UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

MAURÍCIO DE ABREU CORDEIRO

MIDDLE:
UM MIDDLEWARE ORIENTADO A SERVIÇOS PARA A INTERNET DAS COISAS

Vitória da Conquista
2017

MAURÍCIO DE ABREU CORDEIRO

MIDDLE:
UM MIDDLEWARE ORIENTADO A SERVIÇOS PARA INTERNET DAS COISAS

Monografia apresentada ao Curso de Graduação em Ciência da Computação da Universidade Estadual do Sudoeste da Bahia, como requisito parcial para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Prof. M.e Marco Antônio D. Ramos

Vitória da Conquista

2017

MAURÍCIO DE ABREU CORDEIRO

MIDDLE:

UM MIDDLEWARE ORIENTADO A SERVIÇOS PARA A INTERNET DAS COISAS

Aprovado em ___ / ___ / _____

BANCA EXAMINATÓRIA

_____

Prof. M.e Marco Antônio D. Ramos

Universidade Estadual do Sudoeste da Bahia – UESB

Orientador

_____

Prof. Dr. Fábio Moura Pereira

Universidade Estadual do Sudoeste da Bahia – UESB

_____

Prof. M. ª Cleia Santos Libarino

Instituto Federal da Bahia - IFBA

“Sonhei / que todos os meus dentes caíram / mas a língua sobreviveu / para contar a história. ”

(Lawrence Ferlinghetti)

RESUMO

Desde que surgiu, a Internet passou por uma série de mudanças, que a levaram pelos estágios de comunicação entre humanos e programas para comunicação entre programas, com a Internet dos Serviços. Agora, a Internet avança em um novo estágio: a Internet das Coisas. Um novo paradigma que propõe a integração de objetos ao mundo digital, possibilitando a criação de ambientes inteligentes, capazes de perceber a agir de acordo a eventos do meio em que estão inseridos. Para que isso ocorra, é necessário disponibilizar uma infraestrutura adequada de hardware e de software que dê suporte para o desenvolvimento desses ambientes. Desse modo, um middleware se apresenta como a solução de software escolhida para suprir as necessidades da Internet das Coisas. Este trabalho apresenta um middleware baseado em serviços web, para atender as necessidades de integração e gerenciamento, a nível de software, da Internet das Coisas.

Palavras-chave: middleware, Internet das Coisas, serviços web, integração, ambientes inteligentes.

ABSTRACT

Since the beginning, the Internet has been through a series of changes, which have led it through the stages of communication between humans and programs for communication among programs, with Internet of Services. Now, the Internet advances in a new stage: the Internet of Things. A new paradigm that proposes the integration of objects into the digital world, allowing the creation of smart environments, capable of perceiving and to act according to environment's events which they are inserted. However, for this to occur, it is necessary to provide adequate hardware and software infrastructure to support the development of these environments. Thereby, middleware presents itself as the software solution chosen to supply the Internet of Things needs. This work presents the web services-based middleware design and implementation to provides the integration and management, in a software approach, that Internet of Things demands.

Keywords: middleware, Internet of Things, web services, integration, smart environments.

LISTA DE ABREVIAÇÕES

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	Application Programming Interface
CoAP	Constrained Application Protocol
CORBA	Common Object Request Broker Architecture
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
DCOM	Distributed Component Object Model
DTLS	Datagram Transport Layer Security
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDL	Interface Definition Language
IPSO	IP for Smart Objects
IoT	Internet of Things
J2EE	Java 2 Platform, Enterprise Edition
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
Modelo E-R	Modelo Entidade-Relacionamento
MVCC	Multi-Version Concurrency Control
NFC	Near Field Communication
OASIS	Organization for the Advancement of Structured Information Standards
ORB	Object Request Broker
P2P	peer-to-peer
QoS	Quality of Service
RAM	Random Access Memory
REST	REpresentational State Transfer
RFID	Radio-Frequency IDentification
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAML	Security Assertions Markup Language
SGBD	Sistema Gerenciador de Banco de Dados
SMTP	Simple Mail Transfer Protocol

SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SODA	Service Oriented Device Architecture
SOM	Service Oriented Middleware
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TSL	Transport Layer Security
UDDI	Universal Description Discovery and Integration.
URI	Uniform Resource Identifier
Web	World Wide Web
WISP	Wireless Identification and Sensing Platform
WSAN	Wireless Sensor and Actuator Networks
WSDL	Web Service Description Language
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A fronteira entre o mundo físico e o mundo digital fica mais tênue a cada dia. A inclusão de ferramentas computacionais na rotina diária da sociedade humana é notadamente crescente e a qual estamos nos tornando habituados, ao ponto de não percebermos a presença de tal tecnologia. É nesse cenário que a Internet das Coisas se apresenta.

A Internet das Coisas (IoT, sigla em inglês para *Internet of Things*) é um paradigma que propõe a conexão de diversos tipos de objetos à Internet, como a fechadura de uma porta, luminárias ou uma geladeira; ou ainda, todo o maquinário de uma fábrica, sistemas de irrigação, colheitadeiras e máquinas de transporte e processamento de produtos agrícolas; além de carros, ônibus e trens de um sistema de transporte urbano, sistemas de iluminação e monitoramento de vias públicas, etc.

Seu potencial é tão grande, que a integração da Internet das Coisas e da Internet dos Serviços com processos de fabricação deu início à Quarta Revolução Industrial, ou a Indústria 4.0 (KAGERMANN et al., 2013, apud HERMANN, OTTO, 2016).

Além da aplicação industrial, é possível aplicar IoT na construção de casas inteligentes, capazes de realizar de forma autônoma tarefas como irrigação de jardim, controle de ar-condicionado, controle de persianas, monitoramento de consumo de água, luz e gás em tempo real, identificar vazamentos, alertar quando estiver faltando algum produto na sua geladeira, dentre outras coisas. O mesmo vale na construção de cidades inteligentes, hospitais, prédios, rodovias.

No Brasil existem iniciativas como a criação do primeiro Polo de Inovação em IoT[1], com foco em segurança pública, na cidade paulista de São José dos Campos. O projeto é uma parceria entre o Ministério da Ciência, Tecnologia, Inovações e Comunicações (MCTIC) e a empresa sueca de tecnologia Ericsson, que tem realizado investimentos em diversos projetos para o desenvolvimento de soluções IoT no país.

O MCTIC apresentou recentemente o Plano Nacional de IoT[2], e vem realizando consultas públicas a fim de estabelecer diretrizes para nortear as políticas públicas voltadas para o desenvolvimento do mercado de IoT no Brasil até 2022.

---

[1] http://www.mcti.gov.br/[...]parceria-para-o-desenvolvimento-de-tecnologias-de-iot
[2] http://www.mcti.gov.br/[...]plano-nacional-de-internet-das-coisas-nos-proximos-dias

O leque de objetos que podem integrar um sistema IoT é bastante abrangente. No entanto, interconectá-los e torná-los parte de um único sistema é uma tarefa que requer a definição de protocolos de comunicação e de tratamento dos dados enviados e recebidos, além de soluções para questões de escalabilidade do sistema e identificação de objetos.

Uma forma de estabelecer tais padrões é com a implantação de um middleware, ou seja, uma camada de software que fornece os recursos necessários para a interação entre objetos e aplicações, abstraindo as características específicas de software ou de componentes de hardware simplificando o tratamento dos dados recebidos e manipulados, gerenciando informações de identificação e endereçamento e fornecendo serviços para as outras camadas do sistema.

Atualmente, uma nova área de conhecimento vem tomando forma e formatando e estimulando o desenvolvimento de sistemas que apliquem os conceitos de IoT. Conhecidos como Sistemas Ciberfísicos, estas estruturas são formadas por dispositivos diversos, meios de comunicação diversos além do software necessário para processamento dos dados, geração de sinais e interfaces de comunicação.

Como toda nova área de conhecimento, existem divergências quanto a definição dos Sistemas Ciberfísicos. Uma corrente de pesquisadores defende que tais sistemas são uma evolução da mecatrônica, ou seja, não possuem interseção com elementos mecatrônicos. Já outra corrente de pesquisadores defende que, os Sistemas Ciberfísicos usam elementos mecatrônicos, além de novos elementos, criando uma nova área.

Tanto IoT, quanto os Sistemas Ciberfísicos são utilizados hoje em dia em diversas soluções, desde soluções para controle doméstico quanto para controle de plantas industriais e *Smart-grids*[3]. Esta variedade de aplicações traz também uma complexidade de grande ordem no desenvolvimento e validação de tais sistemas. Hoje, as ferramentas de desenvolvimento não possuem modelos específicos para tais aplicações, bem como os testes funcionais e de desempenho. No entanto, mesmo assim diversos sistemas deste tipo vêm sendo desenvolvidos com certa eficiência.

Este trabalho, se ateve ao estudo de middlewares aplicados especificamente no desenvolvimento de aplicações voltadas a IoT, bem como no desenvolvimento de uma aplicação utilizando uma das soluções estudadas. Sendo

[3] Rede de distribuição de energia elétrica com alto índice de automação.

assim, após o estudo a arquitetura de middleware escolhida foi a SOA (*Service Oriented Architecture*). A descrição da arquitetura bem como o desenvolvimento e estrutura da aplicação será discutida no capítulo 3.

Assim, além desta introdução, esta monografia se organiza da seguinte forma: o capítulo 2 apresenta o embasamento teórico do trabalho, incluindo trabalhos relacionados. O capítulo 3 apresenta os detalhes arquiteturais do middleware desenvolvido. O capítulo 4 detalha as questões relacionadas a implementação do middleware. No capítulo 5 são apresentadas as conclusões obtidas durante o desenvolvimento do trabalho.

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver uma solução de middleware baseado na Arquitetura Orientada a Serviços.

1.1.2 Objetivos específicos

- Investigar as soluções mais adequadas para o desenvolvimento de um middleware voltado para a Internet das Coisas;
- Modelar uma arquitetura para o middleware proposto que se adeque aos padrões estabelecidos;
- Implementar o middleware.

1.2 Metodologia

Este trabalho foi realizado seguindo a metodologia analítica experimental, uma vez que soluções de middlewares para a IoT foram estudadas e uma solução própria foi desenvolvida à título de experimentação.

2 CONCEITOS E TECNOLOGIAS

Este capítulo tem por objetivo apresentar a fundamentação teórica para o trabalho desenvolvido.

2.1 Sistemas Ciberfísicos

Sistemas Ciberfísicos são, juntamente com IoT, pilares da chamada Indústria 4.0. São sistemas nos quais existem forte integração e forte coordenação entre suas componentes físicas e componentes computacionais e são classificados, a depender da linha de pesquisa, como uma evolução dos sistemas mecatrônicos ou como o resultado da junção de sistemas embarcados com sistemas de tempo real. A Figura 1 mostra a relação entre sistemas ciberfísicos e sistemas mecatrônicos.

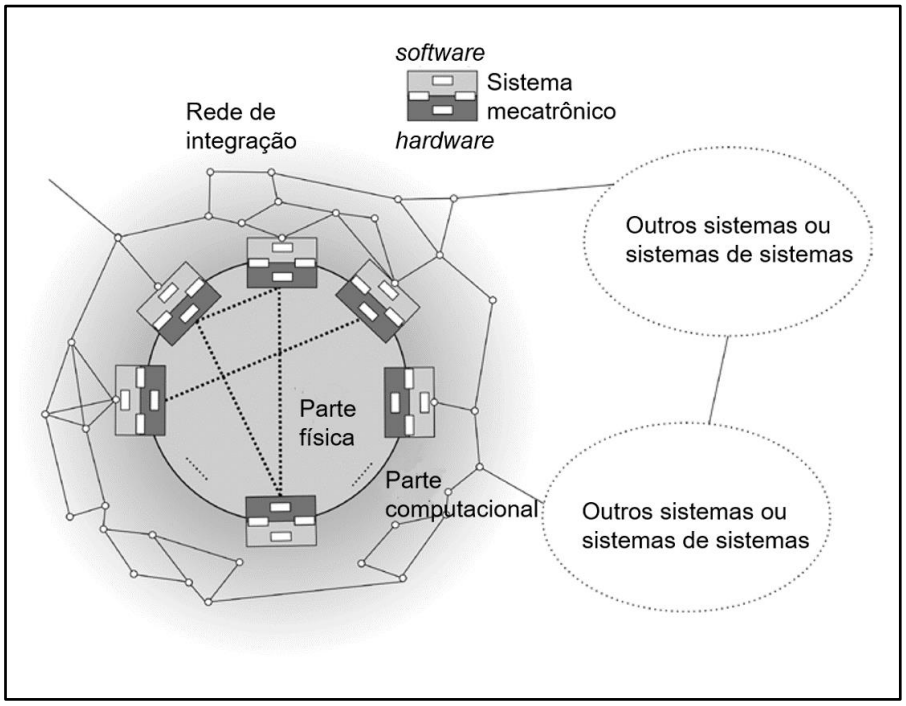

Figura 1: Sistemas ciberfísicos. (SILVA, 2017)

Esse tipo de sistema tem aplicação em monitoramento de ambientes, seja ele doméstico, profissional ou externo, monitoramento de pessoas (pacientes médicos, por exemplo), *Smart-grids*, controle de fábricas e em áreas onde o monitoramento e controle possa ser feito de maneira autônoma ou com pouca interferência humana.

A Figura 2 apresenta um mapa conceitual dos sistemas ciberfísicos, como as áreas de aplicação, características necessárias para a sua implantação e as respostas e esses sistemas podem fornecer.

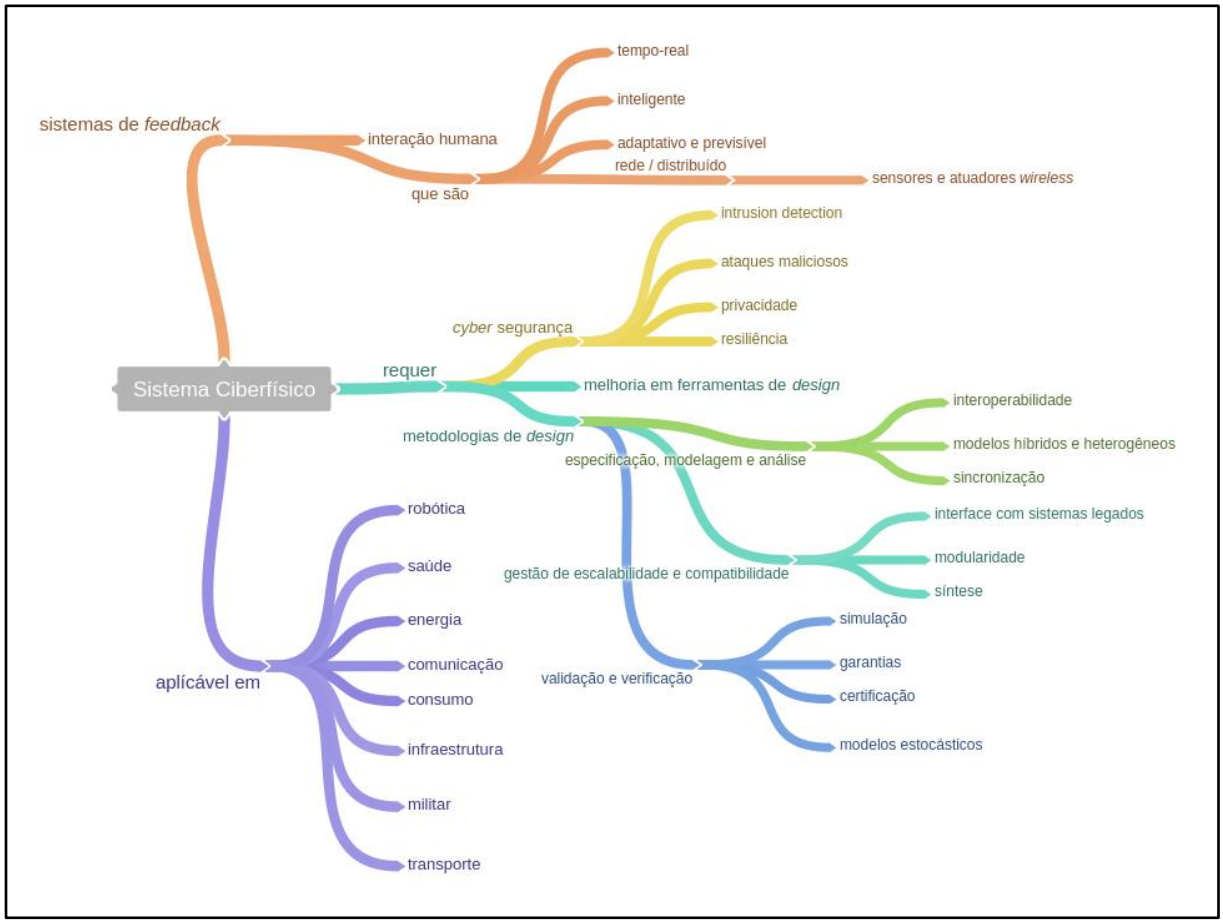

Figura 2: Mapa conceitual dos sistemas ciberfísicos (SILVA, 2017)

No desenvolvimento de tais sistemas são utilizados, em muitos casos a infraestrutura oferecida pela IoT, a qual é melhor descrita na seção 2.2.

2.2 Internet das coisas

A Internet das Coisas (*Internet of Things* ou IoT) é um paradigma da computação em franca expansão, principalmente no que diz respeito às tecnologias de comunicação *wireless*, onde a presença pervasiva de coisas ou objetos inteligentes – sejam etiquetas RFID, sensores, atuadores, telefones móveis – que interagem e cooperam entre si, possibilita a realização de tarefas de monitoramento, controle e fornecimento de informações com o mínimo de interferência humana.

Nesse contexto, através da troca de suas informações e de informações do ambiente, e da realização de ações em resposta às informações recebidas, com ou sem interferência humana, as “coisas” se tornam agentes participantes nos processos de negócio, sociais e informacionais do mundo físico (GUBBI, et. al, 2012).

Segundo Atzori et. al. (2010), Internet das Coisas é a junção de três visões da computação (Figura 3) – visão orientada às coisas (sensores e atuadores), visão orientada à Internet (*middleware*) e visão orientada à semântica (conhecimento). Apesar da multidisciplinariedade dessa definição, a usabilidade da IoT só pode ser mensurada na interseção dessas três visões (GUBBI, et. al, 2012).

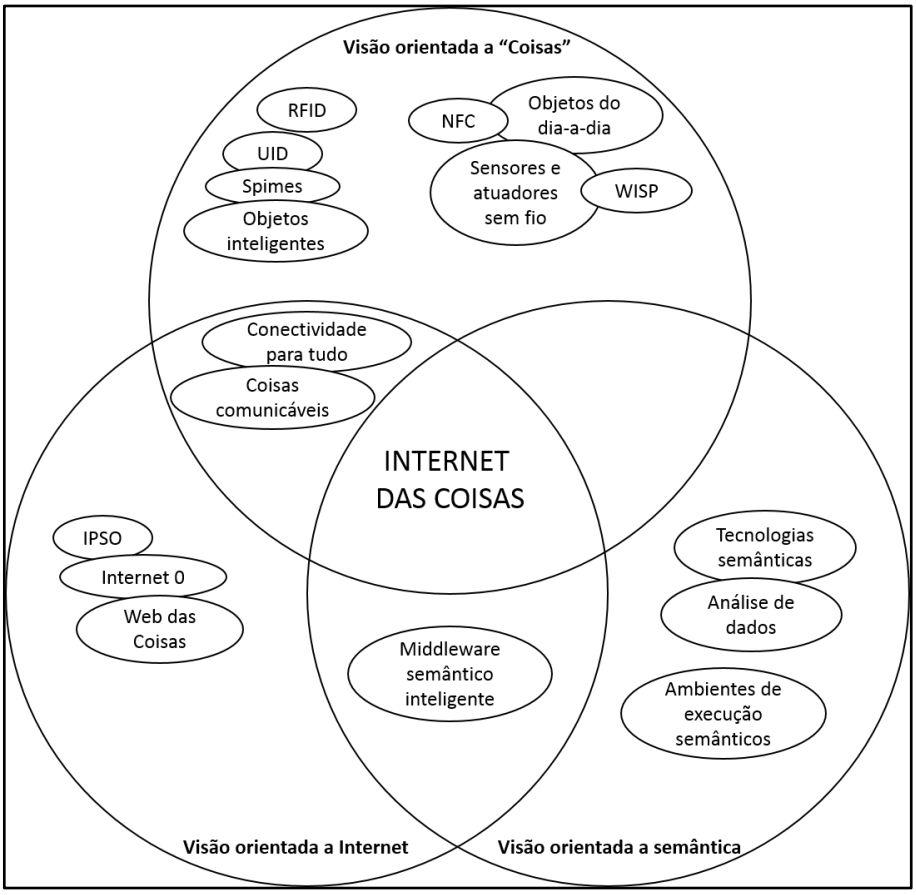

Figura 3: Visões da Internet das Coisas (ATZORI et. al., 2010)

A visão orientada a “coisas” se refere aos objetos, sua capacidade de trabalhar com dados (objetos inteligentes) e a forma como eles estão conectados. E na vanguarda das tecnologias utilizadas nessa visão está o RFID (*Radio-Frequency IDentification*), devido a sua maturidade, baixo custo e amplo suporte. NFC (*Near Field Communication*), Redes de Sensores e Atuadores Sem fio (WSAN), Plataformas de Identificação e Detecção Sem fio (WISP), *spime* (objeto que pode ser rastreado

através do espaço-tempo) seguem crescendo como alternativas ou complemento ao RFID no desenvolvimento da perspectiva das coisas na IoT.

Ainda referente a visão orientada a coisas, define-se “objetos inteligentes” (a implementação de *spime*) como aqueles que, além de possuir memória, capacidade de processamento e comunicação sem fio, são capazes de comportamento autônomo, sensível ao contexto em que estão inseridos e de comunicação colaborativa com outros objetos do ambiente que integram (ATZORI et.al., 2010).

MIORANDI et al (2012), define objetos inteligentes (ou coisas) como entidades com uma forma física, tenham uma identidade única e possuam, pelo menos um nome (para identificação por humanos) e um endereço (para identificação por máquinas), que sejam capazes de um mínimo conjunto de funcionalidades de comunicação – receber uma mensagem e responder a ela –, que tenham alguma capacidade computacional e que possam sentir ou atuar no ambiente.

A visão orientada a Internet diz respeito a infraestrutura necessária para que os objetos se conectem, como a identificação de objetos e protocolos de comunicação. Para tal, em 2008, o fórum IPSO[4] (IP para Objetos Inteligentes) foi formado com o objetivo de promover o uso do IP para conexão de objetos inteligentes ao redor do mundo pois, segundo o IPSO, o IP já possui características que permitem o seu uso na IoT, como ser um protocolo leve e de baixo consumo, já interligar grande quantidade de dispositivos e funcionar com dispositivos embarcados, e além do IPSO, existe a Internet Ø, com pretensões de simplificar o protocolo IP para que este “funcione em tudo” (ATZORI et.al., 2010).

A visão orientada a semântica trata do significado e da utilização dos objetos conectados, isto é, definir o que é o objeto e como utilizado dentro de um contexto. Uma vez que o número de objetos conectados à Internet tende a ser bastante grande e variado, torna-se necessário desenvolver uma forma de armazenar, interconectar, organizar e acessar as informações geradas. Desse modo, ambientes semânticos e arquiteturas que preenchem os requisitos da IoT podem trazer a solução adequada para essa problemática (ATZORI et.al., 2010).

Além da definição dada em ATZORI et.al., 2010, Internet das Coisas pode ser definida, de forma mais concisa, como:

> Interconexão de sensores e atuadores que proveem a capacidade de compartilhar informações entre plataformas através de uma estrutura

[4] *IP for Smart Objects*.

> unificada, desenvolvendo um cenário comum para aplicações inovadoras. Isso é alcançado pela junção de computação ubíqua, análise de dados e representação de informação integrado com Computação em Nuvem (GUBBI, et. al, 2012).[5]

Em nível de sistema, aplicações para a Internet das Coisas devem suportar aspectos como heterogeneidade, pois uma das principais características da IoT é a possibilidade de englobar os mais variados objetos e aplicações em um único sistema; a escalabilidade, pois novos objetos se conectam a infraestrutura de comunicação diariamente, levantando problemas como (i) endereçamento, (ii) conexão e troca de dados, (iii) gerenciamento de informações, uma vez que é possível criar uma versão digital de objetos, que geram, processam e armazenam dados, e (iv) gerenciamento do alto número e variedade de serviços disponibilizados para suprir as necessidades dos objetos (MIORANDI et al, 2012).

Além disso, aplicações IoT devem suportar comunicação ubíqua através de tecnologias sem fio; recursos de localização e rastreamento, importantes para aplicação em logística e monitoramento de ciclo-de-vida de objetos; serem capazes de reagir aos mais diversos cenários e situações, com o mínimo de intervenção humana possível; soluções com baixo consumo de energia, devido as características funcionais dos objetos; gerenciamento semântico de interoperabilidade e de dados, devido a necessidade de dar significado e padronização a enorme quantidade de dados que pode ser obtida com a IoT, buscando garantir a interoperabilidade entre aplicações; e, por fim, garantir a segurança e privacidade dos dados.(MIORANDI et al, 2012).

2.3 Middleware

O termo *middleware* é usado para definir diferentes tipos de software. De maneira geral, ele é um software que conecta dois ou mais software em um mesmo conjunto de sistemas, como mostrado na Figura 4.

Middleware pode ser definido como um software assistente de uma aplicação para interagir ou comunicar com outras aplicações, redes, hardware e/ou

[5] Texto original: "*Interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless ubiquitous sensing, data analytics and information representation with Cloud computing as the unifying framework*". Tradução nossa.

sistemas operacionais. Esse software auxilia programadores aliviando o trabalho com as complexas conexões necessárias em um sistema distribuído. Ele fornece ferramentas para melhorar a QoS, segurança, passagem de mensagem, diretório de serviços, serviços de arquivos, etc. que são invisíveis ao usuário (BISHOP, KARNE 2003).

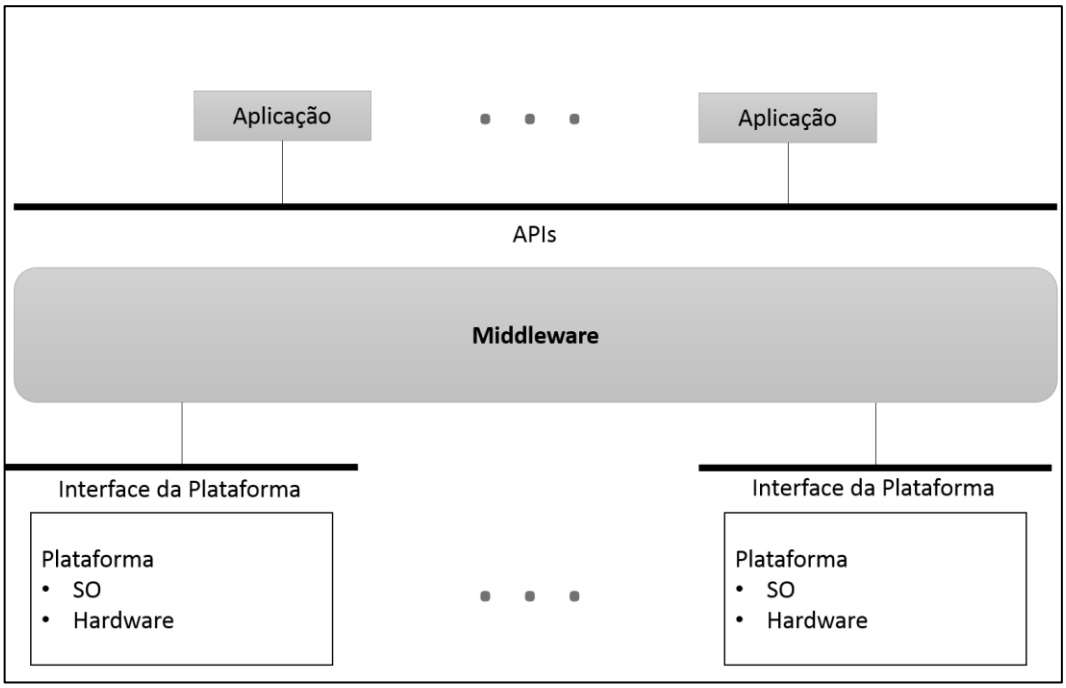

Figura 4: Middleware (BERNSTEIN 1996)

O middleware deve fornecer uma gama de serviços de propósito geral tanto para a plataforma quanto para as aplicações. Ele deve ser capaz de atender a várias aplicações, facilitando a comunicação entre elas em diversos formatos; de funcionar independente de plataforma, promovendo interoperabilidade entre diversos sistemas; de fornecer acesso remoto a outras aplicações e serviços; suportar padrões de protocolo de comunicação, como por exemplo o TCP/IP; e dar suporte a uma API padrão de forma transparente ao usuário (BERNSTEIN, 1996).

Existem diversos tipos de middleware, classificados de acordo com suas características de implementação, protocolos de comunicação que são usados, interface e número de usuários. Esses tipos podem ser organizados em duas grandes categorias: middlewares de integração e middlewares de aplicação (BISHOP, KARNE 2003), como ilustra a Figura 5.

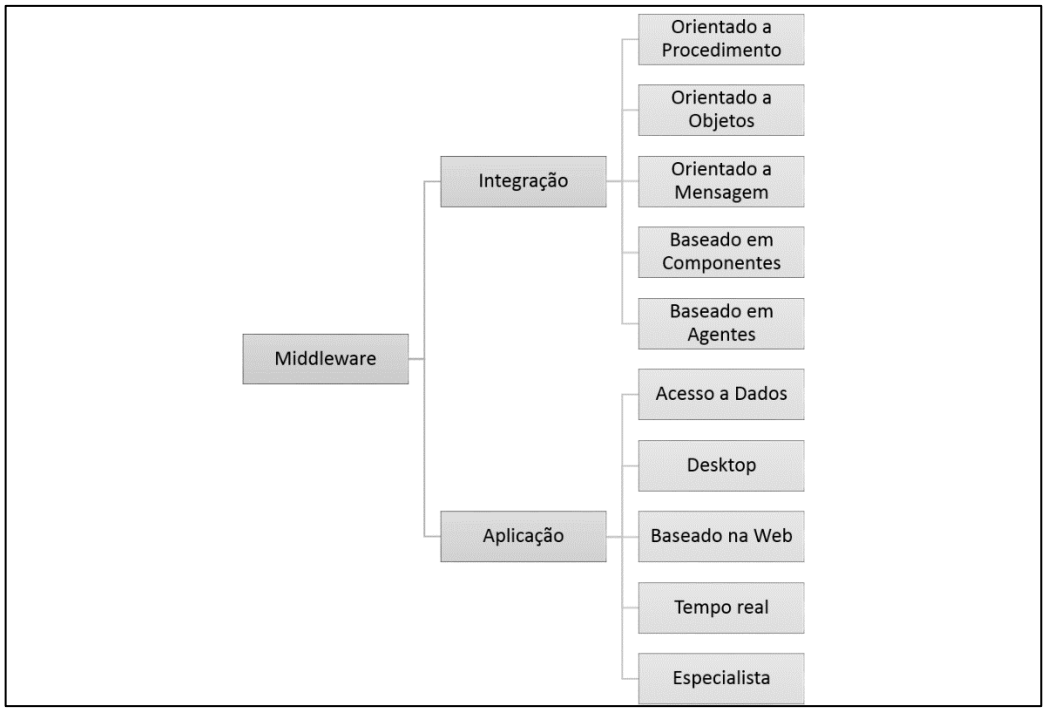

Figura 5: Categorias de middlewares (BISHOP, KARNE 2003)

2.3.1 Middlewares de Integração

Middlewares de Integração são aqueles que possuem maneiras específicas de integração dentro de um ambiente heterogêneo. Cada middleware desse tipo tem diferentes protocolos de comunicação ou padrões de operação com outros softwares.

No *middleware orientado a procedimento*, o cliente converte os parâmetros do procedimento em uma mensagem e a envia ao servidor. Este converte a mensagem de volta em parâmetros e executa a chamada ao procedimento. Após a execução, o retorno é devolvido ao cliente também em forma de mensagem. Sua vantagem está no uso de padrões para nomes de serviço, procedimentos remotos e na possibilidade de existir retorno mesmo com problemas na rede. Suas desvantagens incluem a pouca escalabilidade e a impossibilidade de retornar dados para outros programas.

O *middleware orientado a objetos* suporta requisições distribuídas de objetos e a comunicação entre eles pode ser síncrona ou assíncrona. Nele, o cliente executa uma chamada a um objeto remoto, a requisição é convertida em mensagem e transmitida por um ORB (*Object Request Broker*). Ao chegar no servidor, a

mensagem é convertida de volta e a requisição é enviada ao objeto responsável pelo seu processamento. O processo inverso é feito para enviar a resposta ao cliente. Suas vantagens são a escalabilidade e a possibilidade de retornar dados a outros programas, além de operar com múltiplos tipos de dados e estados e suportar operação simultânea usando múltiplas threads. As desvantagens incluem a necessidade de um link previamente estabelecido para a execução, e o envelopamento do código de sistemas legados.

O *middleware orientado a mensagem* pode funcionar de duas formas: Passagem e enfileiramento de mensagens ou Publicação de mensagem. Na passagem de mensagem, o cliente envia a requisição do serviço ao servidor e este enfileira a requisição, com base em alguma ordem de prioridade, e as distribui para serem processadas. Na publicação de mensagem, o cliente envia a requisição para um barramento onde o servidor “escuta” a chegada de requisições. Após o processamento, o retorno é publicado e “escutado” pelo cliente.

O *middleware baseado em componentes*, ou middleware reflexivo, consiste em um conjunto de componentes – programas com funções específicas e facilmente interoperáveis – que operam entre si e com outras aplicações. Eles são configuráveis tanto em tempo de compilação quanto em tempo de execução, o que o torna flexível, fácil de expandir e reconfigurar.

Em um *middleware baseado em agentes*, as entidades, os meios de comunicação e as regras que os agentes seguem se combinam para interagir com o ambiente e fornecer os serviços que lhe são requisitados. A autonomia e adaptabilidade desse tipo de middleware, advinda das características nativas dos agentes, o torna ideal para lidar com ambientes heterogêneos. Sua desvantagem está na alta complexidade e necessidade de grande poder computacional para operar.

2.3.2 Middlewares de Aplicação

Esta categoria abrange middlewares desenvolvidos para funcionar com tipos específicos de aplicações.

Middlewares de acesso a dados são aqueles com o propósito de interagir com aplicações – locais ou remotas – de banco de dados. Ele inclui funcionalidades como suporte a transações, gateways de banco de dados e processamento distribuído de transações, além de implementar as propriedades ACID e ferramentas de

segurança. Sua principal vantagem está na transparência na comunicação entre múltiplas fontes de dados e na comunicação direta com SGBD.

De modo geral, um *middleware de desktop* funciona como um sistema operacional em um computador distribuído, fornecendo serviços como gestão de serviços (emulação de terminal e, transferência de arquivos), gestão de arquivos e diretórios, gestão de banco de dados, gestão de threads e escalonamento de processos, notificação de eventos, assistente de instalação, encriptação e controle de acesso.

Middleware baseado na Web é aquele cujo objetivo é lidar com questões referentes ao funcionamento de aplicações na Internet. Ele fornece serviços como autenticação de grande número de aplicações, comunicação interprocessos que seja independente de linguagem, sistema operacional, protocolo de rede e plataforma de hardware. Esse tipo de middleware pode se conectar diretamente com as aplicações (geralmente usando HTTP) e seus serviços podem incluir ainda servidores de e-mail, acesso remoto a dados e aplicações remotas. Ele é largamente usado em sistemas de comércio eletrônico, e computação móvel.

Sistema de tempo real pode ser caracterizado com um sistema onde não só a corretude do dado gerado é importante, mas também o tempo em que este dado é apresentado, caso contrário, o dado deixa de estar correto. Um *middleware de tempo real* provê serviços característicos desse tipo de sistema, como requisições sensíveis ao tempo e políticas de escalonamento. Sua vantagem está na capacidade de fornecer assistentes capazes de auxiliar a resolução de questões relacionadas à gestão de *deadline* de processos e na escolha da melhor abordagem para solucionar problemas de processos sensíveis ao tempo.

Middlewares especialistas são aqueles desenvolvidos para resolver questões muito específicas e que não entram em nenhuma das categorias anteriores.

2.4 RMI Java

RMI Java (*Remote Method Invocation*) é uma solução da plataforma Java para o desenvolvimento de aplicações distribuídas, que permite objetos invocarem métodos em objetos remotos com a mesma sintaxe de uma invocação de métodos locais.

Baseada na arquitetura cliente-servidor, as aplicações distribuídas que implementam RMI Java devem ser capazes de localizar objetos remotos, seja por registro de objetos (disponibilizado pela plataforma) ou por passagem de referência aos objetos remotos. A comunicação entre esses objetos é tratada pelo RMI, se tornando transparente ao usuário.

A principal desvantagem dessa solução é o fato de interligar apenas aplicações escritas em Java, se tornando inadequada para uso em um cenário heterogêneo como a IoT.

2.5 CORBA

O CORBA (*Common Object Request Broker Architecture*) é uma arquitetura que define padrões para a troca de dados em sistemas distribuídos heterogêneos.

Seu funcionamento é semelhante ao do Java RMI, com a vantagem de ser independente de linguagem, sendo os objetos CORBA descritos de interfaces IDL (*Interface Definition Language*) que são compiladas para uma linguagem específica, podendo então ser acessada pelo cliente.

Apesar de permitir interoperalibidade independente de linguagem, a implementação da arquitetura CORBA é complexa, e a sua implantação pode enfrentar problemas de comunicação, uma vez que é executado sobre o protocolo TCP e utiliza portas arbitrárias que podem estar bloqueadas por *firewalls*.

2.6 Arquitetura Orientada a Serviços

A Arquitetura Orientada a Serviços, ou SOA, é um modelo de arquitetura de software que permite a criação de serviços interoperáveis que podem ser usados por diferentes sistemas. A organização OASIS define SOA da seguinte forma:

> Um paradigma para organização e utilização de recursos distribuídos que podem estar sob o controle de diferentes domínios. Isso provê uma maneira uniforme de oferecer, descobrir, interagir e usar esses recursos para produzir os efeitos desejados compatíveis com as

> precondições e expectativas mensuradas. (MACKENZIE. et al, 2006, p. 29, tradução nossa[6]).

Surgiu como uma evolução de modelos como Engenharia de Software Baseada em Componentes, Desenvolvimento Baseado em Interface (orientado a objetos) e sistemas distribuídos da década de 1990, como, por exemplo, CORBA, DCOM e J2EE.

Nesse modelo, um serviço é uma funcionalidade da aplicação que se encontra bem definida, autocontida e que não depende de outros serviços para funcionar (SILVA, 2004). Um serviço pode também ser a combinação de dois ou mais serviços diferentes.

A arquitetura SOA é formada por três componentes distintos: os fornecedores, os consumidores e o barramento de serviços. Os fornecedores são aplicações que prestam os serviços; os consumidores são aplicações que utilizam os serviços; o barramento de serviços é onde os serviços são publicados pelos fornecedores e encontrados pelos consumidores. A Figura 6 ilustra o funcionamento da SOA.

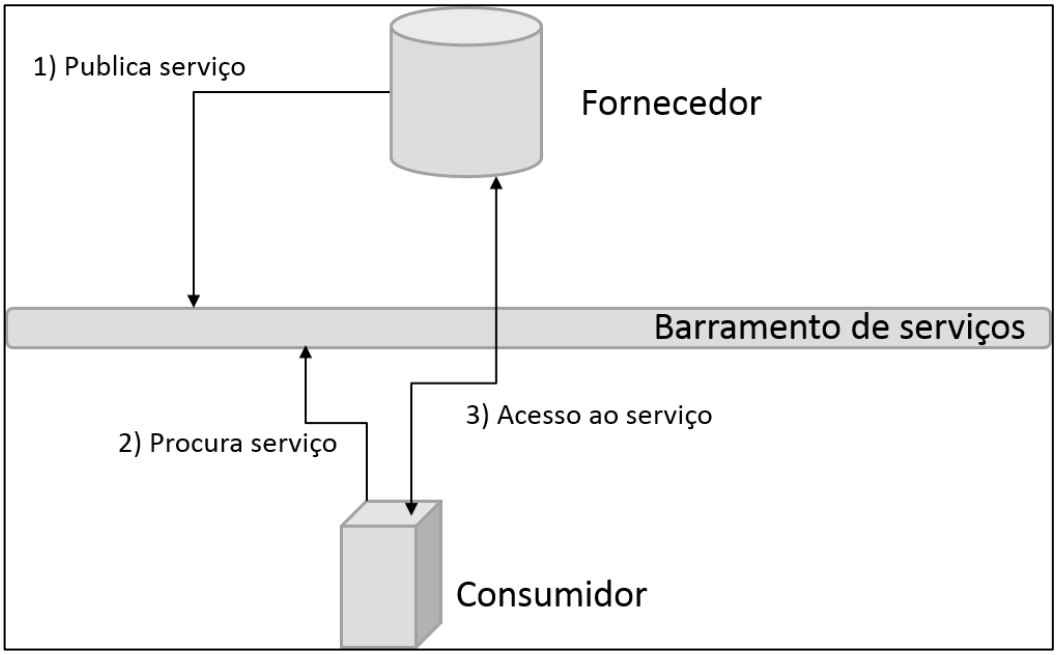

Figura 6: Funcionamento da SOA.

[6] Texto original: "*A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations*."

Do ponto de vista funcional e organizacional, a SOA possui características como a descoberta e ligação dinâmica de serviços, ou seja, a descoberta e acesso a serviços podem ser feitos pelos consumidores em tempo de execução. Além disso, um serviço é modular e pode ser composto por outros serviços, é interoperável, independente da linguagem usada na implementação. Os módulos de serviço são independentes entre si, permitindo reutilização de código e independência entre fornecedores e consumidores, além de possibilitar a composição de aplicações a partir da junção de módulos de serviços. Os serviços também são tolerantes a falhas, continuando a operar com o mínimo de interferência humana (VALIPOUR, 2009).

2.6.1 Web Services

Web service como um padrão para a implementação da arquitetura SOA sobre a Internet, que utiliza os padrões de comunicação da Web para o desenvolvimento de uma camada de abstração e serviços que podem não ser oferecidas pelo software integrado. Ele pode ser implementado com base em protocolos: SOAP, REST.

- **SOAP**

O protocolo SOAP (*Simple Object Access Protocol*) define um padrão para troca de mensagens, especificado em XML, sobre diferentes protocolos de transporte como HTTP ou SMTP. Ele é análogo ao padrão RPC, embora seja mais lento, devido à alta carga de texto característica do formato XML.

Um web service baseado em SOAP é composto pelas camadas de Descoberta, Descrição, Envelopamento, Transporte e Rede (SILVA, 2004).

A camada de Descoberta é responsável por fornecer a descrição dos serviços dos fornecedores. O registro e descoberta do serviço são feitos de acordo com a especificação UDDI. A camada de Descrição mantém a descrição dos serviços do fornecedor. O WSDL é comumente usado para descrever um web service. A camada de Envelopamento prepara a mensagem para o consumidor. Essa etapa é feita através de SOAP, que utiliza XML para definir padrões de troca de mensagens. A camada de Transporte é responsável pela comunicação entre as aplicações,

geralmente utilizando o protocolo HTTP e a camada de Rede é responsável pela base da comunicação entre as aplicações, análoga à camada de rede do modelo TCP/IP.

A Figura 7 apresenta um esquema do funcionamento de um web service baseado em SOAP.

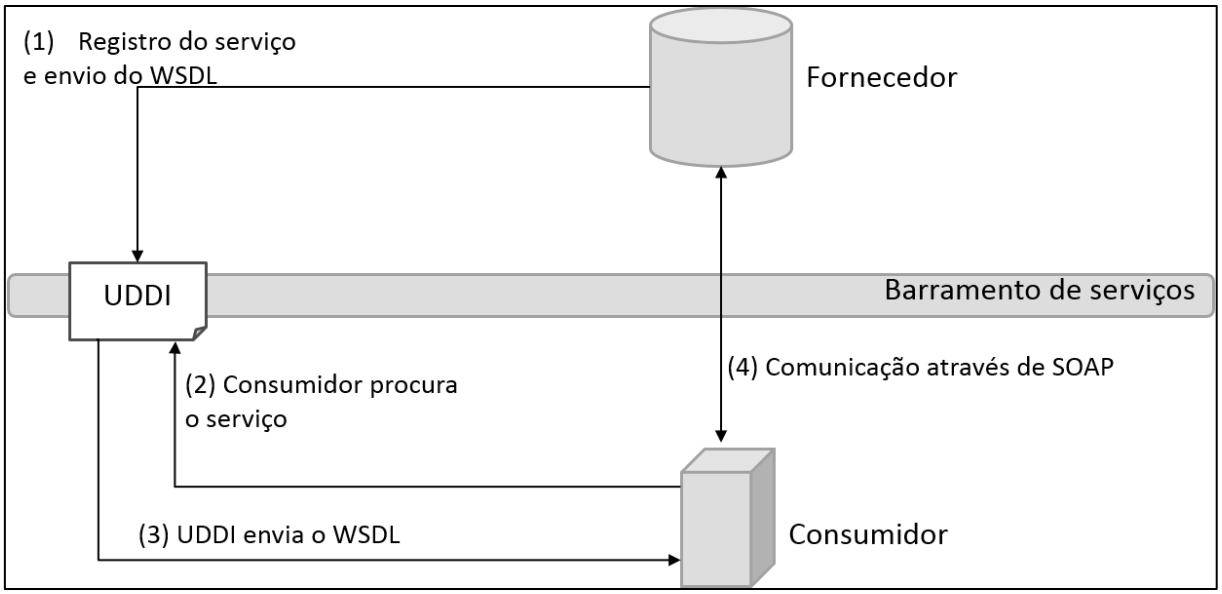

Figura 7: Funcionamento de um web service SOAP.

- **REST**

O padrão REST (*REpresentational State Tranfer*) descreve um conjunto de princípios arquiteturais pelos quais dados podem ser transmitidos sobre uma interface padronizada (como o HTTP), como mostra a Figura 8.

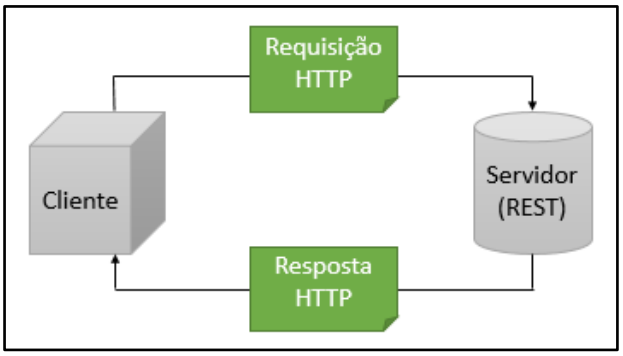

Figura 8: Funcionamento de um web service REST.

Um cliente pode acessar o serviço usando uma URI única e uma representação do serviço é retornada. Durante o acesso ao serviço através de HTTP, a URL do serviço funciona como um identificador e os métodos padrões do HTTP (GET, POST, PUT, DELETE e HEAD) são operações realizadas pelo serviço.

2.7 Arquitetura Orientada a Serviços em Dispositivos

A Arquitetura Orientada a Serviços em Dispositivos, ou SODA (*Service Oriented Device Architecture*), é uma adaptação da arquitetura SOA para dispositivos de hardware, conectando estes ao barramento de serviços para que sejam acessados por consumidores como se fossem serviços, abstraindo os componentes de hardware em software.

A implementação da SODA é baseada em três componentes: o adaptador de dispositivo, que é responsável por gerar uma abstração dos dispositivos de hardware; o barramento adaptador, que realiza a comunicação entre os dispositivos e a rede; e o registro de serviço de dispositivo, que guarda as informações necessárias para o descobrimento dos serviços.

Além de possuir as características apresentadas na SOA, esse modelo busca fornecer alto-nível de abstração dos objetos do mundo físico, além de isolar o desenvolvimento de sistemas corporativos do crescente número de padrões de interface de dispositivos e interligar os mundos físico e digital através de serviços (DEUGD, 2006).

- **CoAP**

O CoAP (*Constrained Application Protocol*) é um protocolo de comunicação para dispositivos de baixo poder computacional e redes de curto alcance e com alta perca de dados. Foi desenhado especificamente para aplicações máquina-maquina, como um sistema de monitoramento e controle de ambientes inteligentes. Definido no RFC 7252[7], o CoAP possui suporte a requisições e respostas, via HTTP, entre dispositivos, além de suporte a transmissão *multicast*, baixo *overhead* de mensagem e simplicidade em ambientes restritos.

[7] Atualizado pelo RFC 7959 *Block-Wise Transfers in the Constrained Application Protocol (CoAP).*

Em suma, o CoAP é um protocolo baseado em REST para dispositivos com baixo poder computacional, com foco em comunicação máquina-máquina, característico da IoT.

2.8 Middleware Orientado a Serviços

A aceitação cada vez maior da SOA como opção para o desenvolvimento de sistemas distribuídos e a heterogeneidade dos componentes desses sistemas levou ao desenvolvimento de um tipo de middleware capaz de dar suporte às requisições que esse paradigma demanda, o middleware orientado a serviços.

Assim, o Middleware Orientado a Serviços, ou SOM (*Service Oriented Middleware*), é um middleware projetado como uma coleção de serviços que pode ser usado para facilitar o desenvolvimento, execução e o gerenciamento de aplicações orientadas a serviços (AL-JAROODI; MOHAMED, 2014). Para isso, é necessário que o SOM implemente funcionalidades da arquitetura SOA.

De modo geral, um middleware orientado a serviços deve oferecer os seguintes recursos:

- *Padronização e composição de serviço*: disponibilizar APIs que facilitem o processo de desenvolvimento de aplicações para integração com o middleware, eliminando a necessidade de desenvolvedores se envolverem na definição de padrões de comunicação;
- *Registro e publicação de serviços*: deve ser feita de forma a não interromper o funcionamento do middleware;
- *Descoberta e integração de serviços*: suportar um serviço que possibilite aos clientes descobrir e acessar os outros serviços do middleware;
- *Heterogeneidade*: o middleware deve ser capaz de abstrair e tornar transparente ao usuário a heterogeneidade do ambiente;
- *Integração transparente com aplicações clientes*: As aplicações clientes devem ser capazes de se integrar ao middleware sem se preocupar com o funcionamento do serviço, apenas com sua resposta;
- *Adaptação e autonomia*;
- *Escalabilidade e eficiência*;

- *Confiabilidade e segurança*;
- *Requisitos de QoS*.

2.9 Trabalhos relacionados

Eisenhauer et al (2009) apresenta um middleware baseado em serviços para redes de sistemas embarcados, o Hydra middleware. Com foco na interoperabilidade, o trabalho desenvolveu protótipos em três diferentes escopos: automação residencial, monitoramento de saúde e agricultura.

Valente (2011) desenvolveu uma plataforma onde os serviços representam as funcionalidades dos dispositivos, fornecendo aos clientes uma interface de comunicação genérica capaz de trabalhar com a heterogeneidade desses dispositivos. O trabalho tem ainda suporte para busca semântica de serviços, uma vez que os serviços disponíveis podem ser numerosos e diversificados.

Kostelnik et.al. (2013) apresentam uma plataforma de middleware para sistemas embarcados, implementada combinando a arquitetura SOA com redes P2P e tecnologias de web service semântico.

Hachem (2014) desenvolveu um protótipo de uma solução de middleware, baseado na arquitetura SOA, para trabalhar com a grande quantidade e a heterogeneidade de dispositivos móveis da IoT. O trabalho inclui detalhes acerca da implementação e dos testes executados sobre a solução desenvolvida.

RAZZAQUE et.al. (2016), estabelece duas categorias de requisitos que um middleware para a IoT deve atender: i) os serviços que ele deve oferecer e ii) a arquitetura que ele deve suportar.

A primeira diz respeito, como requisito funcional, (a) à descoberta de recursos, uma vez que o dinamismo da infraestrutura da IoT não permite um total conhecimento determinístico dos recursos disponíveis, é preciso disponibilizar formas para que seja possível descobri-los dinamicamente; (b) ao gerenciamento de dados, pois dados são um ponto chave na IoT, e demandam um nível eficiente de aquisição, processamento e armazenamento; (c) gerenciamento de eventos disparados tanto por aplicações quanto por objetos do sistema IoT; e (d) gerenciamento de código-fonte para garantir dinamismo na implantação e migração de novos módulos do sistema. Como requisito não-funcional, o middleware deve ser escalável, confiável, estar

disponível o máximo possível, garantir segurança e privacidade e ser de fácil implantação.

A segunda, se refere ao suporte a outras aplicações como (a) abstração de programa, fornecendo uma API para desenvolvimento de aplicações de integração com o middleware; (b) interoperabilidade, garantindo a capacidade do middleware de funcionar em um ambiente heterogêneo, característico da IoT; (c) a capacidade de adaptação do middleware às mudanças que podem acontecer no ambiente IoT, o que leva a necessidade de (d) sensibilidade ao contexto no qual está inserido; (e) capacidade de atuar com autonomia, em resposta a eventos do ambiente e; (f) ser distribuído, uma vez que um middleware centralizado pode não ser capaz de suportar sistemas IoT de larga escala, cujas parte podem estar, inclusive, geograficamente distribuídas.

3 O MIDDLE

O middleware desenvolvido neste trabalho é, então, um middleware de integração baseado em mensagens e de aplicação, implementado segundo as especificações definidas nas arquiteturas SOM e SODA, que são especializações da SOA, ou seja, seus recursos são disponibilizados e acessados na forma de serviços web. Esses serviços foram implementados seguindo protocolos SOAP, REST e CoAP, por contarem com documentação e comunidade reconhecidos no mercado.

Neste capítulo serão expostas as suas definições de projeto, como a arquitetura e a descrição dos módulos, o modelo de dados, o fluxo de funcionamento e os serviços que foram disponibilizados.

3.1 Arquitetura

Os módulos que compõem o middleware são: *ThingsGateway*, *WSGateway*, Reflexão, Implantação, Serviços, Agendamentos e Persistência (Figura 9).

Os módulos *Gateway*[8] são responsáveis pela comunicação do middleware com o ambiente exterior. O módulo ***ThingsGateway*** é responsável por manter uma interface de comunicação entres os dispositivos IoT[9] e destes com o middleware. Ele foi implementado com suporte para troca de mensagem nos protocolos CoAP e REST, e atua recebendo dos dispositivos a descrição detalhada do serviço a ser invocado e devolve aos objetos a resposta dada pelo serviço.

Paralelo ao *ThingsGateway* há o ***WSGateway***, que é o responsável pela interface de comunicação do middleware com as aplicações na Internet e de aplicações entre si. Ele está implementado como um web service, com suporte para troca de mensagens nos protocolos SOAP e REST, também recebendo a descrição do serviço e a ser acessado e devolvendo o resultado do serviço a aplicação.

Trabalhando dentro dos limites do middleware, está o módulo de **Reflexão**, responsável por invocar os métodos que representam os serviços do middleware. A informação sobre o nome do serviço e os parâmetros para acessá-lo, ou seja, sua

[8] Para fins práticos, *ThingsGateway* e *WSGateway* foram abstraídos em apenas *Gateway* quando suas características em comum forem referenciadas.
[9] Leia-se “Objetos inteligentes”.

descrição, são recebidas dos módulos *Gateway* e então a chamado para o método que represente o serviço (dentro do módulo Serviço) é realizada.

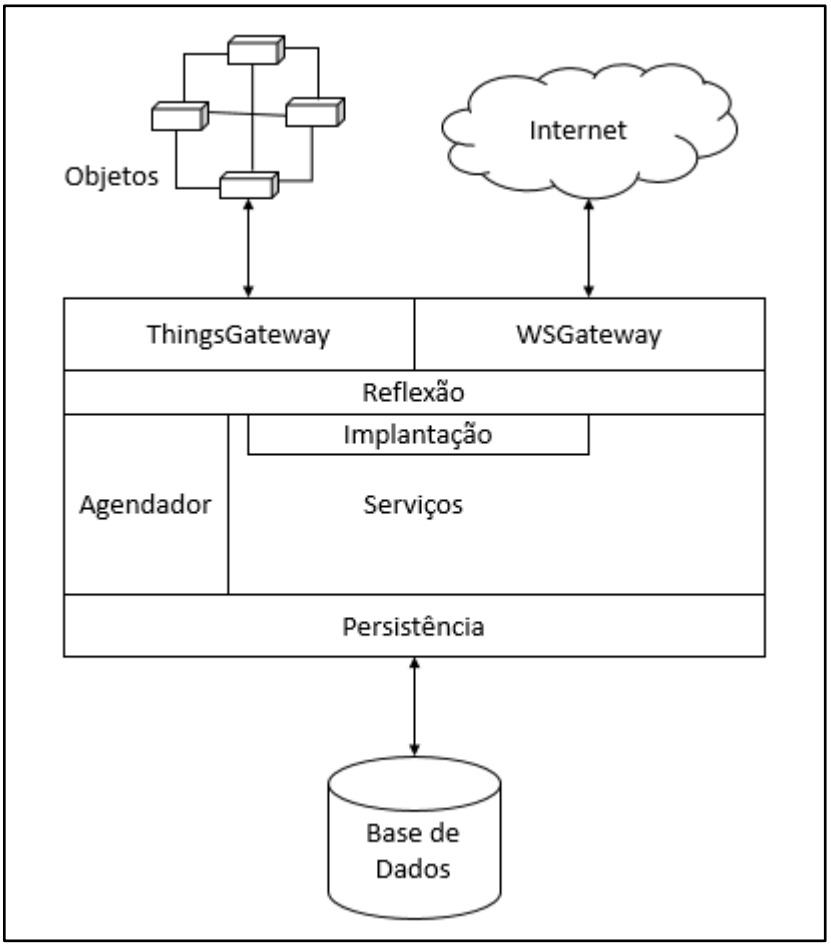

Figura 9: Arquitetura do middleware

O módulo de **Implantação** representa o serviço do middleware responsável por implantar novos serviços. Ele recebe seu código-fonte, o compila e o executa sem que seja necessário interromper o funcionamento do middleware. Desse modo, é possível garantir a escalabilidade do sistema, a nível de *software* e *hardware*, uma vez que novos dispositivos e o código-fonte que os controla podem ser incluídos dinamicamente.

Através do módulo **Agendador**, o middleware permite que a invocação de serviços possa ser agendada por dispositivos e aplicações. Assim, é possível agendar leitura de sensores, tarefas de atuadores e envio de mensagens para aplicações conectadas ao middleware.

O módulo **Persistência** é responsável por persistir informações essenciais para o funcionamento do middleware, como perfil dos dispositivos, aplicações e

usuários conectados, descrição de serviços agendados, histórico de dados capturados por sensores IoT e histórico de acesso ao middleware.

Além dos módulos, são partes integrantes da arquitetura do middleware, embora externos a ele, a rede de dispositivos IoT conectados ao middleware, as aplicações conectadas ao middleware através da Internet e o sistema de banco de dados conectado ao módulo de persistência do middleware.

3.2 Modelo de dados

O middleware conta com uma base de dados relacional, modelada em três áreas conceituais: Geral, Dispositivos e Serviços, mostrado de forma simplificada na Figura 10.

A área Geral é onde os dados de usuários do middleware são armazenados, bem como o histórico de acessos. É composta pelas entidades PESSOA, USUARIO e LOG.

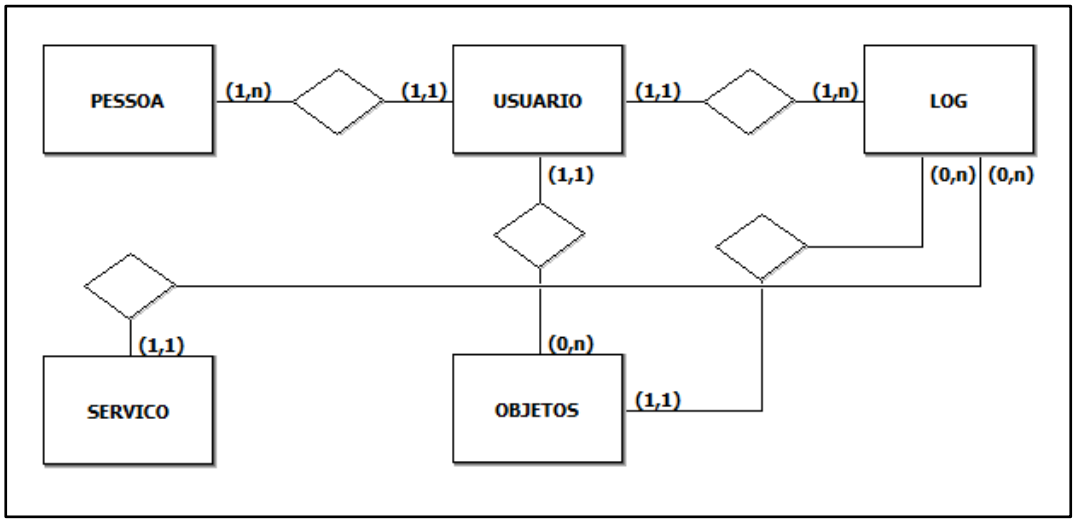

Figura 10: Modelo E-R do middleware

A área Dispositivos é a responsável por armazenar informações sobre os dispositivos conectados ao middleware. Além da entidade OBJETO, ela conta com tabelas específicas para armazenar informações de dispositivos de GPS, RFID e câmeras.

A área Serviços mantém o registro dos serviços disponíveis no middleware, bem como suas informações de agendamento de execução. Ela é composta pela

entidade SERVICO, e outras entidades que armazenam informações sobre operações e parâmetros dos serviços, além de informações sobre seus agendamentos.

O Apêndice A contém diagramas detalhados do modelo conceitual da base de dados do middleware.

3.3 Serviços

Para acessar qualquer serviço do middleware, o cliente, seja ele uma aplicação na Internet ou um dispositivo IoT, deve fazê-lo através de uma das três diretivas dos módulos ***ThingsGateway*** e ***WSGateway***. São elas: INVOKE, DEPLOY e SCHEDULE.

A diretiva INVOKE, acessada via SOAP, REST ou CoAP, representa a porta de entrada para um serviço previamente implantado no middleware. Ela recebe como parâmetro o nome do serviço, a operação do serviço e parâmetros usados pela operação. Ao receber a requisição, o *Gateway* extrai os dados do serviço e os repassa a camada de reflexão e esta, por sua vez, realiza a chamada indicada, devolvendo o resultado. O *Gateway* então envelopa o resultado no formato adequado (SOAP, REST ou CoAP) e o envia ao cliente. Um serviço pode ou não persistir dados no middleware, a depender de suas características. A Figura 11 ilustra o funcionamento da diretiva.

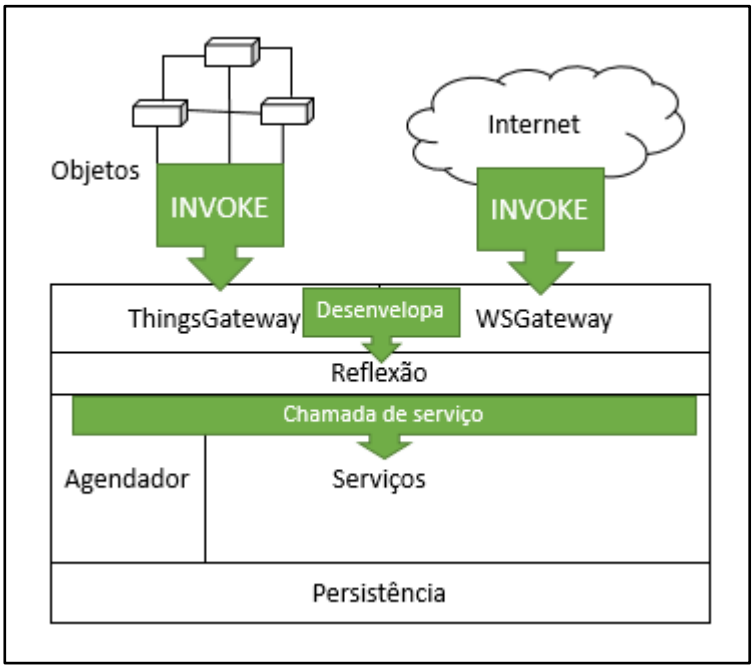

Figura 11: Funcionamento da diretiva INVOKE.

A implantação de um novo serviço no middleware é feita através da diretiva DEPLOY. Ela recebe como parâmetro o código-fonte do serviço – que pode ser a abstração em software de um objeto inteligente, permitindo controla-lo através de serviços –, o compila e o coloca em execução, além de persistir informações desse serviço na base de dados do middleware. A partir de então, o serviço estará disponível através da diretiva INVOKE. A Figura 12 ilustra o funcionamento da diretiva DEPLOY.

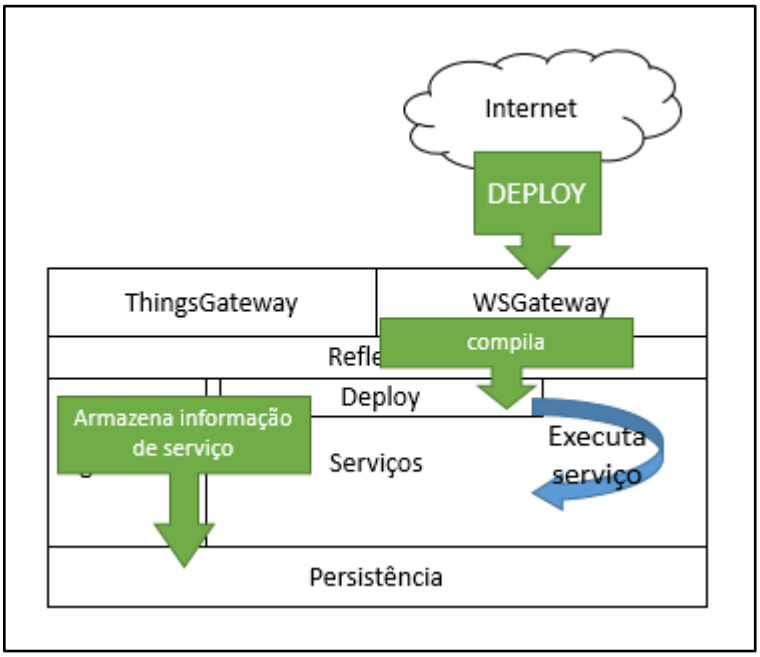

Figura 12: Funcionamento da diretiva DEPLOY.

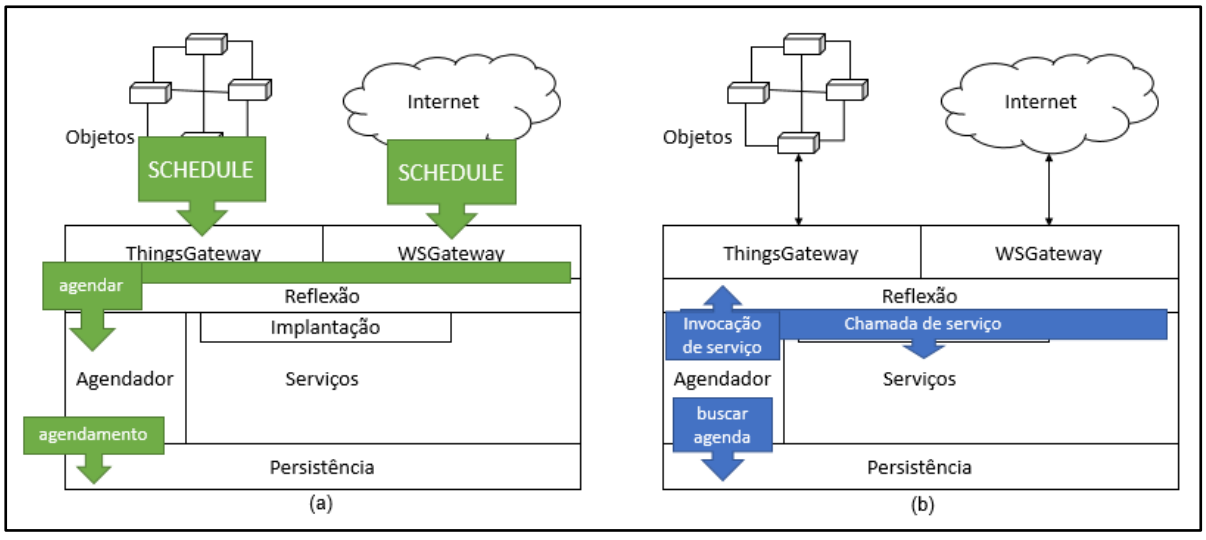

Figura 13: Funcionamento da diretiva SCHEDULE. (a) agendamento de execução de serviço e (b) fluxo de execução de serviço agendado.

A diretiva SCHEDULE permite que a invocação de serviços seja agendada no middleware. Ela recebe como parâmetro o nome do serviço e o nome da operação a ser agendada, bem como os parâmetros para a execução da operação, além da periodicidade com que o serviço será invocado e o início e final do período de agendamento. Todas essas informações serão persistidas na base de dados do middleware e utilizadas pelo módulo **Agendador** para realizar as invocações. A Figura 13 mostra o funcionamento da diretiva SCHEDULE (a) e o fluxo de execução de serviços agendados (b).

4 IMPLEMENTAÇÃO

Um middleware que atenda as demandas da IoT precisa ser, em primeira análise, independente de plataforma. Esse requisito elimina a solução apresentada pelo RMI Java. CORBA, apesar da independência de plataforma, tem sua implementação complexa, pois precisa de “tradução” do objeto para a linguagem do cliente, além do fato de que seu uso sobre os padrões da Web não ocorrer de forma natural.

Dessa forma, por ser independente de plataforma e linguagem de programação, além de ser facilmente implementada sobre os padrões da Web, a arquitetura SOA, mais especificamente suas derivações já apresentadas SOM e SODA, seguindo os padrões já consolidados dos protocolos SOAP, REST e CoAP, é a escolha adequada para o desenvolvimento de um middleware para IoT.

Este capítulo apresenta detalhes da implementação do middleware proposto, como as ferramentas utilizadas e a implementação de cada um de seus módulos.

4.1 Ferramentas Utilizadas

4.1.1 Java

Java[10] é uma linguagem de programação Orientada a Objetos, lançada pela Sun Microsystems em 1995 e adquirida pela Oracle em 2009. Possui uma sintaxe semelhante ao C/C++, mas, ao contrário desta, é independente de plataforma.

Ao compilar um código-fonte escrito em Java, ao invés de um código em linguagem de máquina ser gerado para execução, é gerado um *bytecode* e este é executado pela JVM (máquina virtual Java) da máquina hospedeira. Como dito anteriormente, isso faz com que programas escritos em Java sejam independentes de plataforma. Mas o fato de precisar de uma máquina virtual para ser executado pode tornar lenta a execução dos programas, se comparado com programas escritos em C/C++.

[10] https://www.oracle.com/java/

4.1.2 Apache Tomcat

O Apache Tomcat[11] é um servidor de aplicações Java para a web. Ou seja, é uma implementação de código aberto de tecnologias como Java Servlet – aplicações Java para web que, ao invés de serem executadas em navegadores web, são executadas em servidores – e JavaServer Pages – pode ser considerado como uma abstração em alto nível de um Java Servlet.

Na prática, ele torna as aplicações Java acessíveis através do protocolo HTTP. Para tal, é necessário indicar o caminho (diretório) onde essas aplicações estão localizados. O Apêndice B apresenta o arquivo de configuração do Tomcat (chamado de contexto) e onde encontra-lo.

Foi utilizada a versão 7 do Apache Tomcat no desenvolvimento do middleware.

4.1.3 Apache Axis2

O Apache Axis2[12] é um *framework* que implementa a arquitetura SOA, seguindo o protocolo SOAP, disponível nas linguagens Java e C. É através dele que classes Java serão disponibilizadas para acesso através de requisições SOAP.

4.1.4 Jersey

O Jersey[13] é um *framework* que implementa a arquitetura SOA, seguindo o protocolo REST, disponível para a linguagem Java. Embora o Axis2 tenha suporte também para o protocolo REST, o desenvolvimento de uma aplicação RESTful com Jersey é mais simples, uma vez que este abstrai configurações de baixo nível da comunicação cliente-servidor, possibilitando a implementação utilizando apenas *Java Annotations*[14].

---

[11] http://tomcat.apache.org/
[12] http://axis.apache.org/axis2/java/core/
[13] https://jersey.github.io/
[14] Metadado incluído no código-fonte, mas que não é propriamente parte dele. Vide https://docs.oracle.com/javase/tutorial/java/annotations/.

4.1.5 Californium

O Californium[15] é um *framework* que implementa a arquitetura SODA, seguindo o protocolo CoAP, com foco na escalabilidade e usabilidade.

4.1.6 PostgreSQL

O PostgreSQL[16] é um SGBD objeto-relacional, de código aberto, nascido na Universidade de Berkley, Califórnia (EUA). Possui recursos sofisticados como MVCC (controle de concorrência multi-versão), controle nativo de transações e backup online.

É reconhecidamente o melhor SGBD de código aberto do mercado, tendo ganhado diversos prêmios[17], como *2006 Linux Journal Editors' Choice Awards for Best Database* e *2008 Developer.com Product of the Year, Database Tool*.

4.2 O módulo *ThingsGateway*

O módulo ***ThingsGateway*** foi implementado para ter suporte a requisições do protocolo CoAP, e para tal, foi utilizado sua implementação Californium. Sendo um protocolo baseado em REST, a classe *CoAPGateway*, que implementa este módulo, possui métodos responsáveis por tratar a requisição HTTP GET, identificando qual diretiva do módulo está sendo executada.

4.3 O módulo *WSGateway*

O módulo ***WSGateway*** foi implementado com suporte para requisições tendo SOAP quanto REST. Desse modo, ele conta com as classes *SOAPGateway* e *RESTGateway*.

A implantação da classe *SOAPGateway* se dá com a geração de um arquivo WSDL correspondente e a disponibilização dessa classe como um serviço a ser acessado com o auxílio do Axis2.

---

[15] https://eclipse.org/californium/
[16] https://www.postgresql.org/
[17] https://www.postgresql.org/about/awards/

A implantação da classe *RESTGateway* se dá com a declaração de um *servlet* correspondente no arquivo de configuração web.xml sendo que o acesso a ele será realizado através do Jersey.

4.4 O módulo Reflexão

O Java possui um recurso chamado *Reflection* com o qual é possível indicar dinamicamente, em tempo de execução, um método a ser chamado. Desse modo, com a informação sobre a classe, o método e os parâmetros que representam o serviço recebidos pelos módulos *Gateway*, o módulo Reflexão realiza a chamada indicada.

Por motivos de segurança, o módulo **Reflexão** é capaz de chamar apenas métodos de classes que possuam a *annotation* @ReflectionService, caso contrário, seria possível invocar qualquer classe dentro do middleware, sendo ela um serviço ou não.

4.5 O módulo Implantação

Para implementar características da arquitetura SOA, como a composição e publicação de serviços, utilizamos um recurso Java que permite a compilação e execução de classes em tempo de execução.

A classe *Deploy* implementa o módulo Implantação e consiste em dois métodos que recebem ou o nome da classe a ser implantada e o código-fonte (em formato de texto) ou recebe o endereço do arquivo com o código-fonte, o compila e o executa.

4.6 O módulo Serviços

Este módulo contém todos os serviços do middleware (exceto os serviços especiais de Implantação e Agendamento). Aqui é onde os serviços criados através do módulo Implantação estão contidos e é onde o módulo **Reflexão** encontra os serviços requisitados.

Todas as classes deste módulo devem possuir a *annotation* @ReflectionService.

4.7 O módulo Agendador

Como o nome sugere, este módulo é o responsável por agendar e executar serviços. Uma vez que o serviço esteja implantado no middleware (pela diretiva DEPLOY), ele pode ser agendado. O indicativo de agendamento está armazenado na base de dados do middleware e, com base nessas informações, o módulo **Agendador** irá executar os serviços indicados.

4.8 O módulo Persistência

O módulo **Persistência** é o responsável pelo armazenamento de informações relevantes na base de dados do middleware. Implementado seguindo o padrão de projeto DAO, ele possui uma classe responsável pelas operações CRUD para cada uma das entidades da base de dados.

4.9 Resultados

Para ilustrar o funcionamento do middleware desenvolvido, um serviço de consultas meteorológicas (*Weather*) e de rastreamento por GPS (*GPSService*) foram implementados.

O serviço *Weather* é uma composição do serviço de consulta meteorológicas a partir das coordenadas geográficas de uma localidade, disponibilizado pela Dark Sky[18], e do serviço de codificação de endereços Google Maps Geocoding[19], responsável por traduzir um endereço em coordenadas geográficas. Desse modo, o serviço *Weather* é capaz de fornecer informações meteorológicas a partir de um endereço ou de uma coordenada geográfica. O formato da resposta do serviço depende da tecnologia utilizada para o acesso (SOAP, REST ou CoAP).

---

[18] https://darksky.net/dev/
[19] https://developers.google.com/maps/documentation/geocoding/start

O serviço *GPSService* consiste em um aplicativo android que envia a sua coordenada geográfica para a base de dados do middleware, e esses dados podem ser organizados de modo a fornecer o trajeto percorrido pelo dispositivo enquanto o aplicativo esteve em funcionamento.

Para realizar as requisições das coordenadas do dispositivo, foi implementado um cliente utilizando a linguagem de programação Python, mais especificamente a biblioteca SOAPpy, para acesso através do *SOAPGateway*.

O Apêndice E apresenta algumas ilustrações dos resultados citados.

4.10 Segurança

Segurança em sistemas de middleware, principalmente para a IoT, é uma área tão extensa e complexa que pode, por si só, ser objeto de estudo de um novo trabalho. Assim, este trabalho não se deteve na implementação dessas soluções, embora elas tenham sido, de fato, estudadas.

Desse modo, além do uso de HTTPS com SSL/TLS[20] para autenticação na troca de mensagens, existe a possibilidade de incluir etiquetas com informações para autenticação dentro de mensagens SOAP, através da extensão SAML (*Security Assertions Markup Language*). Além disso, é possível criptografar mensagens SOAP utilizando a extensão XKMS (*XML Key Management Specification*), para registro e troca de chaves públicas.

Ainda a respeito de segurança para troca de mensagens SOAP, existe a especificação WS-Security, que define formatos para a segurança de credenciais, além da preservação e confidencialidade das mensagens. Tudo isso, utilizando criptografia baseada em chaves públicas e privadas, além da inserção de informações de credenciamento de clientes e servidores.

Em REST, a segurança se baseia no uso de HTTPS, com SSL/TLS, uma vez que seu formato de mensagem não permite que informações de controle sejam inseridas, como acontece com o SOAP.

[20] HTTPS (*HyperText Transfer Protocol Secure*) SSL (*Secure Sockets Layer*) e seu sucessor TLS (*Transport Layer Security*) são protocolos que implementam soluções de segurança no protocolo HTTP.

Similar ao REST, a segurança no protocolo CoAP utiliza o protocolo DTLS[21], uma solução baseada no protocolo TLS implementada para comunicação UDP.

4.11 Considerações finais

Devido à complexidade das chamadas de métodos dentro do código-fonte do middleware, este não foi incluído aqui em sua totalidade, mas apenas interfaces, declaração de classes e assinaturas de métodos. Assim, sendo este trabalho uma aplicação de código aberto, o código-fonte está disponível em sua totalidade no sítio https://github.com/mauriciocordeiro/middle.

É importante informar que, além do Axis2 e do Jersey, estão incluídas no projeto bibliotecas para uso do Tomcat, do PostgreSQL, dependências Maven[22], como o Californium e o *framework* Midgard[23], que contém uma série de utilidades para o desenvolvimento de aplicações Java, como a classe *Result*, utilizada como retorno de alguns métodos do middleware e a Interface *Dao*.

Por fim, os métodos que executam a diretiva INVOKE e SCHEDULE, nos módulos *Gateway*, recebem informações como o nome do serviço, a operação do serviço e uma lista de parâmetros que devem ser repassados a operação. Buscando simplificar a passagem dessa lista ao *Gateway*, estabelecemos um objeto JSON que indica o tipo e o valor do parâmetro, como mostrado abaixo.

```
"[{type:\"Integer\",value:5}, {type:\"String\",value:\"teste\"}]".
```

O Apêndice C deste trabalho apresenta o arcabouço das principais classes implementadas no middleware. Já o Apêndice D apresenta Diagramas de Sequência que ilustram o funcionamento das diretivas INVOKE, DEPLOY e SCHEDULE.

[21] *Datagram Transport Layer Security*
[22] https://maven.apache.org/
[23] https://github.com/mauriciocordeiro/midgard

5 CONCLUSÃO

A Internet das Coisas é um novo paradigma, resultado da confluência de distintas áreas da computação, como sistemas embarcados e robótica – sistemas ciberfísicos –, Internet e tratamento semântico de dados. Como resultado, a IoT possibilita a criação de ambientes inteligentes, capazes de monitorar eventos e reagir a eles com pouca ou nenhuma interferência humana.

Os serviços oferecidos pela infraestrutura da IoT possibilitam a implementação de diversos sistemas com alto grau de complexidade, tais como os sistemas Ciberfísicos. São sistemas com alto grau de heterogeneidade em todos os níveis de implementação, tanto vertical quanto horizontalmente. Além do mais, devem oferecer um alto nível de extensibilidade, ou seja, devem suportar que novos dispositivos e serviços possam ser adicionados ao sistema de maneira transparente e segura.

Desta forma, o sistema desenvolvido como estudo de caso neste trabalho, oferece, em certo grau, tais características. É possível adicionar novos dispositivos e serviços ao sistema de maneira transparente e segura. Como discutido no capítulo 4, o trabalho apresentou o Middle, um middleware orientado a serviços, capaz de interconectar objetos inteligentes entre si, através do módulo ***ThingsGateway***, que implementa a arquitetura SODA; e capaz de conectar aplicações na Internet com os dispositivos, através dos módulos ***WSGateway***, que implementa a arquitetura SOA.

Além disso, a solução apresentada permite que novos serviços sejam implantados sem a necessidade se interromper o funcionamento do middleware, função realizada pelo módulo **Implantação**. E esses novos serviços são capazes de interagir com serviços já em funcionamento.

Com o objetivo de minimizar a necessidade de interferência humana, o Middle possui um módulo capaz de agendar a execução de serviços. Tarefa realizada pelo módulo **Agendador**.

Todos os serviços do middleware podem ser acessados através de web service, por meio dos protocolos, já consolidados no mercado, SOAP, REST (quando acessados pelo ***WSGateway***) e CoAP (quando acessados pelo ***ThingsGateway***).

Por fim, o middleware possui de uma camada de persistência, onde dados capturados por sensores conectados são armazenados, podendo ser acessados por qualquer serviço.

5.1 Trabalhos futuros

Como trabalho futuro, será importante implementar soluções de segurança para troca de mensagens, além de suporte aos requisitos semânticos de um middleware para a Internet das Coisas, como a descoberta dinâmica de serviços e de objetos inteligentes com base em informações do contexto onde o Middle está inserido.

A fim de demonstrar utilização da solução desenvolvida em um ambiente real, deseja-se implementar um ambiente inteligente e controla-lo através de uma aplicação web.

REFERÊNCIAS

ATZORI, Luigi. et al. The Internet of Things: A survey, **Computer Networks**, v 54, n. 15, 2010, p. 2787-2805. Disponível em <http://dx.doi.org/10.1016/[...]>. Acessado em 28 mar. 2015.

AL-JAROODI, Jameela; MOHAMED, Nader. Service-oriented middleware: A survey, **Journal of Network and Computer Applications**, v. 35, n. 1, janeiro 2014, p. 211 - 220. Disponível em <http://www.sciencedirect.com/[...]S1084804511001512>. Acessado em 27 mai 2015.

BERNSTEIN, Philip A. Middleware: a model for distributed system services. **Communications of the ACM**, v. 39, n. 2, p. 86-98, 1996. Disponível em <http://dl.acm.org/citation.cfm?id=230809>. Acesado em 10 jun 2015.

BISHOP, Toni A.; KARNE, Ramesh K. A Survey of Middleware. **Computers and Their Applications**. 2003. p. 254-258. Disponível em <http://citeseerx.ist.psu.edu/[...]>. Acessado em 27 mai 2015.

DEUGD, Scott. et al. SODA: Service Oriented Device Architecture. **IEEE Pervasive Computing**, v. 5, n. 3, 2006, p. 94 – 96. Disponível em <http://www.idi.ntnu.no/[...]>. Acessado em 22 abr. 2015.

EISENHAUER, Markus. et al. A development plataform for integrating wireless devices and sensors into Ambient Intelligence systems. **Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops '09. 6th Annual IEEE Communications Society Conference on**, junho 2009, p. 1 – 3, 22 – 26. Disponível em <http://ieeexplore.ieee.org/[...]5172907>. Acessado em 16 abr. 2015.

FERREIRA, Higo Gabriel Cerqueira. **Arquitetura de middleware para Internet das Coisas**. 125 f. Dissertação (Mestrado) – Departamento de Engenharia Elétrica, Faculdade de Tecnologia da Universidade de Brasília, 2014. Disponível em <http://repositorio.unb.br/handle/10482/17251>. Acessado em 13 mar 2015.

GUBBI, Jayavardhana. et al. Internet of Things (IoT): A vision, architectural elements, and future directions. **Future Generation Computer Systems**, v. 29, n. 7, setembro 2013, p. 1645-1660. Disponível em <http://www.sciencedirect.com/[...]>. Acessado em 28 mar. 2015.

HACHEM, Sara. **Service-Oriented Middleware for the Large-Scale Mobile Internet of Things**. 201 f. Tese (Doutorado) – Escola de Ciência e Tecnologia de Versailles, Universidade de Versailles, 2014. Disponível em <https://tel.archives-ouvertes.fr/tel-00960026>. Acessado em 05 abr. 2015.

HERMANN, Mario; PENTEK, Tobias; OTTO, Boris. Design principles for industrie 4.0 scenarios. In: **System Sciences (HICSS), 2016 49th Hawaii International Conference on**. IEEE, 2016. p. 3928-3937. Disponível em <http://ieeexplore.ieee.org/document/7427673/>. Acessado em 17 jun 2017.

JAYASINGHE, Deepal; AZEEZ, Afkham. **Apache Axis2 Web Services**. Packt Publishing Ltd, 2011.

MACKENZIE, C. Matthew et al. Reference model for service oriented architecture 1.0. **OASIS Standard**, v. 12, 2006. Disponível em <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>. Acessado em 25 ago 2015

MIORANDI, Daniele. et al. Internet of things: Vision, applications and research challenges. **Ad Hoc Networks**, v. 10, n. 7, setembro 2012, p. 1497-1516. Disponível em <http://dx.doi.org/10.1016/j.adhoc.2012.02.016>. Acessado em 05 abr. 2015.

NEWCOMER, Eric. **Understanding Web Services: XML, Wsdl, Soap, and UDDI**. Addison-Wesley Professional, 2002.

ROUSE, Margaret. **SOAP (Simple Object Access Protocol) definition**. Disponível em: <http://searchsoa.techtarget.com/definition/SOAP>. Acesso em: 03 maio 2015.

ROUSE, Margaret. **REST (representational state transfer).** Disponível em: <http://searchsoa.techtarget.com/definition/REST>. Acesso em: 03 maio 2015.

SHELBY, Z.; HARTKE, K.; BORMANN, C.. **RFC 7252: The Constrained Application Protocol (CoAP).** 2014. Disponível em: <https://tools.ietf.org/html/rfc7252>. Acesso em: 12 jun. 2017.

SILVA, Hugo. **Service-Oriented Architectures e Interoperabilidade de Aplicações utilizando Web Services**. 52 f. Relatório de Projeto – Departamento de Engenharia Informática, Instituto Politécnico do Porto, 2004.

SILVA, Wellington Lacerda Silveira da. **Detecção de falhas bizantinas em sistemas síncronos particionados.** 2017. 90 f. Dissertação (Mestrado) – Pós-Graduação em Mecatrônica, Universidade Federal da Bahia, Salvador, 2017.

VALENTE, Bruno Alexandre Loureiro. **Um *middleware* para a Internet das coisas**. 92 f. Dissertação (Mestrado) – Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa, 2011. Disponível em <http://hdl.handle.net/10455/6769>. Acessado em 28 mar. 2015.

VALIPOUR, Mohammad H. et al. A brief survey of software architecture concepts and service oriented architecture. **Proceedings of Second IEEE International Conference on International Computer Science and Information Technology (ICCSIT)**, 2009, p. 34 – 38. Disponível em <http://dx.doi.org/10.1109[...]>. Acessado em 20 abr. 2015.

APÊNDICE A – MODELO DE DADOS

Neste apêndice são apresentados os diagramas conceituais do modelo de dados Entidade-Relacional do middleware, divididos em três regiões semânticas: Geral, Serviço e Dispositivos

GNR – Área geral

A área geral corresponde às tabelas para armazenamento de informações do perfil de usuários do middleware (pessoas, dispositivos e aplicações) e o do armazenamento de registros de log.

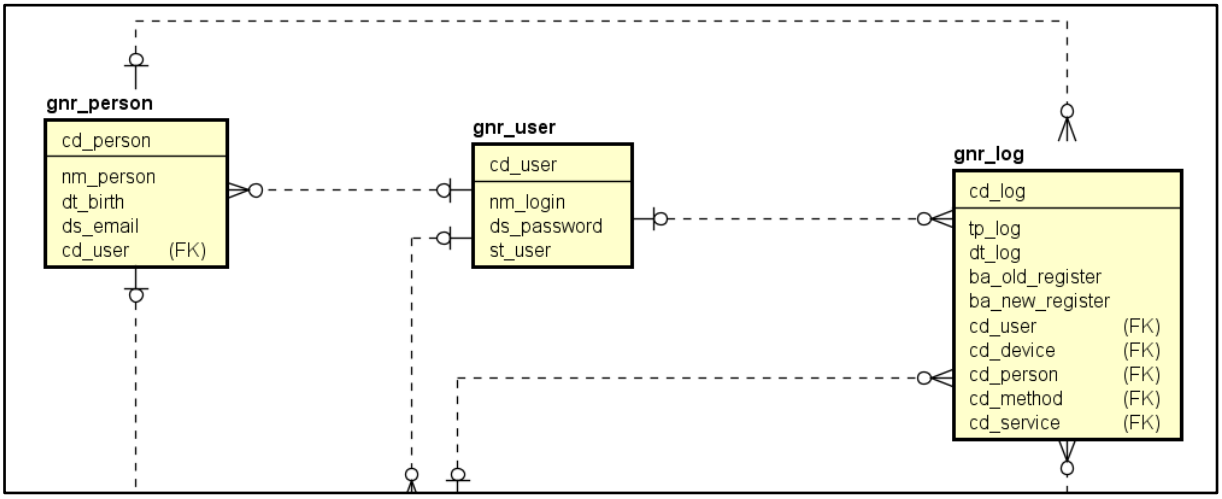

Figura 14: Área geral do modelo de dados do middleware

SVC – Serviços

A área se serviços corresponde às tabelas para armazenamento de informações sobre os serviços disponíveis no middleware, como nome, parâmetros, descrição, código-fonte (no caso de serviços implantados em tempo de execução), bem como informações sobre a execução de serviços agendados.

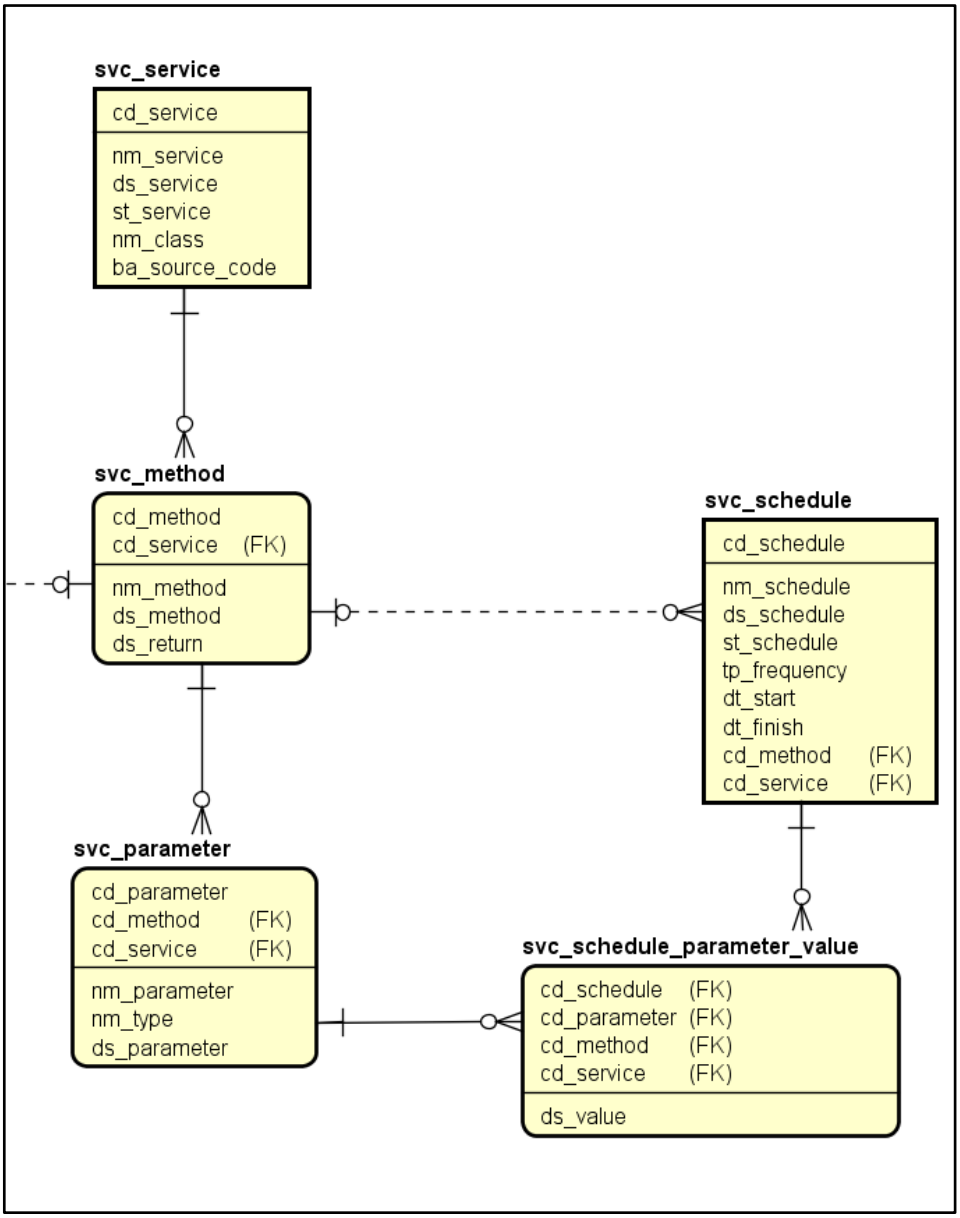

Figura 15: Área de serviços do modelo de dados do middleware

SVC – Área de dispositivos

A área de serviços corresponde às tabelas para armazenamento de informações sobre os dispositivos de IoT conectados ao middleware, como endereço de sensores e atuadores, e o registro histórico dos dados captados por sensores.

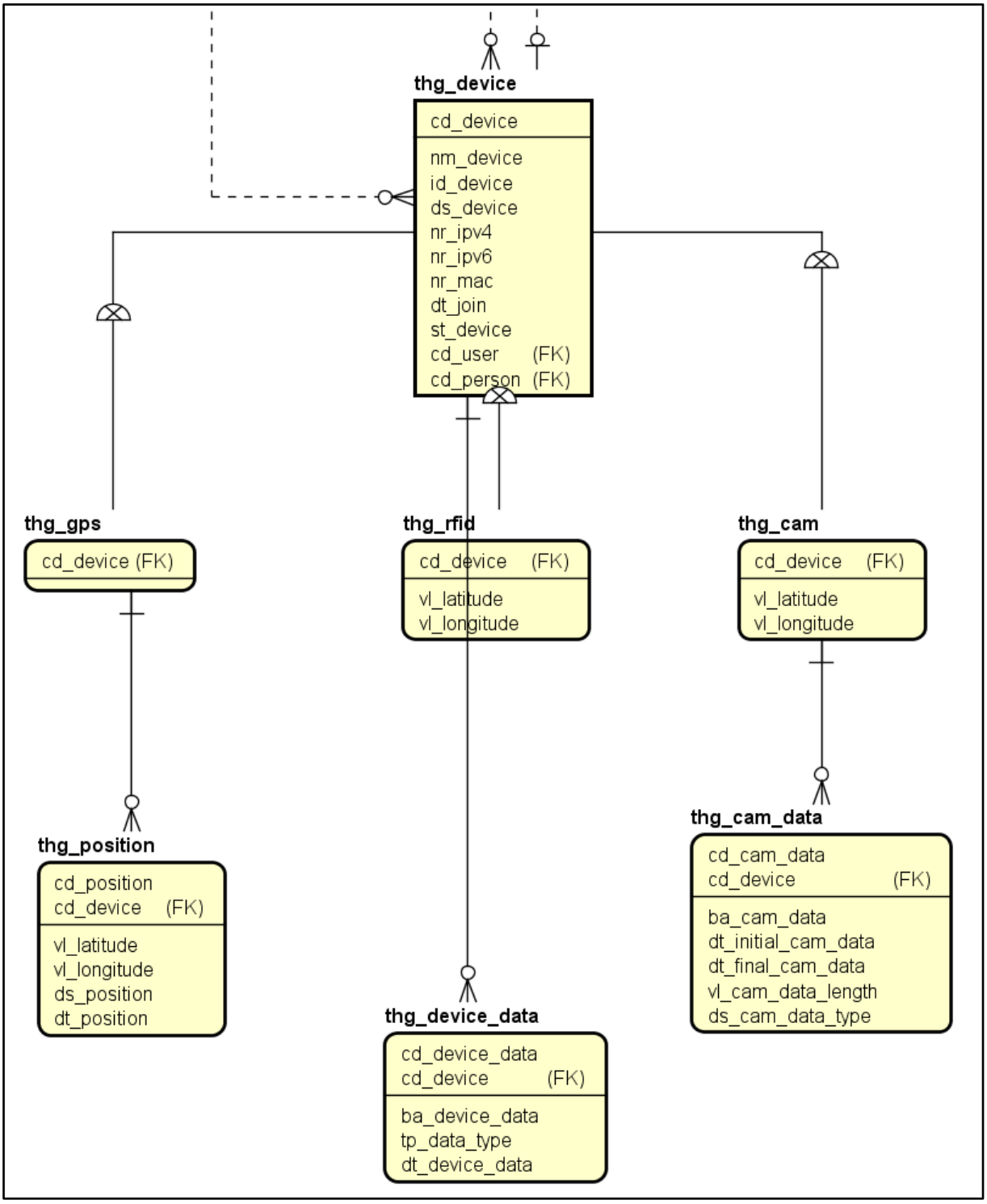

Figura 16: Área de dispositivos do modelo de dados do middleware

APÊNDICE B – CONTEXTO DO APACHE TOMCAT

O arquivo de configuração de contexto do Apache Tomcat está localizado em

`<diretório de instalação do Tomcat>\conf\Catalina\localhost`

Consiste em um arquivo no formato .xml, e seu nome representa o nome do contexto e será usado para acessar os *servlets*, indicado no parâmetro `docBase`.

middle.xml

```
1 <Context docBase="C:\Users\Maurício\Desenvolvimento\middle\web"
2         antiResourceLocking="false" antiJARLocking="false" reloadable="true">
3 </Context>
```

Figura 17: Arquivo de configuração de contexto do Apache Tomcat.

APÊNDICE C – CÓDIGO-FONTE DOS MÓDULOS

C.1 A classe *CoAPGateway*

Classe que implementa o módulo ***ThingsGateway***.

```
1  package br.org.mac.middle.gateway;
2
3  import java.util.Date;
4  import org.eclipse.californium.core.CoapResource;
5  import org.eclipse.californium.core.server.resources.CoapExchange;
6
7  public class CoAPGateway extends CoapResource implements ThingsGateway {
8
9    public CoAPGateway(String name) {
10     super(name);
11   }
12
13   @Override
14   public void handleGET(CoapExchange exchange) {
15     //<trata requisições GET recebidas, identificando a diretiva
16     //e devolve um resultado>
17   }
18
19   @Override
20   public void invoke(String service, String operation, String parameters) {
21     // <executa a diretiva INVOKE>
22   }
23
24   @Override
25   public void deploy(String sourcecode) {
26     // <executa a diretiva DEPLOY>
27   }
28
29   @Override
30   public void schedule(String service, String operation, String parameters,
31                        int periodType, Date start, Date finnish) {
32     // <executa a diretiva SCHEDULE>
33   }
34 }
```

Figura 18: Arcabouço da classe *CoAPGateway*

C.2 As classes que implementam o módulo ***WSGateway***

```java
package br.org.mac.middle.gateway;

import java.util.Date;
import org.json.JSONArray;
import org.json.JSONObject;
import br.org.mac.middle.reflection.ReflectionCaller;
import br.org.mac.midgard.util.Result;

public class SOAPGateway implements WSGateway<String> {

  @Override
  public String invoke(String service, String operation, String parameters) {
    // <executa a diretiva INVOKE>
  }

  @Override
  public String deploy(String sourcecode) {
    // <executa a diretiva DEPLOY>
  }

  @Override
  public String schedule(String service, String operation, String parameters,
                         int periodType, Date start, Date finnish) {
    // <executa a diretiva SCHEDULE>
  }
}
```

Figura 19: Arcabouço da classe *SOAPGateway*

```java
package br.org.mac.middle.gateway;
//<imports>

@Path("/RESTGateway")
public class RESTGateway implements WSGateway<Response> {

  @GET
  @Path("/invoke/{service}/{operation}")
  @Produces("application/json")
  public Response invoke(@PathParam("service") String service,
                         @PathParam("operation") String operation) {
    return invoke(service, operation, null);
  }

  @GET
  @Path("/invoke")
  @Produces("application/json")
  public Response invoke(@QueryParam("service") String service,
                         @QueryParam("operation") String operation,
                         @QueryParam("parameters") String parameters) {
    // <executa a diretiva INVOKE>
  }
}
```

Figura 20: Arcabouço da classe *RESTGateway*.

```java
package br.org.mac.middle.gateway;

//<imports>

@Path("/RESTGateway")
public class RESTGateway implements WSGateway<Response> {
  @GET
  @Path("/deploy/{sourcecode}")
  @Produces("application/json")
  @Override
  public Response deploy(String sourcecode) {
    // <executa a diretiva DEPLOY>
  }

  @GET
  @Path("/schedule")
  @Produces("application/json")
  @Override
  public Response schedule(@QueryParam("service") String service,
                           @QueryParam("operation") String operation,
                           @QueryParam("parameters")String parameters,
                           @QueryParam("periodType")int periodType,
                           @QueryParam("start")Date start,
                           @QueryParam("finnish")Date finnish) {
    // <executa a diretiva SCHEDULE>
  }
}
```

Figura 21: Arcabouço da classe *RESTGateway.*

C.3 A classe *ReflectionCaller*

Classe que implementa o módulo Reflexão.

```java
package br.org.mac.middle.reflection;

import java.lang.reflect.Method;

import br.org.mac.middle.util.ReflectionService;
import br.org.mac.midgard.util.Result;

public class ReflectionCaller {
  public static Result call(String className, String methodName,
                            Class<?>[] types, Object[] args) {
    //<código-fonte do método>
  }
}
```

Figura 22: Arcabouço da classe *ReflectionCaller*

C.4 A classe *Deploy*

Classe que implementa o módulo **Implantação**.

```java
package br.org.mac.middle.services;
//<imports>

public class Deploy {

    /**
     * compila a classe className escrita em sourcecode
     */
    public void compile(String className, String sourcecode) {
        //<compila e executa a classe>
    }

    /**
     * compila a classe indicada em filePath (arquivo .java)
     */
    public void compile(String... filePath) {
        //<compila e executa a classe>
    }
}
```

Figura 23: Arcabouço da classe *Deploy*

C.5 A classe *Scheduler*

Classe que implementa o módulo **Agendador**

```java
package br.org.mac.middle.services;

import java.util.ArrayList;
import br.org.mac.middle.persistence.Schedule;

public class Scheduler {

  private ArrayList<Schedule> schedule; //lista de serviços em execução

  public Scheduler() {
    schedule = new ArrayList<>();
  }

  public void runService(String className, String methodName) {
    //<usa o findService e executa o servico encontrado>
  }

  public void stopService(String className, String methodName) {
    //<cancela o agendamento de um serviço>
  }

  public Schedule findService(String className, String methodName) {
    //busca na base de dados um serviço agendado
  }
}
```

Figura 24: Arcabouço da classe *Scheduler*

C.6 A interface *Dao*

```
 1 package br.org.mac.midgard.dao;
 2
 3 public interface Dao<T> {
 4   public int insert(T obj);
 5   public int insert(T obj, java.sql.Connection connection);
 6   public int update(T obj);
 7   public int update(T obj, java.sql.Connection connection);
 8   public int delete(T obj);
 9   public int delete(T obj, java.sql.Connection connection);
10   public T get(int... keys);
11   public T get(java.sql.Connection connection, int... keys);
12   public ResultSetMap select(SqlFilter  criteria);
13   public ResultSetMap select(SqlFilter  criteria,
14                              java.sql.Connection connection);
15 }
```

Figura 25: Interface *Dao*

APÊNDICE D – DIAGRAMAS DE SEQUÊNCIA

Diagrama de Sequência da diretiva INVOKE (Figura 26). Note que a linha-do-tempo *Gateway* representa os módulos ***ThingsGateway*** e ***WSGateway***.

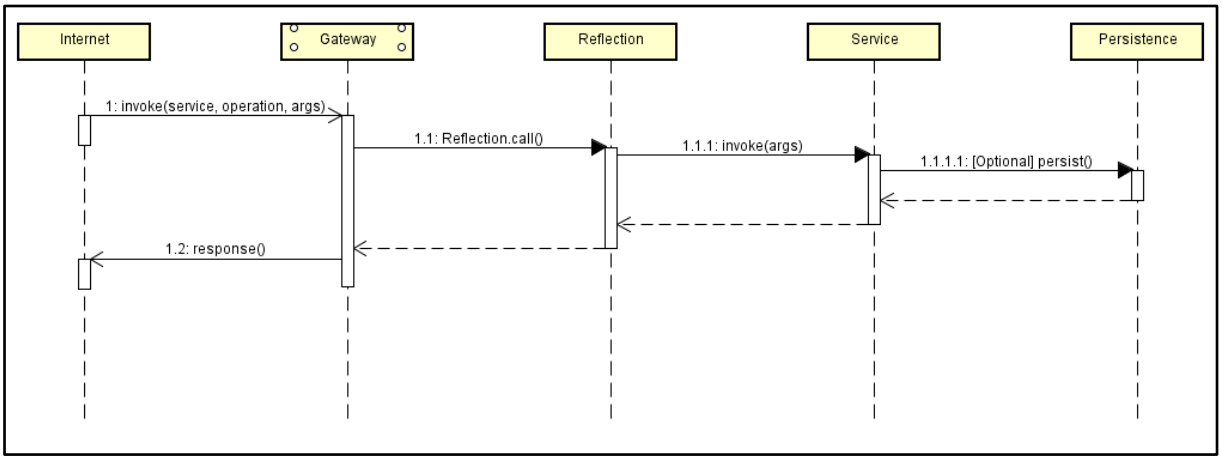

Figura 26: Diagrama de Sequência da diretiva INVOKE

A Figura 27 mostra o Diagrama de Sequência da diretiva DEPLOY.

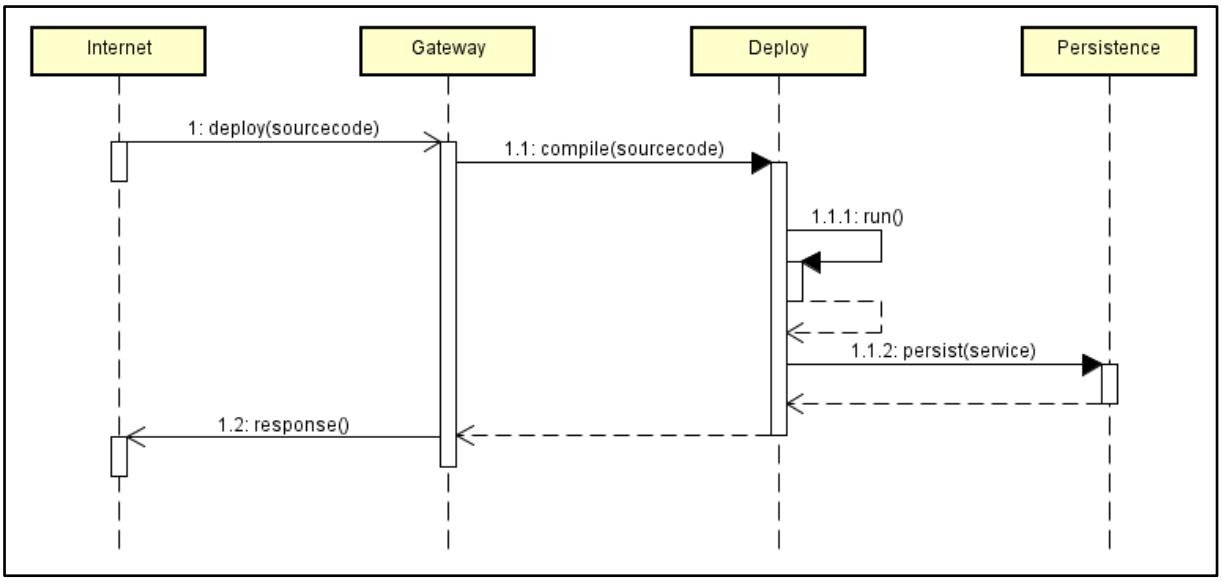

Figura 27: Diagrama de Sequência da diretiva DEPLOY

A Figura 28 apresenta o Diagrama de Sequência da diretiva SCHEDULE, incluindo a sequência de chamadas para a execução de serviços agendados.

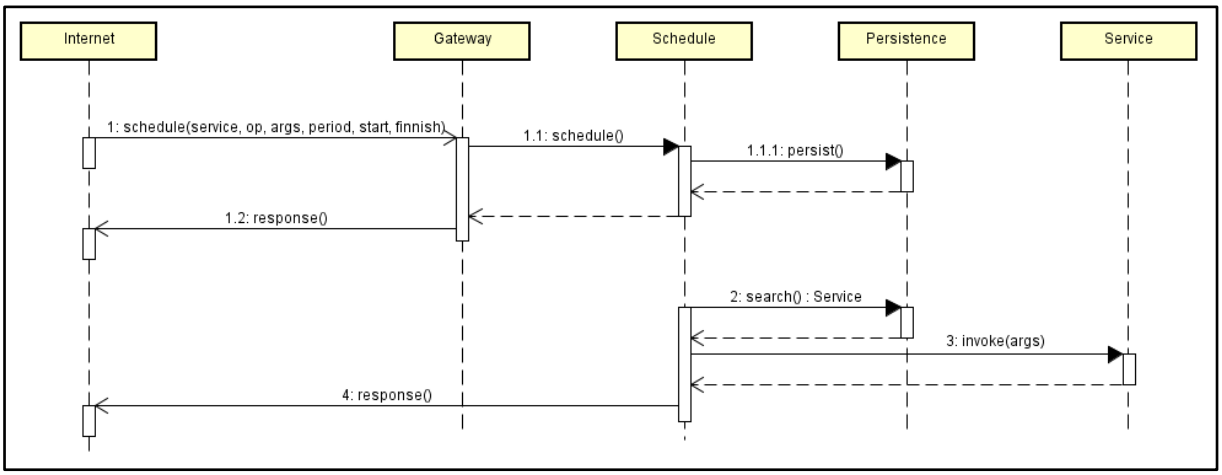

Figura 28: Diagrama de Sequência da diretiva SCHEDULE

APÊNDICE E - RESULTADOS

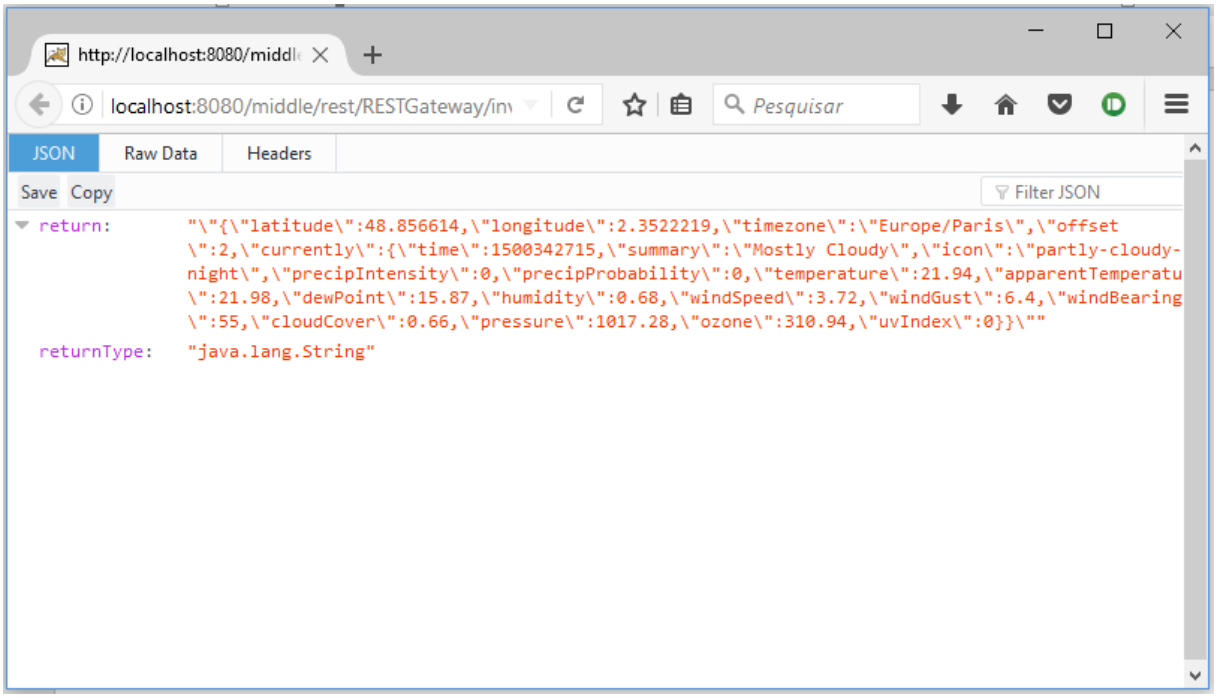

Figura 29: Serviço *Weather* acessado através do RESTGateway

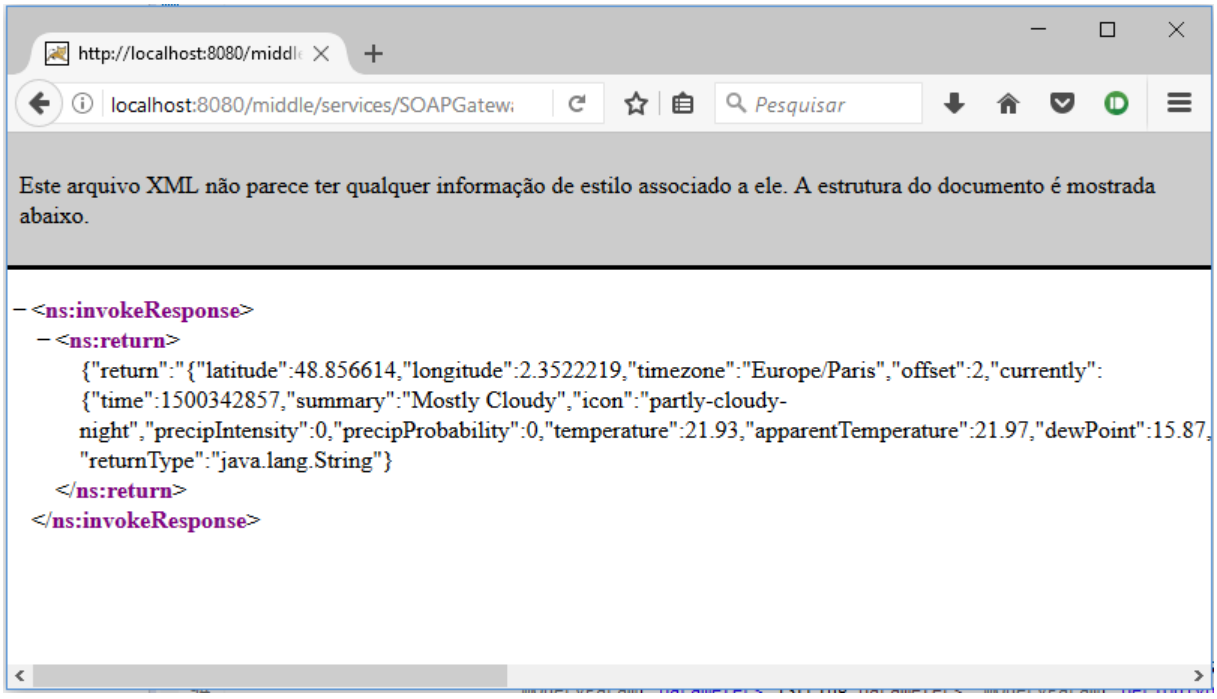

Figura 30: Serviço *Weather* acessado através do SOAPGateway