAAL 2014*, MDPI Publishing, http://www.mdpi.com, v. 15, n. 6, p. 14180–14206, june 2015. Disponível em: <http://www.mdpi.com/1424-8220/15/6/14180>.

MARTIN, D. et al. Bringing semantics to web services with owl-s. *World Wide Web*, Kluwer Academic Publishers, Hingham, MA, USA, v. 10, n. 3, p. 243–277, set. 2007. ISSN 1386-145X. Disponível em: <http://dx.doi.org/10.1007/s11280-007-0033-x>.

MAYER, S.; GUINARD, D. An extensible discovery service for smart things. In: *Proceedings of the Second International Workshop on Web of Things*. New York, NY, USA: ACM, 2011.

PAGANELLI, F.; PARLANTI, D. A dht-based discovery service for the internet of things. *Journal of Computer Networks and Communications*, v. 2012, p. 107041:1–107041:11, 2012.

PEREIRA, V. d. C. *Perspectives and Approaches for the Internet of Things*. Dissertação (Mestrado) — Faculdade de CiÃªncias e Tecnologia, 2014.

PERERA, C. et al. Context-aware sensor search, selection and ranking model for internet of things middleware. In: *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management - Volume 01*. Washington, DC, USA: IEEE Computer Society, 2013. (MDM '13), p. 314–322. ISBN 978-0-7695-4973-6. Disponível em: <http://dx.doi.org/10.1109/MDM.2013.46>.

PERERA, C. et al. Context aware computing for the internet of things: A survey. *CoRR*, abs/1305.0982, 2013. Disponível em: <http://arxiv.org/abs/1305.0982>.

PETERS, S.; SHROBE, H. E. Using semantic networks for knowledge representation in an intelligent environment. In: *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on.* [S.l.: s.n.], 2003. p. 323–329.

PIRES, P. et al. A platform for integrating physical devices in the internet of things. In: *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on.* [S.l.: s.n.], 2014. p. 234–241.

SOLDATOS, J.; SERRANO, M.; HAUSWIRTH, M. Convergence of utility computing with the internet-of-things. In: *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on.* [S.l.: s.n.], 2012. p. 874–879.

SPALAZZI, L.; TACCARI, G.; BERNARDINI, A. An internet of things ontology for earthquake emergency evaluation and response. In: *Collaboration Technologies and Systems (CTS), 2014 International Conference on.* [S.l.: s.n.], 2014. p. 528–534.

STRANG, T.; LINNHOFF-POPIEN, C. A context modeling survey. In: *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England.* [S.l.: s.n.], 2004.

WEERASINGHE, T.; WARREN, I. Odin: Context-aware middleware for mobile services. In: *Services (SERVICES-1), 2010 6th World Congress on.* [S.l.: s.n.], 2010. p. 661–666.

WEI, Q.; JIN, Z. Service discovery for internet of things: A context-awareness perspective. In: *Proceedings of the Fourth Asia-Pacific Symposium on Internetware.* New York, NY, USA: ACM, 2012. p. 25:1–25:6.

WEISER, M. Some computer science issues in ubiquitous computing. *Commun. ACM*, ACM, New York, NY, USA, v. 36, n. 7, p. 75–84, jul. 1993. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/159544.159617>.

WEISER, M. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 3, n. 3, p. 3–11, jul. 1999. ISSN 1559-1662.

ZARGHAMI, S. *Middleware for Internet of Things.* Dissertação (Mestrado) — Faculty of Electrical Engineering, Mathematics and Computer Sciente, University of Twente, 2013.

ZHAO, R.; WANG, J. Visualizing the research on pervasive and ubiquitous computing. *Scientometrics*, v. 86, n. 3, p. 593–612, 2010.

ZHOU, X. et al. Spbca: Semantic pattern-based context-aware middleware. In: *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on.* [S.l.: s.n.], 2009. p. 891–895.