Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

# Uma Arquitetura Orientada a Serviços para Integração de Redes de Sensores e Atuadores Heterogêneos na Internet das Coisas

Yuri Farias Gomes

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Hyggo Oliveira de Almeida e Angelo Perkusich
(Orientadores)

Campina Grande, Paraíba, Brasil

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG**

G633a Gomes, Yuri Farias.

Uma arquitetura orientada a serviços para integração de Redes de Sensores e Atuadores Heterogêneos na Internet das coisas / Yuri Farias Gomes. – Campina Grande, 2016.

73 f. il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2016.

"Orientação: Prof. Dr. Hyggo Oliveira de Almeida e Prof. Dr. Angelo Perkusich".

Referências.

1. Internet das Coisas. 2. Redes de Sensores sem Fio. 3. Arquitetura Orientada a Serviços. 4. Modelos de Dados. 5. Infraestrutura para internet das Coisas. 6. Internet- Serviços Pervasivos. I. Almeida, Hyggo Oliveira de. II. Perkusich, Angelo. III. Título.

CDU 004.414.28 (043)

# Resumo

A visão da Internet das Coisas possibilitou o desenvolvimento de uma diversidade de aplicações e serviços que antes não era possível devido a uma série de limitações. Apesar de algumas dificuldades ainda existentes no *hardware*, como poder de processamento limitado e utilizações de baterias, pesquisas indicam que no ano de 2016 mais de 6,4 bilhões de dispositivos estarão conectados. A alta diversidade destes aparelhos cria a necessidade de infraestruturas capazes de lidar com dispositivos altamente heterogêneos e suas limitações de *hardware*. Neste trabalho propõe-se uma arquitetura orientada a serviços para integrar dispositivos na Internet das Coisas e resolver grande parte dos problemas que essa integração ocasiona. A partir desta arquitetura, serviços e aplicações poderão acessar sensores e atuadores através da web utilizando modelos de dados definidos a partir de padrões na Internet. O gerenciamento dos nós conectados a esta infraestrutura é realizado a partir de um *middleware* conectado a dispositivos ou *gateways* para a tradução de informações na tecnologia de comunicação utilizada (ie. Bluetooth, ZigBee, entre outros). Esta proposta foi avaliada com o desenvolvimento de um *middleware* baseado na especificação UPnP e uma aplicação Android para simulação dos dados de sensores. Resultados do experimento demonstram a viabilidade de utilização da arquitetura proposta na Integração com aplicações, serviços e outras arquiteturas disponíveis na Internet através da web e modelos de dados padronizados.

**Palavras-Chave**: Internet das Coisas; Redes de Sensores sem Fio; Arquitetura Orientada a Serviços; Modelos de Dados; Infraestrutura para Internet das Coisas; Serviços Pervasivos.

# Abstract

The vision of Internet of Things enabled the development of a diverse range of applications and service not possible before due to a number of limitations. Despite some remaining problems still exists on hardware, such as limited processing power and battery usage, researches indicates that in 2016, more than 6.4 billion devices will be connected. The high diversity of these devices creates the need of a infrastructure capable of managing highly heterogeneous devices and their hardware limitations. This work proposes an service-oriented architecture to integrate IoT devices and solve most issues that this integration brings. Using this architecture, services and applications will be able to access sensors and actuators from the web using data models from repositories on the Internet. The management of connected devices is performed by a middleware that can be connected directly to the devices or through gateways that can translate information to the communication technology used (ie. Bluetooth, ZigBee, and others). This proposal was evaluated with the development of a middleware based on the UPnP specification and an Android application to simulate sensor data. Results from this evaluation shows the feasibility of the solution with the integration with applications, services and other architectures available on the Internet through the web using the same data model.

**Keywords**: Internet of Things; Wireless Sensor Networks; Service-Oriented Architecture; Data Models; Infrastructure for Internet of Things; Pervasive Services.

# Agradecimentos

Primeiramente, eu gostaria de agradecer à Deus por ter me privilegiado com a saúde e conhecimentos necessários para concluir este trabalho. Meus pais e minhas irmãs, por ter me apoiado em diversos momentos, estando juntos e incentivando na conclusão de mais uma etapa da minha vida.

Agradeço aos meus orientadores e doutorandos que dedicaram seu tempo para me auxiliar na orientação, pesquisa e revisão deste trabalho. Sem essa ajuda, este trabalho nunca teria chegado onde chegou, e por isso eu agradeço imensamente.

Agradeço também aos meus amigos pessoais, que sempre estiveram juntos nos momentos bons e também nos momentos difíceis de toda essa jornada. Muito obrigado por tudo!

Gostaria também de fazer um agradecimento especial a minha comunidade e amigos na fé cristã, que sempre estiveram orando e intercedendo para que Deus pudesse me capacitar a concluir este trabalho.

Também agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro.

Por fim, gostaria de dedicar este trabalho a minha irmã Carolina, que apesar de não estar mais fisicamente ao nosso lado, tenho a certeza de que ela iria se alegrar e festejar bastante, a conclusão de mais essa etapa da minha vida. Obrigado cacazinha.

# Conteúdo

# Lista de Abreviações

API - *Application Programming Interface*

BLE - *Bluetooth Low Energy*

CP - *Control Point*

CoAP - *Constrained Application Protocol*

HTTP - *Hypertext Transfer Protocol*

IETF - *Internet Engineering Task Force*

IP - *Internet Protocol*

IoT - *Internet of Things*

JID - *Jabber Identifier*

JSON - *JavaScript Object Notation*

LAN - *Local Area Network*

MD - *Model Data*

MQTT - *MQ Telemetry Transport*

OCF - *Open Connectivity Foundation*

OSI - *Open Systems Interconnection*

PubSub - *Publish/Subscribe*

QoS - *Quality of Service*

REST - *Representational State Transfer*

RFID - *Radio-Frequency ID*

SOA - *Service-Oriented Architecture*

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*

UPnP - *Universal Plug and Play*

UUID - *Universally Unique Identifier*

WAN - *Wide Area Network*

WSAN - *Wireless Sensor and Actuator Network*

WSN - *Wireless Sensor Network*

XML - *eXtensible Markup Language*

XMPP - *Extensible Messaging and Presence Protocol*

# Lista de Figuras

# Lista de Tabelas

# Lista de Códigos Fonte

# Capítulo 1

# Introdução

Com uma quantidade cada vez maior de dispositivos coletando dados de maneira autônoma, surgem novos serviços capazes de utilizar esses dados afim de extrair informações úteis para tomada de decisões. Na visão da Internet das Coisas, estes dispositivos são capazes de se interconectarem e estarem disponíveis em uma rede mundial tal como a Internet. Tais aparelhos são dispostos nos mais diversos lugares, podendo enviar informações do mundo físico para o meio virtual, bem como também realizar atividades mecânicas e elétricas, como ativar uma trava de porta, a partir de dados recebidos por meios de comunicação [1].

Um grupo de dispositivos que atuam em conjunto e podem fazer parte da Internet das Coisas são as Redes de Sensores sem Fio (do inglês, *Wireless Sensor Network - WSN*), que são redes formadas por nós capazes de captar informações do ambiente em que estão dispostos [2]. Esses sensores, usualmente, são dispositivos que possuem uma capacidade de processamento, memória e bateria bastante limitados, fazendo com que a eficiência do *software* executado seja de extrema importância. Dessa forma, tais redes podem ser utilizadas em diversas aplicações, tais como a verificação e detecção de incêndios [3], nas áreas de saúde pessoal ou sistemas médicos [4, 5] e automação doméstica. Na área de saúde, por exemplo, existe um grande avanço na idade média da população mundial, a qual irá trazer um aumento no número de pacientes mais velhos que sofrem de doenças crônicas, e com isso, cria-se a necessidade de um monitoramento remoto em tempo real para o controle de enfermidades [5].

Devido às limitações de processamento, memória e energia, que os dispositivos integrantes da rede de sensores possuem, os dados gerados por esses aparelhos são comumente

enviados para outras entidades no sistema, para análise e processamento. Nesse sentido, comumente um dos nós da rede de sensores atua como um receptor de dados, e faz o envio dos dados para um dispositivo com maior poder de processamento em uma rede externa. Tais entidades podem ser chamadas de *Gateway* ou *Sink Node* [6]. Na literatura, vemos diversas soluções que utilizam *Sink Nodes* [3, 7, 8] para expor os dados para a Internet das Coisas e também para repassá-los a outras entidades responsáveis pelo processamento, como servidores na nuvem [9].

O gerenciamento de dados da Internet das Coisas é extremamente complexo devido às mais diferentes propriedades que cada dispositivo possui. Dados da IoT são provenientes de uma grande variedade de objetos e sensores, cada um com sua própria representação. Além disso, o aumento da quantidade de dispositivos conectados na rede irá trazer uma rápida explosão na escala dos dados coletados [10].

Os dados extraídos da Internet das Coisas, se isolados, não possuem valor algum. Eles precisam ser analisados, exibidos e utilizados por terceiros para terem valor. A agregação desses dados gera informações úteis para usuários, podendo originar estatísticas e previsões nas mais diferentes áreas. *Machine Learning* e *Big Data* são grandes exemplos, onde uma gigantesca quantidade de dados é necessária para que inferências possam ser realizadas. Uma Arquitetura Orientada a Serviços é promissora para resolver a complexidade de escalabilidade da troca de informações na IoT e ainda de fornecer acesso aos dados de dispositivos de maneira uniforme e padronizada para agentes externos.

## 1.1 Problemática

Dispositivos IoT podem ser classificados em dois grandes tipos: os sensores, que têm por característica principal transformar informações do mundo físico em um dado digital, e os atuadores que, através de instruções recebidas, transformam o meio físico em que estão dispostos, através de partes mecânicas atreladas aos mesmos. Estes dispositivos, comumente, não possuem interface direta com o usuário e, por isso, é necessária uma arquitetura que faça uso destes aparelhos de forma eficiente e exponha seus dados para aplicações e serviços externos, para que possa existir o agregamento, processamento de dados e exposição para usuários [11].

Para utilização dos dados provenientes dos dispositivos, uma das maiores dificuldades é o formato em que seus dados estão organizados. Muitos aparelhos que possuem o mesmo propósito, como sensores de temperatura ou batimentos cardíacos, capturam as mesmas informações, mas devido a diferenças e falta de padronização entre fabricantes, a estrutura dessas informações torna extremamente difícil a integração com serviços na Internet. Essa heterogeneidade faz com que interessados tenham que se adaptar a cada novo dispositivo ou fabricante lançado no mercado, aumentando o custo de desenvolvimento de suas aplicações.

*Gateways* para intermediar informações entre dispositivos e serviços são comumente utilizadas para transcrever pacotes e estruturá-los no formato utilizado por serviços. Os *gateways* têm por característica serem aparelhos com poder de processamento mais robusto, e podem se conectar a diversos outros nós simultaneamente [12].

Um exemplo do uso de *gateways* são os aparelhos que utilizam o protocolo ZigBee[1]. A grande maioria destes utilizam um ponto de controle para os atuadores e sensores conectados, tornando-os mais simples e diminuindo seu custo para produção em larga escala.

A IoT tem a característica de envolver uma quantidade massiva de "coisas" conectadas. Estas, por sua vez, possuem uma grande probabilidade de falha, devido, muitas vezes, à mobilidade e limitações de *hardware*, como uso de bateria e conectividade simples. Dispositivos podem entrar e sair da rede a qualquer momento devido a mudanças em sua disposição, ou interferências no seu ambiente. Este aspecto traz a necessidade de um mecanismo para gerenciamento eficiente e escalável que leve em conta suas características extremamente dinâmicas [13].

Segurança é outro fator de extrema importância no âmbito da IoT. As informações provenientes de sensores geralmente possuem um valor significativo e devem ser tratadas com o mais alto cuidado. Todas as principais arquiteturas em IoT possuem maneiras de lidar com a privacidade das informações a fim de proteger a segurança de seus usuários.

Como enunciado no início desta seção, dispositivos IoT, normalmente, não possuem interface direta com o usuário, e por isso a integração com outros serviços e aplicações é de extrema importância, do contrário, seus dados não poderão ser utilizados.

Sensores possuem a característica de enviar eventos extraídos do mundo real para o âmbito digital. Em diversas aplicações o tempo de reação entre uma leitura de um sensor e uma

[1] http://www.zigbee.org/

ação tomada é de extrema importância, como nas áreas de saúde [5]. Por esta peculiaridade, arquiteturas IoT devem possuir mecanismos para a extração destas informações de maneira eficiente e rápida, visto que os sensores possuem limitações em seus *hardwares*.

Com as informações descritas neste Capítulo, podemos enumerar os principais problemas encontrados em arquiteturas IoT: limitações de *hardware*; heterogeneidade de tecnologias; gerenciamento de dispositivos; segurança; integração com serviços externos; controle de dados baseado em eventos; e endereçamento único. Estes são os problemas abordados neste trabalho.

## 1.2 Objetivos

Neste trabalho, tem-se como objetivo a concepção de uma infraestrutura para Internet das Coisas que considera os seguintes requisitos: limitações de hardware; heterogeneidade de tecnologias; gerenciamento de dispositivos; segurança e privacidade; integração com serviços externos; controle de dados baseado em eventos; e endereçamento único. Com base nos 7 problemas anteriores, os seguintes objetivos específicos são elencados:

1. Elaborar uma arquitetura de integração para dispositivos IoT, onde são definidos componentes responsáveis por tratar cada um dos problemas mencionados anteriormente.

2. Desenvolver um *middleware* para intermédio da comunicação dos dispositivos IoT e da arquitetura criada.

3. Criação de uma API para expor os dispositivos da plataforma de maneira uniforme, através de um modelo de dados, para serviços na Internet.

4. Definição de um modelo de dados único, a fim de possibilitar a integração desta infraestrutura com aplicações, serviços e outras infraestruturas dispostas na Internet.

Por fim, este trabalho visa possibilitar que serviços pervasivos possam utilizar esta infraestrutura como base, e assim permitir que clientes tenham acesso a dispositivos em qualquer lugar e a qualquer momento.

## 1.3 Relevância

A partir da infraestrutura proposta neste trabalho, novas aplicações e serviços podem realizar a integração de dispositivos de diferentes tecnologias. O modelo de dados utilizado servirá também como base para integração e desenvolvimento de futuros arcabouços que se comunicam e integram diferentes protocolos de comunicação.

Diversos outros trabalhos na literatura visam cumprir estes objetivos, mas não encontramos um que tenha a meta de cumprir todos estes objetivos de integração de dispositivos e exposição de dados de maneira padronizada, facilitando a integração com outras arquiteturas.

Considerando os objetivos dessa proposta, temos as principais contribuições:

1. **Utilização de um modelo de dados para diferentes dispositivos IoT.** A definição proposta auxilia desenvolvedores na integração de dispositivos IoT de diferentes tecnologias de comunicação para uso de maneira homogênea para serem utilizados por serviços externos.

2. **Infraestrutura que virtualiza dispositivos através de modelos de dados para nuvem,** possibilitando a sua utilização por aplicações e serviços dispostos na Internet.

3. **Modelo de referência para implementação de serviços para dispositivos IoT na nuvem utilizando o modelo de dados definido.** Desta forma, facilita-se a construção de serviços que utilizam a mesma interface definida para comunicação com diferentes dispositivos.

4. **Implementação de uma API para utilização de recursos provenientes da infraestrutura.** Desta forma, desenvolvedores de aplicações poderão utilizar os recursos provenientes de dispositivos IoT, sem a necessidade de modificações em código por mudanças na infraestrutura.

Este trabalho contribui para os avanços das pesquisas no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded) da Universidade Federal de Campina Grande (UFCG), mais especificamente na área de Internet das Coisas.

## 1.4 Organização do Documento

O restante do documento se organiza da seguinte forma:

- no Capítulo 2 apresenta-se a fundamentação teórica, abrangendo assuntos como a Internet das Coisas, Serviços Pervasivos e Redes de Sensores sem Fio;
- no Capítulo 3, apresentam-se os trabalhos relacionados, onde é apresentado pesquisas que se assemelham a este trabalho e procuram resolver parcialmente ou precisamente o mesmo problema;
- o Capítulo 4 e o Capítulo 5 tratam da arquitetura proposta e a avaliação realizada na mesma;
- Por fim, no Capítulo 6, apresentam-se as conclusões e trabalhos futuros.

# Capítulo 2

# Fundamentação Teórica

Neste Capítulo apresentam-se os principais conceitos relacionados ao trabalho. No início, temos a apresentação do tema principal da Internet das Coisas. Em seguida serão expostas as principais noções sobre Serviços Pervasivos incluindo um caso de uso para os mesmos. Depois, nas Seções 2.3 e 2.4, encontram-se informações sobre uma classe de dispositivos que faz parte da Internet das Coisas, as Redes de Sensores sem Fio (do inglês - *Wireless Sensor Networks - WSN*), seus diversos tipos, limitações, características e protocolos usualmente utilizados com as mesmas. Em seguida, são apresentados conceitos relacionados a infraestruturas da Internet das Coisas e iniciativa UPnP+. Por fim, apresentam-se Arquitetura Orientada a Serviços e Modelos de Dados, definições importantes para composição da arquitetura proposta neste trabalho.

## 2.1 Internet das Coisas

Com uma quantidade cada vez maior de sensores coletando dados de maneira autônoma, surgem novos tipos de serviços, como o monitoramento de entidades, e quando essas informações se tornam disponíveis na Internet, traz-se à realidade a visão da Internet das Coisas. Sua definição evoluiu bastante nos últimos anos desde sua origem, em meados dos anos 2000, nos laboratórios do Massachusetts Institute of Technology (MIT), quando cientistas desenvolviam uma maneira de identificação única através de rádio frequência, o também chamado de *Radio-Frequency ID* (RFID). Hoje em dia, o conceito tornou-se muito mais abrangente, e consiste de um grande número de objetos físicos e virtuais, unicamente endereçáveis, que

se comunicam uns com os outros através de uma rede mundial dinâmica, tal como a Internet [10].

Esses dispositivos se conectam com o mundo real de forma transparente, podendo enviar informações do mundo físico para o meio virtual e também realizar outras atividades mecânicas a partir de dados recebidos por meios de comunicação.

## 2.2 Serviços Pervasivos

A visão de computação pervasiva objetiva criar ambientes computacionais integrados ao dia-a-dia das pessoas de uma maneira imperceptível, utilizando-se de objetos familiares e utilizados por todos, em qualquer lugar [14].

Serviços são amplamente utilizados na área de Tecnologia da Informação. Os chamados *Web Services* são um exemplo que tem ganhado bastante popularidade nos últimos anos com o avanço da Web 2.0 e das aplicações móveis. O termo serviço se refere a uma entidade que provê um conjunto de funcionalidades e que possui uma interface para comunicação com clientes e o envio de possíveis parâmetros [14].

O conceito de computação pervasiva baseada em serviços surgiu inicialmente no trabalho de Zhou [15], que utilizava serviços web como uma solução para o paradigma da computação pervasiva. Porém, serviços pervasivos possuem grandes diferenças, apesar da similaridade com serviços web. Em um ambiente pervasivo, dispositivos devem identificar serviços existentes no ambiente e comunicar-se com os mesmos através de mínima ou nenhuma interação por parte do usuário, trazendo assim o princípio da invisibilidade da computação pervasiva.

Um caso de uso bastante comum para serviços pervasivos, nos dias de hoje, é a casa inteligente. O usuário, ao entrar em sua casa, tem a presença reconhecida e recebe uma notificação com as principais informações relativas ao ambiente doméstico. Se o sistema perceber uma diferença muito grande na temperatura média do ambiente, o mesmo questiona o usuário para ativar o condicionador de ar ou outros dispositivos que alterem a temperatura do ambiente para tornar-se mais confortável. Desta forma, o exemplo anterior evidencia o uso de serviços pervasivos que utilizam o contexto do usuário sobre seu ambiente para alterar a temperatura.

## 2.3 Redes de Sensores sem Fio

As Redes de Sensores sem Fio (do inglês, *Wireless Sensor Network - WSN*) são conjuntos de dispositivos que capturam informações do meio em que estão dispostos e podem vir a fazer parte da Internet das Coisas. Esses sensores, usualmente, são dispositivos que possuem um poder de processamento, memória e bateria bastante limitados, fazendo com que a eficiência do *software* executado seja de extrema importância.

Tais redes podem ser utilizadas em diversas aplicações, como para a verificação e detecção de incêndios [3], nas áreas de saúde pessoal ou sistemas médicos [4][5], automação doméstica, entre outros. Elas também podem ser dispostas em diversos lugares, como por exemplo, de maneira subaquática, para monitoramento da temperatura da água, e também subterrâneas, para monitoramento da umidade do solo [2]. Cada uma dessas redes possuem propriedades e restrições diferentes, por exemplo, redes subterrâneas necessitam de um tempo de vida muito maior que as WSNs terrestres, devido às dificuldades para troca de bateria dos sensores.

A necessidade das WSNs cresce em áreas como a de saúde, tendo em vista que o avanço na idade média da população mundial irá trazer um aumento no número de pacientes mais velhos que sofrem de doenças crônicas, e com isso, cria-se a necessidade de um monitoramento remoto em tempo real para o controle de enfermidades [5].

Devido às limitações de processamento, memória e energia, que os dispositivos integrantes da Rede de Sensores sem Fio possuem, os dados gerados por esses aparelhos são comumente enviados para outras entidades no sistema, para análise. Para que isso ocorra, comumente um dos nós da WSN atua como um receptor de dados, e faz o envio dos mesmos a um dispositivo com maior poder de processamento em uma rede externa, os quais podem ser chamados de *Gateways* [6]. Na literatura, vemos diversas soluções que utilizam *Gateways* [3, 7, 8] para expor os dados para a Internet e também para repassá-los a outras entidades responsáveis pelo processamento, como servidores na nuvem [9].

## 2.4 Protocolos das Redes de Sensores sem Fio

Os protocolos de comunicação utilizados pelas Redes de Sensores sem Fio têm um papel importante para suprir as limitações de *hardware* que as redes possuem. Na literatura, diversos protocolos vêm sendo utilizados para várias aplicações e em todas as camadas do modelo OSI [1] [16, 17, 18, 19]. Considerando o objetivo deste trabalho, apenas alguns desses protocolos são apresentados.

### 2.4.1 Message Queue Telemetry Transport (MQTT)

Um dos mais famosos protocolos de pouca sobrecarga, o MQTT[2], teve sua primeira versão criada pela IBM[3] em 1999. Ele executa sobre o protocolo TCP/IP[4] e foi completamente projetado para dispositivos que possuem memória e capacidade de banda de transmissão limitados. Baseado em uma arquitetura *publisher/subscriber*, o MQTT é hoje utilizado em diversas aplicações, tais como o Facebook Messenger[5].

A tecnologia também possui suporte a 3 níveis de QoS (*Quality of Service*). No primeiro nível, a mensagem é enviada sem nenhuma garantia de entrega, no segundo, a mensagem é enviada e o destinatário responde com uma mensagem ACK, para confirmação de recebimento. Finalmente, no último nível QoS, é realizado um *4-way handshake* para garantir que a mensagem chegue apenas uma vez no destinatário [19].

### 2.4.2 Constrained Application Protocol (CoAP)

O CoAP [20] é um protocolo que reside na camada de aplicação do modelo OSI, executa sobre o UDP, e foi desenhado para comunicação máquina-para-máquina de dispositivos da Internet das Coisas. A tecnologia é uma versão altamente simplificada do popular HTTP (*Hypertext Transfer Protocol*), possuindo mecanismos para cache, serviços REST (*Representational State Transfer*) e subscrição de recursos.

[1] https://en.wikipedia.org/wiki/OSI_model

[2] http://mqtt.org/

[3] http://www.ibm.com/

[4] https://technet.microsoft.com/pt-br/library/cc786900%28v=ws.10%29.aspx

[5] https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920

## 2.5 Infraestrutura para Internet das Coisas

Para o monitoramento remoto e rastreamento de entidades utilizando as WSN, o uso de uma infraestrutura para Internet é necessária [5]. Essas infraestruturas têm como papel principal agir como ponte entre as Redes de Sensores sem Fio e aplicações. Na literatura, encontramos diversas soluções tecnológicas para criar-se essa infraestrutura [21]. Além disso, outras soluções foram desenvolvidas para preencher a lacuna entre dispositivos e aplicações. Entre estas, temos as mais populares como o Weave[6], do Google, o AllJoyn[7], da AllSeen Alliance e também o IoTivity[8] da Open Connectivity Foundation.

### 2.5.1 Universal Plug and Play (UPnP)

A iniciativa do UPnP especifica uma infraestrutura de comunicação e tem como objetivo principal permitir que aparelhos se conectem e enviem dados entre si de maneira uniforme e independente de fabricantes. A especificação é baseada em protocolos tais como TCP/IP, UDP, HTTP, XML e vários outros. A arquitetura do UPnP define dispositivos que proveem serviços para aplicações de controle (*Control Points*) que podem ser aparelhos como videogames, *smartphones* e computadores, entre outros.

A pilha tecnológica do UPnP consiste de 6 camadas sendo a última opcional, são elas: descoberta, descrição, controle, eventos, apresentação e endereçamento. A camada de descoberta permite que novos dispositivos UPnP encontrem outros aparelhos na rede através do envio de uma mensagem *multicast* UDP para todos os endereços ativos da rede na porta 1900. Os destinatários que recebem a mensagem e têm suporte UPnP, respondem a mensagem com uma descrição em XML dos serviços que possuem disponíveis. Exemplos de serviços UPnP podem ser, iluminação de ambientes, controle de televisão entre outros. Para endereçar unicamente seus dispositivos, a especificação define o uso dos *Universally Unique Identifiers* (UUIDs), os quais possuem regras específicas para sua geração [22].

A comunidade do UPnP é bastante ativa e está sempre discutindo novas especificações. Em uma destas discussões, surgiu o UPnP+, uma expansão da já existente especificação do UPnP, com o objetivo de virtualizar os dispositivos para a IoT. O UPnP foi escolhido

[6]https://developers.google.com/weave/

[7]https://allseenalliance.org/framework

[8]https://www.iotivity.org/

neste trabalho por possuir uma grande quantidade de bibliotecas disponíveis na Internet nas mais variadas tecnologias, facilitando a integração com outros sistemas e possuir milhões de dispositivos existentes.

A iniciativa do UPnP+ visa exportar os dispositivos UPnP, acessíveis previamente apenas por rede local, para a Internet. Quaisquer aparelhos com certificação UPnP conseguem se comunicar entre si em uma rede local, e aqueles do tipo *Cloud Device*, conseguem além disso, se comunicar com a *UPnP Cloud*, que é um mediador, na nuvem, entre os mesmos.

No diagrama ilustrado na Figura 2.1, podemos ver os diversos ambientes em que o *UPnP Cloud* pode se comunicar.

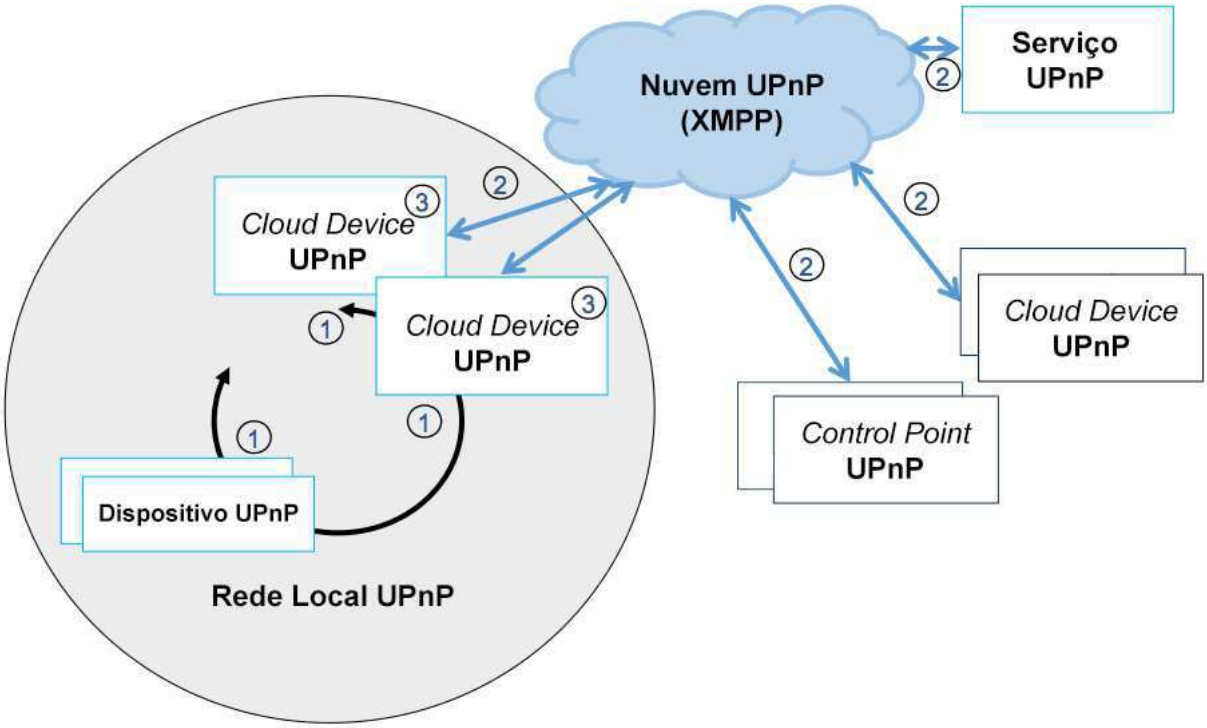


Figura 2.1: Diagrama com uma visão geral de uso do UPnP Cloud.

Primeiro, indicados pelo número 1, temos as interfaces LAN, que participam da definição original do UPnP, e seus dispositivos têm por característica a comunicação em uma rede local. A extensão do UPnP+ trouxe a definição da comunicação através da Nuvem UPnP, indicada pelo número 2, permitindo assim que todos os aparelhos conectados na infraestrutura possam trocar mensagens entre si. Indicados pelo número 3, os *UPnP Cloud Devices* possuem um endereçamento único dentro da rede local, através do UUID, e também na Nuvem

UPnP, utilizando os *Jabber Identifiers* (JIDs). Estes identificadores fazem parte do *Extensible Messaging and Presence Protocol* (XMPP) para endereçar clientes conectados através do protocolo, e por isso também são utilizados para endereçar todas as interfaces conectadas ao *UPnP Cloud*.

### 2.5.2 Extensible Messaging and Presence Protocol (XMPP)

O *Extensible Messaging and Presence Protocol* (XMPP) [23] é uma família de protocolos padronizada pela IETF[9] e bastante utilizada na Internet. Seu conjunto de protocolos está disponível nas principais linguagens de programação e sistemas operacionais, como Android, iOS e Java.

O XMPP foi projetado para ser um protocolo de comunicação em tempo-real, com baixo custo na sua troca de mensagens e latência mínima. JIDs proveem um mecanismo para identificação global, tal como endereços de e-mail. Diferentes tipos de mensagens, chamadas de *stanzas*, permitem uma comunicação bidirecional baseada no padrão *PubSub* permitindo o envio de dados em tempo-real. Sua arquitetura é descentralizada, permitindo alta escalabilidade. Infraestruturas XMPP podem ser compostas por um simples servidor e podendo ser escalado até múltiplos servidores.

## 2.6 Arquitetura Orientada a Serviços

*Softwares* desenvolvidos nos últimos anos estão cada vez mais complexos e integrados uns com os outros. Um aplicativo móvel pode se comunicar com outro aplicativo para extrair uma informação, que pode em seguida enviá-la para um servidor em nuvem para depois então mostrar uma resposta ao usuário final. Para diminuir essa complexidade, uma série de abordagens foram propostas, a fim de facilitar a integração e manutenção desses sistemas [24].

Uma Arquitetura Orientada a Serviços (do inglês, *Service-oriented Architecture*, ou *SOA*), é uma arquitetura de *software* que modulariza serviços, trazendo a flexibilidade de localização e escolha das tecnologias utilizadas na implementação para os provedores e consumidores destes serviços [24].

[9] https://www.ietf.org/

Infraestruturas que fazem uso desta técnica proveem uma interface única e padronizada para realizar a comunicação de provedores (como objetos da Internet das Coisas) e consumidores (sistemas externos, como redes de hospitais), escondendo toda a camada de *hardware* e sua heterogeneidade [25].

## 2.7 Modelos de Informação e Modelos de Dados

Atualmente, existe uma diversidade enorme de linguagens para definir objetos gerenciados por sistemas. Dentre estas estão a *Structure of Management Information* (SMI) [26], a *Structure of Policy Provisioning Information* (SPPI) [27] e a *Managed Object Format* (MOF) [28]. Dentro destas linguagens de definição, temos duas grandes categorias principais: os Modelos de Informação e os Modelos de Dados.

Modelos de Informação são utilizados principalmente para descrever o ambiente do objeto que será gerenciado, para que pessoas entendam os objetos modelados e serve como guia para a descrever suas funcionalidade em um Modelo de Dados. Estes modelos podem ser comumente descritos informalmente em linguagem natural [29].

Algumas características importantes dos Modelos de Informação é que os mesmos geralmente especificam os relacionamentos entre objetos. Além disso, organizações que queiram criar Modelos de Dados baseados nos mesmos podem decidir utilizar apenas um subconjunto das funcionalidades de um Modelo de Informação [29]. Na Figura 2.2 podemos ver a hierarquia em que diversos modelos de dados podem surgir a partir de um único modelo de informação.

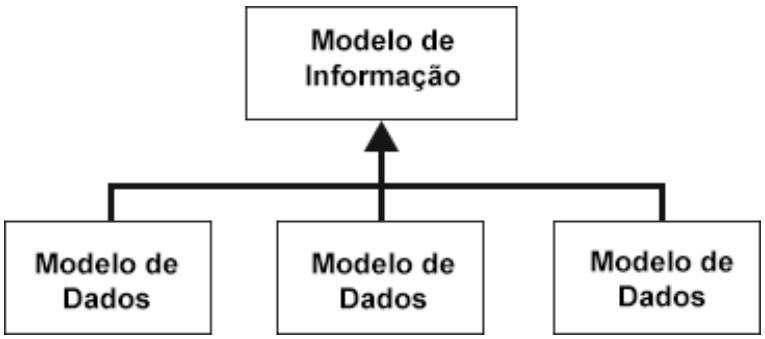


Figura 2.2: Hierarquia entre um Modelo de Informação e seus Modelos de Dados

Modelos de Dados definem seus objetos em um nível menor de abstração. Eles incluem

especificidades de protocolos e implementação em suas descrições, como por exemplo, regras de mapeamento do objeto para protocolos de baixo nível. A maior parte dos modelos padronizados atualmente são modelos de dados, incluindo o SMI, SPPI e MOF.

## 2.8 Considerações Finais do Capítulo

Neste capítulo foram apresentados conceitos e tecnologias importantes e que foram utilizados durante o desenvolvimento deste trabalho. Inicialmente, foi apresentado o paradigma da Internet das Coisas, área na qual este trabalho se enquadra. Em seguida, os conceitos de Serviços Pervasivos e Redes de Sensores sem Fio foram introduzidos, estes são importantes para o entendimento do funcionamento da arquitetura apresentada nos próximos capítulos. Posteriormente, algumas tecnologias utilizadas nos experimentos deste trabalho são apresentadas, o UPnP e XMPP.

Por fim, foram apresentados os conceitos de Arquitetura Orientada a Serviços, Modelos de Informação e Modelos de Dados, os quais também são necessários para a compreensão da arquitetura que será apresentada.

# Capítulo 3

# Trabalhos Relacionados

Neste capítulo serão apresentados trabalhos que definem o estado da arte, suas limitações e semelhanças com a solução proposta. *Middlewares* para IoT são uma área bastante ativa com diversas soluções sendo propostas e implementadas, em vista dos vários problemas em aberto que ainda existem na área [30]. Essas soluções se diversificam bastante baseada em seus objetivos, níveis de abstração e projeto da infraestrutura. *Razzaque* [30] classificam estas soluções nos seguintes tipos: *event-based*, *VM-based*, *agent-based*, *tuple-spaces*, *database-oriented* e *service-oriented*. Serão apresentados a seguir uma pequena descrição do design destas soluções e em seguida, os mais similares a proposta deste trabalho.

Soluções baseadas em eventos (do inglês, *event-based*) [31, 32, 33], são aquelas que as iterações entre os principais atores acontecem por meio de eventos. Geralmente é utilizado o padrão *Publish/Subscribe* devido a sua eficiência no gerenciamento de mensagens. Os atores envolvidos nesta solução são classificados entre produtores e consumidores e suas mensagens podem possuir informações extras relacionadas ao contexto da informação que está sendo enviada. Soluções deste tipo são geralmente utilizadas por sistemas que levam em consideração requisitos de disponibilidade, informações em tempo real e escalabilidade.

A virtualização de nós também é bastante popular, e soluções deste tipo podem ser chamadas de *VM-based* [34, 35]. Arquiteturas deste tipo se caracterizam por possuir uma camada de virtualização em cada nó da rede, fornecendo assim um maior nível de abstração para ser utilizado por aplicações clientes. A maior vantagem destas soluções é a adaptatividade mesmo em dispositivos heterogêneos.

Além das anteriores, existem os trabalhos baseados em agentes móveis (do inglês *agent-*

*based*) [36, 37]. Dispositivos são utilizados como recursos para execução de programas modulares. Estes fragmentos de *software* podem ter seu estado de execução transferido entre agentes e também utilizar outros dispositivos para executar suas tarefas. Geralmente, um *middleware* na nuvem é utilizado para mediar e gerenciar os agentes envolvidos.

Adiante, existem soluções *tuple-spaces* [38, 39], onde nós da rede guardam uma estrutura de dados chamadas de tuplas-espaciais como repositório de dados. *Gateways* entre dispositivos e clientes possuem uma tupla-espacial formada pela estrutura de dados dos dispositivos conectados a ele, facilitando o acesso externo dos dados.

Trabalhos orientados a bancos de dados (do inglês, *database-oriented*) [40, 41] são aqueles que tratam a arquitetura como um gigantesco banco de dados relacional, onde as requisições são divididas para os diversos dispositivos conectados ao *middleware* possibilitando a formulação de consultas altamente complexas.

Finalmente, existem também as soluções orientadas a serviços (do inglês, *service-oriented*) [42, 43, 44], que utilizam os mesmos princípios da arquitetura mostrada na seção 2.6. A utilização da SOA na IoT é caracterizada por utilizar os dispositivos na forma de serviços para aplicações, fornecendo uma interface padronizada e escondendo detalhes intrínsecos dos aparelhos.

Além destes, existem os trabalhos que utilizam soluções híbridas, que trazem as vantagens de dois ou mais tipos apresentados anteriormente.

## 3.1 Trabalhos Relacionados baseados em Serviços

A seguir serão apresentados trabalhos que fazem uso de arquiteturas baseadas em serviços, pois são os projetos que mais se assimilam a arquitetura proposta neste trabalho.

### 3.1.1 Uma arquitetura baseada em micro-serviços

O trabalho projetado por *Vresk* [45] utiliza uma arquitetura baseada em micro-serviços. Cada micro-serviço é responsável por gerenciar um conjunto de dispositivos em seu protocolo proprietário (ZigBee, Bluetooth, etc) e cada um destes pode comunicar-se entre si para troca de informações. No exemplo dado pelo autor, um conjunto de dispositivo envia dados para outro micro-serviço responsável por executar tarefas relacionadas a *Machine Learning*. A

solução é bastante modularizada através da implementação dos micro-serviços que podem integrar diferentes sistemas de domínios variados.

A Figura 3.1 desenvolvida pelo autor, mostra na *Network A*, um conjunto de sensores, que se comunica com o serviço *ModbusTCP*, que por sua vez, troca informações com outros micro-serviços que estão conectados a outros sistemas, como é o caso do *SOAP Web Service*. Essa troca de mensagens em um canal único de comunicação facilita o consumo das mensagens por interessados, possibilitando uma maior integração com outros serviços.

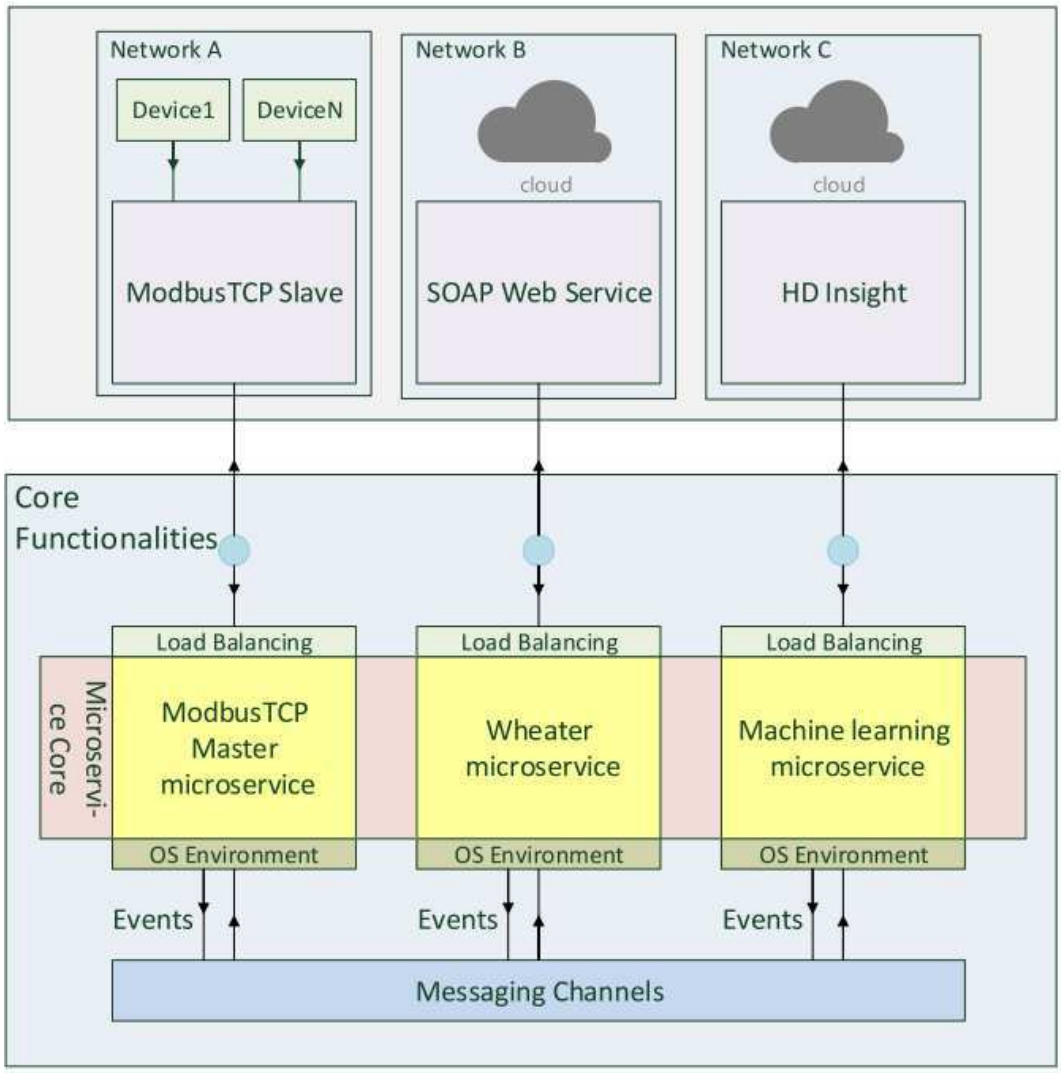


Figura 3.1: Diagrama ilustrando a arquitetura apresentada por *Vresk* [45].

Algumas limitações ainda existem, como a necessidade da implementação de um micro-serviço para cada diferente dispositivo utilizado, e os autores também não deixam claro qual o formato das mensagens utilizado na arquitetura, fator que influencia diretamente

no modelo de dados que seria utilizado por consumidores.

### 3.1.2 Uma arquitetura composta por um *IoT Hub* para integração de objetos inteligentes heterogêneos

Para lidar com os problemas de grande volume de dispositivos e sua alta heterogeneidade, os autores *Cirani* [46] propõem a adição de um elemento de rede na borda das WSANs, que age como um intermediador entre dispositivos e serviços na Internet.

O *IoT Hub*, com sua arquitetura ilustrada no diagrama da Figura 3.2, age como um nó central e nos experimentos do autor, foi implementado utilizando CoAP.

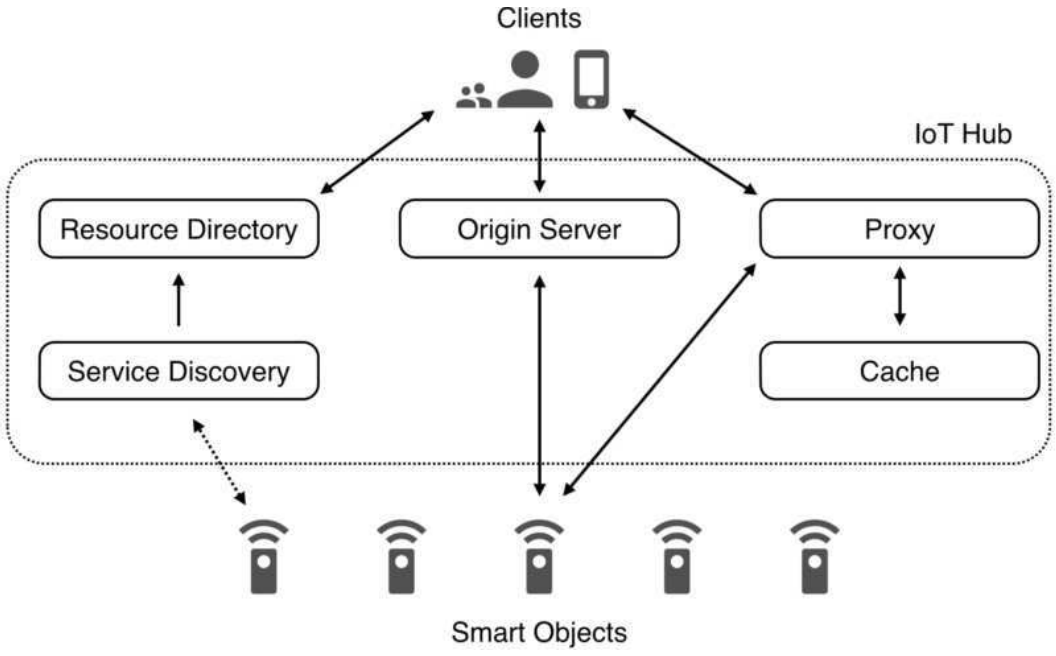


Figura 3.2: Diagrama com a arquitetura desenvolvida por *Cirani* [46].

Ele possui dois elementos para gerenciar os dispositivos, os quais são chamados de *Resource Directory*, que expõe os recursos conectado a clientes, e o *Service Discovery*, que recebe notificações de novos objetos conectados. Além disso, o *middleware* utiliza *proxys* com sistemas de *cache* para intermediar as requisições de clientes a dispositivos, reduzindo o tráfego de informações quando existir solicitações repetidas.

Para a comunicação com aparelhos mais limitados e que não possuem suporte ao CoAP, a arquitetura faz o uso do *Origin Server*, que faz a tradução de dados na comunicação com

estes dispositivos.

O CoAP é um protocolo bastante utilizado em IoT e a solução proposta pelos autores age como uma ponte para que aplicações consigam acessar os dispositivos por meio do CoAP, solucionando problemas de descoberta. Apesar disso, os autores não detalham e nem mesmo utilizam em seus experimentos o *Origin Server*, criando uma lacuna entre a solução proposta e uma grande parcela de dispositivos que não utilizam o CoAP como protocolo de comunicação.

### 3.1.3 Uma proposta para integração de WSNs com a Web

Em [6], os autores utilizam o protocolo CoAP para comunicação entre as diferentes WSNs. Para exposição dos dados, é utilizado um *gateway* conectado a Internet que possui um servidor web e um cliente CoAP, que se comunica com as Redes de Sensores sem Fio para captura dos dados, como mostrado na Figura 3.3. O *gateway*, armazena os dados das redes em um banco de dados não relacional. Esses dados podem ser visualizados através do protocolo HTTP, utilizando o formato JSON, mas o modelo de dados utilizado não foi especificado pelo autor. A maior limitação deste trabalho está na maneira simplificada do gerenciamento de dados, além disso, não foi avaliado seu uso com grande volume dados, e nem especificado o modelo de dados utilizado para expor as informações das redes.

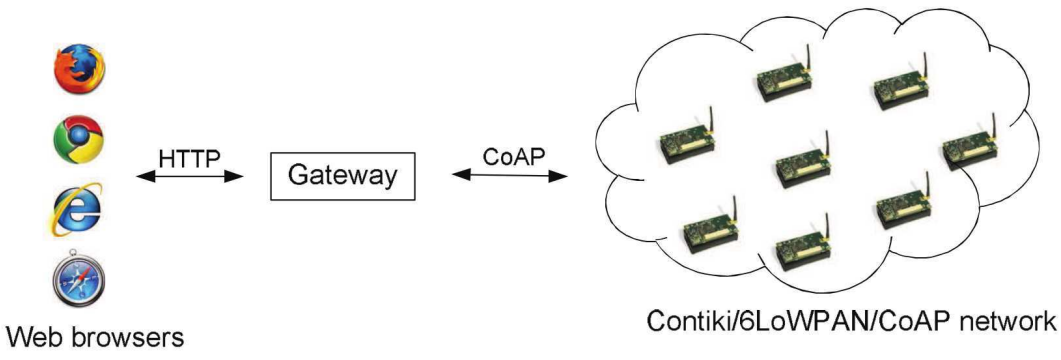


Figura 3.3: Diagrama ilustrando a arquitetura apresentada por *Colitti* [6].

### 3.1.4 Uma infraestrutura de serviço para a Internet das Coisas baseada em XMPP

Outro trabalho similar é o de *Bendel* [21], onde uma infraestrutura baseada em XMPP é proposta para a comunicação com dispositivos IoT. A arquitetura é baseada em uma Plataforma de Serviço (do inglês *Service Platform*), que age na rede como uma central de gerenciamento de dados. Dispositivos conectados na rede enviam pacotes já no formato XMPP e caso não seja possível, conversores são utilizados, como no caso de dispositivos Bluetooth que podem fazer uso do *Bluetooth-2-XMPP Converter*, mostrado no exemplo da Figura 3.4. Para tornar a visão de IoT possível, os dispositivos da infraestrutura são unicamente identificáveis através de um endereço JID, o qual provê uma maneira de identificação única para clientes XMPP.

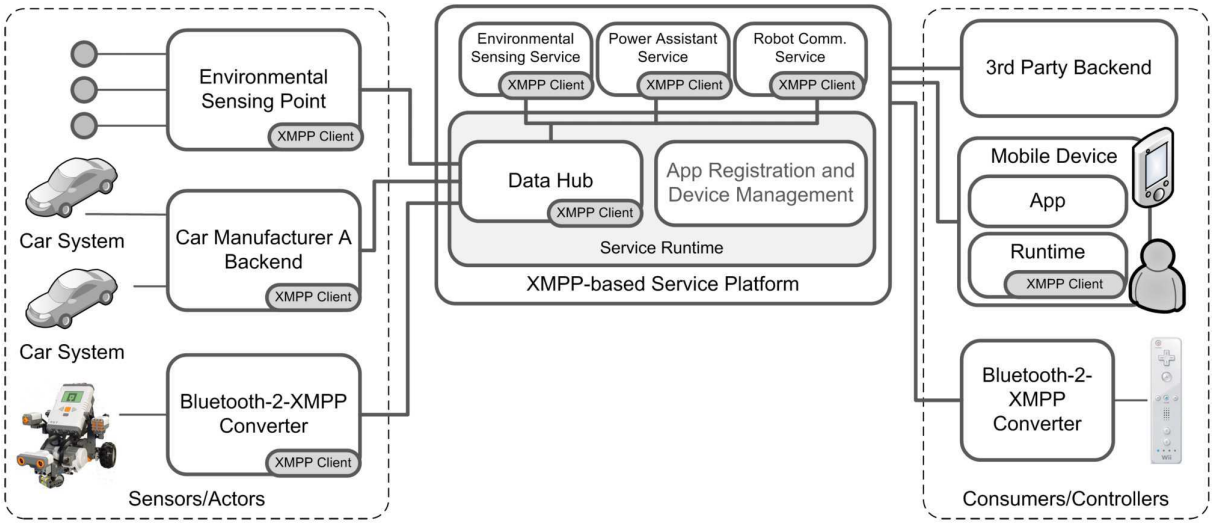


Figura 3.4: Diagrama ilustrando a arquitetura apresentada por *Bendel* [21].

Uma infraestrutura baseada em XMPP se torna bastante eficiente para integração de novos serviços, visto que é uma tecnologia difundida, e existem diversas bibliotecas para realizar a conversão em diversas tecnologias. Além disso, é possível o acesso e controle dos dispositivos na rede local através da Internet, permitindo que os mesmos façam parte da visão IoT. A maior limitação dessa plataforma está na integração com outras plataformas e dispositivos já existentes. Para utilizar aparelhos já presentes no mercado, seria necessário a criação de conversores, e um novo elemento provavelmente seria preciso para expor os dados

da plataforma em modelos de dados padrões.

### 3.1.5 Uma arquitetura para integrar WSNs heterogêneas em um ambiente de monitoramento para saúde

O trabalho que mais se assimila a solução proposta no Capítulo 4 é o projeto apresentado por *Corchado* [47]. O autores propõem a utilização de uma infraestrutura chamada SYLPH, baseada em uma Arquitetura Orientada a Serviços (do inglês, *Service-Oriented Architecture* - SOA) para a integração de Redes de Sensores sem Fio Heterogêneas em Ambientes Inteligentes e realização de um monitoramento remoto de pacientes. Um caso de uso na área de saúde foi evidenciada pelo autor e pode ser notada na Figura 3.5, mostrando como funciona a comunicação de dispositivos com serviços na Internet.

Dentro da plataforma SYLPH, WSANs de diferentes tecnologias tem seus pacotes convertidos para um protocolo comum chamado SSP (*SYLPH Services Protocol*). Estes permitem que pacotes sejam enviados de um nó da plataforma para outro, independente da rede a qual pertencem. A mensagem deste protocolo especifica um nó origem, um nó alvo e uma invocação de serviço, definido por uma linguagem chamada SSDL (*SYLPH Services Definition Language*). A arquitetura é distribuída e possui um ou mais nós da rede sendo SDNs (*SYLPH Directory Nodes*), onde estes contem uma listagem dos nós da rede e seus serviços disponíveis. Além disso, *gateways* SYLPH são necessários para a interconexão de WSANs de diferentes tecnologias de comunicação.

A proposição de arquitetura feita por [47] cobre diversos objetivos abordados no Capítulo 4. A mesma possui uma arquitetura baseada em serviços, o que facilita a integração com terceiros. Existe uma integração de Redes de Sensores e Atuadores sem Fio Heterogêneas e a solução é baseada em uma infraestrutura distribuída, onde não é necessário um nó central na rede para gerenciar os outros componentes. As maiores limitações desta arquitetura em relação a solução proposta neste trabalho é que o sistema SYLPH utiliza padrões novos a comunidade, o que se faz necessário o desenvolvimento novas ferramentas, tais como bibliotecas, para a integração de serviços existentes com o SSP.

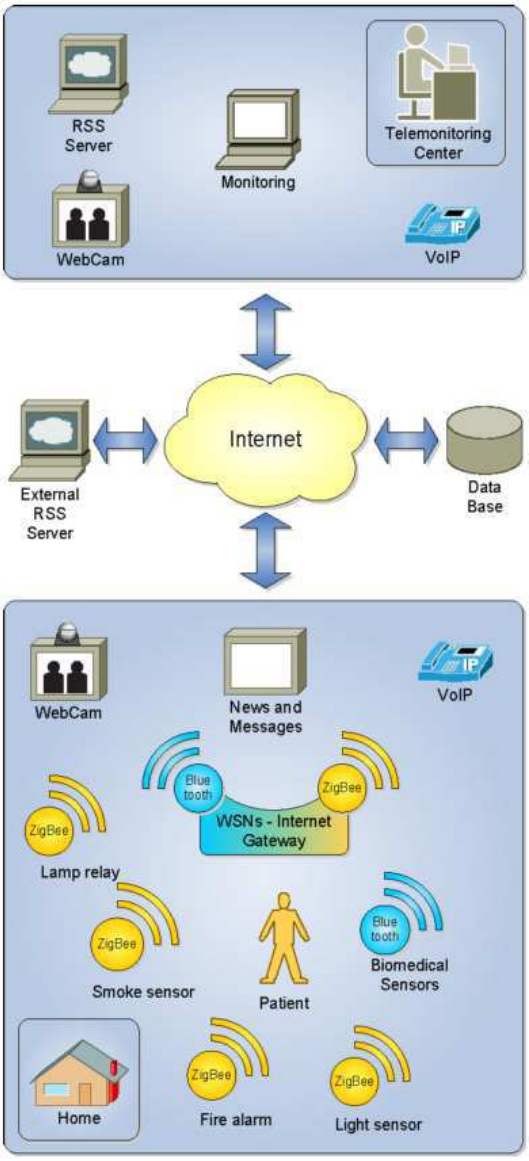


Figura 3.5: Diagrama ilustrando a arquitetura apresentada por *Corchado* [47].

## 3.2 Análise Comparativa e Discussão dos Trabalhos

Todos os trabalhos relacionados enunciados visam solucionar o problema de interoperabilidade de acesso a dispositivos por serviços externos na IoT. Visando uma análise comparativa entre os mesmos, a Tabela 3.1 foi criada tomando como parâmetros objetivos comumente utilizados em *middlewares* IoT e descritos em mais detalhes a seguir:

**Descoberta e Gerenciamento**

Remete a capacidade de suportar a descoberta de novos dispositivos, antes não participantes da rede, e gerenciar de maneira eficiente a alta mobilidade das WSANs.

**Abstração de Dados**

Visando a utilização de dispositivos por serviços e aplicações, a abstração de dados é necessária para que não seja necessário implementações especificas para cada tipo diferente de dispositivo, quando estes forem utilizados para o mesmo fim (ie. sensor de temperatura).

**Interoperabilidade**

Refere-se ao suporte na utilização e integração de dispositivos que utilizam tecnologias de comunicação diferente da empregada pelo *middleware*. Isso significa, por exemplo, na capacidade de incorporar aparelhos de tecnologias Bluetooth, CoAP, ZigBee para utilização na mesma arquitetura.

**Volume de Dados e *Eventing***

Um fator importante relacionado as arquiteturas avaliadas é a questão da sua eficiência em lidar com um grande volume de dados. Essa é uma característica inerente as WSANs e deve ser tratado com efetividade. Além disso, a atualização de eventos em tempo real, chamado aqui de *Eventing*, é outro fator importante, em vista aplicações relacionadas a sistemas de monitoramento.

**Integração Externa**

Finalmente, o parâmetro de integração com outras arquiteturas e sistemas é avaliado entre soluções. Diversas soluções existem e outras ainda estão em desenvolvimento, então a facilidade de integração com outras arquiteturas por parte de aplicações e serviços é importante. Essa medida avaliada se remete a capacidade de se comunicar por um padrão já existente em outras arquiteturas.

O trabalho de *Vrest* [45], primeiro avaliado, possui o gerenciamento de dispositivos feito por micro-serviços, o que deixa os elementos de descoberta e gerencia a cargo da implementação do micro-serviço. O funcionamento da abstração de dados não foi detalhada pelo autor e sua interoperabilidade pode ser atingida através da implementação de micro-serviços. O tratamento de volume de dados e eventos existe através do canal único de comunicação, chamado de *Messasing Channels*, apesar de que o volume de dados não foi avaliado pelo autor. A integração externa não foi referenciada em momento algum do texto, porém, suspeita-se que isso é alcançado através de um micro-serviço especifico para este fim.

| **Solução** | Descoberta e Gerenciamento | Abstração de Dados | Interoperabilidade | Volume de Dados e Eventing | Integração Externa |
|---|---|---|---|---|---|
| *Vrest* | Deve ser implementada em micro-serviços | Não | Sim | Possui suporte | Não |
| *Cirani* | Sim | Sim | Parcial, utiliza *Origin Server* | Sim | Apenas com serviços CoAP |
| *Colitti* | Sim | Sim, mas não especificada | Não | Sim, mas não foi avaliada | Utiliza HTTP/-REST |
| *Bendel* | Sim | Parcial, autor diz que possui mas não entra em detalhes | Sim | Sim | Necessário a implementação de conversor XMPP |
| *Corchado* | Apenas Sensores | Sim | Apenas Sensores | Sim | Não |
| **Este Trabalho** | Sim | Sim | Sim | Possui suporte, mas não foi avaliado | Sim, através de modelos de dados padrões |

Tabela 3.1: Tabela comparativa de trabalhos relacionados.

*Cirani* [46] trata o gerenciamento com o *Resource Directory* e *Service Discovery*. Sua abstração de dados e interoperabilidade acontece por meio do CoAP, dificultando a comunicação apenas de dispositivos que não possuem suporte ao protocolo. Para amenizar isso, é utilizado o *Origin Server*, o qual não tem seu funcionamento detalhado pelo autor. A questão de eventos e dados é gerenciada pelo protocolo CoAP, e sua integração externa acontece utilizando a *CoRE Link Format*[1].

O *gateway* sugerido por *Colitti* [6] possui características semelhantes a solução anterior, visto que ambos utilizam o protocolo CoAP como base de comunicação. O principal diferencial está exposição para serviços na Internet através do protocolo HTTP. Seu gerenciamento acontece no próprio *gateway*, enquanto a descoberta é feita através do CoAP. A abstração de dados é feita através do formato JSON, mas não teve seu modelo de dados detalhado. Assim como na solução anterior, a interoperabilidade acontece por meio do CoAP, possuindo as mesmas limitações. Em seguida, temos o volume de dados e *eventing*, que é tratado também com o protocolo, mas não foi avaliado no trabalho do autor. Finalmente, a integração externa acontece através do HTTP, mas não tem modelo de dados especificado.

Em seguida, temos a *Service Platform* proposta por *Bendel* [21]. Seu gerenciamento e descoberta acontece pelo protocolo XMPP, em conjunto do modulo *App Registration and Device Management*. A abstração de dados e interoperabilidade acontece por meio do XMPP, exigindo o uso de conversores. O autor não entra em detalhes de como funciona a abstração de dados para dispositivos de mesma finalidade. Devido ao uso do XMPP, o *Eventing* e o fator de grande volume de dados são tratados. E finalmente, a integração externa acontece utilizando o próprio protocolo, mas sendo necessário um conversor para abstrair dados da mesma maneira que em outras plataformas, visto que o XMPP é apenas uma tecnologia de comunicação e não possui modelo de dados definido para modelagem de entidades.

O trabalho de *Corchado* [47] aparenta o mais completo dentre os outros, sendo limitado principalmente pelo fator de ter uma aplicação especifica de monitoramento. A solução possui o gerenciamento e descoberta, abstração de dados e interoperabilidade através de um protocolo especifico, o *SSP*, lida com grandes volumes de dados e eventos em tempo real, inclusive avaliados pelo autor, mas sua integração externa é limitada devido ao uso de protocolos e abstrações definidos apenas pelo autor.

[1] https://tools.ietf.org/html/rfc6690

Finalmente, temos a arquitetura proposta neste trabalho, que será mais detalhada no Capítulo 4. A descoberta, gerenciamento e abstração de dados é feita através da utilização do UPnP+. A interoperabilidade acontece através de *gateways* para transcrever dados de dispositivos que utilizam outras tecnologias de comunicação. O volume de dados e *eventing* é suportado pela arquitetura, mas não foi avaliado neste trabalho, a análise mais detalhada pode ser vista no Capítulo 5. Finalmente, a adoção dos modelo de dados padrões faz com que a integração com outras plataformas aconteça de maneira homogênea e simplificada.

A segurança é outro parâmetro importante, mas que não foi mencionado nesta comparação. Todas as soluções apresentadas neste trabalho possuem suporte para comunicação segura, mesmo que esse seja realizado de maneira mais simples, apenas utilizando o protocolo de comunicação, como é o caso das soluções CoAP, ou uma forma mais complexa, como as soluções utilizando XMPP.

## 3.3 Considerações Finais do Capítulo

Neste capítulo foram apresentados detalhes das diferentes classificações de *middlewares* para Internet das Coisas. De forma mais aprofundada, foi exposto cinco trabalhos que utilizam arquiteturas *service-oriented*. Ao final, uma tabela comparativa entre os trabalhos foi exposta com uma discussão comparando as soluções com a infraestrutura proposta neste trabalho. Foi possível verificar que a integração com outras arquiteturas ainda é algo pouco explorado na literatura.

# Capítulo 4

# Arquitetura de Integração

Neste capítulo será apresenta uma arquitetura orientada a serviços para expor dispositivos para a Internet das Coisas e possibilitar sua utilização de maneira uniforme por aplicações e serviços na Internet. Inicialmente, serão mostrados os principais objetivos desta arquitetura e as lacunas que ela visa preencher. Posteriormente, apresenta-se uma visão geral da arquitetura, os principais atores envolvidos, dispositivos, redes IoT e a comunicação entre eles. Em seguida, será mostrado o funcionamento do *middleware* que faz a intermediação de informações entre aplicações ou serviços com dispositivos. Adiante, serão descritos modelos de dados e sua utilização dentro da arquitetura. Por fim, os fluxos mais utilizados dentro da arquitetura serão detalhados para exemplificar seu funcionamento.

## 4.1 Objetivos da Arquitetura

Para a construção de uma aplicação que utiliza dispositivos provenientes da IoT, desenvolvedores têm de atacar uma série de problemas na construção do software. Portanto, a arquitetura apresentada neste trabalho tem como principal objetivo facilitar o desenvolvimento de novas aplicações, otimizando o uso de tempo dedicado a lógica principal e não na integração de novos dispositivos IoT.

Um exemplo de caso de uso em que esta arquitetura é necessária é quando são dispostos múltiplos sensores, de diferentes fabricantes para capturar um mesmo dado, como por exemplo a temperatura. Todos os sensores captam a mesma informação, mas por possuírem fabricantes diferentes, ou até mesmo modelos diferentes, o formato em que seus dados estão

estruturados é diferente. Para contornar este problema, é necessário que o desenvolvedor do aplicativo tenha conhecimento prévio de todos os aparelhos utilizados e realize a comunicação com os mesmos. Utilizando uma arquitetura que exponha os dispositivos para a Internet e utilize um modelo de dados padronizado, todo o fardo de conectar-se com os aparelhos e considerar suas diferentes características, provenientes dos fabricantes, serão removidos.

Para que esta arquitetura realize o caso de uso, uma série de metas devem ser satisfeitas:

**1. Suportar a descoberta e uso de dispositivos aos consumidores de serviços**

A arquitetura deve possibilitar que aplicações clientes utilizem e descubram os dispositivos registrados na plataforma. A mesma também deve viabilizar a integração dos dispositivos com aplicações.

**2. Oferecer transparência de serviços a aplicações clientes**

O acesso aos recursos de dispositivos por aplicações deve acontecer de maneira transparente, sem que sejam expostos detalhes de implementação.

**3. Abstrair camadas inferiores das redes de dispositivos**

Todos os detalhes de heterogeneidade e *hardware* devem estar escondidos de aplicações. A arquitetura deve oferecer interfaces de alto nível para utilizar os recursos dos dispositivos cadastrados sem que as aplicações tenham de lidar diretamente com a heterogeneidade e mudanças na organização da rede.

**4. Oferecer suporte a mecanismos de auto-organização**

Mecanismos de auto-organização remetem à autonomia da arquitetura para lidar com problemas relacionados à dinamicidade. Dispositivos IoT podem entrar e sair da rede a qualquer momento devido a suas limitações de *hardware* e até mesmo interferências no meio em que estão. Devido a estas características, existe a possibilidade que alguns aparelhos estejam disponíveis apenas por um período limitado, e por isso, a arquitetura deve possuir mecanismos para lidar com a descoberta e gerência de dispositivos. Isso inclui notificar aplicações clientes caso o nó procurado não esteja disponível para utilização.

**5. Interoperabilidade com uma variedade de dispositivos**

A arquitetura deve oferecer suporte independente do tipo de dispositivo e de tecnologia de comunicação utilizada.

**6. Lidar eficientemente com grande volume de dados e altas cargas de comunicação**

As redes de sensores sem fio frequentemente possuem um grande volume de dados gerados e por isso a arquitetura deve ser eficiente na comunicação entre elas, levando em consideração a capacidade de banda de transmissão, atrasos, consumo de energia e a confiança que a informação trocada é a mesma gerada por suas fontes.

**7. Auxiliar na integração com outros sistemas**

A arquitetura deve viabilizar a integração destes dispositivos com outros sistemas ou serviços, possibilitando o uso das informações por terceiros. Um exemplo seria a utilização dos dados das WSN para o monitoramento de pacientes. Neste caso, a arquitetura deve possibilitar que aplicações médicas utilizem os dados provenientes das redes de sensores sem fio para mostrar informações do usuário sendo monitorado.

## 4.2 Arquitetura de Integração

Nesta seção, será apresentada uma visão geral da arquitetura que satisfaz os objetivos mencionados anteriormente.

A arquitetura ilustrada na Figura 4.1 utiliza o protocolo IP para se comunicar com as redes de sensores e atuadores sem fio. O *IoT Device Intermediator* conectado às WSANs, tem por objetivo gerenciá-las e ser o principal ponto de acesso para sua comunicação. Para uniformizar o uso dos dispositivos, o *Device Manager* é responsável por encapsular suas informações em Modelos de Dados disponíveis em bancos de dados na Internet. Além disto, o *Device Manager* também possui um banco de dados interno para monitorar o estado de cada aparelho, a fim de que não haja a necessidade de novas requisições aos dispositivos quando receber de aplicações e serviços externos, evidenciados na parte inferior esquerda da Figura 4.1.

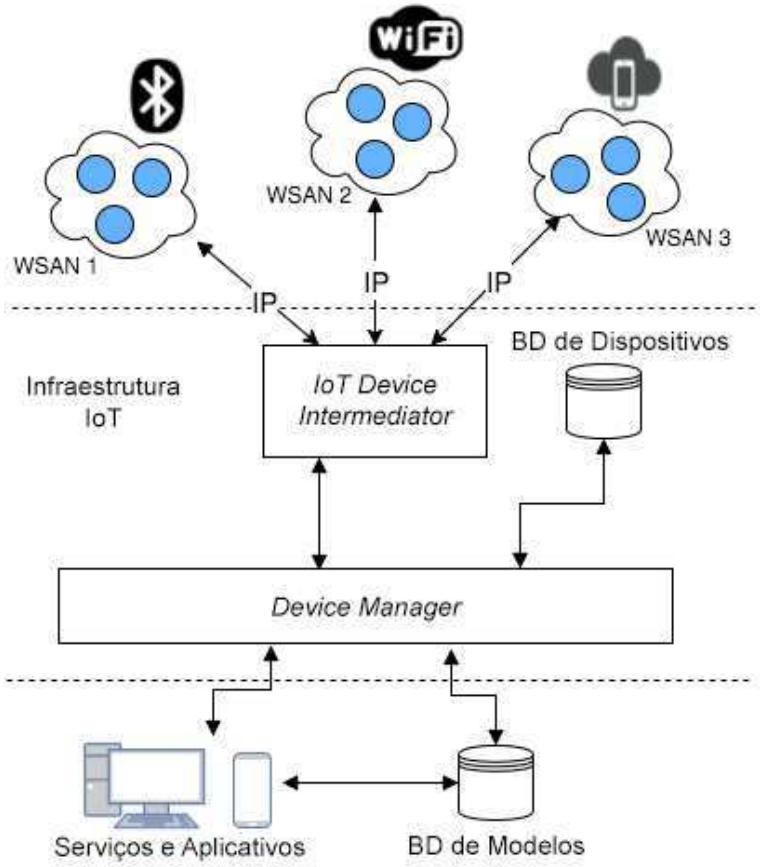


Figura 4.1: Diagrama com uma visão geral da arquitetura

### 4.2.1 Comunicação com WSANs

Devido à heterogeneidade de tecnologias e limitações de *hardware*, nem todos os dispositivos podem se comunicar através do protocolo IP, sendo necessário o uso de *gateways* para tradução e envio de informações.

Este fragmento da arquitetura, ilustrado na parte superior da Figura 4.2, é responsável por todo o tráfego entre os recursos utilizados pelas aplicações e o arcabouço proposto. Se necessário, os dados nativos de cada dispositivo são traduzidos para um formato padrão através dos *gateways*. Quando acontecem eventos de conexão, desconexão e captura de dados, pacotes são enviados para o *IoT Device Intermediator* que em seguida envia as informações para interessados. Caso aplicações ou serviços queiram executar ações em atuadores, a mensagem também é enviada pelo mesmo intermediador, que encaminha a mensagem para a WSN alvo, e caso seja necessário, o *gateway* traduz a mensagem para o formato nativo do atuador.

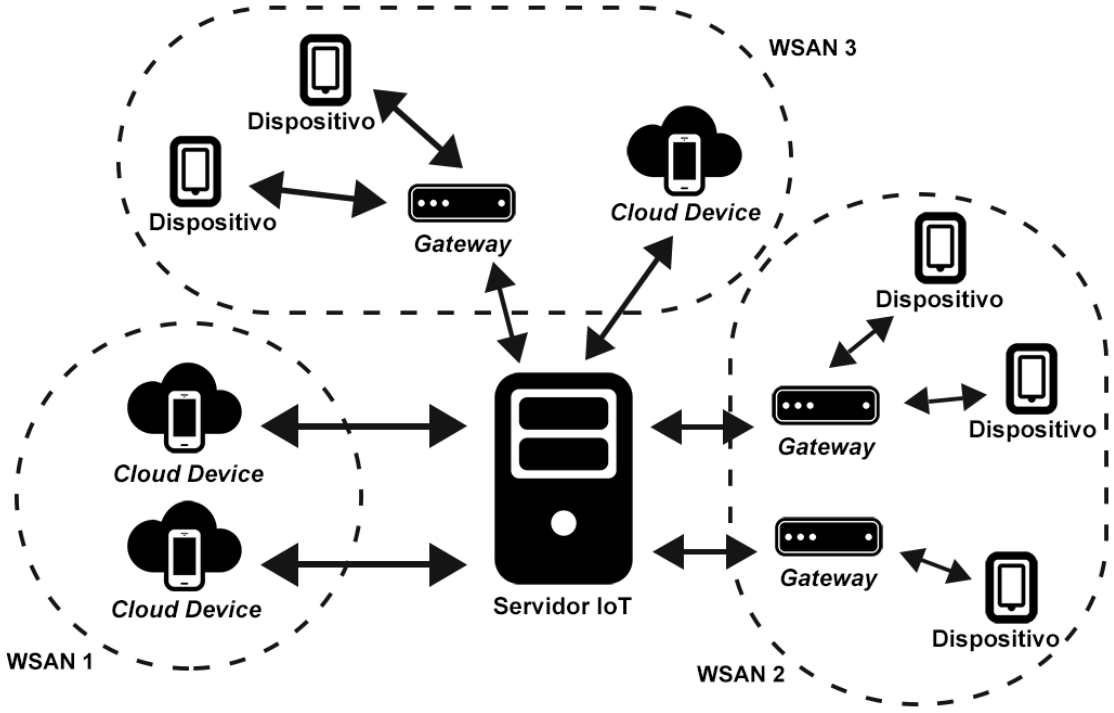


Figura 4.2: Comunicação de diferentes redes de sensores sem fio.

Na Figura 4.2 pode-se observar três tipos de WSANs que podem fazer parte da infraestrutura. A WSAN 1 é formada por *Cloud Devices*, aparelhos capazes de comunicar-se diretamente com o servidor em nuvem, estes já possuem suporte próprio para comunicação e abstração de suas informações no formato apropriado, sem a necessidade de intermédio por terceiros. Geralmente, *Cloud Devices* são aparelhos com uma capacidade de processamento maior e mais modernos. Já na WSAN 2, temos uma disposição de dispositivos associados a *gateways*, que agem como canal de comunicação. Nesta organização, toda a lógica de envio e abstração de dados pertencem ao *gateway*, permitindo assim a utilização de dispositivos mais limitados em *hardware*. Esta disposição é comumente associada a redes que utilizam tecnologias como Bluetooth, ZigBee, entre outras. Por fim, temos a WSAN 3 que possui aparelhos limitados associados a um *gateway* e também *Cloud Devices*. Apesar dos dispositivos possuírem características diferentes, eles podem fazer parte da mesma rede na infraestrutura.

### 4.2.2 Middleware para Internet das Coisas

O *middleware* a ser descrito a seguir é a principal contribuição deste trabalho. Ele é responsável por solucionar os problemas enunciados anteriormente, e está dividido em dois componentes principais, ilustrado na Figura 4.3.

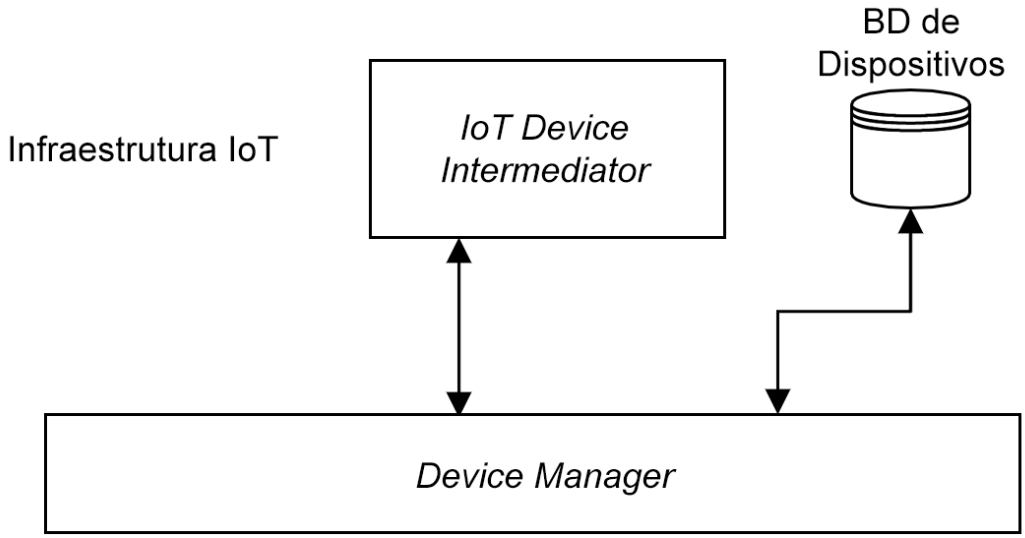


Figura 4.3: Diagrama com componentes envolvidos na integração.

Primeiro, temos o *IoT Device Intermediator*, que atua como um intermediário dentro do padrão *Publish/Subscribe*. Ele é responsável pela comunicação direta com as diferentes WSANs, e com isso, fica encarregado de gerenciar o tráfego de dados e dispositivos conectados. Este componente é agnóstico às informações trocadas e não armazena o estado de cada dispositivo. Sua eficiência vem justamente do padrão *PubSub*, no qual dispositivos interessados inscrevem-se para receber eventos de dispositivos que publicam suas informações. Este mecanismo torna a troca de dados mais eficiente, em vista das limitações de *hardware* e grande volume de dados. Além disso, o mesmo deve possuir maneiras de endereçamento único para os dispositivos conectados e habilitar a troca segura dos dados com as WSANs.

O segundo componente importante do *middleware* é o *Device Manager*. Este é conectado diretamente ao *IoT Device Intermediator* e se inscreve para obter informações de atualização (descoberta e desconexão), dos dispositivos conectados. Ele é responsável pela integração com agentes externos (por exemplo, serviços pervasivos), e abstração em modelos de dados. O banco de dados conectado é utilizado para manter o estado de dispositivos conectados, evitando assim requisições extras para obter informações frequentemente utilizadas. Mais

detalhes desse componente serão mostrados na próxima seção.

### 4.2.3 Integração Externa

Para facilitar a integração com serviços na Internet e aplicações externas, uma API de acesso é oferecida a desenvolvedores. Os principais componentes envolvidos nessa integração são apresentados no diagrama da Figura 4.4.

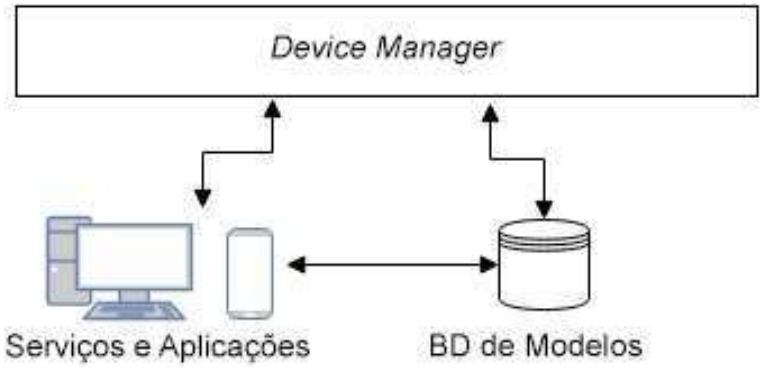


Figura 4.4: Diagrama com componentes envolvidos na integração externa.

O *Device Manager* atua como uma camada de abstração e acesso para aplicações e serviços na Internet. Ele é responsável por encapsular os dados dos dispositivos em modelos de dados, disponíveis na Internet. Este encapsulamento é importante para uniformizar a maneira em que os dispositivos IoT são acessados e assim abstrair seu *hardware* e especifidades de cada fabricante. Mais detalhes sobre os modelos de dados serão dados nas próximas seções.

Esta funcionalidade de abstração de dados em um formato único e padronizado é também necessária para facilitar a integração com aplicações e serviços que já utilizam os modelos de dados disponíveis na Internet com arcabouços diferentes, como IoTivity[1], AllJoyn[2], Weave[3] entre outras. Esta conveniência otimiza o tempo que desenvolvedores levam para integrar suas soluções com novas arquiteturas existentes.

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

[2]https://allseenalliance.org/framework

[3]https://developers.google.com/weave/

### 4.2.4 Serviços de Integração

A flexibilidade desta infraestrutura possibilita a adição de outros serviços que utilizam os dados provenientes das WSANs e são necessários para tipos diferentes de aplicações. A agregação e extração de informações semântica dos dados é um dos principais serviços utilizados com os dados de redes IoT. Como mostrado na Figura 4.5, utilizando a agregação de dados podemos adicionar um novo componente em nossa infraestrutura, que se inscreve para receber dados de dispositivos interessados, e extrai informações dos mesmos, deixando-as disponíveis para aplicações.

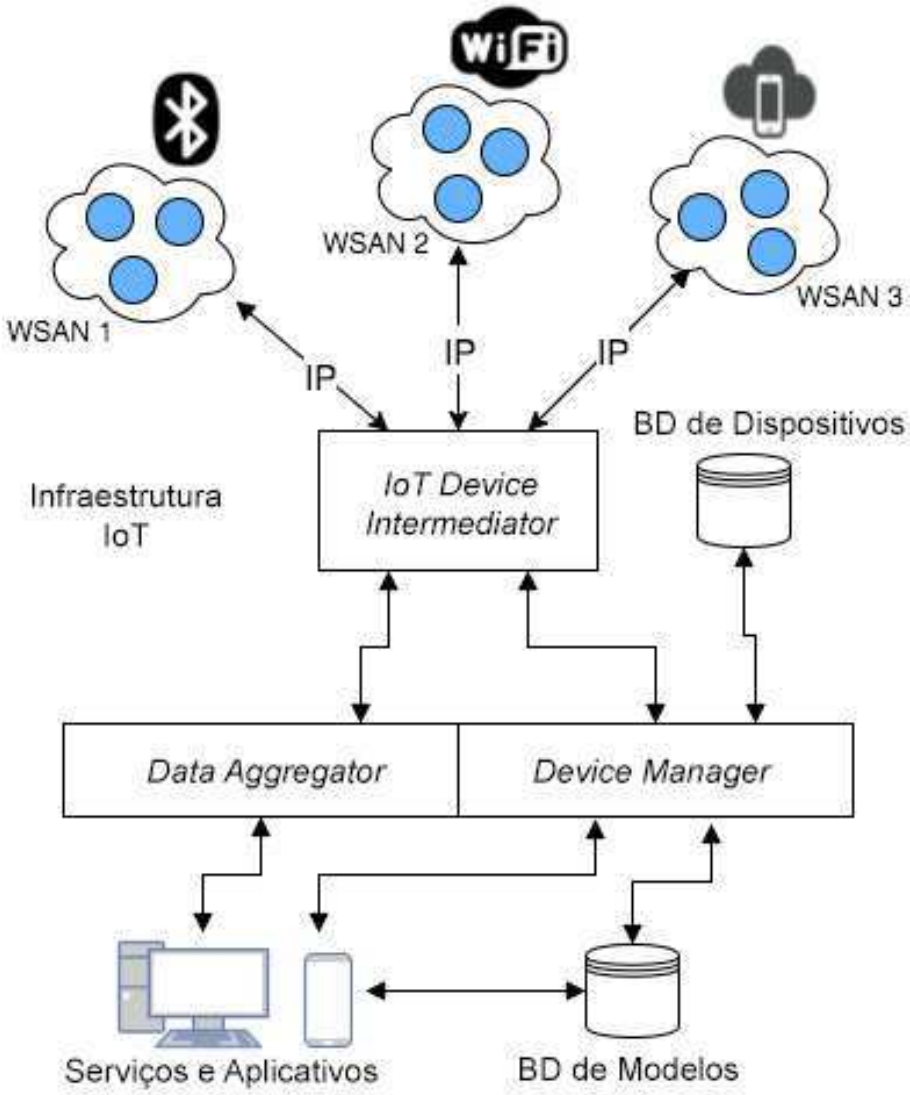


Figura 4.5: Arquitetura contendo um novo serviço de agregação de dados.

Os campos de *Machine Learning* e *Big Data* podem fazer uso desta funcionalidade para

alimentar seus sistemas com informações em tempo real, e assim garantir sempre uma inferência mais precisa para usuários.

### 4.2.5 Modelos de Dados

Para solucionar a dificuldade na comunicação de dispositivos que possuem características semelhantes, mas enviam seus dados em estruturas diferentes, são utilizados os *Modelos de Dados* (MD).

Os MD auxiliam na padronização da comunicação com serviços e aplicações interessadas em consumir as informações de sensores e executar ações em atuadores em redes IoT.

Grandes empresas já notaram o grande problema que esta falta de uniformidade causa, e assim surgiram iniciativas como a Open Connectivity Foundation[4]. Esta é uma das maiores iniciativas para padronização da informação, possuindo um banco de dados em versão inicial já disponível na Internet, o oneIoTa[5]. Outras iniciativas também já tentam padronizar seus modelos de dados, como o AllJoyn da Allseen Alliance[6], que utiliza formatos em XML para especificar seus DMs, um exemplo pode ser visualizado no Código 4.1. Neste trabalho, foi utilizado o banco de dados da OCF em virtude da sua facilidade de integração, popularidade da iniciativa, e acervo já existente de DMs disponíveis. Trabalhos que utilizam o oneIoTa ainda são escassos na literatura, visto que o mesmo teve suas versões iniciais divulgadas apenas no ano de 2016.

Código Fonte 4.1: Modelo de Dados do AllJoyn para uma porta inteligente.

```
<node>
  <interface name="org.allseenalliance.door">
    <property name="Location" type="s"/>
    <property name="Open" type="b"/>

    <signal name="PersonPassedThrough">
      <arg name="Who" type="s"/>
    </signal>

    <method name="ChangeState">
```

[4] https://openconnectivity.org/

[5] http://oneiota.org/

[6] https://allseenalliance.org

```
      <arg name="Open" type="b"/>
    </method>
  </interface>
</node>
```

O *Modelo de Dados* é utilizado para padronizar dispositivos que possuem características semelhantes. Na Figura 4.6 podemos notar três lâmpadas diferentes, onde duas delas são do mesmo modelo, mas uma terceira, possivelmente de outra fabricante, utiliza uma outra maneira de estruturar os mesmos dados que são compartilhados entre os 3 modelos de lâmpadas. O modelo de Lâmpada Regulável, surge para generalizar as características semelhantes as 3 lâmpadas e organizar as informações para que serviços e aplicações que desejam utilizar lâmpadas deste modelo, consigam se comunicar com elas sem o conhecimento prévio das características intrínsecas de cada uma delas.

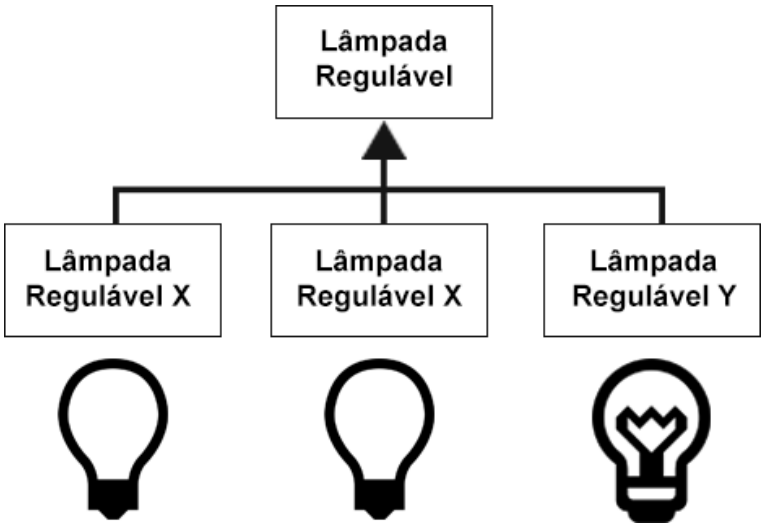


Figura 4.6: Categoria de modelos de dados para um mesmo dispositivo

No Código 4.2 é mostrada a definição simplificada de um modelo para sensores de temperatura feita pela Open Connectivity Foundation[7]. No inicio, é evidenciado o título do modelo, no campo *title*, em formato legível a usuários finais. Em seguida, temos a definição do modelo de dados caracterizado pela sua identificação única, *oic.r.temperature*. Dentro dela, é possível visualizar que se trata de um objeto, que possui a propriedade *temperature* de um tipo simples numérico. Mais detalhes do formato dos modelos de dados da OCF

[7]https://openconnectivity.org/

podem ser encontrados no site do oneIoTa[8].

Código Fonte 4.2: Definição de um modelo de dados simplificado da Open Connectivity Foundation.

```
{
  "schema": "http://json-schema.org/draft-04/schema#",
  "description" : "Copyright (c) 2016 Open Connectivity Foundation, Inc.
      All rights reserved.",
  "title": "Temperature",
  "definitions": {
    "oic.r.temperature": {
      "type": "object",
      "properties": {
        "temperature":  {
          "type": "number",
          "description": "Current temperature setting or measurement"
        }
      }
    }
  },
  "type": "object",
  "allOf": [
    {"$ref": "oic.core.json#/definitions/oic.core"},
    {"$ref": "oic.baseResource.json#/definitions/oic.r.baseresource"},
    {"$ref": "#/definitions/oic.r.temperature"}
  ],
  "required": ["temperature"]
}
```

Além disso, aplicações que possuam conhecimento mais específico, podem também fazer uso de propriedades e funcionalidades únicas do fabricante que podem não ser características genéricas da categoria do aparelho, graças a hierarquização e multiplicidade dos modelos de dados que um único dispositivo pode ter. Por exemplo, um sensor que captura temperatura e pressão ambiente pode tanto se enquadrar no modelo de dados de Temperatura, quanto no de Pressão Ambiente.

[8]http://oneiota.org/

Código Fonte 4.3: Utilização do modelo de temperatura no formato JSON.

```
{
    "id": "unique_example_id",
    "temperature": 18.0
}
```

A definição mostrada no Código 4.2 se trata de uma descrição de como os serviços interessados devem processar os dados que se enquadram naquele modelo de dados. Já no Código 4.3 é possível visualizar um exemplo da utilização de dados de temperatura, estruturados de maneira que se encaixe na definição mostrada anteriormente.

As abstrações nestes formatos podem ser facilmente processadas e utilizadas por sistemas, mesmo que novos dispositivos sejam incorporados no mercado com o passar do tempo.

## 4.3 Funcionamento

Nesta seção, iremos descrever alguns fluxos de dados comumente utilizados dentro da infraestrutura proposta neste trabalho.

### 4.3.1 Descoberta, Desconexão e Eventos Sensoriais

Para ilustrar o fluxo de descoberta, a Figura 4.7 apresenta a sequência de passos necessários para que o sistema reconheça um novo dispositivo conectado. No passo 1, quando um novo sensor conecta-se na WSAN, o *gateway* recebe a requisição na tecnologia de comunicação utilizada (por exemplo, ZigBee ou Bluetooth) e encapsula a informação em um formato padronizado. No passo 2, o *gateway* envia o novo pacote, em conjunto com as características do nó conectado, para o *IoT Device Intermediator* via TCP/IP. *Cloud Devices* possuem a capacidade de encapsular seus dados no formato apropriado e enviar diretamente suas informações, sem a necessidade de dois passos nesta etapa inicial.

Finalmente, no passo 3, o *IoT Device Intermediator* encaminha o evento para serviços interessados, no caso da nossa infraestrutura básica, a mensagem é reenviada para o *Device Manager*. Ao chegar neste último componente, ele pode guardar o estado deste novo dispositivo em seu banco de dados, para assim evitar novas requisições ao receber chamadas REST.

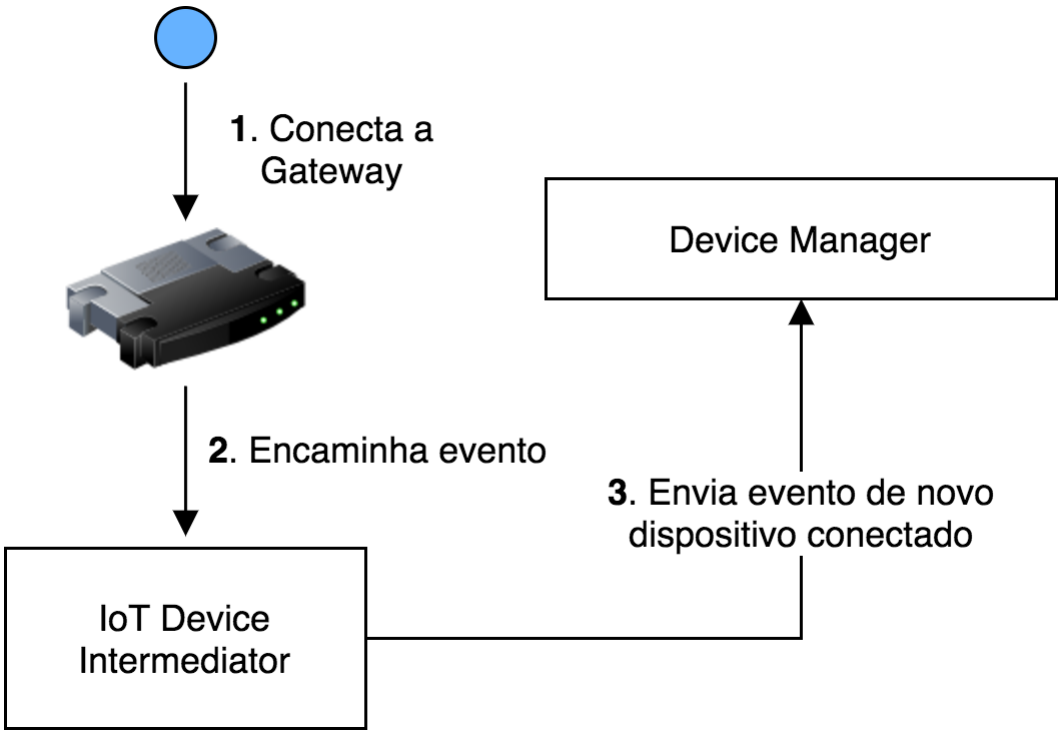


Figura 4.7: Fluxo de mensagens para descoberta de novos dispositivos.

O fluxo para a desconexão e eventos sensoriais é bastante semelhante, as únicas mudanças ficam em sua semântica e no conteúdo das mensagens trocadas.

### 4.3.2 Ativação de um Atuador

A Figura 4.8 ilustra o fluxo para aplicações ou serviços invocarem uma ação de um atuador. Assim como no fluxo anterior, o *middleware* age como um intermediador para tradução de informações e controle do fluxo de mensagens.

Inicialmente, a aplicação deve estruturar suas informações baseadas no Modelo de Dados utilizado pelo *middleware*. Em seguida, no passo 2, o serviço ou aplicação envia a ação através de uma API Web para o *Device Manager*, que transcreve a mensagem para um formato interno padrão. No passo 3, as informações da ação são encaminhadas para o *IoT Device Intermediator* via TCP/IP, que já no passo 4, encaminha a ação para o *gateway*, ou *Cloud Device*. Caso o dispositivo final se encontre em uma rede interna, o *gateway* faz a tradução de informações para o formato apropriado (ie. Bluetooth), e a envia para o receptor final. *Cloud Devices* que recebem o pacote diretamente do *IoT Device Intermediator*, são capazes

de processar as informações e executar a ação final.

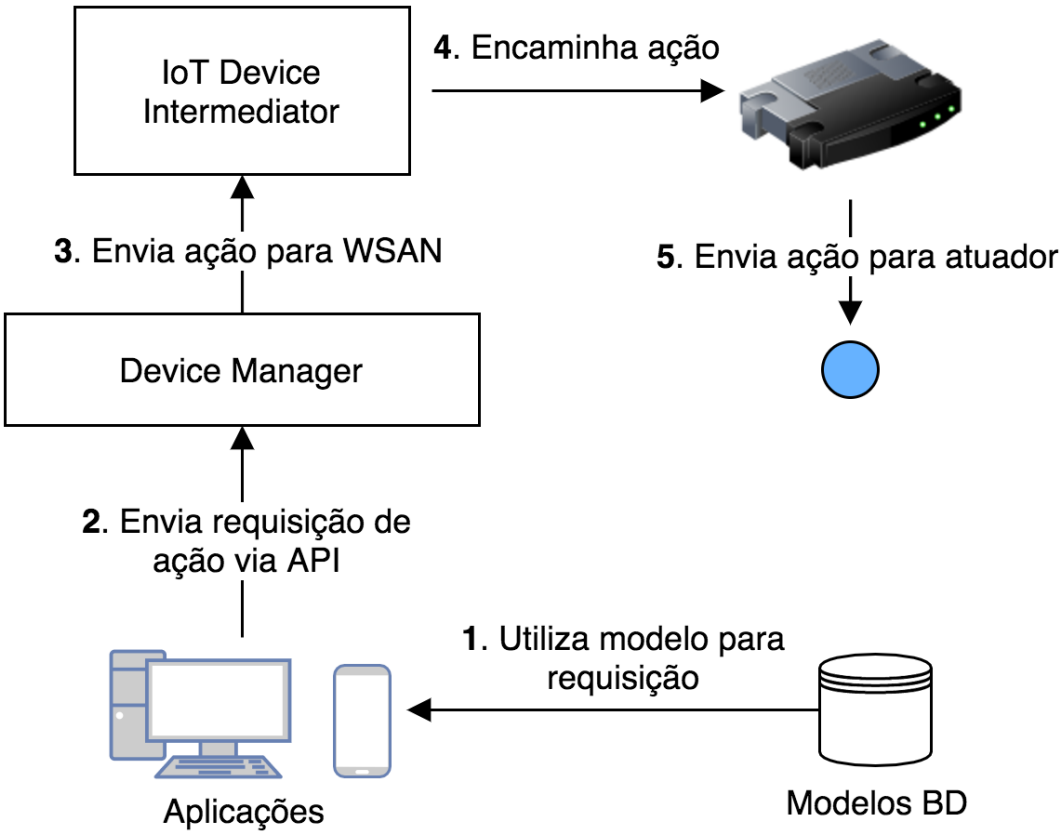


Figura 4.8: Fluxo de mensagens para execução de ação em um atuador.

## 4.4 Considerações Finais do Capítulo

No início deste capítulo, os objetivos da arquitetura foram apresentados. Em seguida, foram expostos os detalhes da arquitetura que satisfaz os objetivos mencionados anteriormente, seus principais atores e a comunicação entre eles. Por fim, é especificado o funcionamento de alguns dos fluxos mais utilizados na arquitetura, com o objetivo de exemplificar, em detalhes, cada uma das etapas que a arquitetura executa para realizar fluxos comuns de *middlewares* para a Internet das Coisas, como o de descoberta, envio de eventos, entre outros.

Ao final deste Capítulo, pode-se observar uma arquitetura que planeja satisfazer os objetivos citados anteriormente. A utilização do *IoT Device Intermediator* em conjunto com outros componentes pretende lidar com o grande volume de dados que as WSANs enviam,

oferecer suporte a mecanismos de auto-organização e também auxiliar no suporte a descoberta e uso de dispositivos. Modelos de Dados são utilizados para oferecer oferecer transparência, abstrair camadas inferiores e auxiliar na integração com outros sistemas que utilizam os mesmos modelos de dados padrões. Finalmente, a interoperabilidade com vários dispositivos é atingida através do uso de *gateways* na borda das WSANs, fazendo o intermedio e tradução de suas informações, quando necessário, para comunicação com o IoT Device Intermediator.

Baseado na proposta para atingir os requisitos citados, foi realizada uma avaliação, a fim de verificar se a arquitetura consegue de fato cumprir os objetivos elencados no inicio do Capítulo. Está avaliação será apresentada na próxima sessão.

# Capítulo 5

# Avaliação da Arquitetura

Nas próximas seções será descrita a concretização da infraestrutura proposta. Primeiro, serão enunciadas as tecnologias utilizadas e como elas foram integradas para formar o *middleware*. Em seguida, será descrito o fluxo de mensagens utilizando as tecnologias empregadas. Por fim, os experimentos executados para avaliar a arquitetura são discutidos.

## 5.1 Avaliação do Middleware

Para concretizar a infraestrutura proposta, foi empregado em sua base o UPnP (*Universal Plug and Play*) [22]. O UPnP possui hoje um grande número de dispositivos já prontos para serem utilizados, entre eles, sensores, *settop boxes* e *stereo systems*. Além disso, a tecnologia está sendo cada vez mais incorporado dentro do meio IoT. Desde 2016, o UPnP foi integrado a uma das iniciativas mais promissoras para a padronização da comunicação na IoT, a OCF[1] (Open Connectivity Foundation).

Tendo na sua lista de membros empresas como Intel, Samsung, Cisco, Microsoft, Qualcomm entre outras, a OCF é um dos maiores projetos com o objetivo de projetar uma especificação para possibilitar a comunicação entre bilhões de dispositivos que estarão conectados a Internet das Coisas. Apesar de ainda estar em fase de desenvolvimento, já é possível notar em parte de sua padronização, especificações provenientes do UPnP[2]. Além disso, a iniciativa planeja o desenvolvimento de pontes[3] para incorporar dispositivos de fabricantes que

[1] https://openconnectivity.org/

[2] https://openconnectivity.org/upnp/specifications

[3] https://openconnectivity.org/upnp/iot

utilizam as especificações do UPnP.

O UPnP foi escolhido neste trabalho devido a sua capacidade de integração com dispositivos no mercado que já possuem suporte à certificação, além de possuir comunidade ativa em seu desenvolvimento.

Com as informações do UPnP/UPnP+ introduzidas no Capítulo 2, é possível realizar um mapeamento entre a arquitetura proposta nesse trabalho e o UPnP+, como apresentado na Tabela 5.1.

O *Intermediator* da arquitetura apresentada no Capítulo 4 fica mapeado a um servidor *UPnP Cloud*, que é um servidor que utiliza o XMPP[4] (*Extensible Messaging and Presence Protocol*). Já os *Cloud Devices* e *Gateways* do Capítulo 4, podem ser vistos como *UPnP Cloud Devices*, que são dispositivos capazes de comunicar-se diretamente com o servidor XMPP, e por isso devem ser mais robustos.

| Componente da Infraestrutura | Mapeamento para UPnP+ |
|---|---|
| Intermediator | UPnP Cloud (XMPP) |
| Cloud Device | UPnP Cloud Device |
| Gateway | UPnP Cloud Device |
| Device Manager | UPnP Cloud Service |

Tabela 5.1: Tabela de mapeamento para UPnP+.

Para a avaliação da nossa arquitetura utilizando UPnP e UPnP+, foi utilizado o *ejabberd* 16.16[5] para o servidor XMPP definido para a *UPnP Cloud*. Já para o desenvolvimento da API Web, foi utilizado um servidor em Java, utilizando as bibliotecas *Jersey*[6] para a API REST, e *Smack* 4.1.5[7] para a comunicação com o servidor XMPP.

### 5.1.1 Caso de Uso

Com o objetivo de demonstrar o uso da arquitetura, um caso de uso foi definido para exemplificar casos reais de utilização.

[4]https://xmpp.org/

[5]https://www.ejabberd.im/

[6]https://jersey.java.net/

[7]http://www.igniterealtime.org/projects/smack/

O caso de uso definido neste trabalho envolve o monitoramento de temperatura corporal. Para isso, foram simulados sensores com especificações diferentes, que capturam temperatura do corpo humano. O sensor mais simples captura apenas informações da temperatura, enquanto um sensor mais complexo consegue capturar tanto as informações do sensor mais simples mas também a localização de onde a temperatura foi medida. Ambos os sensores são utilizados para calcular uma temperatura mais precisa do corpo humano, baseada em múltiplas fontes de dados.

Para a demonstração do caso de uso definido foi implementada a *Nuvem UPnP+*, que se comunica com dispositivos através do protocolo XMPP e os expõe através de uma interface REST. Os sensores foram simulados através de uma aplicação Android que atua como um *Cloud Device* trocando dados tal como os dispositivos UPnP. A seguir são apresentados mais detalhes das implementações.

### 5.1.2 Nuvem UPnP+

Esta implementação teve como base o exemplo criado pela TP Vision[8] para Android[9]. A empresa criou aplicativos Android que utilizavam UPnP+ na nuvem para intermédio da comunicação. Um dos aplicativos funciona como simulador de uma lâmpada, e outro como um controlador da lâmpada.

Como os exemplos foram criados no final de 2013, algumas bibliotecas estavam desatualizadas, e por isso uma migração foi necessária para utilizar suas novas versões e também a conversão para utilizar serviços baseados inteiramente em Java SE, em vez de Android.

A implementação do UPnP+ foi dividida em 3 módulos principais:

- Módulo REST;
- Módulo de Comunicação XMPP;
- Módulo de Modelagem.

O módulo XMPP utiliza a biblioteca Smack 4.1.5 para comunicar-se com o servidor XMPP. O mesmo é composto em sua grande maioria por objetos que tem por objetivo converter, processar e encapsular objetos para o formato XMPP.

[8] http://www.tpvision.com/

[9] https://bitbucket.org/gmekenkamp/upnp-smgt

Os modelos do sistema são criados e gerenciados pelo módulo de modelagem. Seus gerenciadores são responsáveis por encapsular os dispositivos UPnP em objetos Java.

Por fim, o módulo REST é responsável por manter o serviço Web disponível para aplicações externas. Um outro componente importante deste módulo é o de transcrição de dados para o *oneIoTa*. Quando visualizados pela API REST, os dispositivos tem suas informações estruturadas seguindo os modelos de dados padrões em formato JSON (*JavaScript Object Notation*).

### Sequência de ações ao encontrar novo dispositivo

Para exemplificar o funcionamento da Nuvem UPnP+, a Figura 5.1 ilustra um exemplo da troca de informações entre o módulo XMPP, o *UPnP Manager*, que é utilizado pelo módulo REST, e o *Device Description Fetcher*, parte do módulo de Modelagem. As entidades envolvidas no fluxo apresentado fazem parte da implementação do *Device Manager* neste trabalho.

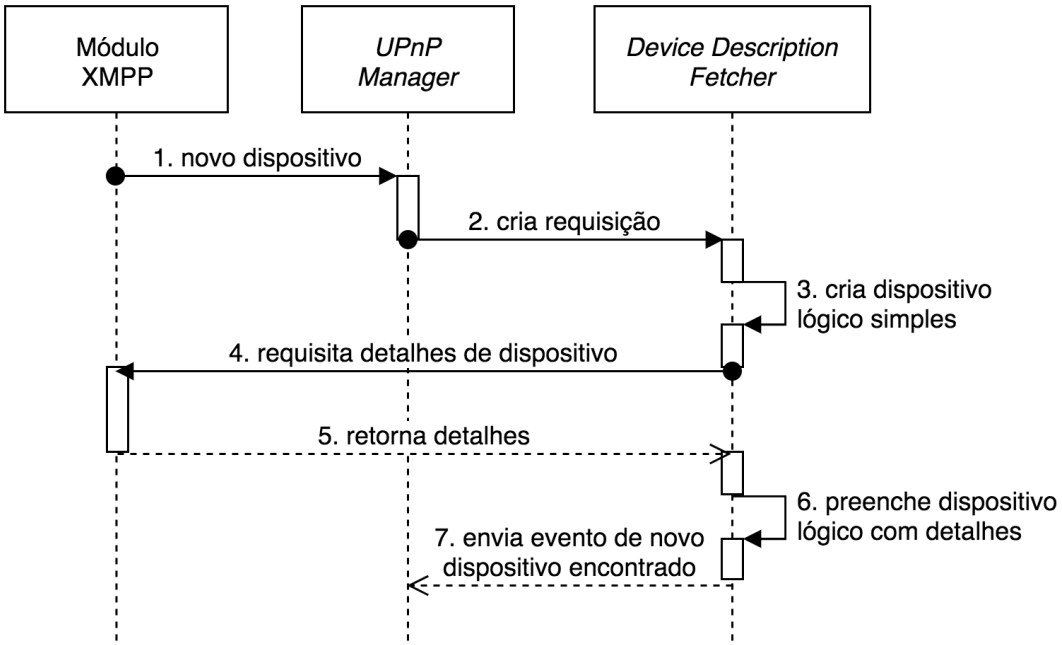


Figura 5.1: Diagrama de sequência de um novo dispositivo.

No passo 1, o sistema recebe uma mensagem de presença XMPP com a disponibilidade de um novo dispositivo na rede. Em seguida, no passo 2, o *UPnP Manager* requisita ao *Device Description Fetcher* para obter mais informações do dispositivo. Este, por sua vez, cria

um modelo básico do dispositivo, apenas com as informações disponíveis naquele momento, mostrado no passo 3. Em seguida, na quarta etapa, requisita ao dispositivo sua descrição detalhada por meio do módulo XMPP. Ao receber a resposta, no passo 5, ele preenche a entidade criada previamente com os novos detalhes, na etapa 6, e finalmente no passo 7, a envia ao *UPnP Manager*, entidade que gerencia os dispositivos UPnP, e tem contato direto com o módulo REST.

### Captação de dados de sensores

Para seguir a especificação definida pelo *IoT Management and Control* do UPnP, sensores devem possuir na sua descrição XML a ação *ReadSensor*, que realiza a leitura de dados ao ser invocada [48]. Esta ação recebe como parâmetros o sensor específico do dispositivo e retorna os valores mais atuais que foram capturados pelo aparelho. O fluxo de dentro do *Device Manager*, implementado neste trabalho, pode ser visualizado na Figura 5.2 e tem o envolvimento do Módulo XMPP, o *UPnP Manager* e o *BaseSensorPooling*, parte do módulo de Modelagem.

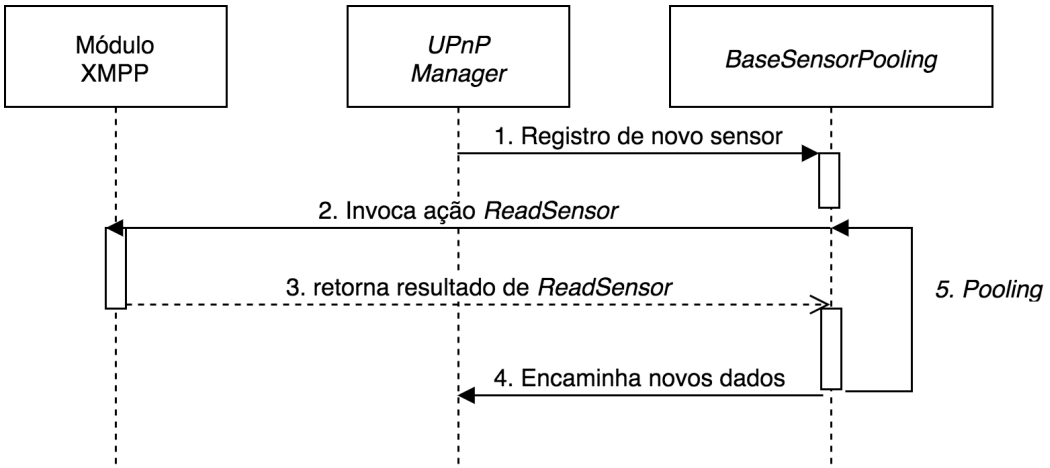


Figura 5.2: Diagrama de sequência da requisição de dados de um sensor.

Primeiro, assim que um novo sensor é registrado, o módulo *UPnP Manager* envia uma mensagem ao *BaseSensorPooling* para que ele faça a atualização dos dados. O *BaseSensor-Pooling* tem por objetivo fazer as requisições dos dados de sensores e as manter sempre no estado mais atualizado. Para isso, ele utiliza o mecanismo de *Pooling*, que faz requisições durante um período fixo de tempo para que os dados não fiquem desatualizados por muito

tempo. Quando chega o momento de requisitar novos dados, o *BaseSensorPooling* cria uma requisição de invocação da ação *ReadSensor*, mostrado no passo 2, com as informações do sensor registrado pelo *UPnP Manager*. Essa mensagem é encaminhada via XMPP para o dispositivo através do servidor XMPP. Quando o aparelho recebe a mensagem, ele processa as informações e envia como resposta os dados mais recentes do sensor, como mostrado no passo 3, que então é recebido no *BaseSensorPooling*, e finalmente no passo 4, é enviado ao *UPnP Manager* para atualização dos dados disponíveis a aplicações e serviços na Internet. No passo 5, é mostrado como funciona a etapa de *Pooling*, que faz repetir a sequência de passos anteriores.

Uma oportunidade para trabalhos futuros será avaliar a utilização do UPnP com GENA[10] (*General Event Notification Architecture*), que retira a necessidade de requisições da nuvem para cada dispositivo. Utilizando este princípio, os dispositivos enviam seus novos valores no momento em que eles forem capturados, evitando assim tráfego desnecessário na rede quando não existirem dados novos.

### API REST

Com o objetivo de expor os dispositivos na Internet, foi criada uma interface Web para utilização dos aparelhos por serviços externos.

A interface foi desenvolvida seguindo os padrões REST (*Representational State Transfer*), com um modelo de representação no formato JSON, para uma melhor adequação aos padrões Web atuais e a modelagem feita pelo *oneIoTa*.

| Endpoint | Descrição simples |
|---|---|
| /devices/all | Lista dispositivos conectados na rede. |
| /devices?id=XID | Mostra detalhes do dispositivo com ID=XID |

Tabela 5.2: Tabela mostrando os *endpoints* da API REST desenvolvida.

A Tabela 5.2 evidencia a descrição dos *endpoints* mais importantes da API. O primeiro *endpoint* lista todos os dispositivos conectados para que serviços externos possam tomar conhecimento dos recursos disponíveis naquele momento. Um exemplo de resposta para

[10]https://tools.ietf.org/html/draft-cohen-gena-p-base-01

este *endpoint* pode ser visto no Código Fonte 5.1. As informações mais relevantes apresentadas para cada dispositivo são seu código de referência único, chamado de ID, e o modelo de dados que ele se enquadra. Para o exemplo da requisição, temos sensor de ID "382a2eec8f2f3efbb2981db382ebff4c" que se enquadra no modelo de dados *oneIoTa* "oic.r.temperature.json". Os identificadores gerados por este *middleware* são funções MD5 dos metadados dos dispositivos, como UUID, e outros. Estas informações quando colocadas em conjunto, podem identificar unicamente um sensor UPnP, mesmo que o mesmo esteja sendo utilizado em conjunto com outros em um mesmo dispositivo físico.

Código Fonte 5.1: Exemplo de requisição para *endpoint* /devices/all

```
1  Request:
2  GET /devices/all HTTP/1.1
3  Host: localhost:8080
4  Connection: close
5  User-Agent: Paw/3.0.6 (Macintosh; OS X/10.11.6) GCDHTTPRequest
6
7  Response:
8  HTTP/1.1 200 OK
9  Content-Type: application/json
10 Connection: close
11 Content-Length: 87
12
13 {
14   "devices": [
15     {
16       "id": "382a2eec8f2f3efbb2981db382ebff4c",
17       "type": "oic.r.temperature.json"
18     }
19   ]
20 }
```

O segundo *endpoint* descreve em detalhes um dispositivo a partir do seu modelo de dados. Neste método da API, é necessário o envio do identificador único do aparelho a ser descrito. Um exemplo de requisição é descrita no Código Fonte 5.2. É mostrado o identificador do dispositivo, um identificador do modelo de dados utilizado para estruturar suas informações, e finalmente, os dados do aparelho. Neste exemplo, temos um aparelho que segue o modelo

de dados de um sensor de temperatura simples. A definição deste modelo de dados é descrita no Código Fonte 4.2.

Código Fonte 5.2: Exemplo de requisição para *endpoint* /devices?id=XID

```
1  Request:
2  GET /devices?id=382a2eec8f2f3efbb2981db382ebff4c HTTP/1.1
3  Host: localhost:8080
4  Connection: close
5  User-Agent: Paw/3.0.6 (Macintosh; OS X/10.11.6) GCDHTTPRequest
6
7  Response:
8  HTTP/1.1 200 OK
9  Content-Type: application/json
10 Connection: close
11 Content-Length: 104
12
13 {
14   "id": "382a2eec8f2f3efbb2981db382ebff4c",
15   "type": "oic.r.temperature.json",
16   "body": {
17     "temperature": "-20.9"
18   }
19 }
```

### 5.1.3 Simulação de Dispositivos

Como sensores que possuem suporte ao *UPnP Cloud* ainda não estão disponíveis no mercado, este trabalho utiliza a simulação através de um aplicativo Android para gerar dados tal como um sensor UPnP.

Para o desenvolvimento do simulador, foi tomado como base os exemplos contidos no repositório da comunidade UPnP[11]. Os dispositivos simulados no código da comunidade foram feitos utilizando o *framework* Qt[12], que utiliza C++ como linguagem base, e por isso foi necessária uma ampla migração para utilizar serviços Android, em Java, e bibliotecas

[11] https://github.com/upnpforum

[12] https://www.qt.io/

semelhantes. Além disso, módulos do código criado pela TP Vision, mencionados no desenvolvimento do *middleware*, também foram utilizados para modelagem dos sensores.


Figura 5.3: Tela de seleção do aplicativo para simular sensores.

O aplicativo foi criado para simular dois dispositivos, os quais podem ser ilustrados através da captura de tela apresentada na Figura 5.3. Para isso, foram selecionados os sensores de temperatura simples e temperatura corporal. O primeiro gera, em intervalos fixos, valores aleatórios para temperatura. O segundo, além de gerar dados tal como o primeiro, também gera um valor de localização no corpo humano para identificar onde o valor de temperatura foi capturado. Para o simulador construído, as localizações de exemplo definidas foram: boca, testa e orelhas.

Os valores gerados são colocados dentro de *DataItems*, que são elementos dentro de

arquivos de descrição detalhados na especificação *IoT Management and Control* do UPnP [48], que são posteriormente enviados para o *middleware*, que verifica o tipo de dispositivo e converte os valores dos *DataItems* para o modelo de dados *oneIoTa*. Na Figura 5.4 é apresentada uma captura de tela do aplicativo com um sensor de temperatura conectado ao servidor XMPP.

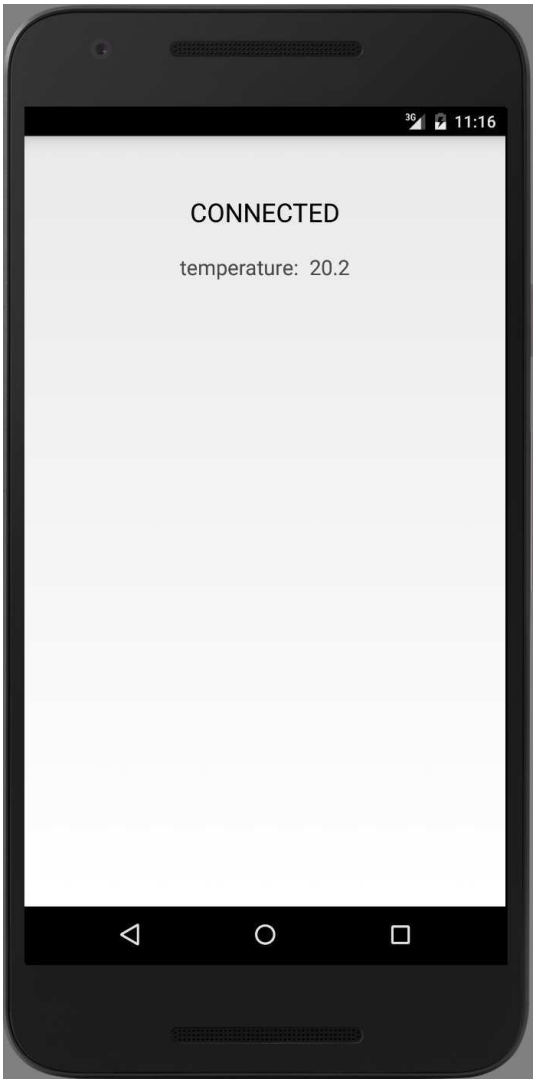


Figura 5.4: Captura de tela exibindo o sensor simulado conectado e seu valor atual.

A partir do momento que o simulador está conectado, os fluxos de descoberta, descritos a seguir, acontecem e os dados gerados se tornam disponíveis para serem acessados.

### 5.1.4 Fluxo de Mensagens

Nesta seção serão apresentados os fluxos de mensagens mais utilizados na arquitetura desenvolvida. Os testes foram executados em um computador com OS X 10.11, processador Intel i7 3GHz e 8GB de memória RAM, os pacotes foram capturados utilizando o Wireshark[13] 2.0.5. Toda a comunicação é feita sobre o protocolo TCP/IP. Para fins explicativos, apenas mensagens que contêm conteúdo XMPP em seu corpo foram analisadas a seguir.

#### Descoberta e Desconexão

A descoberta é definida como a ação do dispositivo em entrar na lista de aparelhos no *middleware*, ficando disponível para utilização. Este fluxo envolve uma autenticação inicial com o servidor XMPP, o envio da presença para o *Device Manager*, a requisição da descrição e finalmente, sua resposta.

Logo depois do servidor XMPP registrar um novo dispositivo, uma notificação de presença é enviada ao *Device Manager*. O corpo desta requisição é apresentada no Código Fonte 5.3: na linha 1 é possível ver o tipo da requisição *presence*, enquanto na linha 2 é possível ver o nome do sensor. Na resposta, é feita uma requisição de sua descrição, que finalmente é respondida com mais detalhes sobre o sensor. Um exemplo de requisição utilizada nos testes pode ser visualizada no Código Fonte 5.4: a *tag GetValuesResponse* na linha 5 indica a resposta para a requisição, e no seu corpo, na linha 7, temos a descrição do dispositivo, que foi omitida do Código Fonte 5.4 por questões de simplicidade.

Código Fonte 5.3: Requisição de presença de um novo dispositivo para *Device Manager*.

```
1 <presence from='test@brigadeiro.local/urn:schemas-upnp-
    org:device:SensorManagement:1:uuid:6e151c45-1b2a-4c29-aad2-27
    e342e0885b' to='test@brigadeiro.local/WebAPI' id='UuVoH-4'>
2       <status>Temperature Sensor</status>
3       <priority>42</priority>
4 </presence>
```

Ao final deste fluxo, o dispositivo fica disponível na listagem feita através da API Web. As informações relacionadas aos dados capturados ainda não se tornam acessíveis neste momento, apenas após uma leitura de dados estes dados se tornam disponíveis.

[13] https://www.wireshark.org/

Código Fonte 5.4: Resposta de requisição de detalhes proveniente de sensor.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <iq id="FALi4-15" to="test@brigadeiro.local/WebAPI" from="test@brigadeiro
       .local/urn:schemas-upnp-org:device:SensorManagement:1:uuid:6e151c45-1
       b2a-4c29-aad2-27e342e0885b" type="result">
3      <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" xmlns=
           "upnpcloud" s:encodingStyle="http://schemas.xmlsoap.org/soap/
           encoding/">
4          <s:Body>
5              <u:GetValuesResponse xmlns:u="GetValues">
6                  <ParameterValueList>
7                      ... Device Description XML Encoded ...
8                  </ParameterValueList>
9              </u:GetValuesResponse>
10         </s:Body>
11     </s:Envelope>
12 </iq>
```

O fluxo de desconexão é bastante similar, sendo necessária apenas uma mensagem no formato XMPP mudando o estado de presença do dispositivo. O Código Fonte 5.5 apresenta um exemplo do corpo da mensagem.

Código Fonte 5.5: Requisição de desconexão.

```
1  <presence id="0JPzG-15" type="unavailable"></presence>
```

Após o informe do dispositivo, a mensagem é encaminhada pelo servidor XMPP para todos os dispositivos interessados. O *Device Manager* ao receber a mensagem, retira da lista o dispositivo para que ele não possa mais ser acessado por aplicações ou serviços.

### Leitura de dados do sensor

Para que as informações capturadas pelo sensor estejam disponíveis à usuários finais, elas precisam ser lidas previamente. Para que isso aconteça, uma leitura dos dados deve ser realizada pelo *Device Manager*.

O fluxo de mensagens para leitura não envolve requisições de autenticação, visto que as entidades envolvidas já estão conectadas com o servidor XMPP. Seguindo a especificação

*IoT Management and Control* do UPnP [48], é necessária a realização de uma invocação de método chamado *ReadSensor* e como parâmetro, devem ser especificados os dados relacionados à informação capturada pelo sensor. Um exemplo desta requisição é evidenciada no Código Fonte 5.6. Pode-se observar na linha 5, a descrição da ação *ReadSensor*, e nas linhas 6 a 11, os parâmetros de descrição do sensor.

Código Fonte 5.6: Invocação da ação *ReadSensor* para temperatura.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <iq from="test@brigadeiro.local/WebAPI" to="test@brigadeiro.local/urn:
       schemas-upnp-org:device:SensorManagement:1:uuid:c18b693b-14dd-4107-
       a04e-36e823d51f28" xml:lang="en" id="Tq2nF-29" type="set">
3       <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" xmlns
            ="upnpcloud" s:encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/">
4           <s:Body>
5               <u:ReadSensor xmlns:u="urn:schemas-upnp-org:service:
                    SensorTransportGeneric:1">
6                   <SensorID>Sensor0001</SensorID>
7                   <SensorRecordInfo>&lt;?xml version="1.0" encoding="UTF
                        -8"?&gt;&lt;SensorRecordInfo xmlns="urn:schemas-upnp-
                        org:smgt:srecinfo" xmlns:xsi="http://www.w3.org/2001/
                        XMLSchema-instance" xsi:schemaLocation="urn:schemas-
                        upnp-org:smgt:srecinfo http://www.upnp.org/schemas/
                        smgt/srecinfo.xsd" &gt;&lt;sensorrecord&gt;&lt;field
                        name="temperature" /&gt;&lt;/sensorrecord&gt;&lt;/
                        SensorRecordInfo&gt;</SensorRecordInfo>
8                   <SensorURN>urn:upnp-org:smgt-surn:temperature:
                        AcmeSensorsCorp-com:AcmeIntegratedController:
                        TemperatureCorp:rf217acrs:monitor</SensorURN>
9                   <DataRecordCount>1</DataRecordCount>
10                  <SensorClientID>SensorClientIDSensor0001</SensorClientID>
11                  <SensorDataTypeEnable>0</SensorDataTypeEnable>
12              </u:ReadSensor>
13          </s:Body>
14      </s:Envelope>
15 </iq>
```

A resposta da requisição é apresentada no Código Fonte 5.7. Na linha 6 é possível ver o número **20.3**, que indica o valor da temperatura para esta leitura. O conteúdo da *tag DataRecords* é a resposta da invocação da ação *ReadSensor*.

Código Fonte 5.7: Resposta da ação *ReadSensor* para temperatura

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <iq id="Tq2nF-29" to="test@brigadeiro.local/WebAPI" from="test@brigadeiro
       .local/urn:schemas-upnp-org:device:SensorManagement:1:uuid:c18b693b-14
       dd-4107-a04e-36e823d51f28" type="result">
3       <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" xmlns
            ="upnpcloud" s:encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/">
4           <s:Body>
5               <u:ReadSensorResponse xmlns:u="ReadSensor">
6                   <DataRecords>&lt;?xml version='1.0' encoding='UTF-8'
                        standalone='no' ?&gt;&lt;DataRecords xsi:
                        schemaLocation="urn:schemas-upnp-org:ds:drecs http://
                        www.upnp.org/schemas/ds/drecs-v1.xsd" xmlns:xsi="http
                        ://www.w3.org/2001/XMLSchema-instance" xmlns="urn:
                        schemas-upnp-org:ds:drecs"&gt;&lt;datarecord&gt;&lt;
                        field name="temperature" encoding="ascii"&gt;20.3&lt;/
                        field&gt;&lt;/datarecord&gt;&lt;/DataRecords&gt;</
                        DataRecords>
7               </u:ReadSensorResponse>
8           </s:Body>
9       </s:Envelope>
10 </iq>
```

### 5.1.5 Avaliação de Desempenho

Uma análise de desempenho foi executada para averiguar se a arquitetura proposta pode ter fins práticos para casos de uso reais. O caso de uso base, para a análise de desempenho, foi de um monitoramento remoto em tempo real, onde as métricas de atualização e captura de novos dados são de extrema importância. Um exemplo deste uso é no monitoramento de pacientes, onde os eventos, como medidas fora das faixas de seguranças, precisam ser recebidas com velocidade, para que médicos e profissionais da saúde sejam notificados. As

métricas abordadas por este trabalho foram as mesmas utilizadas em outros trabalhos da literatura, como o de Bendel [21]. Estas foram: o tempo de atualização, que envolve conexão e desconexão, e o tempo de captura de novos dados feitos por um sensor.

Em todos os experimentos executados, foi utilizado o *middleware* desenvolvido e descrito neste Capítulo, em conjunto com o aplicativo Android para simulação de sensores. O ambiente foi simulado em uma única máquina com sistema operacional Ubuntu 15.10, processador Intel i7-4790 de 3.60GHz e 32 GB de memória RAM. Um emulador Android foi utilizado com 1.5GB de RAM disponíveis e a versão Lollipop[14] do sistema operacional.

Para simular ambientes reais, a ferramenta *NETEM*[15] versão 1.0.0-2 foi utilizada. O *NETEM* é um *software* desenvolvido pela *Linux Foundation*[16] para realizar simulações de redes reais com o objetivo de testar protocolos de comunicação. Baseado nos valores de referência contidos nos exemplos da ferramenta, este trabalho utilizou 3 diferentes configurações de perda de pacotes e atraso, utilizando variações aleatórias e uma pequena correlação na perda de pacotes para simular *packet bursts*, que são vários pacotes perdidos sequencialmente. Foram definidos três cenários diferentes para testar diferentes configurações de redes:

**Ambiente sem ruído**

Experimentos com esta configuração não tiveram o uso do *NETEM*.

**Ambiente com baixo ruído**

Visando simular uma rede real, esta configuração utilizou um atraso de $100ms$ com variações de $\pm 30ms$ com uma perda de pacotes de $1\%$ e correlação de $25\%$, que significa que o próximo pacote tem um quarto de probabilidade de perda dado que o pacote anterior foi perdido.

**Ambiente com alto ruído**

Para simular uma rede com problemas de conexão, foram utilizados valores de atraso ainda maiores, com $300ms$ e variações $100ms$, com uma perda de pacotes de $10\%$ e correlação de $25\%$, tal como o ambiente anterior.

[14] `https://www.android.com/intl/en_us/versions/lollipop-5-0/`

[15] `https://wiki.linuxfoundation.org/networking/netem`

[16] `https://www.linuxfoundation.org/`

### Tempo de Atualização

Para avaliar o tempo de atualização foi realizado um experimento com uma amostra de **70** pontos. O fluxo executado pelo sensor foi de uma simples conexão, uma espera de **20** segundos e em seguida a desconexão com o *middleware*. Dentro do simulador de sensores foram capturados *timestamps* no momento em que é enviada a requisição de conexão e desconexão, enquanto no *middleware*, os *timestamps* foram registrados no momento em que o sensor se torna disponível e indisponível na API Web.

A Figura 5.5 apresenta um *boxplot* com todos os **70** pontos capturados, em ambiente sem ruído, para a duração em que o dispositivo leva para iniciar o processo de conexão e estar disponível para aplicações e serviços. Pode-se notar uma baixa variação dos dados no valor mediano de **0,723** segundos.

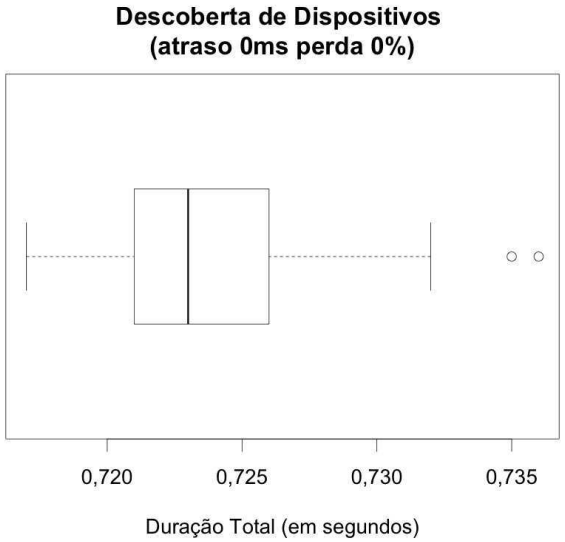


Figura 5.5: *Boxplot* com valores para tempo de descoberta de um sensor simulado em ambiente sem ruído.

A faixa da duração ficou entre **0,717** e **0,736** segundos, enquanto o 1o quartil e 3o quartil entre **0,721** e **0,726** segundos, o que significa que metade dos valores capturados se encontram dentro dessa faixa. O histograma da Figura 5.6, mostra por uma outra perspectiva a distribuição de frequência dos valores capturados. Pode-se ver que os valores na faixa de **0,720** e **0,727** possuem uma frequência maior.

A duração de desconexão foi realizada no mesmo experimento, e por isso foi capturada a mesma quantidade de **70** pontos. O *boxplot* dos dados é evidenciado na Figura 5.7. Nele,

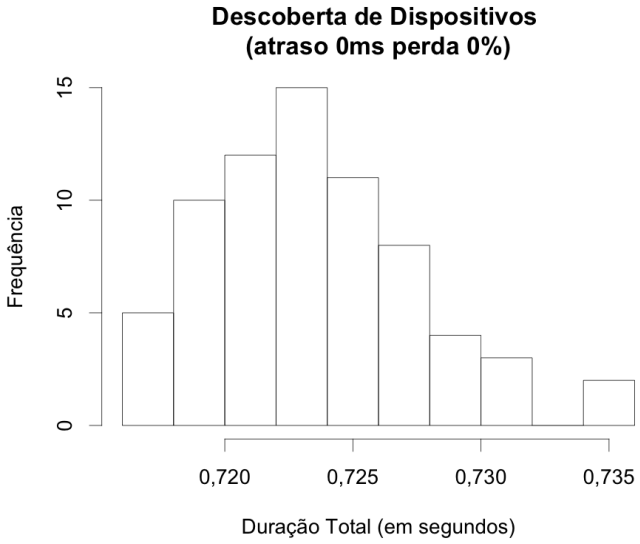


Figura 5.6: Histograma com valores para tempo de descoberta de um sensor simulado em ambiente sem ruído.

é possível notar uma maior frequência dos dados em torno da média de **0,076** segundos. A faixa de duração teve seus valores entre o mínimo de **0,074** e **0,087** segundos. A maior concentração dos dados ficou entre o 1o e 3o quartil, de valores **0,075** e **0,077**.

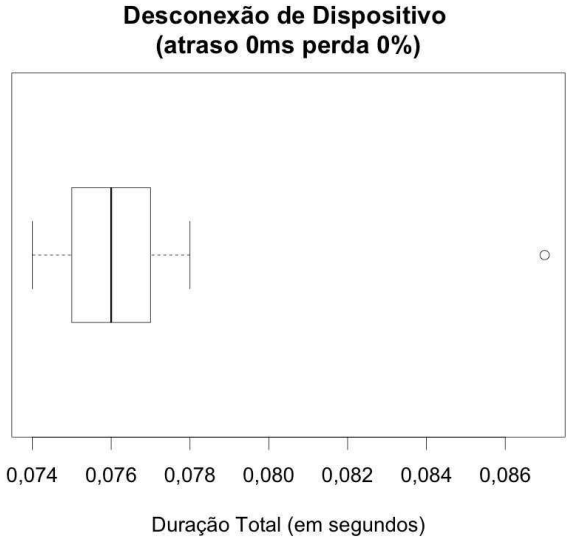


Figura 5.7: *Boxplot* com valores para tempo de desconexão de um sensor simulado em ambiente sem ruído.

No histograma evidenciado na Figura 5.8 é possível visualizar a frequência dos dados em torno da média. Pouquíssimos valores se afastam do centro de concentração, com frequên-

cias abaixo até mesmo dos 5 pontos.

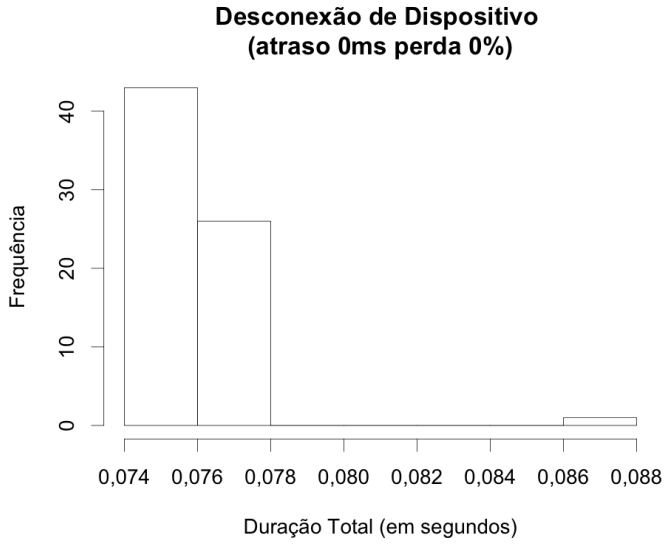


Figura 5.8: Histograma com valores para tempo de desconexão de um sensor simulado em ambiente sem ruído.

A fim de simular ambientes reais, também foram executados testes no ambiente simulado com baixo ruído. Neste experimento, houve a captura de **70** pontos. A Figura 5.9 apresenta um *boxplot* com os valores capturados.

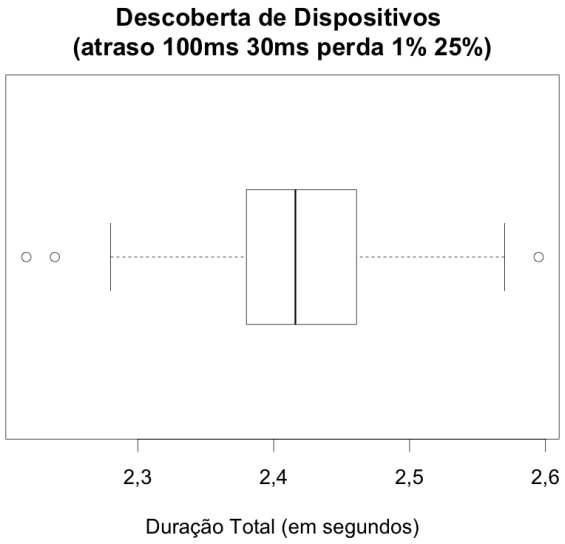


Figura 5.9: *Boxplot* com valores para tempo de descoberta de um sensor simulado em um ambiente com pouco ruído.

A média neste experimento teve o valor de **2,418** segundos, e faixas de valores entre

**2,218** e **2,595**. O 1o quartil foi de **2,380** e 3o quartil de **2,460**.

Um *boxplot* com o resultado do experimento, utilizando a configuração de alto ruído, é exposto pela Figura 5.10.

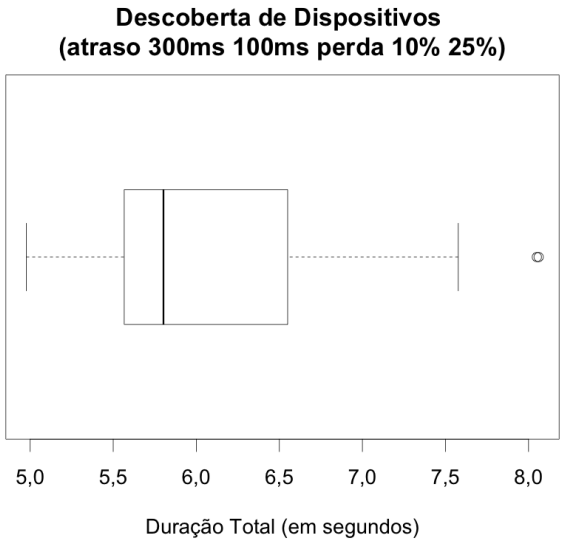


Figura 5.10: *Boxplot* com valores para tempo de descoberta de um sensor simulado em um ambiente com alto ruído.

É possível notar que os valores estiveram entre a faixa de **4,977** e **8,061**, na média de **6,045**, 1o quartil de **5,570** e 3o quartil de **6,528**, todos os valores são dados em segundos.

Finalmente, temos a simulação de ambientes para o tempo de desconexão. Primeiro, é apresentado pelo *boxplot* da Figura 5.11, os resultados da captura de **70** pontos em um ambiente de baixo ruído.

Para a desconexão em ambiente de baixo ruído, os valores tiveram uma média de **0,740** segundos, valor mínimo de **0,685**, máximo de **0,787**, 1o quartil em **0,725** e 3o quartil em **0,757**.

Já para o ambiente com alto ruido, a faixa ficou entre **0,666** e **0,984**, com média de **0,807**, 1o quartil de **0,752** e 3o quartil de **0,851**.

A partir dos dados apresentados, pode-se notar que o tempo de atualização, para ambientes sem ruídos, em ambos os casos se encontram abaixo dos **1000ms**. Já para a simulação de ambientes reais com pouco ruído, que pode ser considerado um caso mais comum para redes WAN (*Wide Area Network*), encontramos valores abaixo dos **3000ms**, indicando que o sistema pode ser utilizado por aplicações de monitoramento remoto de pacientes em tempo

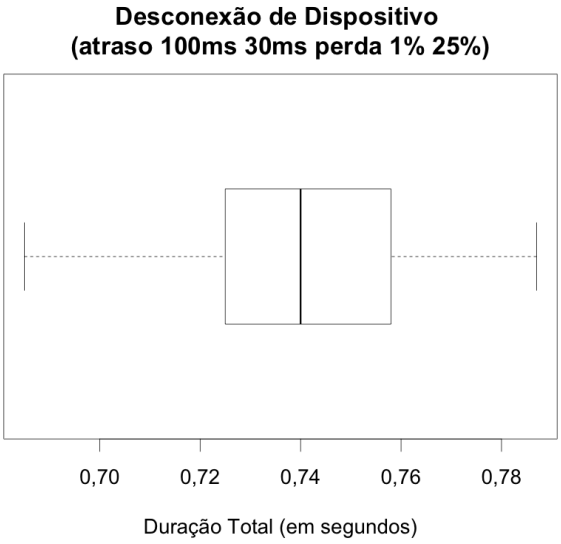


Figura 5.11: *Boxplot* com valores para tempo de desconexão de um sensor simulado em um ambiente com pouco ruído.

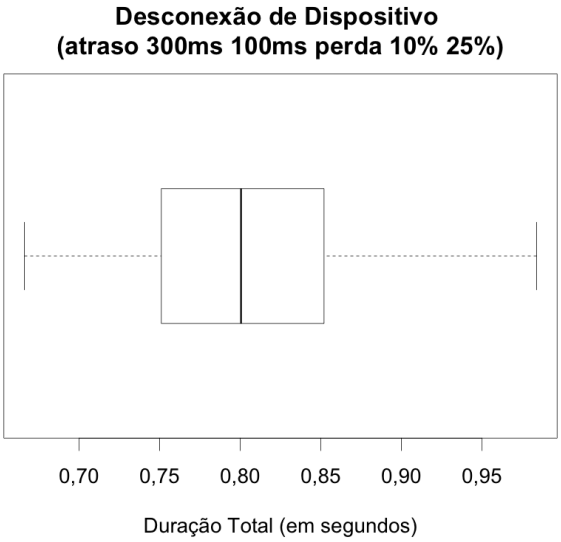


Figura 5.12: *Boxplot* com valores para tempo de desconexão de um sensor simulado em um ambiente com alto ruído.

real [49]. Estas informações evidenciam que não existe grande atraso por influência do *middleware* intermediário.

Nos experimentos de simulação de um ambiente com alto ruído, que são considerados casos extremos, o *middleware* foi capaz de completar todas as requisições, indicando que o experimento conseguiu lidar bem com as perdas em *bursts*.

### Leitura de Dados

Para a avaliação do desempenho da leitura dos dados, foi utilizado o fluxo de invocação da ação *ReadSensor* que é executada pelo *Device Manager* no padrão *Pooling*. Este padrão faz com que este fluxo seja executado repetidamente em um tempo definido, para que os dados sejam atualizados.

O fluxo avaliado a seguir também é o mesmo utilizado para invocar ações em atuadores, visto que os mesmos também utilizam o mesmo mecanismo de *Invoke* para receber instruções.

Nos três ambientes simulados, a sequência a seguir obteve **119** valores capturados para a duração do momento em que a ação é invocada, e da chegada de sua resposta, contendo os valores requisitados. A Figura 5.13 apresenta um *boxplot* com o experimento no ambiente simulado sem ruído. É possível notar uma variação baixa dos valores em torno da média de **0,007** segundos. O 1o e o 3o quartil tiveram valores de **0,005** e **0,008** respectivamente.

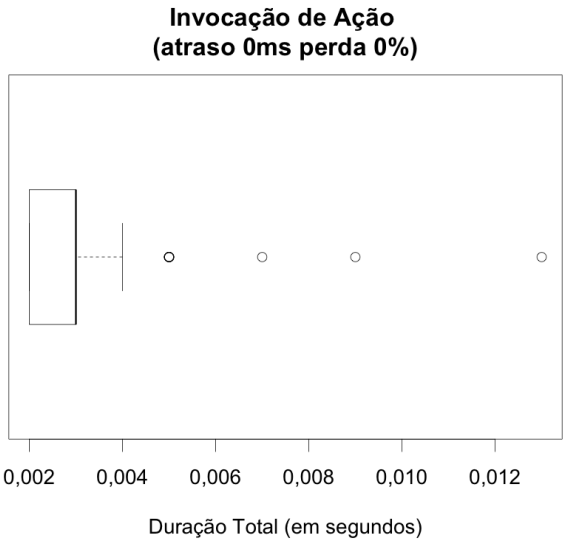


Figura 5.13: *Boxplot* com valores para tempo de invocação da ação *ReadSensor* em ambiente sem ruído.

O histograma da Figura 5.14 apresenta a frequência dos valores, entre as faixas do valor máximo de **0,029** e mínimo de **0,004** segundos. Poucos valores se distanciaram da média e teve sua maior concentração nos valores mais próximos do mínimo.

Na Figura 5.15 é apresentado o *boxplot* dos resultados em ambiente com pouco ruído. A média teve um valor de **0,402** segundos, o valor máximo de **0,478**, mínimo de **0,318**, 1o

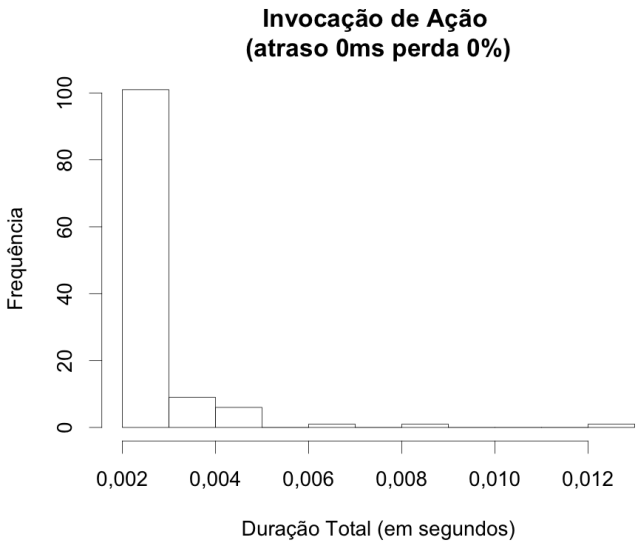


Figura 5.14: Histograma com valores para tempo de invocação da ação *ReadSensor* em ambiente sem ruído.

quartil em **0,376** e 3o quartil em **0,429**.

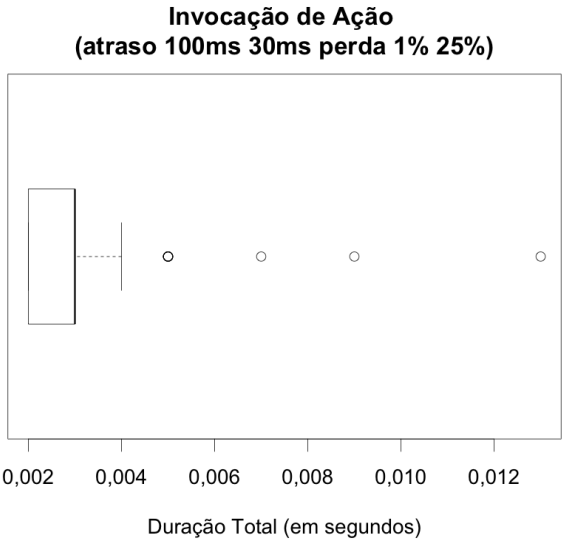


Figura 5.15: *Boxplot* com valores para tempo de invocação da ação *ReadSensor* em ambiente com pouco ruído.

Nos experimentos utilizando a configuração do ambiente com alto ruído, a média ficou em **1,286**, entre as faixas de **0,950** e **2,373**, 1o quartil de **1,138** e 3o quartil **1,312**. Estes resultados podem ser visualizados no *boxplot* da Figura 5.16.

Como apresentado nos dados anteriores, a leitura e invocação de ações em ambiente

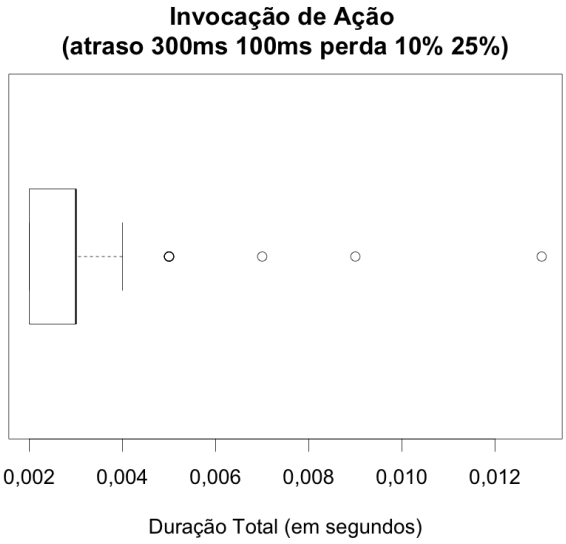


Figura 5.16: *Boxplot* com valores para tempo de invocação da ação *ReadSensor* em ambiente com alto ruído.

sem ruído é realizada abaixo dos **50ms**, destacando ainda mais sua utilização por aplicações de tempo-real e facilitando para aplicações que exijam que comandos de execução sejam tomados rapidamente por atuadores e também a rápida atualização dos dados provenientes de sensores.

Os resultados para ambientes simulados de baixo ruído ainda obtiveram uma duração abaixo dos **1000ms**, com uma baixa variação de valores em torno da média. Os ambientes simulados de alto ruído, tiveram a grande maioria dos seus resultados abaixo dos **2000ms**, apresentando alguns valores mais elevados podendo chegar próximos dos **2400ms**, provavelmente em casos de *packet bursts*, em que se foi necessário o reenvio do mesmo pacote mais de uma vez. Em todos os casos apresentados, o sistema conseguiu completar todas as requisições em tempo abaixo de **3000ms**, valor indicado para mudanças no estado do paciente na especificação IEEE 11073:00101 [49].

### Análise de Resultados

A partir dos dados apresentados na seção anterior, é possível notar com facilidade que o evento de descoberta teve valores bem maiores que os eventos de desconexão e invocação de ação. Isso é devido ao fluxo de autenticação realizado pelo protocolo XMPP, que pode ser ainda mais elevado caso seja utilizado encriptação por SSL. Porém, como eventos deste tipo

são realizados de maneira menos frequente, esses valores mais elevados se tornam viáveis em implementações com dispositivos reais.

Já os eventos de desconexão e leitura de dados possuem valores bem mais baixos comparados a descoberta. Na análise de pacotes realizada em ambientes sem ruídos, é possível ver que invocações de ações possuem uma troca de apenas **2** pacotes, não sendo necessária a realização de autenticação ou outros fluxos que acarretam no atraso da ação. O objetivo primordial desta arquitetura é a utilização destes dispositivos, e por isso possuir o mínimo esforço na realização do evento de invocação de ação é imprescindível.

Ao final desta análise, podemos concluir que os valores de desempenho estão apropriados para utilização em aplicações de tempo real, mesmo em ambientes de alto ruído. A especificação IEEE 11073:00101 define até **3000ms** para leitura de parâmetros fisiológicos, que indicam alteração no estado do paciente [49]. Além disso, outros trabalhos como o de *Bendel et tal* [21], detalhado no Capítulo 3, obteve um valor médio de **108** ms para leitura de dados em um ambiente controlado e sem ruído, sugerindo que este trabalho possa obter uma melhor eficiência se comparado a outras soluções na literatura.

Uma métrica não avaliada nessa análise foi o volume de dados e *eventing*, apresentado no Capítulo 3. Para executar está avaliação, bastaria a execução de uma grande quantidade de dispositivos simulados, afim de estressar o sistema e detectar possíveis falhas e atrasos na troca de informações.

## 5.2 Considerações Finais do Capítulo

Neste capítulo foram apresentados detalhes do desenvolvimento do *middleware* que concretiza a infraestrutura proposta. Inicialmente, foram expostos os detalhes das tecnologias utilizadas e como elas foram empregadas no desenvolvimento. Em seguida, foi definido um caso de uso real para a utilização da arquitetura e logo após, foi apresentado o funcionamento da arquitetura para o exemplo concebido. Posteriormente, o desenvolvimento do simulador de dispositivos foi apresentado com detalhes do fluxo de mensagens gerados pelo simulador desenvolvimento. Por fim, foi realizada uma avaliação experimental para analisar o uso do *middleware* em ambientes reais e sua influência no tempo de envio da informação transmitida. Uma análise dos resultados do experimento foi apresentada ao final, como fechamento

do capítulo.

A partir desta análise, pode-se elencar algumas conclusões da arquitetura apresentada neste trabalho e também possibilidades para trabalhos futuros, os quais serão apresentados no próximo Capítulo.

# Capítulo 6

# Conclusões

O *middleware* apresentado neste trabalho tem por objetivo solucionar alguns dos principais problemas encontrados na integração de dispositivos heterogêneos na IoT. Para isso, foi proposta uma arquitetura que visa integrar aparelhos de diferentes fabricantes, gerenciá-los e virtualizá-los para que possam ser utilizados por aplicações e serviços na Internet.

A fim de avaliar a proposta, foi implementado um *middleware* utilizando a especificação UPnP/UPnP+ com um servidor Java. Para simular os dispositivos, foi criado um aplicativo Android que simula dois tipos diferentes de sensores de temperatura, para avaliar a abstração dos dados na plataforma. O modelo de dados *oneIoTa* foi utilizado para integração com outras arquiteturas em desenvolvimento como a *IoTivity* da OCF.

Ao final da avaliação realizada e a partir dos objetivos estipulados no início do Capítulo 4, pode-se concluir, para cada um deles:

**Suportar a descoberta e uso de dispositivos aos consumidores de serviços**

Este objetivo foi atingido através da execução do caso de uso definido, onde se verifica a descoberta de novos dispositivos através da interface REST, disponível a consumidores de serviços.

**Oferecer transparência de serviços e abstrair camadas inferiores**

O *middleware* implementado neste trabalho trata destas metas por meio da utilização e exposição de dispositivos através de modelos de dados que utilizam apenas informações comuns aos mesmos, oferecendo assim a transparência e abstraindo camadas inferiores. A avaliação também foi realizada através do caso de uso definido, que faz

a verificação deste requisito.

**Oferecer suporte a mecanismos de auto-organização**

O componente *Iot Device Intermediator*, responsável por intermediar a comunicação com as WSANs, recebe os eventos de conexão e desconexão, e em seguida atualiza o *Device Manager*, responsável por prover a interface REST a clientes. Apesar da arquitetura possuir o suporte a mecanismos de auto-organização, casos de uso mais complexos são necessários para melhor avaliar este requisito.

**Interoperabilidade com uma variedade de dispositivos**

Para atingir este objetivo, a arquitetura utiliza a proposta de *gateways* para a tradução de informações em um modelo de dados únicos interno a plataforma. Tal como o requisito anterior, casos de uso utilizando dispositivos reais ainda se fazem necessários para uma melhor avaliação, apesar do suporte existir na arquitetura.

**Lidar eficientemente com grande volume de dados e altas cargas de comunicação**

A utilização da tecnologia XMPP no componente *IoT Device Intermediator* trás uma proposta de grande eficiência no controle de volumes de dados e altas cargas de comunicação, tal eficiência também utilizada por grandes empresas, como apresentado no Capítulo 2. A avaliação feita neste trabalho ainda não foi suficiente para cobrir este requisito, fazendo-se necessário a execução de novos experimentos contendo uma maior quantidade de dispositivos gerenciados pelo *middleware*.

**Auxiliar na integração com outros sistemas**

Finalmente, o requisito de integração com outros sistemas foi realizado através do uso de modelos de dados padrões, disponíveis em bancos de dados na Internet. O caso de uso avaliado verifica a utilização destes modelos de dados. Uma avaliação mais profunda poderá ser realizada quando novas arquiteturas que utilizem os mesmos MDs se tornarem disponíveis, tal como o IoTivity[1].

O desenvolvimento de soluções para resolver os problemas atacados neste trabalho é bastante ativo [30]. Projetos como o Connctd[2] oferecem uma abordagem bastante similar

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

[2] http://www.connctd.com/

em termos de API para serviços. O diferencial com este trabalho está na tecnologia utilizada (UPnP), e a maior integração com outras plataformas, como a já mencionada *IoTivity*.

## 6.1 Contribuições

Baseado nos objetivos citados no Capítulo 4, a proposta deste trabalho e seu desenvolvimento, podemos citar as principais contribuições:

**Uma arquitetura para virtualizar dispositivos na IoT**

A primeira contribuição desse trabalho é a proposta da arquitetura apresentada no Capítulo 3. A infraestrutura é capaz de lidar com os principais problemas na utilização de dispositivos IoT: descoberta, oferecer abstração e transparência, ser interoperável, lidar com grande volume de dados e altas cargas de comunicação, e oferecer suporte na integração com outras plataformas.

**Oferecer uma implementação base para utilização de modelos de dados padrões**

Uma das maiores limitações das arquiteturas propostas em outros trabalhos está na dificuldade de integração com outras arquiteturas. Modelos de dados padrões, como o *oneIoTa*, trazem o beneficio de padronizar a comunicação entre as plataformas e facilitar o desenvolvimento de aplicações e serviços que utilizem múltiplas plataformas. Este trabalho oferece uma implementação base para que futuros *middlewares* utilizem o banco de modelos de dados *oneIoTa*.

**Criação de simuladores para *Cloud Devices* UPnP+**

Uma contribuição indireta deste trabalho foi a criação de simuladores para *Cloud Devices* UPnP+. A pequena quantidade de dispositivos com esta característica disponíveis no mercado ainda é bastante limitada, e por isso simuladores como o desenvolvido neste trabalho poderão auxiliar desenvolvedores na criação de novas soluções que façam utilização dos *Cloud Devices*.

## 6.2 Trabalhos Futuros

Apesar das contribuições realizadas neste trabalho, promover uma arquitetura para resolver os problemas enunciados no Capítulo 1 ainda é uma tarefa que exige respostas para questões em diversas áreas e diferentes testes para comprovar a eficácia da solução. Por este motivo, como trabalhos futuros propõem-se:

**Casos de uso mais complexos**

*Middlewares* IoT são *softwares* extremamente complexos que envolvem diversos componentes e que visam resolver vários problemas. Os casos de uso abordados neste trabalho, apesar de cobrirem as funcionalidades mais básicas de *middlewares* IoT, ainda não abrangem todos os problemas que foram propostos, como a métrica de volume de dados e *eventing*. Para isso, é necessário a concepção de casos de uso mais complexos, com uma simulação maior de dispositivos diferentes.

**Testes em Ambientes Reais**

Apesar da simulação feita com perda de pacotes e atrasos, ainda é necessária uma realização de testes com dispositivos e *gateways* reais, e o *middleware* sendo executado na nuvem, para que possamos ter conclusões ainda mais próximas de um caso de uso em condições reais.

**Integração com outras plataformas**

A proposta de integração com outras plataformas que utilizam o banco de modelos de dados *oneIoTa* é suportada, mas devido à novidade na tecnologia, ainda não foi possível realizar testes em aplicativos que integram múltiplas plataformas que façam uso do *oneIoTa*.

**Segurança**

Apesar da plataforma ter em sua base a segurança dos protocolos que ela utiliza, como SSL, XMPP e outros, testes em segurança não foram realizados, o que pode indicar que ainda existem vulnerabilidades expostas por esta solução.

# Bibliografia


[1] Daniele Miorandi et al. “Internet of things: Vision, applications and research challenges”. Em: *Ad Hoc Networks* 10.7 (set. de 2012), pp. 1497–1516. ISSN: 15708705. DOI: 10.1016/j.adhoc.2012.02.016. URL: http://linkinghub.elsevier.com/retrieve/pii/S1570870512000674.

[2] Jennifer Yick, Biswanath Mukherjee e Dipak Ghosal. “Wireless sensor network survey”. Em: *Computer Networks* 52.12 (ago. de 2008), pp. 2292–2330. ISSN: 13891286. DOI: 10.1016/j.comnet.2008.04.002. URL: http://linkinghub.elsevier.com/retrieve/pii/S1389128608001254.

[3] Jaime Lloret et al. “A wireless sensor network deployment for rural and forest fire detection and verification”. Em: *Sensors* 9 (2009), pp. 8722–8747. ISSN: 14248220. DOI: 10.3390/s91108722.

[4] “Wireless Sensor Networks for In-Home Healthcare:” em: *High Confidence Medical Devices, Software, and Systems* (2005), pp. 2–3. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.5768.

[5] Hande Alemdar e Cem Ersoy. “Wireless sensor networks for healthcare: A survey”. Em: *Computer Networks* 54.15 (out. de 2010), pp. 2688–2710. ISSN: 13891286. DOI: 10.1016/j.comnet.2010.05.003. URL: http://linkinghub.elsevier.com/retrieve/pii/S1389128610001398.

[6] Walter Colitti, Kris Steenhaut e Niccolò De Caro. “Integrating wireless sensor networks with the web”. Em: *Conference on Information Processing in Sensor Networks (IPSN)* (2011), pp. 2–6. ISSN: 1089-7801. URL: http://hinrg.cs.jhu.edu/joomla/images/stories/IPSN%5C_2011%5C_koliti.pdf.

[7] Ga-won Lee et al. “Smart Home and Cloud Interworking System ( SHCI ) Architecture Design for a Cloud broker based Smart Home Environment”. Em: 5.12 (2013), pp. 441–447.

[8] Rui M. L. Santos, Instituto Superior Técnico e Universidade Técnica De Lisboa. “StoNes – Multiprotocol Gateway for Wireless Sensor Networks”. Em: (2010), pp. 1–10.

[9] Iván Corredor, José F. Martínez e Miguel S. Familiar. “Bringing pervasive embedded networks to the service cloud: A lightweight middleware approach”. Em: *Journal of Systems Architecture* 57.10 (nov. de 2011), pp. 916–933. ISSN: 13837621. DOI: 10.1016/j.sysarc.2011.04.005. URL: http://linkinghub.elsevier.com/retrieve/pii/S1383762111000543.

[10] Eleonora Borgia. “The Internet of Things vision : Key features , applications and open issues”. Em: *Computer Communications* 54 (2014), pp. 1–31. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2014.09.008. URL: http://dx.doi.org/10.1016/j.comcom.2014.09.008.

[11] Jayavardhana Gubbi et al. “Internet of Things (IoT): A vision, architectural elements, and future directions”. Em: *Future Generation Computer Systems* 29.7 (set. de 2013), pp. 1645–1660. ISSN: 0167739X. DOI: 10.1016/j.future.2013.01.010. URL: http://linkinghub.elsevier.com/retrieve/pii/S0167739X13000241.

[12] Luigi Atzori, Antonio Iera e Giacomo Morabito. “The Internet of Things: A survey”. Em: *Computer Networks* 54.15 (out. de 2010), pp. 2787–2805. ISSN: 13891286. DOI: 10.1016/j.comnet.2010.05.010. URL: http://linkinghub.elsevier.com/retrieve/pii/S1389128610001568.

[13] Ghofrane Fersi. “Middleware for internet of things: A study”. Em: *Proceedings - IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS 2015* (2015), pp. 230–235. ISSN: 09763252. DOI: 10.1109/DCOSS.2015.43.

[14] Abdelhadi Bouain, Abdelaziz El Fazziki e Mohammed Sadgal. “Pervasive services vs. Web services: Survey and comparison”. Em: *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*. IEEE. 2014, pp. 552–557.

[15] Jiehan Zhou, J. Riekki e Junzhao Sun. “Pervasive Service Computing toward Accommodating Service Coordination and Collaboration”. Em: *Frontier of Computer Science and Technology, 2009. FCST ’09. Fourth International Conference on*. Dez. de 2009, pp. 686–691. DOI: 10.1109/FCST.2009.39.

[16] Karim Seada et al. “Energy-efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks”. Em: *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*. SenSys ’04. Baltimore, MD, USA: ACM, 2004, pp. 108–121. ISBN: 1-58113-879-2. DOI: 10.1145/1031495.1031509. URL: http://doi.acm.org/10.1145/1031495.1031509.

[17] Venkatesh Rajendran, Katia Obraczka e J. J. Garcia-Luna-Aceves. “Energy-efficient, Collision-free Medium Access Control for Wireless Sensor Networks”. Em: *Wirel. Netw.* 12.1 (fev. de 2006), pp. 63–78. ISSN: 1022-0038. DOI: 10.1007/s11276-006-6151-z. URL: http://dx.doi.org/10.1007/s11276-006-6151-z.

[18] Joseph Polastre et al. “A Unifying Link Abstraction for Wireless Sensor Networks”. Em: *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*. SenSys ’05. San Diego, California, USA: ACM, 2005, pp. 76–89. ISBN: 1-59593-054-X. DOI: 10.1145/1098918.1098928. URL: http://doi.acm.org/10.1145/1098918.1098928.

[19] U. Hunkeler, Hong Linh Truong e A. Stanford-Clark. “MQTT-S: A publish/subscribe protocol for Wireless Sensor Networks”. Em: *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*. Jan. de 2008, pp. 791–798. DOI: 10.1109/COMSWA.2008.4554519.

[20] IETF. *CoAP RFC 7252*. https://datatracker.ietf.org/doc/rfc7252/. Acesso em 24/02/2015.

[21] Sven Bendel et al. “A Service Infrastructure for the Internet of Things based on XMPP”. Em: March (2013), pp. 385–388.

[22] Fairman Donoho Roe e Tourzan. “UPnP Device Architecture 2.0”. Em: (2015), pp. 1–195.

[23] P. Saint-Andre. “Extensible messaging and presence protocol (XMPP): Core”. Em: *IETF RFC 6120* (2015).

[24] Mohammad Hadi Valipour et al. “A brief survey of software architecture concepts and service oriented architecture”. Em: *2009 2nd IEEE International Conference on Computer Science and Information Technology* (2009), pp. 34–38. DOI: 10.1109/ICCSIT.2009.5235004. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5235004.

[25] E. Mingozzi et al. “An open framework for accessing Things as a service”. Em: (2013), pp. 1–5. ISSN: 1347-6890.

[26] Network Working Group. *RFC 2578*. Vol. 1. 2015, pp. 1689–1699. ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

[27] Network Working Group. *RFC 3159*. 1. 2014, pp. 1–5. ISBN: 9780874216561. DOI: 10.1007/s13398-014-0173-7.2. arXiv: arXiv:1011.1669v3.

[28] DMTF. “Managed Object Format (MOF) Specification”. Em: (2012), pp. 1–56.

[29] Network Working Group. “RFC 3444”. Em: *Igarss 2014* 1 (2014), pp. 1–5. ISSN: 0717-6163. DOI: 10.1007/s13398-014-0173-7.2. arXiv: arXiv:1011.1669v3.

[30] M A Razzaque, Marija Milojevic-jevric e Andrei Palade. “Middleware for Internet of Things : a Survey”. Em: (2015), pp. 1–26. DOI: 10.1109/JIOT.2015.2498900.

[31] P. R. Pietzuch e J. M. Bacon. “Hermes: a distributed event-based middleware architecture”. Em: *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*. 2002, pp. 611–618. DOI: 10.1109/ICDCSW.2002.1030837.

[32] P. Costa et al. “The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario”. Em: *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07)*. Mar. de 2007, pp. 69–78. DOI: 10.1109/PERCOM.2007.36.

[33] R. Meier e V. Cahill. “STEAM: event-based middleware for wireless ad hoc networks”. Em: *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*. 2002, pp. 639–644. DOI: 10.1109/ICDCSW.2002.1030841.

[34] Athanassios Boulis et al. “SensorWare: Programming Sensor Networks Beyond Code Update and Querying”. Em: *Pervasive Mob. Comput.* 3.4 (ago. de 2007), pp. 386–412. ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2007.04.007. URL: http://dx.doi.org/10.1016/j.pmcj.2007.04.007.

[35] R Müller, G Alonso e D Kossmann. “SwissQM: Next Generation Data Processing in Sensor Networks”. Em: *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research CIDR07* (2007), pp. 1–9. URL: http://dblp.uni-trier.de/db/conf/cidr/cidr2007.html.

[36] Ting Liu e Margaret Martonosi. “Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems”. Em: *SIGPLAN Not.* 38.10 (jun. de 2003), pp. 107–118. ISSN: 0362-1340. DOI: 10.1145/966049.781516. URL: http://doi.acm.org/10.1145/966049.781516.

[37] YoungMin Kwon et al. “ActorNet: An Actor Platform for Wireless Sensor Networks”. Em: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS ’06. Hakodate, Japan: ACM, 2006, pp. 1297–1300. ISBN: 1-59593-303-4. DOI: 10.1145/1160633.1160871. URL: http://doi.acm.org/10.1145/1160633.1160871.

[38] A. L. Murphy, G. P. Picco e G. C. Roman. “LIME: a middleware for physical and logical mobility”. Em: *Distributed Computing Systems, 2001. 21st International Conference on.* Abr. de 2001, pp. 524–533. DOI: 10.1109/ICDSC.2001.918983.

[39] C. Curino et al. “TinyLIME: bridging mobile and sensor networks through middleware”. Em: *Third IEEE International Conference on Pervasive Computing and Communications*. Mar. de 2005, pp. 61–72. DOI: 10.1109/PERCOM.2005.48.

[40] Chien-Chung Shen, C. Srisathapornphat e C. Jaikaeo. “Sensor information networking architecture and applications”. Em: *IEEE Personal Communications* 8.4 (ago. de 2001), pp. 52–59. ISSN: 1070-9916. DOI: 10.1109/98.944004.

[41] P. B. Gibbons et al. “IrisNet: an architecture for a worldwide sensor Web”. Em: *IEEE Pervasive Computing* 2.4 (out. de 2003), pp. 22–33. ISSN: 1536-1268. DOI: 10.1109/MPRV.2003.1251166.

[42] M. Eisenhauer, P. Rosengren e P. Antolin. “A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems”. Em: *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*. Jun. de 2009, pp. 1–3. DOI: 10.1109/SAHCNW.2009.5172913.

[43] P. Evensen e H. Meling. “SenseWrap: A service oriented middleware with sensor virtualization and self-configuration”. Em: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2009 5th International Conference on*. Dez. de 2009, pp. 261–266. DOI: 10.1109/ISSNIP.2009.5416827.

[44] D. Guinard et al. “Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services”. Em: *IEEE Transactions on Services Computing* 3.3 (jul. de 2010), pp. 223–235. ISSN: 1939-1374. DOI: 10.1109/TSC.2010.3.

[45] T Vresk e I Cavrak. “Architecture of an interoperable IoT platform based on microservices”. Em: *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (2016), pp. 1196–1201. DOI: 10.1109/MIPRO.2016.7522321.

[46] Simone Cirani et al. “The IoT hub: a fog node for seamless management of heterogeneous connected smart objects”. Em: *Sensing, Communication, and Networking-Workshops (SECON Workshops), 2015 12th Annual IEEE International Conference on*. IEEE. 2015, pp. 1–6.

[47] J.M. Corchado et al. “Using Heterogeneous Wireless Sensor Networks in a Telemonitoring System for Healthcare”. Em: *IEEE Transactions on Information Technology in Biomedicine* 14.2 (2010), pp. 234–240. ISSN: 1089-7771. DOI: 10.1109/TITB.2009.2034369.

[48] UPnP. *IoT Management and Control DataModel Service*. 2015.

[49] Ieee 11073Tm Standard Committee of the Ieee Engineering in Medicine Society e Biology. *Health informatics—PoC medical device communication—Part 00101: Guide—Guidelines for the use of RF wireless technology*. December. 2008, pp. 1–109. ISBN: 9780738158129. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=%7B%5C&%7Darnumber=4736537$%5Cbackslash$npapers2://publication/uuid/9762BD90-9EF6-4FE9-AC55-9E92F887F765.