UNIVERSIDADE FEDERAL FLUMINENSE

FELIPE RALPH DA COSTA SANTOS

FRAMEWORK PARA CONSTRUÇÃO DE SISTEMAS DE MONITORAMENTO BASEADO EM REDES DE SENSORES

NITERÓI

2015

UNIVERSIDADE FEDERAL FLUMINENSE

FELIPE RALPH DA COSTA SANTOS

FRAMEWORK PARA CONSTRUÇÃO DE SISTEMAS DE MONITORAMENTO BASEADO EM REDES DE SENSORES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Sistemas de Computação

Orientador:
Christiano de Oliveira Braga, D.Sc.

Co-orientador:
Raphael Pereira de Oliveira Guerra, Dr.-Ing.

NITERÓI

2015

FELIPE RALPH DA COSTA SANTOS

FRAMEWORK PARA CONSTRUÇÃO DE SISTEMAS DE MONITORAMENTO BASEADO EM REDES DE SENSORES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Sistemas de Computação

Aprovada em Setembro de 2015.

BANCA EXAMINADORA

Prof. D.Sc. Christiano de Oliveira Braga - Orientador, UFF

Prof. D.Sc. Marcos Kalinowski, UFF

Prof. D.Sc. Alexandre Sztajnberg, UERJ

Prof. Dr.-Ing. Raphael Pereira de Oliveira Guerra, UFF

Niterói

2015

Dedico este trabalho a todos que se esforçam para fazer do mundo um lugar melhor, através do progresso coletivo.

Agradecimentos

Agradeço a minha mãe pelo apoio. Ao meu pai por algumas conversas, quando mais novo, que me deixaram com interrogações. A todos que me incentivaram nos momentos mais complicados, pois isso foi muito importante. Ao Raphael Guerra pela paciência. E as condições adversas que me fizeram sair da zona de conforto e tornaram a realização da pós uma experiência única.

Resumo

Há vários desafios na criação de sistemas de monitoramento com redes de sensores: componentes físicos são altamente heterogêneos, sofrem avaria, são substituídos, dados são gerados massivamente e devem ser gerenciados, armazenados e compartilhados com outros sistemas. Neste trabalho, propomos o OSIRIS, um framework para construção de sistemas de monitoramento baseado em redes de sensores. Ele oferece recursos para coletar, processar, armazenar e disponibilizar dados sensoriais para outras aplicações e/ou sistemas, além de monitorar a própria rede sensora. O diferencial do OSIRIS é a flexibilidade na criação de sistemas de monitoramento diversos e o desacoplamento entre a rede de sensores e a aplicação final, que é obtido a partir de um conjunto de abstrações definidos no framework. A proposta apresentada aqui foi avaliada através de diferentes experimentos de submissão e consumo de dados sensoriais com o intuito de mensurar a performance da implementação do OSIRIS.

Palavras-chave: Sistemas de monitoramento, redes de sensores, framework, sensor virtual, abstração de dados.

Abstract

Deploying monitoring systems with sensor networks is very challenging task: physical components are highly heterogeneous, suffer damage, are replaced, data is generated massively and must be managed, stored and made available to other systems. In this work, we propose OSIRIS, a framework for building monitoring systems based on sensing networks. This framework provides resources for monitoring the sensing networks, collecting, processing, and storing data, and an interface for providing data to other applications and/or systems. OSIRIS uses a set of abstractions to offer flexibility for the creation of various monitoring systems and to decouple network physical sensors from data consuming applications. This proposal was evaluated by different experiments covering submitting and consuming of sensorial data to measure the performance of this build of the Osiris.

Keywords: Monitoring systems, sensing networks, framework, virtual sensor, data abstraction.

Lista de Figuras

1.1 Arquitetura SenseWeb [17]. 8

1.2 Arquitetura Sensor Cloud [20]. 8

3.1 Arquitetura do OSIRIS 15

3.2 Digrama de fluxo de dados parcial da arquitetura de base 19

4.1 Difusão de informações um para muitos 21

4.2 Requisição e resposta 22

4.3 Representação da utilização de um grupo de mensagens 26

4.4 Palavras-chave para classificação de informação 29

5.1 Estados e suas transições 36

5.2 Exemplo de rede multicoletada, com 2 coletores 38

5.3 Níveis do processo de abstração de dados 53

5.4 Ilustração da abordagem do sensor virtual 54

5.5 Especializações do VSensor 58

6.1 Pubilsh-subscribe com o RabbitMQ 68

6.2 Abordagem cliente-servidor sobre o RabbitMQ 69

6.3 Abordagem de comunicação síncrona e assíncrona no RabbitMQ 70

6.4 Diagrama de classes do modelo do SensorNet 77

6.5 Diagrama de classes do modelo do VirtualSensorNet 80

8.1 Circuito do OSIRIS utilizado nos experimentos 108

8.2 Resultado para o experimento de medir o tempo para uma transmissão . . 109

8.3 Resultado para o experimento de transmissões simultâneas 110

8.4 Resultado para o experimento de transmissões simultâneas, com conexão dedicada 112

8.5 Resultado para o experimento de composição de sensores virtuais 114

8.6 Resultado para o experimento de vários módulos Collector 115

Lista de Tabelas

4.1 Estrutura da mensagem de requisição OMCP 24

4.2 Estrutura da mensagem de reposta OMCP 25

4.3 Códigos de resposta do OMCP 25

4.4 Esquema de endereçamento de recursos do OMCP 27

4.5 Convenção de endereçamento de recursos do OSIRIS 28

4.6 Recursos nativos do OSIRIS 28

5.1 Tupla para dados de sensoriamento, Value 31

5.2 Definição dos tipos de valores 31

5.3 Tupla para informar nível de recurso fungível, Consumable 32

5.4 Tupla de expressão condicional dos recursos fungíveis, ConsumablesRules . 33

5.5 Definição dos operadores lógicos para expressão condicional 33

5.6 Definição das unidades de tempo do intervalo de captura 34

5.7 Definição dos estados de operação de um recurso 35

5.8 Pacote de publicação do Collector 37

5.9 Campos do recurso Network 40

5.10 Campos do recurso Collector 41

5.11 Campos do recurso Sensor 42

5.12 Definição dos níveis de criticidade para as notificações 45

5.13 Campos da mensagem de notificação 45

5.14 Lista dos recursos e operações OMCP do módulo SensorNet 47

5.15 Definição de Query Strings para a requisição das revisões 48

5.16 Tupla para determinar o tipo de parâmetro, *Param* 49

5.17 Campos do recurso Interface 49

5.18 Definição para os modos de operação do módulo Function 50

5.19 Tupla de valor parametrizado, *Value* 50

5.20 Campos do recurso RequestObject 50

5.21 Campos do recurso ResponseObject 51

5.22 Lista dos recursos e operações OMCP do módulo Function 52

5.23 Campos do recurso DataType 55

5.24 Campos do recurso Converter 56

5.25 Campos do recurso Function 57

5.26 Tupla para dados de sensoriamento do recurso VSensor, *Value* 57

5.27 Campos do recurso VSensor 59

5.28 Campos do elemento *Field* 60

5.29 Campos do recurso VSensor-Link 61

5.30 Campos do recurso VSensor-Composite 61

5.31 Campos do recurso VSensor-Blending 62

5.32 Lista dos recursos e operações OMCP do módulo VirtualSensorNet 65

6.1 Classes To para o módulo Collector 72

6.2 Classes To para o módulo SensorNet 72

6.3 Classes TO para o módulo VirtualSensorNet 73

6.4 Classes TO para o módulo Function 73

6.5 Objeto TO para a mensagem de notificação 74

6.6 Lista de enumerados da API com as definições do OSIRIS 74

7.1 Dados coletados de um dispositivo da rede de sensores do TMON 86

7.2 Dados do objeto InterfaceFnTo 98

7.3 Lista de códigos disponíveis para a consulta 103

Lista de Abreviaturas e Siglas

AMQP	:	Advanced Message Queuing Protocol;
API	:	Application Programming Interface;
HTTP	:	Hypertext Transfer Protocol ;
JVM	:	Java Virtual Machine;
JPA	:	Java Persistence API;
JSON	:	JavaScript Object Notation;
J2SE/JavaSE	:	Java Platform, Standard Edition;
MVC	:	Model-View-Controller ;
REST	:	Representational State Transfer;
RPC	:	Remote Procedure Call;
SATA	:	Serial ATA/Serial AT Attachment;
SOA	:	Service-Oriented Architecture;
URI	:	Universal Resource Identifier;
URL	:	Uniform Resource Locator;

Sumário

1 Introdução **1**

1.1 Definições 2

1.1.1 Sistema de monitoramento 3

1.1.2 Fonte geradora 3

1.1.3 Amostras 3

1.2 Problema 3

1.2.1 Coleta de dados distribuída 3

1.2.2 Disponibilização de dados para múltiplas aplicações concorrentes . . 4

1.2.3 Processamento de dados 4

1.2.4 Redes de sensores com dispositivos heterogêneos 4

1.2.5 Gerência da rede de sensores 5

1.3 Proposta 5

1.4 Trabalhos Relacionados 7

1.5 Organização do trabalho 9

2 Conceitos Básicos **10**

2.1 Comunicação síncrona e assíncrona 10

2.2 Publish-Subscribe 11

2.3 REST - *Representational State Transfer* 11

2.4 Visão geral sobre o middleware RabbitMQ 12

3 Visão Arquitetural do OSIRIS **14**

3.1 Collector 16
3.2 SensorNet 16
3.3 VirtualSensorNet 17
3.4 Functions 17
3.5 Services 18
3.6 Externals 18
3.7 Fluxo de dados 18

4 Especificação da Camada de Comunicação 20

4.1 Requisitos da camada de comunicação 20
4.1.1 Requisito da notificação 21
4.1.2 Requisito da requisição 21
4.1.3 Requisito do baixo acoplamento 22
4.2 OMCP - *OSIRIS Module Communication Protocol* 23
4.3 Comunicação entre módulos 24
4.3.1 Cliente-servidor 25
4.3.2 Publish-subscribe e o grupo de mensagens 26
4.4 Recursos no OMCP 27
4.4.1 Convenção de nomes para recursos do OSIRIS 27
4.4.2 Publicação e inscrição nos grupos de mensagens 28

5 Especificação dos Módulos 30

5.1 Collector 30
5.1.1 Tipos de dados 30
5.1.1.1 Value, dado de sensoriamento 31
5.1.1.2 Consumable, recursos fungíveis dos dispositivos 32
5.1.1.3 ConsumablesRules, regras para a verificação de níveis de recursos fungíveis 32

5.1.1.4 Info, informações extras 33
5.1.1.5 AcquisitionTimestamp, CaptureTimestamp e CapturePrecisionInNano, dados temporais 34
5.1.1.6 CaptureInterval e CaptureIntervalTimeUnit, intervalo de captura de dados da rede sensora 34
5.1.1.7 State, estados de operação 35
5.1.2 Empacotamento e publicação 35
5.1.3 Rede de sensores multicoleta 38
5.2 SensorNet 39
5.2.1 Recursos do SensorNet 39
5.2.1.1 Network 39
5.2.1.2 Collector 40
5.2.1.3 Sensor 40
5.2.2 Funcionamento do módulo 41
5.2.2.1 Armazenamento de dados sensoriais mais recentes 43
5.2.2.2 Inferência do estado de operação de um recurso 43
5.2.2.3 Difusão de atualização dos recursos 44
5.2.2.4 Emissão de alertas de notificação 44
Suporte para notificações do OSIRIS 44
Alertas de troca de estado de operação 45
Alertas para recursos fungíveis 46
5.2.3 Interação com o módulo 46
5.3 Function 48
5.3.1 Recursos do módulo Function 48
5.3.1.1 Interface 49
5.3.1.2 RequestObject 50
5.3.1.3 ResponseObject 50

5.3.2 Funcionamento do módulo 51
5.3.3 Interação com o módulo Function 52
5.4 VirtualSensorNet 52
5.4.1 Camada de abstração 53
5.4.1.1 Processo de abstração de dados 53
5.4.1.2 O sensor virtual 54
5.4.2 Recursos do módulo VirtualSensorNet 55
5.4.2.1 DataType 55
5.4.2.2 Converter 55
5.4.2.3 Function 56
5.4.2.4 VSensor 57
Link: 60
Composite: 61
Blending: 61
5.4.3 Demais funcionalidades do VirtualSensorNet 62
5.4.3.1 Armazenamento de dados sensoriais mais recentes 62
5.4.3.2 Difusão de informações 63
5.4.3.3 Emissão de alerta para notificações 63
5.4.4 Interação com o módulo 64
5.5 Service e External 64

6 Implementação do OSIRIS 67

6.1 Implementação da Camada de Comunicação 67
6.1.1 OMCP sobre RabbitMQ 68
6.1.2 Componentes cliente e servidor 69
6.2 API do OSIRIS 71
6.2.1 Controle abstrato e pacote de utilidades 71

6.2.2 Objetos de transferência 71
6.2.2.1 Collector TO 72
6.2.2.2 SensorNet TO 72
6.2.2.3 VirtualSensorNet TO 73
6.2.2.4 Function TO 73
6.2.2.5 Objeto TO para notificações 74
6.2.3 Definições do OSIRIS 74
6.3 Implementação dos módulos 75
6.3.1 SensorNet 75
6.3.1.1 Detalhes da implementação 76
6.3.1.2 Critério de armazenamento de dados sensoriais 77
6.3.1.3 Inferência do estado de operação do recurso 77
6.3.1.4 Difusão de informações e emissão de alertas de notificação 78
6.3.2 VirtualSensorNet 78
6.3.2.1 Detalhes da implementação 78
6.3.2.2 Camada de abstração 80
Link 81
Composite 81
Blending 81
6.3.2.3 Critério de armazenamento de dados sensoriais 82
6.3.2.4 Inferência do estado de operação do recurso 82
6.3.2.5 Difusão de informações e emissão de alertas de notificação 82

7 Prova de Conceito 83

7.1 TMON 83
7.2 A implementação do TMON com o OSIRIS 84
7.3 Preparando a infraestrutura do framework 84

7.4 Criando um módulo Collector 86
7.4.1 Captura dos dados da rede de sensores 86
7.4.1.1 Dados dos dispositivos do TMON 86
7.4.1.2 Configurando o objeto SensorCoTo 87
7.4.1.3 Configurando recursos fungíveis para o objeto SensorCoTo 88
7.4.1.4 Configurando o objeto CollectorCoTo 89
7.4.1.5 Configurando o objeto NetworkCoTo 89
7.4.2 Submissão de dados 89
7.5 Consumindo dados do SensorNet e VirtualSensorNet 90
7.5.1 Requisições 90
7.5.2 Mensagens dos grupos de mensagens 91
7.6 Configurando o módulo VirtualSensorNet 92
7.6.1 Criando sensores virtuais do tipo Link 92
7.6.2 Criando sensores virtuais do tipo Composite 95
7.6.3 Criando sensores virtuais do tipo Blending 96
7.6.3.1 Criação do recurso 96
7.6.3.2 Definição dos parâmetros de processamento 97
7.7 Removendo recursos do SensorNet e VirtualSensorNet 100
7.8 Criando um módulo Function 100
7.9 Lista com endereços de códigos implementados 102

8 Experimentos 105

8.1 Métricas de avaliação do framework 106
8.2 Configuração do ambiente de experimentações 107
8.3 Resultados 108
8.3.1 Latência para uma transmissão 109
8.3.2 Latência para transmissões simultâneas 110

8.3.3 Latência para transmissões simultâneas com conexão dedicada . . . 111

8.3.4 Comportamento da composição de sensores virtuais 113

8.3.5 Latência para envio de amostras sensoriais por múltiplos módulos Collector 113

9 Conclusão e Trabalhos Futuros 116

9.1 Considerações finais 116

9.2 Contribuições 117

9.3 Trabalhos futuros 118

9.3.1 Módulos distribuídos 119

9.3.2 Camada de comunicação 119

9.3.3 Composição de sensores virtuais mais eficiente 119

9.3.4 Armazenamento das revisões mais eficiente 119

9.3.5 Criação de uma interface de administração 120

9.3.6 Criação da API em outras linguagens 120

Referências 121

Capítulo 1

Introdução

Durante toda a existência, o homem buscou compreender o mundo a sua volta. Observamos a dinâmica das marés, a mudança das estações e o movimento das estrelas. Mapeamos os mais diversos sistemas da natureza simplesmente observando seus fenômenos. Hoje, ainda encontramo-nos em meio a esse mesmo processo de compreensão maior do Universo, o qual, apesar de similar aos nossos antepassados, é diferenciado pelo auxílio de ferramentas que expandem as capacidades humanas, transformando o intangível em tangível. O telescópio e o microscópio são bons exemplos de ferramentas amplificadoras do sentido da visão, duas dentre as inúmeras ferramentas já criadas pelo homem. O telescópio amplia o sentido da visão de modo a enxergar o distante, possibilitando a observação dos mais diversos corpos no Universo. Já o microscópio amplia o sentido da visão, desvelando o mundo microscópico, permitindo a visualização de incríveis estruturas ínfimas, até então, inimagináveis pelo homem. Com o progresso fluindo em diferentes áreas do conhecimento, foi-se necessário buscar novos meios de observação para suprir a demanda investigativa do ser humano. Surgiram os recursos eletroeletrônicos e dispositivos computadorizados. O advento tecnológico nos possibilitou estender nossos sentidos, escutar frequências inaudíveis, aumentar a visão periférica e realizar a telecomunicação. Somos capazes de observar grandes extensões territoriais ao mesmo tempo em que não perdemos de vista o equilíbrio secreto da natureza. Podemos ampliar infinitamente a nossa capacidade de explorar este maravilhoso mundo que nos cerca, utilizando os mais diversos recursos sensoriais construídos pelo homem.

É um fato a existência de redes de sensores [5], onde um conjunto de dispositivos independentes trabalham orquestrados para propiciar entendimento sobre um ambiente desconhecido. É possível confeccionar redes de sensores para as mais diversas finalidades. Sensoriamento térmico [37], sensoriamento luminoso [19], sensoriamento sísmico [36],

sensoriamento acústico [30]. Mares, rios, lagoas, reservatórios, dutos de petróleo e gás, trânsito de automóveis ou pedestres são alguns exemplos de ambientes possíveis de serem monitorados. Existe um campo enorme de possibilidades aguardando a aplicação de redes de sensores para serem entendidos com precisão. Para citar como exemplo de ambiente, temos a Baía de Guanabara. A Baía de Guanabara é um ambiente complexo, com diversas possibilidades de monitoramento, passível de monitoramento de correntes marítimas, salinidade, por conta do encontro de águas de rios com as águas do oceano atlântico, nível de poluição, temperatura, entre outros. Para um futuro próximo, poderemos implantar uma rede de sensores em toda a baía para o monitoramento de diversos parâmetros. Os dados coletados pela rede sensores podem ser disponibilizados online para todos os interessados, contemplando pesquisadores que estudam este ambiente, autoridades e a sociedade civil.

Como é possível perceber, as redes de sensores são ótimas ferramentas, porém construí-las não é uma tarefa simples. São necessários dispositivos, software embarcado, um sistema de informação para armazenar e distribuir dados, ou seja, todo um conjunto de ferramentas tecnológicas atuando para que a ação de monitorar um ambiente seja realizada. Devido ao Laboratório Tempo pesquisar sobre redes de sensores e sistemas embarcados, nós empreendemos esforços para construir uma estrutura capaz de lidar com a aquisição e a disponibilização de dados de rede de sensores. Este trabalho propõe um framework para tratar redes de sensores com dispositivos heterogêneos, a coleta de dados distribuída, a disponibilização de dados para múltiplas aplicações concorrentes, o processamento de dados e a gerência da rede de sensores, enquanto oculta os detalhes de como os dados são obtidos.

Este capítulo está organizado em: a)Seção 1.1: contendo definições, utilizadas durante todo o texto do trabalho; b)Seção 1.2: com apresentação do problema abordado neste trabalho; c)Seção 1.3: com uma breve explanação sobre a solução proposta; d)SeçãoSeção 1.4: listando todos os trabalhos relacionados; e)Seção 1.5: fechando o capítulo com a organização de todo o trabalho, descrevendo um pouco sobre o conteúdo de cada capítulo.

1.1 Definições

Antes de prosseguir com informações sobre o problema e a sua resolução, consideramos pertinente dispor esclarecimentos sobre algumas definições utilizadas no trabalho para melhor entendimento do assunto abordado. O primeiro é o significado do termo "sistema de monitoramento", utilizado no trabalho. Em seguida, abordamos o termo "fonte

geradora". Por último, definimos o que representa o termo "amostra".

1.1.1 Sistema de monitoramento

Entendemos por "sistema de monitoramento" o conjunto de dispositivos e sistemas de informação necessários para monitorar determinado ambiente. Porém, nos capítulos e seções adjacentes, utilizamos o termo "sistema de monitoramento" como sendo o software dedicado a tratar os dados sensoriais. No contexto de utilização deste termo, partimos do pressuposto que a estrutura da rede de sensores já está criada, faltando apenas o sistema de informação para completar a atividade de monitoramento sensorial.

1.1.2 Fonte geradora

O termo "fonte geradora" ou "fonte" exprime a ideia de relação de qualquer recurso que gere dados de sensoriamento e os submeta para o sistema de monitoramento, seja esta oriunda de um dispositivo(hardware) para sensoriamento ou de um componente em software.

1.1.3 Amostras

O leitor irá encontrar, por todo o trabalho, diversas ocorrências relacionadas aos dados sensoriais. Decidimos utilizar o temo "amostra" para especificar os dados sensoriais que têm, como destinatário, um sistema de monitoramento. Os dados sensoriais, as amostras, podem ser oriundos de qualquer tipo de fonte geradora, seja de dispositivos para capturas de estados pontuais do ambiente ou um processo de software.

1.2 Problema

Abordamos aqui, com mais detalhes, os problemas atacados em nossa proposta. Os problemas são a coleta de dados distribuída, a disponibilização de dados para múltiplas aplicações concorrentes, o processamento de dados, redes de sensores com dispositivos heterogêneos e a gerência da rede de sensores.

1.2.1 Coleta de dados distribuída

O consumo energético em uma rede de sensores está relacionado com a quantidade de mensagens trocadas entre os dispositivos. Considerando uma rede de sensores com um

único ponto de coleta como uma árvore, onde a raiz é o coletor e os dispositivos são os nós. Os nós próximo à raiz sofrerão o impacto do maior tráfego de mensagens. A abordagem de um único ponto de coleta gera um consumo energético assimétrico, pois os dispositivos com maior tráfego de mensagens tendem a sofrer substituição de bateria com uma maior frequência. Se existe apenas uma rota de saída de dados e um nó desta rota se torna inoperante, toda a rede de sensores fica inalcançável.

1.2.2 Disponibilização de dados para múltiplas aplicações concorrentes

Disponibilizar dados de sensoriamento para múltiplas aplicações concorrentes pode gerar uma consulta de dados sucessivos, bem como um tráfego extra à rede de sensores, levando a um maior consumo de energia.

Outro problema na disponibilização é que as redes de sensores trabalham com o formato de dados mais conveniente para a sua operação. Uma aplicação que consome dados de diversas redes de sensores precisará ser capaz de lidar com diversos formatos de dados.

Restrições de linguagem de programação, sistemas operacional e protocolos também são um problema para a disponibilização de dados, pois atrapalham a interoperabilidade.

1.2.3 Processamento de dados

O processamento de dados pode ser aplicado para realizar cálculos com diferentes variáveis para gerar um único valor, como por exemplo um valor de velocidade gerado a partir de dados de tempo e posicionamento. O processamento de dados pode ainda ser utilizado para obter tolerância à falha, selecionando um conjunto de valores em detrimento de outros. Em outras situações, os valores podem ser processados, antes de utilizados, para oferecer uma visão mais completa e robusta do ambiente monitorado, técnica conhecida como fusão de dados [6].

1.2.4 Redes de sensores com dispositivos heterogêneos

Os dispositivos de uma rede de sensores podem ser substituídos por conta de avarias e outros problemas. Por outro lado, componentes saem de linha depois de algum tempo por conta da evolução tecnológica. Quando a rede de sensores está incapacitada de comportar dispositivos diferentes, é necessário criar uma outra rede para os novos dispositivos.

Qualquer disparidade de hardware que gere dados em formato diferentes irá impactar a aplicação, pois deverá absorver as mudanças. Realizar adaptações na aplicação para comportar mudanças gera custos, sendo o ideal o consumo de dados em formato constante.

1.2.5 Gerência da rede de sensores

Uma rede com dispositivos eletrônicos necessita ser gerenciada, pois a fonte de energia pode ser exaurida e necessitar de troca, bem como trocar os dispositivos avariados. A medida que a rede de sensores aumenta em quantidade de itens, a manutenção torna-se mais custosa. Quanto maior o tamanho, mais manutenção é necessária e quanto mais manutenção, mais tempo a rede pode permanecer inoperante.

1.3 Proposta

Propomos para este trabalho o OSIRIS, um framework para a construção de sistemas de monitoramento baseado em redes de sensores. O OSIRIS ataca diretamente os problemas, apresentados na Seção 1.2, da coleta de dados distribuída, disponibilização de dados para múltiplas aplicações concorrentes, processamento de dados, redes de sensores com dispositivos heterogêneos e gerência da rede de sensores. A arquitetura do OSIRIS compreende seis módulos: ***Collector***, ***SensorNet***, ***VirtualSensorNet***, ***Function***, ***Service*** e ***External***. A arquitetura define ainda uma camada de comunicação para a comunicação entre módulos e entre módulos e aplicações. A comunicação é realizada utilizando o protocolo OMCP(*OSIRIS Module Communication Protocol*), desenvolvido para este trabalho.

O módulo ***Collector*** é o *gateway* que extrai dados da rede de sensores e transfere para o OSIRIS. Podem coexistir diversos módulos ***Collector*** para uma mesma rede de sensores. O módulo ***SensorNet*** é a representação em software do estado atual da rede física e seus sensores, armazenando dados e metadados, tais como nível de bateria, leituras atuais de cada sensor, árvore de roteamento, entre outros. O módulo ***VirtualSensorNet*** é a entidade que gerencia os sensores virtuais, uma abstração que oculta fontes de dados das aplicações. As fontes de dados podem ser um dispositivo sensor físico ou um elemento de software que processa dados de uma ou mais fontes. O ***Function*** é um módulo compartilhado que trabalha vinculado a um sensor virtual, de modo a realizar o processamento de dados e retornar o resultado deste processamento. O módulo ***Service*** atua como o receptor de mensagens de outros módulos do OSIRIS. O módulo ***External***

é designado como um módulo de funcionalidades não previstas no conjunto de módulos apresentados anteriormente, como por exemplos as aplicações para apresentação de dados ou de repasse de dados para outros sistemas. Já o OMCP regulamenta a troca de mensagens utilizadas na comunicação, suportando troca de mensagens síncronas e assíncronas, com características de baixo acoplamento, sem restrições de linguagem de programação ou sistema operacional.

Para o problema da coleta distribuída, propomos utilizar uma arquitetura com suporte para inúmeros módulos ***Collector*** simultâneos, possibilitando implantar múltiplos pontos de coleta em uma única rede de sensores. Em nossos testes, verificamos que a abordagem arquitetural proposta é capaz de lidar com até 10.000 amostras por segundo.

Para o problema da disponibilização de dados para múltiplas aplicações concorrentes, propomos utilizar os módulos ***SensorNet*** e ***VirtualSensorNet*** realizando o armazenamento dos valores de sensoriamento, evitando consulta para a rede de sensores. Outro ponto da proposta é utilizar um formato de dado padrão, comportando um valor, uma unidade e um símbolo para a representação do dado de sensoriamento. Para interoperar com outros sistemas propomos realizar a comunicação utilizando o protocolo OMCP.

Já para o problema do processamentos de dados, propomos utilizar o módulo ***VirtualSensorNet***, em conjunto com o módulo ***Function***, e a abordagem de sensores virtuais para processar dados de sensoriamento através da combinação ou fusão de dados. O módulo ***Function*** pode ser implementado para as mais diversas finalidades, incluindo seleção de valores para aplicar a tolerância à falha.

Para lidar com redes de sensores com dispositivos heterogêneos, propomos a utilização do formato padronizado de dados de sensoriamento, bem como a abstração sensor virtual para possibilitar a independência hardware. A proposta é simples: uma aplicação deve consumir dados em um único formato e o consumo de dados através do sensor virtual oculta a fonte do dado da aplicação. Quando é necessário alterar uma fonte, basta modificar o sensor virtual, isentando a aplicação de alterações. Para o problema de lidar com diversas redes de sensores ao mesmo tempo, propomos utilizar módulos ***Collector*** dedicados para cada rede existente.

Por último, para o problema da gerência da rede de sensores, propomos o módulo ***SensorNet***, que é a representação em software do estado atual da rede física e seus sensores. O módulo ***SensorNet*** é dotado da característica de notificar sobre o estado e os níveis dos recursos fungíveis(bateria, combustível, entre outros) dos dispositivos existentes na rede de sensores. O módulo ***SensorNet*** também é capaz de notificar falha do ponto

de coleta, bem como falha em toda rede de sensores. Através do módulo ***SensorNet***, para a consulta do estado da rede e os dispositivos ou as notificações, acreditamos que a manutenção da rede de sensores seja precisa, com o conhecimento do nó com defeito e a causa, variando entre os recursos fungíveis e outros problemas.

1.4 Trabalhos Relacionados

Inicialmente, nós observamos alguns sistemas de monitoramento para extrair informações que poderiam ser relevantes para o OSIRIS. Foram observados cinco elementos relacionados ao monitoramento, incluindo aparelhos e sistemas de monitoramento já existentes, que, apesar de não serem trabalhos similares ao OSIRIS, possibilitaram um certo grau de entendimento e inspiração para a etapa de projeto do framework.

Observamos o E-noé [25], um sistema de monitoramento para detectar enchentes e medir o nível de poluição de rios e córregos, utilizando uma rede de sensores sem fio, permitindo que a população e outros órgãos públicos sejam avisados sobre eventuais riscos. No E-noé foram observados os seguintes itens da interface: imagem de um ponto do rio, gráfico para o "índice de perigo", mapa com localização dos sensores, consulta de dados históricos, painéis com gráficos do nível do rio para cada sensor e um gráfico e uma tabela com as medições recentes. Concluímos com a observação do E-noé a necessidade de estabelecer um padrão flexível para lidar com diversos tipos de valores de monitoramento.

O segundo elemento observado foi o Solarwinds [31], um sistema de monitoramento de servidores. O Solarwinds disponibiliza um painel de controle que exibe diversos dados sobre servidores e suas aplicações em execução. Sua observação colaborou para a inclusão dos níveis de recursos fungíveis para o OSIRIS.

Observamos também o Ganglia [21], que é uma ferramenta escalável distribuída para monitoramento de sistemas de computação de alta performance, como *clusters* e *grids*. O Ganglia possibilita a visualização remota de estatísticas, atual ou histórico, de todas as máquinas monitoradas. O Ganglia colaborou para fortalecer a ideia de permitir a consulta de dados históricos para o OSIRIS e a coleta de dados distribuída.

O CEMS [35] é um sistema de monitoramento de gases poluentes chinês. Ele contém uma arquitetura descentralizada para monitorar a emissão de gases poluentes em diferentes localidades. O CEMS contém as atribuições de coleta, processamento e exibição de dados, com funcionalidades de consulta de históricos, via queries, geração de relatórios e serviço de impressão. A arquitetura do CEMS é dividida em três camadas: camada de

apresentação; camada de aplicação e camada de persistência de dados. Para o OSIRIS, a observação do CEMS ajudou a ter uma definição mais clara sobre as camadas do OSIRIS.

Encontramos o trabalho Virtual Sensor [15] com a proposta de abstração de dados para possibilitar o desenvolvedor da aplicação a definir programaticamente requisitos de dados de alto nível da aplicação. O trabalho propõe a abordagem de criar sensores virtuais relacionados para sensores físicos. A aplicação que utiliza o Virtual Sensor não tem conhecimento da fonte de dados, que pode ser alterada de acordo com a necessidade. Notamos que o Virtual Sensor tem uma proposta similar com o conceito dos sensores virtuais utilizado no módulo ***VirtualSensorNet***, porém possui a limitação de exigir a implementação de código, bem como criar toda a estrutura para gerenciar os Virtual Sensors.

Em nossa revisão bibliográfica, encontramos dois trabalhos que se assemelham muito a nossa proposta: o SenseWeb [17] e o Sensor Cloud [20].

O SenseWeb [17] é a proposta de uma infraestrutura para compartilhamento de dados sensoriais que mais se assemelha ao nosso trabalho. Como podemos ver na Figura 1.1, ele define quatro componentes principais: *sense gateway*, *mobile proxy*, *coordinator* e *data transformer*. O *sense gateway* e o *mobile proxy* provêm uma interface uniforme para a comunicação com os sensores. O *coordinator* armazena dados em cache para minimizar o fluxo de dados proveniente diretamente dos sensores e gerencia as necessidades de sensoriamento de cada aplicação para localizar os sensores apropriados. E, finalmente, o *data transformers* manipula os dados coletados antes de repassá-los para as aplicações,

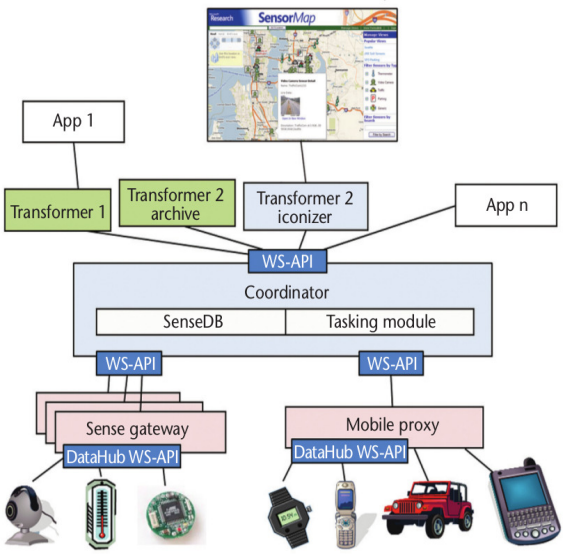

Figura 1.1: Arquitetura SenseWeb [17].

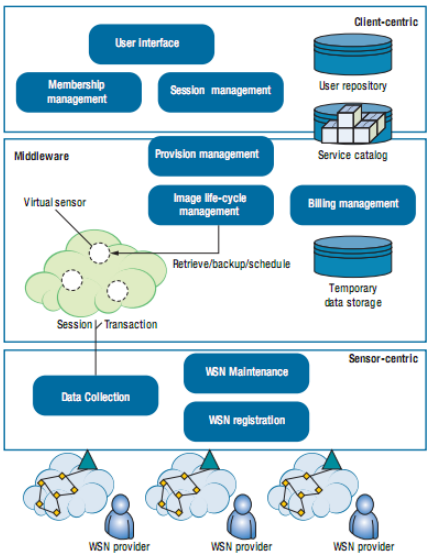

Figura 1.2: Arquitetura Sensor Cloud [20].

seja para melhor visualização, conversão de unidade ou filtragem.

Observamos que não há *cache* ou armazenamento persistente de dados provenientes de *data transformers*. Logo, os dados em estado natural precisam ser manipulados antes de repassados às aplicações e devem ser processados a cada solicitação. Aplicações também precisam estar cientes da necessidade de realizar estas transformações. Além disso, se duas aplicações distintas solicitam dados da mesma rede sensora, haverá tráfego de dados na rede para cada solicitação, o que aumenta o consumo energético. Notamos também que o *coordinator* é um elemento centralizador, o que compromete a escalabilidade. Finalmente, o SenseWeb não oferece um elemento para gerenciar o funcionamento das redes de sensores sem fio, como consumo de energia, estado da árvore de roteamento, etc.

O Sensor Cloud [20] propõe uma arquitetura de nuvem para gerenciar uma rede de sensores virtuais [16]. Os dados dos sensores virtuais são disponibilizados em um serviço de nuvem com suporte para categorias de usuários, pagamentos por serviços, etc. A base da infraestrutura do Sensor Cloud é a camada centrada em sensores, que gerencia diretamente as redes de sensores e repassa os dados amostrais para camadas superiores. Esta camada é funcionalmente equivalente à nossa proposta neste trabalho, e observamos que ela não oferece suporte para vários coletores (*sinks* ou *base stations*) na mesma rede sensora. Ela também não oferece nativamente uma infraestrutura para reconhecer dados replicados oriundos da rede sensora. É muito comum que redes de sensores sem fio repliquem a transmissão de dados para tolerância à falha.

1.5 Organização do trabalho

A dissertação está estruturada da seguinte forma: No Capítulo 3 é descrita a arquitetura que define o framework OSIRIS. No Capítulo 4 apresentamos as especificações para a camada de comunicação e, no Capítulo 5, apresentamos as especificações para os módulos do OSIRIS. No Capítulo 6, descrevemos sucintamente a implementação do framework, realizada em linguagem Java e da camada de comunicação utilizando o RabbitMQ. No Capítulo apresentamos a prova de conceito, com a implementação de um sistema de monitoramento. No Capítulo 8 os experimentos realizados para avaliar a implementação do OSIRIS. Finalmente, o Capítulo 9 traz as conclusões deste trabalho e propostas de trabalhos futuros.

Capítulo 2

Conceitos Básicos

Neste capítulo apresentamos conceitos básicos que serão fundamentais para o entendimento do trabalho em toda a sua completude, abordando, brevemente, elementos de comunicação que podem ser desconhecidos pelo leitor.

O capítulo está organizado em: a)Seção 2.1: abordando definições de comunicação síncrona e assíncrona; b)Seção 2.2: apresentando o publish-subscribe; c)Seção 2.3: explicando brevemente sobre o estilo arquitetural REST; d)Seção 2.4: finalizando com uma visão geral sobre o middleware RabbitMQ.

2.1 Comunicação síncrona e assíncrona

A comunicação entre os módulos do OSIRIS pode ocorrer de modo síncrono ou assíncrono. O conceito de comunicação síncrona e assíncrona utilizado são os seguintes:

- **assíncrono:** comunicação não-bloqueante, onde o remetente envia uma mensagem e continua com o processamento local, sem se preocupar se a mensagem chegou ao seu destino.
- **síncrono:** o remetente da mensagem é bloqueado no envio até receber uma resposta. Algumas implementações podem estabelecer um tempo(*timeout*) para a recepção da resposta, que, uma vez excedido, executa o desbloqueio do remetente. O problema dessa abordagem é: a resposta é descartada pelo remente se chegar após ao desbloqueio.

2.2 Publish-Subscribe

O padrão de troca de mensagens Publish-Subscribe [8] define uma abordagem onde o envio de mensagens entre dois pontos não ocorre diretamente, mas sim, por um intermediário. Este intermediário atua como uma referência onde diversos interessados(*subscribers*) podem se inscrever para obter determinada informação que, após, é enviada por um *publisher* para ser difundida.

O Publish-Subscribe contém as vantagens de baixo acoplamento entre as partes comunicantes e escalabilidade. O baixo acoplamento ocorre por conta dos emissores(*publishers*) não terem ciência da existência dos interessados(*subscribers*), os quais irão consumir as mensagens enviadas. A escalabilidade ocorre por conta da abordagem possibilitar operações paralelas e *caching* e roteamento de mensagens.

2.3 REST - *Representational State Transfer*

O estilo REST [10] é uma abstração dos elementos arquiteturais dentro de um sistema de hipermídia distribuído, com o foco nos papéis dos componentes, considerando suas restrições na interação com outros componentes e a sua interpretação de elementos de dados significativos, ignorando detalhes de implementação e sintaxe do protocolo de um componente. A abstração mandatória no REST é o recurso(*resource*), o qual pode ser uma imagem, um documento HTML ou um objeto de dado qualquer. Os componentes REST executam ações sobre um recurso, utilizando uma representação para capturar ou modificar o estado deste recurso e realizar a transferência da representação entre componentes. O estilo REST contém restrições onde cada recurso possui um identificador, as operações sobre o recurso são padronizadas e a comunicação é realizada sem estado, porém com a representação do estado do recurso.

A implementação mais utilizada do estilo REST é o HTTP. O HTTP(*Hypertext Transfer Protocol*) [9] possibilita a existência de servidores e navegadores escritos em diversas linguagens, rodando sobre inúmeros sistemas operacionais e alocados em lugares diferentes. A comunicação entre o cliente e o servidor possui baixo acoplamento pelo HTTP ser um protocolo sem estado, considerando cada requisição como uma operação independente. Clientes ingressam e se retiram da rede constantemente. Os servidores recebem múltiplas requisições ao mesmo tempo. O HTTP contém recursos endereçáveis e quatro métodos para operar sobre os recursos: GET, POST, PUT e DELETE. O método GET é utilizado

para recuperar informações de um recurso. O método POST é utilizado para criar um novo recurso. O método PUT contém a atribuição de ser utilizado para um processo de atualização de recurso. Já o DELETE deve ser utilizado em situações de remoção de recursos.

2.4 Visão geral sobre o middleware RabbitMQ

O RabbitMQ é um mensageiro capaz de lidar com transmissão dados de um ponto A para um ponto B; desacoplar produtores e consumidores; realizar o enfileiramento e o armazenamento de mensagens para entrega posterior; entregar dados assincronamente; suportar balanceamento de carga e escalabilidade, entre outras características. O RabbitMQ implementa o protocolo AMQP [34], que é um padrão de protocolo de internet aberto para comunicações baseadas em filas de mensagens. O AMQP possibilita comunicação entre componentes construídos em diferentes linguagens de programação, entre sistemas legados e novos, troca de mensagens como um serviço de nuvem, entre outros recursos.

A escolha pelo RabbitMQ se deu por sua característica de possibilitar a implementação de diferentes padrões de trocas de mensagens, ser *opensource*, ser utilizado em aplicações comerciais, com suporte para transações atômicas, difusão de mensagens *broadcast* e *multicast*, garantia de entrega e outros. Possui uma grande comunidade e disponibiliza suporte para diversas plataformas de desenvolvimento (incluindo C#, Java, Python, entre outras) e sistemas operacionais (incluindo Linux e Windows).

Para realizar sua função de middleware de mensagens, o RabbitMQ trabalha com cinco elementos [26]: o produtor, o consumidor, a fila, o *exchange* e os *bindings*. O produtor é o remetente das mensagens, as quais podem ser enviadas para o *exchange* ou para a fila. O *exchange* é o elemento utilizado para rotear mensagens para diversas filas, sendo que cada *exchange* define um algoritmo de roteamento de um conjunto de algoritmos predeterminados do RabbitMQ. A fila é o repositório da mensagem dentro do middleware. As mensagens são organizadas de acordo com a ordem de chegada, armazenadas temporária ou permanentemente, para a posterior entrega aos destinatários, os consumidores. As filas podem se relacionar com os *exchanges*. Esta relação é chamada de *bindings*. Um *binding* pode definir uma *binding key* para realizar uma filtragem de mensagens. Quando uma mensagem aporta em um *exchange*, cópias destas mensagens são repassadas para zero ou mais filas, de acordo com algoritmo de roteamento do *exchange*. Com estes cinco elementos, o RabbitMQ possibilita implementar a comunicação com troca de mensagens

síncronas(RPC, REST, entre outros) e assíncronas(eventos, publish-subscribe e outros), os dois modos que são utilizados no OMCP.

Capítulo 3

Visão Arquitetural do OSIRIS

Neste capítulo apresentamos a arquitetura do OSIRIS, framework para a construção de sistemas de monitoramento baseado em redes de sensores, a qual inclui seis módulos e uma camada de comunicação. A arquitetura de um software pode ser definida como as principais decisões de um projeto [32], e para o OSIRIS, as decisões visam resolver os problemas apresentados na Seção 1.2, da coleta de dados distribuída, disponibilização de dados para múltiplas aplicações concorrentes, processamento de dados, redes de sensores com dispositivos heterogêneos e gerência da rede de sensores.

A Figura 3.1 apresenta a arquitetura do OSIRIS com os seis módulos e a camada de comunicação. Os módulos são: ***Collector***, ***SensorNet***, ***VirtualSensorNet***, ***Function***, ***Service*** e ***External***. Já a camada de comunicação, utilizada para a comunicação entre módulos, utiliza o protocolo OMCP(*OSIRIS Module Communication Protocol*) para regulamentar a troca de mensagens síncronas e assíncronas entre os módulos. Na imagem é possível observar que cada módulo contém um conjunto de ícones para representar suas características funcionais, sendo estas características relacionadas à comunicação, ao tratamento de dados e às operações realizadas por cada módulo.

Definimos para a arquitetura que os módulos apresentados acima da camada de comunicação são componentes pré-implementados do OSIRIS, o ***SensorNet*** e o ***VirtualSensorNet***, enquanto os módulos apresentados abaixo da camada de comunicação (***Collector(s)***, ***Service(s)***, ***Function(s)*** e ***External(s)***) são componentes implementados por demanda, de acordo com a necessidade do sistema de monitoramento. A arquitetura do OSIRIS suporta múltiplos módulos ***Collector***, ***Service***, ***Function*** e ***External***. O conjunto de módulos ***Collector***, ***SensorNet*** e ***VirtualSensorNet*** é essencial para a operação em um sistema de monitoramento implementado com o OSIRIS, pois os módulos

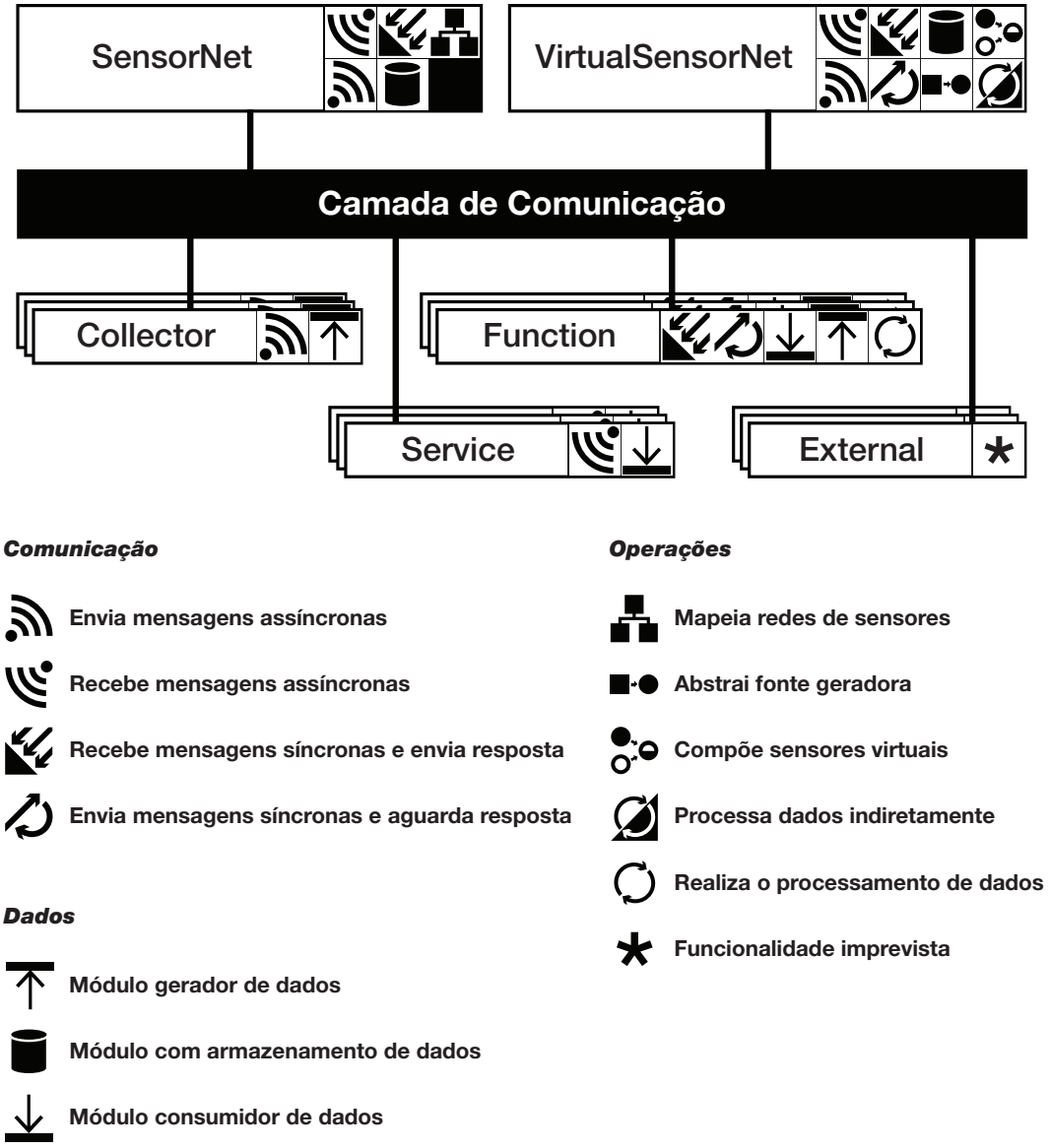

Figura 3.1: Arquitetura do OSIRIS

SensorNet e ***VirtualSensorNet*** operam com dados enviados pelo módulo ***Collector***. O módulo ***Function*** é utilizado quando existe a necessidade de processamento de dados, o módulo ***Service*** é utilizado para receber assincronamente mensagens dos demais módulos e o módulo ***External*** é designado como um módulo de funcionalidades imprevistas na arquitetura.

Não existe restrição de linguagem de programação para a implementação dos módulos, porém a arquitetura impõe a utilização do protocolo OMCP para a troca de mensagens. O OMCP será explicado no Capítulo 4. No Capítulo 5, falaremos mais claramente sobre a especificação para a implementação dos módulos, com definições sobre os recursos que devem ser disponibilizados e o formato de dados utilizado para a comunicação.

O capítulo está organizado em: a)Seção 3.1: abordando características do módulo ***Collector***; b)Seção 3.2: explicando o módulo ***SensorNet***; c)Seção 3.3: abordando aspectos do módulo ***VirtualSensorNet***; d)Seção 3.4: apresentando brevemente o módulo ***Function***; e)Seção 3.5: falando sobre o módulo ***Service***; f)Seção 3.6: abordando aspectos do módulo ***External***; g)Seção 3.7: finalizando com a explicação de como se dá o fluxo de dados entre os módulos ***Collector***, ***SensorNet*** e ***VirtualSensorNet***.

3.1 Collector

O ***Collector*** é um módulo simples, atuando como um *gateway* que extrai dados da rede de sensores e transfere para o OSIRIS. O ***Collector*** adquire os dados da rede de sensores, trabalha estes dados para o formato do OSIRIS e envia-os para a camada de comunicação, via mensagens OMCP assíncronas.

O ***Collector*** pode ser implementado para operar em dispositivos embarcados, como um Arduino [2], BeagleBone [3], Android [12] ou outros sistemas. Uma rede de sensores pode possuir um ou mais módulos ***Collector*** atuando na extração de dados de sensoriamento, abordagem que pode ser utilizada para ampliar os dutos de escoamento de dados. Pelo OSIRIS ter uma arquitetura distribuída, é possível utilizar módulos ***Collector*** trabalhando fisicamente distantes dos outros módulos, bastando apenas que uma conexão com a camada de comunicação possa ser estabelecida.

3.2 SensorNet

O ***SensorNet*** é um dos módulos pré-implementados da arquitetura do OSIRIS, com a atribuição de mapear todas as redes de sensores em operação. Para realizar o mapeamento, o módulo ***SensorNet*** recebe as mensagens assíncronas enviadas pelos módulos ***Collectors*** e cria representações(mapeamento) em software do estado atual das redes de sensores, armazenando dados e metadados com informações sobre as leituras atuais de cada dispositivo sensor, os níveis dos recursos fungíveis(bateria, combustível, entre outros), a topologia de rede, a árvore de roteamento, entre outros. Após construída as representações, o ***SensorNet*** é capaz de observar a atividade de cada dispositivo da rede, detectando alterações nos recursos fungíveis ou uma falha de operação. Quando um dispositivo interrompe abruptamente o envio de dados sensoriais, o módulo ***SensorNet*** detecta essa anomalia e emite uma mensagem OMCP assíncrona para a camada de co-

municação, externalizando a informação que o dispositivo se encontra inativo ou em mau funcionamento. Novos dispositivos ou dispositivos que retornam a operar também geram a emissão de mensagens assíncronas para a camada de comunicação, porém com informação de dispositivo encontrado ou reativado. O módulo possui armazenamento de dados disponível para consultas. As consultas são realizadas por troca de mensagens síncronas, similar a abordagem cliente-servidor, onde o cliente pergunta e o servidor responde.

3.3 VirtualSensorNet

O ***VirtualSensorNet*** é o outro módulo pré-implementado do OSIRIS, com a atribuição de gerenciar os sensores virtuais. O sensor virtual é uma abstração que oculta fontes de dados das aplicações, sendo fontes de dados um dispositivo sensor físico ou um elemento de software que processa dados de uma ou mais fontes. O módulo ***VirtualSensorNet*** ainda é capaz realizar o processamento de dados com o auxílio do módulo ***Function***. O módulo ***VirtualSensorNet*** também recebe mensagens assíncronas emitidas pelos módulos ***Collector***, abastecendo os sensores virtuais com valores de sensoriamento. Conhecendo apenas o sensor virtual, uma aplicação consome o mesmo tipo de dado de sensoriamento, mesmo em casos onde o dispositivo(fonte de dados) é substituído. Com abordagem do sensor virtual, o ***VirtualSensorNet*** se torna capaz de oferecer constância no provimento de informações de monitoramento. O módulo ***VirtualSensorNet*** disponibiliza ainda a composição e mistura de dados de diversos sensores virtuais. A composição de dados opera com dados de dois ou mais sensores virtuais mesclados como um único sensor virtual. Já a mistura opera com o processamento de dados de um ou mais sensor virtual para gerar um novo sensor virtual, o qual terá como valores o resultado do processamento. Para realizar a mistura é necessário utilizar um módulo ***Function*** para realizar o processamento de dados. Todos dos valores de sensoriamento são armazenados e disponibilizados para consultas pelo módulo ***VirtualSensorNet***, similar ao módulo ***SensorNet***. Os sensores virtuais inativos geram a emissão de mensagens assíncronas para a camada de comunicação, a fim de externalizar a informação de inatividade para as aplicações, bem como os sensores virtuais reativados que também informam sua reativação.

3.4 Functions

O ***Function*** é um modulo compartilhado que trabalha vinculado a um sensor virtual, de modo a realizar o processamento de dados e retornar o resultado deste processamento.

A arquitetura do OSIRIS possibilita utilizar diversos módulos ***Function*** simultaneamente, sendo que cada módulo deve ser especializado para um tipo de processamento. O compartilhamento do módulo ***Function*** contém certa similaridade com a abordagem SOA(*Service-Oriented Architecture*) [7], pois os módulos podem ser distribuídos e reutilizados entre diversos sistemas de monitoramento. Com o decorrer do tempo, vislumbramos obter uma biblioteca considerável de módulos ***Function*** para serem reutilizados.

3.5 Services

O módulo ***Service*** é um componente simplificado, similar ao módulo ***Collector***. Ele atua recebendo mensagens assíncronas enviadas para camada de comunicação pelos outros módulos do OSIRIS. O módulo ***Service*** pode ser implementado em interfaces que necessitam de atualização dinâmica dos dados de sensoriamento ou para implantar serviços de notificação de alterações no funcionamento da rede de sensores ou nos sensores virtuais. O módulo ***Service*** possui um baixo consumo de processamento e pode ser implementado para criar aplicativos de *smartphone* para auxiliar a atividade de monitoramento.

3.6 Externals

O módulo ***External*** é designado para implementar funcionalidades imprevistas no conjunto de módulos apresentados anteriormente, como por exemplo aplicações para apresentação de dados ou de repasse de dados para outros sistemas. A arquitetura define este módulo como forma de dar flexibilidade aos utilizadores do OSIRIS de criar novos recursos.

3.7 Fluxo de dados

A arquitetura do OSIRIS define uma estrutura para coletar, armazenar, processar e disponibilizar dados sensoriais. As características são operações simples em uma descrição de alto nível, porém podem ser um pouco confusas no momento de construir um sistema de monitoramento com o framework, pois o entendimento acerca dos dados pode ser pouco clara. Apresentamos um mapa do fluxo de dados na Figura 3.2 para levar ao leitor uma visão de como os módulos ***Collector***, ***SensorNet*** e ***virtualSensorNet*** são integrados para o trabalho conjunto.

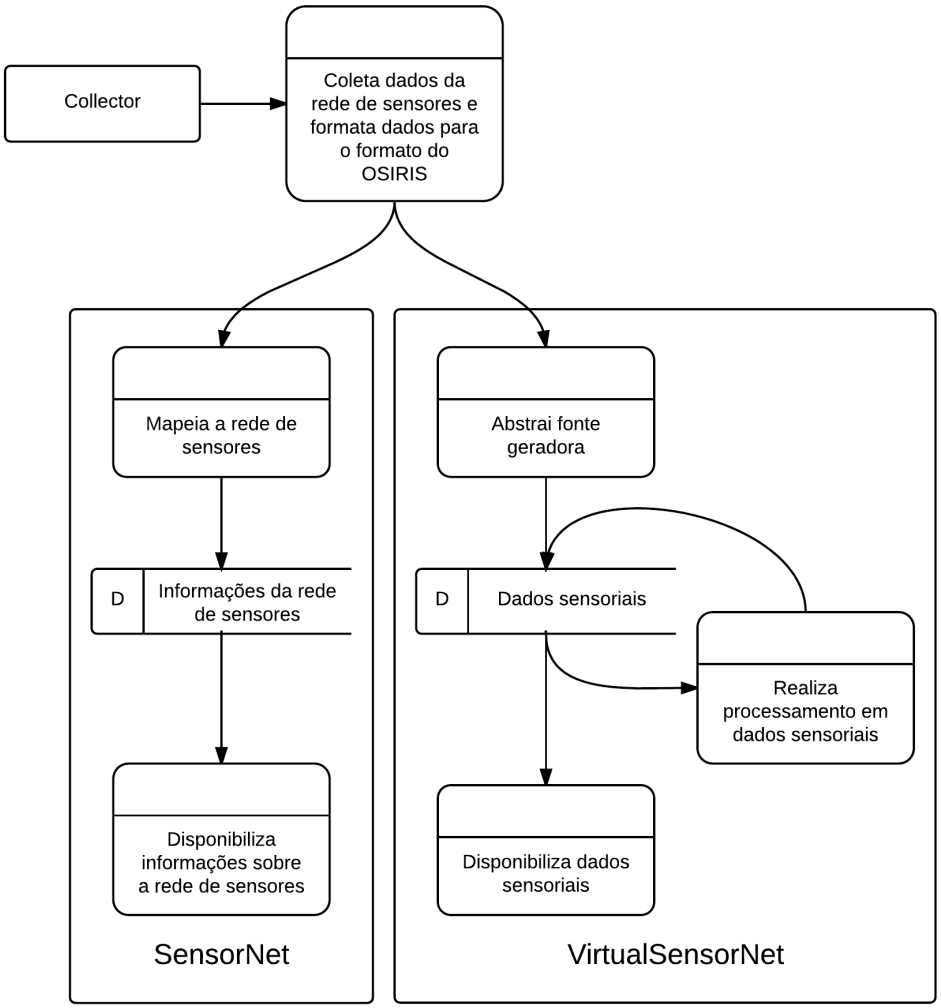

Figura 3.2: Digrama de fluxo de dados parcial da arquitetura de base

A imagem apresenta o módulo ***Collector*** no topo, como o agente responsável por adquirir os dados da rede de sensores e enviar estes dados para o OSIRIS. Em seguida, os dados seguem ao ***SensorNet***, para o mapeamento da rede sensora, e ao ***VirtualSensor-Net***, para abastecer os sensores virtuais com valores de sensoriamento. Uma aplicação interessada nos dados referentes ao monitoramento dos nós da rede de sensores pode utilizar apenas os dados do ***SensorNet***. Já uma aplicação que opera sobre os valores sensoriais deve consumir os dados ***VirtualSensorNet***, pois em caso de substituição do dispositivo sensor, a aplicação não sofrerá o impacto da mudança. Na imagem, é possível observar ainda o processamento de dados do módulo ***VirtualSensorNet***, que ocorre quando auxiliado por um módulo ***Function***.

Capítulo 4

Especificação da Camada de Comunicação

Como explicado no Capítulo 3 a arquitetura está compreendida de seis módulos e a camada de comunicação. Neste capítulo explicamos a especificação da camada de comunicação na arquitetura do OSIRIS. Abordamos as necessidades relacionadas a comunicação entre os módulos que levaram a criação do protocolo OMCP, que regulamenta a troca de mensagens entre os módulos do OSIRIS.

O capítulo está organizado em: a)Seção 4.1: contendo os requisitos da camada da comunicação; b)Seção 4.2: apresentando o OMCP; c)Seção 4.3: abordando os aspectos da comunicação entre os módulos do OSIRIS, utilizando o OMCP;d)Seção 4.4: finalizando com a abordagem da representação de recursos no OMCP.

4.1 Requisitos da camada de comunicação

Por experiência prévia com publish-subscribe [8] no trabalho TMON [37], optamos inicialmente por uma comunicação assíncrona, com mensagens difundidas para um grupo de interessados. Tínhamos em mente utilizar diversos módulos ***Collector*** distribuídos, enviando mensagens assíncronas com informações de sensoriamento para a camada de comunicação. Uma vez submetida para a camada de comunicação, qualquer interessado poderia receber as mensagens. Na prática, essa abordagem se mostrou insuficiente para comportar a consulta do histórico dos valores de sensoriamento disponível no módulo ***VirtualSensorNet***. Deste ponto em diante, estávamos cientes que a comunicação entre os módulos deveria suportar a difusão de informações para inúmeros interessados e a realização de consultas em uma interação pontual. Outro ponto muito debatido foi o acoplamento entre os módulos por conta da comunicação.

Foi definido como os requisitos para a comunicação:

- **notificação:** comunicação baseada em mensagem assíncrona, onde o remente tem o interesse de difundir informações para um ou muitos destinatários, sem se importar se a mensagem chegou ao seu destino;
- **requisição:** comunicação baseada na troca de mensagens síncronas entre dois pontos, similar a abordagem cliente-servidor, onde o cliente realiza uma requisição e o servidor emite uma resposta;
- **baixo acoplamento:** a troca de mensagens entre os módulos deveria simplificada, sem alto custo para a inclusão ou remoção de um módulo do conjunto de módulos do OSIRIS.

4.1.1 Requisito da notificação

Como já explicado, inicialmente pensamos em apenas difundir informações para diversos outros módulos utilizando o publish-subscribe, abordagem de representada pela Figura 4.1. A imagem descreve a ideia difundir informações de um ponto central para diversos pontos adjacentes, sem a necessidade de receber mensagens de resposta confirmando a entrega dos dados. Para este requisito, não existe restrições de limite de tempo para as mensagens atingirem os destinatários.

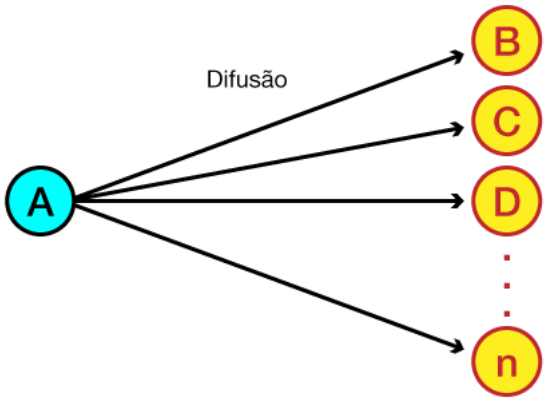

Figura 4.1: Difusão de informações um para muitos

4.1.2 Requisito da requisição

A consulta ao histórico de valores de sensoriamento conta com duas partes, o requisitante e o requisitado, ou o cliente e o servidor, sendo uma ação sincronizada, onde a resposta

ocorre em função do pedido. Seguindo este conceito, a Figura 4.2 demonstra o processo de pergunta e resposta entre os pontos A e B. Se considerarmos o ponto A como um cliente interessado em consultar o histórico de valores de sensoriamento e o ponto B como o módulo ***VirtualSensorNet***, que provê o histórico de valores de sensoriamento, é possível observar que esse paradigma de comunicação é similar ao RPC(Remote Procedure Call), com a execução de funções remotas e o retorno do resultado do processamento.

Figura 4.2: Requisição e resposta

4.1.3 Requisito do baixo acoplamento

A camada de comunicação precisa ser um meio facilitador e não um obstáculo para a adição e remoção de módulos. Neste quesito, o nosso interesse esta diretamente relacionado à possibilidade de ampliar os recursos do framework para permitir o ingresso de módulos escritos em qualquer linguagem e disposto em qualquer sistema operacional.

A comunicação utilizando o publish-subscribe possui um baixo acoplamento para a comunicação, porém funciona apenas para as notificações. Para as requisições, a inspiração partiu da observação do comportamento da Web, onde servidores e navegadores, escritos nas mais diversas linguagens e operando em diferentes sistemas operacionais, se comunicam sem qualquer problema. Na Web, inúmeros clientes entram e saem da rede constantemente sem um alto custo, enquanto os servidores estão sempre aptos a responder requisições de qualquer navegador a qualquer momento. Observamos que o baixo acoplamento da Web ocorre por conta do HTTP(*Hypertext Transfer Protocol*) [9], que implementa o estilo arquitetural REST (*Representational State Transfer*) [10].

Em nosso projeto, optamos em utilizar REST para as requisições, pois o estilo arquitetural atende a necessidade da comunicação síncrona. Descartamos utilizar o RPC pela simples necessidade de conhecer previamente a função remota antes de realizar a requisição, considerando o conhecimento do nome, dos argumentos, do tipo de retorno e etc. Já o estilo REST contém recursos e operações. Cada recurso possui um identificador e está sujeito a um conjunto de operações definidas. A comunicação é realizada sem es-

tado e para ampliar as funcionalidades de um servidor é necessário apenas adicionar mais recursos.

Na próxima seção, descrevemos nossa proposta de protocolo de comunicação, que tenta unir características do REST e do publish-subscribe para atender as necessidades da camada de comunicação.

4.2 OMCP - *OSIRIS Module Communication Protocol*

A comunicação do OSIRIS demanda de um protocolo de comunicação capaz de lidar com requisições, notificações e ter um baixo acoplamento, como explicado na Seção 4.1. De acordo com os requisitos, propomos o OMCP (*OSIRIS Module Communication Protocol*), um protocolo desenvolvido para regulamentar a troca de mensagens entre os módulos do OSIRIS. O OMCP implementa o estilo arquitetural REST, com cinco operações bem definidas, com especificação para formato das mensagens e os códigos retorno. Suporta comunicação síncrona, requisição no paradigma cliente-servidor, e a comunicação assíncrona, notificação, difusão de informações para um ou muitos destinatários.

As cinco operações do protocolo OMCP são: GET, POST, PUT, DELETE e NOTIFY. As quatro primeiras operações são para a comunicação síncrona, com trocas de mensagens de requisição e resposta. Inspirado no HTTP1.1 [9], as operações GET, POST, PUT e DELETE desempenham comportamento similar, sendo o GET utilizado para recuperar informações de um recurso, o POST para criar um novo recurso, o PUT para atualizar um recurso e o DELETE para apagar um recurso existente. Já o NOTIFY é uma operação para a comunicação assíncrona, com o envio mensagens sobre o padrão publish-subscribe.

A Tabela 4.1 mostra a estrutura da mensagem de requisição. A primeira linha consiste dos campos método(operação), recurso e versão do protocolo, respectivamente, separados por um caractere de espaço. O campo método define a operação, o campo recurso contém um identificador de recurso e o campo de versão do protocolo armazena a informação da versão do OMCP utilizada para gerar a mensagem. As outras linhas subsequentes definem os dados de cabeçalho, sendo um campo por linha no formato chave-valor. Os campos são: *date* (data e a hora da requisição), *module* (identificação do módulo destinatário), *source* (identificação do componente de origem da requisição ou difusão), *content-type* (a tipo do dado do conteúdo enviado ou requisitado) e *content-length* (tamanho do conteúdo enviado). Uma linha em branco foi o recurso utilizado para definir onde os dados de

cabeçalho terminam. Todo o conteúdo após a linha em branco é tratado como corpo da mensagem, ou seja, os dados da mensagem. Quando utilizada a operação NOTIFY, o campo recurso é responsável por identificar a natureza da informação enviada. Para as outras operações, o campo recurso endereça, de forma bem específica, um recurso existente em um servidor online.

Tabela 4.1: Estrutura da mensagem de requisição OMCP

<método>	*<recurso>* | *<versão do OMCP>*
date	*<data e hora de envio>*
module	*<módulo destinatário>*
source	*<identificador do requisitante>*
content-type	*<formato do dado(POST/PUT/NOTIFY ou GET para tipo do dado de resposta)>*
content-lenght	*<tamanho do dado(POST/PUT/NOTIFY)>*
<linha em branco>	
<dados(POST/PUT/NOTIFY)>	

Quando utilizada as operações GET, POST, PUT, DELETE é necessário obter uma mensagem de resposta após uma requisição. A Tabela 4.2 descreve a estrutura da mensagem de reposta. A primeira linha contém a versão do protocolo OMCP, o código de retorno e a descrição textual do código de retorno, separados por um caractere de espaço. O campo versão do protocolo OMCP é a versão do protocolo utilizado para montar a mensagem de resposta. O código de retorno é um campo de valor inteiro para informar o resultado da requisição. Todos os códigos de retorno estão descritos na Tabela 4.3. Já a descrição do código de retorno é uma informação textual sobre o código de retorno. Nas linhas subsequentes são distribuídos os dados de cabeçalho. Dentre os dados de cabeçalhos, os campos *module*, *source*, *date*, *content-type* e *content-length* armazenam informações similares da mensagem de requisição. Os cabeçalhos *location* e o *error* contêm, respectivamente, uma URI para endereçar um novo recurso criado(quando utilizado a operação POST), e uma mensagem de erro, caso ocorra algum erro e a requisição não seja efetuada. Após a linha em branco, todos os dados são considerados como corpo da mensagem, abordagem similar utilizada na mensagem de requisição.

4.3 Comunicação entre módulos

O OMCP habilita os módulos do OSIRIS a trocarem mensagens no padrão requisição e resposta e notificação de eventos, pelo conjunto de operações definidos pelo protocolo. A utilização do OMCP para realizar estes dois tipos de comunicação, que abrangem

Tabela 4.2: Estrutura da mensagem de reposta OMCP

<versão do OMCP>	*<código de retorno>*	*<descrição do código de retorno>*
date	*<data e hora de envio>*	
module	*<módulo destinatário>*	
source	*<identificador do requisitante>*	
content-type	*<formato do dado (GET)>*	
content-lenght	*<tamanho do dado (GET)>*	
location	*<caminho onde um recurso foi criado(POST)>*	
error	*<mensagem de erro (em caso de erro)>*	
<linha em branco>		
<dados(GET)>		

Tabela 4.3: Códigos de resposta do OMCP

2xx - Sucesso. Ação recebida com sucesso, entendida e aceita	
200	OK
201	CREATED
4xx - Erro do cliente. A requisição contém sintaxe errônea ou não pode ser realizada	
400	BAD_REQUEST
403	FORBIDDEN
404	NOT_FOUND
405	METHOD_NOT_ALLOWED
5xx - Erro do servidor. Falha do servidor para atender uma requisição aparentemente válida	
500	INTERNAL_SERVER_ERROR
501	NOT_IMPLEMENTED

estruturas cliente-servidor e publish-subscribe, será explicada a seguir.

4.3.1 Cliente-servidor

Criar módulos capacitados a se comunicarem sincronamente, no padrão requisição e resposta, utilizando o OMCP, implica em utilizar as já citadas operações que obrigatoriamente necessitam de uma resposta à requisição, definindo a parte do cliente e a parte o servidor da comunicação. O módulo cliente deve escolher uma operação dentre as operações GET, POST, PUT ou DELETE e um recurso para requisitar o módulo servidor. Por outro lado, o módulo servidor, ao receber a requisição, deve operar sobre o recurso escolhido, se este existir, e emitir ao módulo cliente o resultado de sua operação. A resposta deve conter o código de resposta e, quando necessário, os demais dados requisitados.

4.3.2 Publish-subscribe e o grupo de mensagens

Para a comunicação no paradigma publish-subscrible, o OMCP disponibiliza a operação NOTIFY para o envio de mensagens, como explicado anteriormente. O destaque neste tipo de comunicação é a necessidade da presença do elemento intermediário chamado grupo de mensagens. O grupo de mensagens é o elemento aglutinador, onde interessados sobre um determinado assunto devem se vincular. Na Figura 4.3 é possível observar o esquema utilizando um grupo de mensagens, representado como GM, como elemento de ligação entre o anunciante e os interessados. Cada grupo de mensagens está previamente relacionado a um assunto específico, cabendo aos interessados se vincularem ao grupo quando necessário.

O funcionamento do grupo de mensagens se dá da seguinte maneira: o anunciante deseja divulgar um determinado tipo de informação e, para isso, escolhe um grupo de mensagens específico. Escolhido o grupo de mensagens, o anunciante envia a mensagem para o grupo através da operação NOTIFY. Quando a mensagem aporta no grupo, uma triagem de tema(uma parte do assunto) é realizada e, caso o interesse do membro casar com o tema da mensagem, o mesmo recebe uma cópia da mensagem. É importante destacar que o anunciante não tem qualquer ciência dos interessados e nem os interessados têm ciência do anunciante. Os dois lados conhecem apenas o grupo de mensagens. O OSIRIS possibilita a operação de diversos grupos de mensagens, do mesmo modo que possibilita a operação de diversos módulos personalizados.

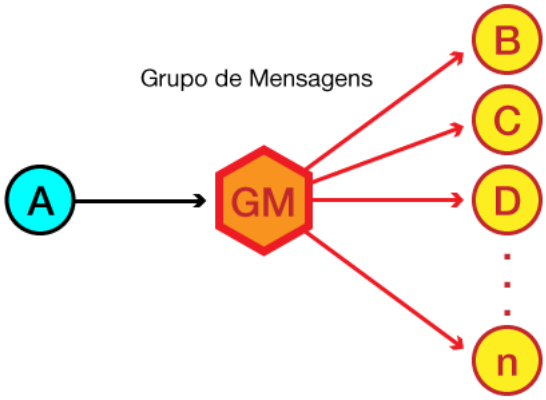

Figura 4.3: Representação da utilização de um grupo de mensagens

4.4 Recursos no OMCP

Definimos para o OMCP um padrão de nomenclatura para endereçamento de módulos, grupos de mensagem e os respectivos recursos. É importante salientar que o OMCP é um protocolo para regular a troca de mensagens entre os módulos. Ele não define um modo de comunicação fim-a-fim, necessitando trabalhar sobre um *middleware* de comunicação com suporte para endereçamento e resolução de nomes. A Tabela 4.4 apresenta o esquema de endereçamento definido para o OMCP. No exemplo de endereço `omcp://virtualsensornet.osiris/vsensor/revisions?from=06/25/2015&limit=10` temos a estrutura composta por nome do protocolo(`omcp://`), seguido do domínio do módulo ou grupo de mensagens. Ainda citando o exemplo, `/vsensor/revisions` representa o caminho e/ou recurso VSensor dentro do servidor ***VirtualSensorNet***, o qual contém o recurso `revisions`. É possível enviar parâmetros para recursos via *Query Strings*. Para utilizar *Query Strings* é necessário adicionar o caractere de `?`, seguido de um ou mais pares de chave-valor, separados pelo caractere `&`.

Tabela 4.4: Esquema de endereçamento de recursos do OMCP

Elementos	Partes do endereço	Objetivo da informação
Protocolo	`omcp://`	Protocolo OMCP
Domínio	`virtualsensornet.osiris`	Endereço do módulo ou grupo de mensagens
Recurso	`/vsensor/revisions`	Caminho e/ou recurso
Query String	`?from=06/25/2015&limit=10`	Conjunto com um ou mais pares de parâmetro-argumento

4.4.1 Convenção de nomes para recursos do OSIRIS

A arquitetura de base do OSIRIS define seis módulos padrões, como apresentado no Capítulo 3. Para todos esses módulos, e mais o grupo de mensagens, foram estabelecidos nomes convencionais de acordo com a Tabela 4.5. A convenção para os módulos nativos do framework é aplicada aos ***SensorNet*** e o ***VirtualSensorNet***, como mostrado na Tabela 4.6. O grupo de mensagens é aplicado para os grupos de mensagens nativo, Tabela 4.6, e para os novos grupos, definidos de acordo com o interesse do sistema de monitoramento criado. Já a convenção para os serviços devem ser aplicada para os módulos ***Service***. Os módulos ***External*** devem utilizar o sufixo `external.osiris` em seu endereço. Os módulos ***Function*** devem utilizar o sufixo `function.osiris`.

Tabela 4.5: Convenção de endereçamento de recursos do OSIRIS

Endereçamento	Descrição
`omcp://{nome}.osiris/`	Módulos nativos do framework
`omcp://{nome}.messagegroup.osiris/`	Grupo de mensagens
`omcp://{nome}.service.osiris/`	Serviço de consumo de mensagens
`omcp://{nome}.external.osiris/`	Módulos do sistema de monitoramento
`omcp://{nome}.function.osiris/`	Módulos de função

Tabela 4.6: Recursos nativos do OSIRIS

Recursos nativos	
`omcp://collector.messagegroup.osiris/`	Grupo de mensagens nativos
`omcp://notification.messagegroup.osiris/`	
`omcp://update.messagegroup.osiris/`	
`omcp://sensornet.osiris/`	Módulos nativos
`omcp://virtualsensornet.osiris/`	

4.4.2 Publicação e inscrição nos grupos de mensagens

Os grupos de mensagens requerem endereçamento para se tornarem encontráveis, tanto por anunciante quanto para o interessado. Porém, há uma ligeira mudança de conceito para ingressar e publicar em grupo de mensagens, pois os recursos de um endereço passam a informar características da mensagem ao invés de apontar para algum recurso existente. Como o grupo de mensagens é um aglutinador de mensagens de único assunto, temas são utilizados para criar subdivisões deste assunto e possibilitar aos interessados obterem especificamente o que desejam. Para exemplificar a estrutura do endereçamento, tanto de ingresso quanto de publicação de mensagens, apresentamos o endereço `omcp://collector.messagegroup.osiris/network0/collector1/sensor2`. Nesta URL, temos `collector.messagegroup.osiris` como identificador do grupo de mensagens e `network0`, `collector1` e `sensor2` como respectivos tema e subtemas. A Figura 4.4 apresenta a classificação da mensagem em temas e subtemas. O tema mais geral é `network0`, com um subtema chamado `collector1`. O subtema `collector1` contém o subtema `sensor2`.

Um anunciante, no momento da publicação para um grupo de mensagens, pode definir um tema e subtemas quaisquer para mensagem enviada, como mostrado na URL exemplo anteriormente. Já um interessado (*i.e* módulo ***Service***) deve ingressar no grupo de mensagens utilizando a mesma estrutura de endereçamento para obter mensagens filtradas por temas. Com esse esquema, um módulo pode receber mensagens apenas de um sensor específico, como no caso o `sensor2` ou de todos os sensores do `collector1`, o qual

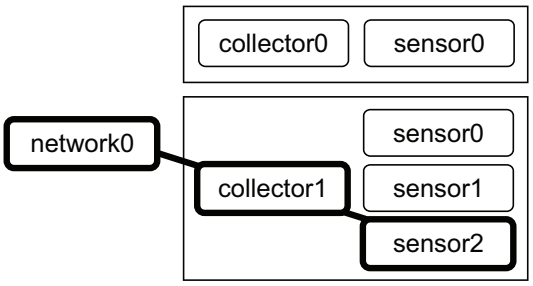

Figura 4.4: Palavras-chave para classificação de informação

contém os sensores `sensor0`, `sensor1` e sensor2.

Capítulo 5

Especificação dos Módulos

Neste capítulo, apresentamos os módulos do OSIRIS, descrevendo suas características básicas para desempenhar sua função dentro do conjunto de módulos do framework.

O capítulo está organizado em a)Seção 5.1: contendo as especificações do módulo ***Collector***; b)Seção 5.2: apresentando as especificações do módulo ***SensorNet***; c)Seção 5.3: definindo as especificações para o módulo ***Function***; d)Seção 5.4: abordando os aspectos do módulo ***VirtualSensorNet***; e)Seção 5.5: finalizando este capítulo com as especificações para os módulos ***Service*** e ***External***.

5.1 Collector

O ***Collector*** é o módulo que realiza a ligação entre a rede de sensores e o sistema de monitoramento. Ele está diretamente acoplado à rede de sensores e, por esse motivo, precisa ser implementado(configurado) para cada novo sistema de monitoramento. O ***Collector*** resolve o problema da captação de dados sensoriais, pois ao receber dados da rede de sensores, ele os empacota para repassá-los ao sistema de monitoramento. Os dados preparados contêm propriedades de rastreabilidade, como a qual rede, coletor e sensor pertencente, valores de monitoramento, informações sobre os recursos fungíveis e informações complementares do dispositivo.

5.1.1 Tipos de dados

As amostras sensoriais submetidas para o OSIRIS englobam um conjunto de dados que devem ser tratados no ***Collector***, sendo este conjunto constituído de dados de sensoriamento e informações relacionadas à configurações. Os dados de sensoriamento são os

dados recebidos dos dispositivos da rede de sensores, como as medições sensoriais capturadas, incluindo informações sobre o tempo da captura. Já as informações de configurações podem englobar os níveis dos recursos fungíveis e as expressões condicionais e um conjunto de informações extras. As informações de configurações são definidas na própria implementação do módulo ***Collector*** e têm por objetivo auxiliar o monitoramento da rede de sensores, que neste trabalho é realizado pelo módulo ***SensorNet***.

5.1.1.1 Value, dado de sensoriamento

Os dados de sensoriamento são valores capturados do ambiente monitorado pelos dispositivos da rede de sensores. Esses dados são recebidos pelo módulo ***Collector*** e passam por uma adequação para o formato de dados de trabalho do OSIRIS, Tabela 5.1. Esta adequação consiste em organizar os dados em forma de tupla, compreendendo informações de valor, nome do campo (*e.g* temperatura, pressão, luminosidade), tipo do valor do dado no ambiente computacional (*e.g* `number`, `logic` e `text`), unidade de medida do valor (*e.g* Celsius, volt, ampere, entre outros) e o símbolo da unidade de medida (*e.g* Hz, V, A, entre outros). A Tabela 5.1 descreve os campos da tupla. O campo `name` comporta o nome. O campo `type` define o tipo do valor, que pode variar entre os tipos `number`, `logic` e `text`. Como descrito na Tabela 5.2, o valor do tipo `number` é utilizado para informar dados numéricos, inteiros ou reais; o tipo `logic` para valores booleanos, verdadeiro ou falso, e o tipo `text` para dados textuais. O campo `value` deve conter o valor do sensorial. O campo `unit` informa a unidade do valor do sensorial. O campo `symbol` armazena o símbolo da unidade do valor do sensorial.

Tabela 5.1: Tupla para dados de sensoriamento, Value

Tipo	**Campo**	**Valor padrão**	**Descrição**
`String`	`name`	N/A	Identificador do dado de sensoriamento
`String`	`type`	`number`, `logic` ou `text`	Tipo do dado do campo valor
`String`	`value`	N/A	Valor sensorial
`String`	`unit`	N/A	Unidade do valor
`String`	`symbol`	N/A	Símbolo da unidade do valor

Tabela 5.2: Definição dos tipos de valores

Tipo	**Descrição**
`number`	Identificação para valores numéricos inteiro ou real
`logic`	Identificação para valores booleanos
`text`	Identificação para valores textuais

Para exemplificar, em um valor de sensoriamento de temperatura de 30,5° Celsius,

temos a seguinte tupla: <“temperatura”,“number”,“30.5”,“Celsius”,“°”>, sendo “temperatura” o nome do sensoriamento, “number” o tipo do valor de sensoriamento, “30.5” o valor da temperatura do tipo real, “Celsius” a unidade de temperatura e “°” o símbolo que representa a unidade de temperatura Celsius. As informações sobre a unidade e o símbolo do valor de sensoriamento têm o objetivo de possibilitar a apresentação correta dos dados de sensoriamento, como, por exemplo, uma aplicação que exibe os dados sensoriais para o operador da rede de sensores.

5.1.1.2 Consumable, recursos fungíveis dos dispositivos

Recursos fungíveis são componentes que sofrem desgaste no decorrer do tempo e devem ser substituídos. Para este trabalho, os recursos fungíveis podem representar os níveis de bateria, combustíveis e outros itens consumíveis, utilizados em um dispositivo sensor. Quando o dispositivo sensor possui a característica de informar os níveis de seus recursos fungíveis, este valor pode ser incluso à amostra sensorial e enviado ao OSIRIS, capacitando, ao módulo ***SensorNet***, disponibilizar estes dados para consulta. O dado para representar o nível do recurso fungível é organizado em forma de tupla, como descrito na Tabela 5.3, <nome, valor>, como por exemplo <bateria, 27>, sendo o valor do tipo inteiro de 1 a 100, representando a porcentagem restante do recurso.

Tabela 5.3: Tupla para informar nível de recurso fungível, Consumable

Tipo	Campo	Valor padrão
String	<nome>	N/A
Integer	<valor>	De 0 a 100(porcentagem)

5.1.1.3 ConsumablesRules, regras para a verificação de níveis de recursos fungíveis

Quando há a necessidade de monitorar o nível dos recursos fungíveis, expressões condicionais podem ser incluídas à amostra sensorial, a fim de que verificações sobre os níveis sejam realizadas. O módulo ***SensorNet*** é capaz de realizar tais verificações e emitir alertas em caso de níveis anormais. A condição segue o formato <nome do alerta, nome do recurso, operador lógico, valor do limite, mensagem para o alerta>, como descrito na Tabela 5.4.

O campo name é o identificador da expressão condicional. O campo consumableName é o nome do recurso fungível presente na amostra sensorial. O campo limitValue define o valor limite do nível do recurso fungível. O campo operator define o operador lógico

Tabela 5.4: Tupla de expressão condicional dos recursos fungíveis, ConsumablesRules

Tipo	Campo	Valor padrão	Descrição
`String`	`name`	N/A	Identificador da condição
`String`	`consumableName`	N/A	Nome do recurso fungível
`String`	`operator`	>, <, >=, <=, == e !=	Operador lógico
`Integer`	`limitValue`	de 0 a 100(porcentagem)	Valor limite do recurso
`String`	`message`	N/A	Mensagem enviada no alerta

utilizado no momento para comparar o nível do recurso fungível e o valor limite definido. O campo `message` pode comportar uma mensagem relacionada a ativação da condição. A mensagem é enviada quando um alerta de recurso fungível é emitido pelo módulo ***SensorNet***. O conjunto dos operadores lógicos contém seis tipos, como descritos na Tabela 5.5: maior, menor, menor ou igual, maior ou igual, igual e diferente. Na mesma tabela é possível reparar o modo de comparação desses operadores, onde x é o valor do recurso fungível e y o limite informado na condicional.

Tabela 5.5: Definição dos operadores lógicos para expressão condicional

Símbolo	Nome	Comparação entre x e y
<	menor	x < y
>	maior	x > y
<=	menor ou igual	x <= y
>=	maior ou igual	x >= y
"=="	igual	x == y
!=	diferente	x != y

5.1.1.4 Info, informações extras

Em algumas situações pode ser necessário que informações de natureza diversa acompanhem a submissão de dados do ***Collector*** para o interior do sistema de monitoramento. Para dar suporte a tal necessidade, incluímos o recurso de informações extras, `info`, o qual constitui em um conjunto de tuplas chave-valor, onde cada tupla representa a informação em si. Em uma tupla, a natureza da informação é definida pela chave, o metadado, enquanto a informação em si é atribuída ao valor. O resultado final é uma lista de chaves-valor, com definições personalizadas, para atender a necessidade do sistema de monitoramento criado. Alguns exemplos de informações utilizadas são: informações sobre o hardware dos dispositivos, sobre o sistema operacional utilizado na rede de sensores, sobre topologia de rede, incluindo elementos de parentesco em uma topologia tipo árvore, entre outros.

5.1.1.5 AcquisitionTimestamp, CaptureTimestamp e CapturePrecisionInNano, dados temporais

Em um determinado momento, o dispositivo de sensoriamento, pertencente à rede de sensores, captura o estado do ambiente através de seus componentes sensoriais. Essa captura é transformada em informação e trafega do dispositivo, passando pela rede de sensores, até um ponto de coleta para ser repassada ao sistema de monitoramento. No ***Collector***, o momento da captura do sensor é mapeado para o campo `captureTimestamp` e o momento da chegada do dado no coletor é mantida no campo `acquisitionTimestamp`. O tempo de captura sensorial pode ser preciso em nano segundos via informação complementar do campo `capturePrecisionInNano`. Estes campos devem ser inclusos como informações de tempo do dado de sensoriamento para a submissão da amostra ao sistema de monitoramento.

5.1.1.6 CaptureInterval e CaptureIntervalTimeUnit, intervalo de captura de dados da rede sensora

Um dispositivo eletrônico trabalha com um intervalo de tempo para suas operações. Esse intervalo também é considerado para o tempo entre as verificações do dispositivo sobre o ambiente monitorado. O interessante do intervalo de monitoramento, quando fixado, é a possibilidade de identificar se um dispositivo se mantém em atividade ou deixou de operar. Por tal motivo, adicionamos o suporte, via campo `captureInterval`, para a informação sobre o intervalo de monitoramento da rede sensora no pacote de dados enviado para o sistema de monitoramento. A precisão pode ser definida como nano segundos, micro segundos, milissegundos, segundos, minutos, horas e dias, como informado na Tabela 5.6. O campo `captureIntervaltimeUnit` é utilizado para informar a unidade de tempo do intervalo de captura.

Tabela 5.6: Definição das unidades de tempo do intervalo de captura

NANOSECONDS
MICROSECONDS
MILLISECONDS
SECONDS
MINUTES
HOURS
DAYS

5.1.1.7 State, estados de operação

Quando os dispositivos da rede de sensores têm a capacidade de informar sobre seu estado de operação, este deve ser realizado através do campo `state`. A Tabela 5.7 descreve os cinco estados de operação suportados. O estado NEW representa uma adição, quando um novo dispositivo é encontrado. O estado INACTIVE representa uma situação onde um dispositivo se encontra desativado. O estado UPDATED representa um estado onde um dispositivo, já existente, é atualizado. O estado REACTIVATED é designado para representar o estado de dispositivos que foram desativados e retornaram a funcionar. O estado MALFUNCTION representa o estado de dispositivos defeituosos ou com funcionamento parcial.

Tabela 5.7: Definição dos estados de operação de um recurso

NEW
INACTIVE
UPDATED
REACTIVATED
MALFUNCTION

A transição entre estados está descrita na Figura 5.1. É possível observar que o estado inicial é o NEW. Do estado NEW é possível transitar para os estados UPDATED, INACTIVE ou MALFUNCTION. O estado INACTIVE pode ser transitado apenas para o estado REACTIVATED. O estado REACTIVATED pode ser transitado para os estados UPDATED, INACTIVE ou MALFUNCTION. O estado UPDATED pode permanecer em seu próprio estado ou ser transitado para os estados INACTIVE ou MALFUNCTION. O estado MALFUNCTION pode ser transitado para os estados INACTIVE ou REACTIVATED.

5.1.2 Empacotamento e publicação

O módulo ***Collector*** precisa enviar dados para o framework e, para o envio, ele deve utilizar uma estrutura única para "empacotar" os dados citados anteriormente. Definimos que o pacote enviado seria dividido em três contêineres para separar a informação por contextos. São estes: **network**, **collector** e **sensor**. O contêiner **network** engloba as informações sobre a rede de sensores, comportando o identificador e informações extras. O contêiner **collector** é o representante do módulo ***Collector***. Este contêiner possui informações relativas ao mecanismo de coleta, como o intervalo de captura da rede de sensores a qual está vinculado. O contêiner **sensor** contém informações inerentes ao

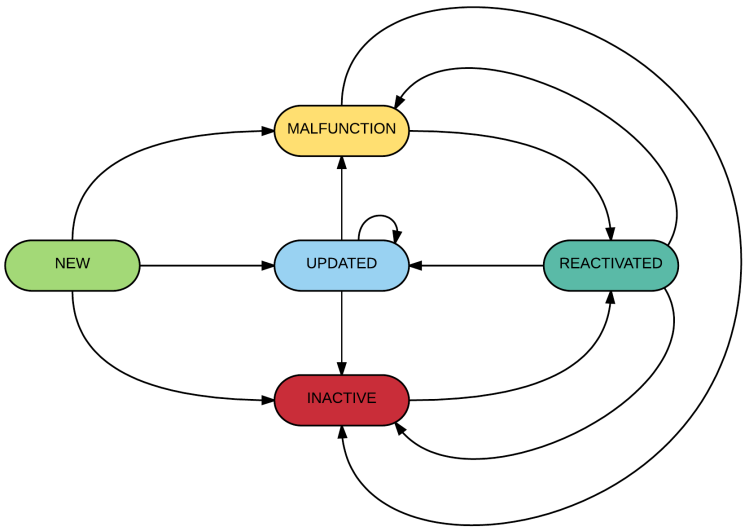

Figura 5.1: Estados e suas transições

dispositivo sensor, englobando os dados sensoriais e o tempo da última captura realizada. A Tabela 5.8 apresenta a organização do pacote completo utilizado pelo ***Collector*** para a submissão de dados amostrais para o OSIRIS.

Contêiner **network** possui os campos `id`, para o identificador da rede, e `info`, para informações diversas atribuídas à rede de sensores. O contêiner **collector** possui os campos `id`, `captureInterval`, `captureIntervaltimeUnit` e `info`, com os respectivos significados de identificador do ponto de coleta, o intervalo de captura de dados da rede de sensores, a unidade de tempo do intervalo de captura e a lista de informações.

O contêiner **sensor** contém os campos `id`, `state`, `captureTimestampInMillis`, `acquisitionTimestampInMillis`, `capturePrecisionInNano`, `value`, `consumables`, `info` e `consumableRules`. O campo `id`, para armazenar o identificador do dispositivo sensor na rede de sensores; o campo `state`, para comportar o estado de funcionamento do dispositivo sensor; o campo `captureTimestampInMillis`, para identificar o momento de captura dos dados pelo dispositivo sensor em milissegundos; o campo `acquisitionTimestampInMillis`, como o instante da aquisição da amostra, oriunda da rede de sensores, pela unidade coletora em milissegundos; o campo `capturePrecisionInNano`, como uma quantidade de tempo em nano segundos para atribuir o instante de captura à precisão de nano segundos; o campo `value`, com um conjunto de tuplas com os valores de sensoriamento. Também há o campo `consumables`, contendo um conjunto de tuplas com os valores dos

Tabela 5.8: Pacote de publicação do Collector

Contêiner	Tipo	Campo	Descrição
network	String	id	Identificador da rede
	String	state	Estado do rede
	Map<String, String>	info	Conjunto de informações da rede
collector	String	id	Identificador do ***Collector***
	String	state	Estado do coletor
	Long	captureInterval	Valor do intervalo de captura
	String	captureIntervalTimeUnit	Unidade de tempo do intervalo de captura
	Map<String, String>	info	Conjunto de informações da rede
sensor	String	id	Identificador do sensor na rede de sensores
	String	state	Estado do dispositivo
	Long	captureTimestampInMillis	Instante da captura em milissegundos
	Long	acquisitionTimestampInMillis	Instante da aquisição da amostra pelo coletor em milissegundos
	Integer	capturePrecisionInNano	Precisão do instante de captura em nano segundo
	List<Value>	values	Lista dos valores da captura
	List<Consumable>	consumables	Lista de valores com os níveis dos recursos fungíveis
	List<ConsumableRule>	consumableRules	Conjunto de regras para os recursos fungíveis
	Map<String, String>	info	Conjunto de informações da rede

níveis dos recursos fungíveis; o campo `consumableRules`, contendo as condicionais de verificação dos recursos fungíveis e, por fim, o campo `info`, com o conjunto de informações adicionais do dispositivo sensor. Uma vez completado o pacote para o envio ao sistema de monitoramento, para a publicação, o módulo coletor deve publicá-lo em `omcp://collector.messagegroup.osiris/`, classificando o conteúdo de modo mais geral para o mais específico, seguindo a ordem das palavras-chave com as respectivas ids de rede, do coletor e do sensor. A seguir um exemplo com o endereço de publicação `omcp://collector.messagegroup.osiris/rede1/coletor1/sensor1/`.

5.1.3 Rede de sensores multicoleta

Para realizar a coleta de dados da rede de sensores é possível utilizar uma abordagem com múltiplos coletores conectados para uma mesma rede de sensores. A Figura 5.2 demonstra um exemplo de rede multicoletada, com dois coletores plugados à rede de sensores. Na imagem, C1 representa o ***Colletor*** 1, C2 representa o ***Colletor*** 2 e o círculo central representa a rede de sensores.

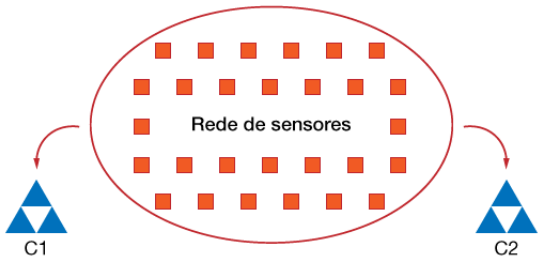

Figura 5.2: Exemplo de rede multicoletada, com 2 coletores

A disposição dos coletores em uma rede multicoletada pode ser dada de duas formas: coletores igualitários, com o mesmo identificador, e coledores distintos, com identificadores diferentes. Quando os coletores são dispostos com o mesmo identificador é gerada uma submissão de dados duplicada, com uma dupla submissão de dados com os mesmos identificadores de rede, coletor e sensor e tempo. Em ocasiões de coletores anexados com identificadores distintos, pode ocorrer uma duplicação dos valores sensoriais com uma dupla submissão de dados com os mesmos identificadores de rede, sensor e tempo (os ids dos coletores são diferentes). Essas abordagens podem ser utilizadas para obter redundância na submissão de dados para o sistema de monitoramento e/ou na rede de sensores.

5.2 SensorNet

O ***SensorNet*** é o módulo responsável por monitorar a rede de sensores e emitir notificações quando alterações ocorrem na rede. O ***SensorNet*** mapeia os dados e metadados da rede de sensores para o ambiente computacional, mantendo uma estrutura de dados interna com representações lógicas de todos os dispositivos em operação na rede de sensores. Como resultado deste mapeamento, o módulo ***SensorNet*** mantém informações sobre os valores de monitoramento, níveis dos recursos fungíveis, estado de operação e dados adicionais e dados extras sobre hardware e software utilizados na rede de sensores.

5.2.1 Recursos do SensorNet

O ***SensorNet*** trabalha com os dados recebidos de todos os módulos ***Collector*** em funcionamento no sistema de monitoramento. Para receber continuamente estes dados, o módulo mantém-se vinculado ao grupo de mensagens `omcp://collector.messagegroup.osiris/`, grupo de mensagens ao qual é utilizado pelo módulo ***Collector*** para a publicação de mensagens relacionadas ao sensoriamento. Obtidas as mensagens pelo ***SensorNet***, as amostras sensoriais e os metadados são utilizados na construção de uma estrutura de dados interna, estrutura esta que é utilizada como referência lógica do estado dos dispositivos existentes na rede de sensores. Para a construção desta estrutura são utilizados três tipos recursos: o Network, recurso para representar a rede de sensores; o Collector, para representar os módulos ***Collector*** em operação no sistema de monitoramento; e o Sensor, representando cada unidade sensora(ou dispositivo) existente na rede de sensores.

A relação entre os recursos segue a hierarquia utilizada pelo módulo ***Collector*** na organização dos dados. O recurso Network é mais geral. O recurso Collector é o intermediário. O recurso Sensor é a representação mais específica. O recurso Network contém diversos recursos Collector. O recurso Collector contém diversos recursos Sensor. Esta estrutura possibilita monitorar diversas redes de sensores, diversos coletores por rede e inúmeros dispositivos sensores.

5.2.1.1 Network

Quando uma nova rede de sensores inicia a emissão de dados sensoriais, a mesma é mapeada para o recurso Network. O recurso Network tem a responsabilidade de englobar recursos, Collectors e Sensors, e manter informações inerentes à rede de sensores como um todo. A Tabela 5.9 descreve os campos do recurso Network. O campo `id` é desti-

nado ao identificador da rede sensora, valor obtido dos metadados enviados pelo módulo ***Collector***. O campo `state` informa estado de operação do recurso. O valor atribuído ao campo `state` advém de uma verificação interna definida pela implementação do módulo ***SensorNet***. Os valores para o estado de operação são os descritos na Seção 5.1.1.7. O campo `lastModified` armazena a data da última modificação do recurso, seja essa modificação ocasionada pela atualização do recurso com novos dados recebidos do ***Collector***, seja por uma mudança de estado de operação. O campo `totalCollectors` informa o número total de coletores em uma rede. Já o campo `totalSensors` informa o número total de sensores da rede. O campo `info` mantém informações extras da rede de sensores.

Tabela 5.9: Campos do recurso Network

Tipo	Campo	Descrição
`String`	`id`	Identificador da rede de sensores
`String`	`state`	Estado de operação da rede
`Long`	`lastModified`	Data da última atualização do recurso
`Integer`	`totalCollectors`	Total de coletores na rede
`Integer`	`totalSensors`	Total de sensores na rede
`Map<String, String>`	`info`	Informações extras sobre a rede

5.2.1.2 Collector

Em uma rede de sensores podem existir um ou mais pontos de coleta de dados, os módulos ***Collector***. Cada ponto de coleta é mapeado para o recurso Collector. O recurso Collector tem o propósito de manter informações relacionadas ao módulo ***Collector*** e englobar os diversos recursos Sensor. A Tabela 5.10 contém informações sobre os campos do recurso Collector. O campo `id` contém o identificador, valor este relacionado ao nome dado para cada módulo Collector. O campo `state` representa o estado de operação do recurso Collector, com o mesmo propósito do campo `state` do recurso Network. O campo `lastModified` guarda o momento da última modificação do recurso, sendo esta atualização por novos valores de metadados ou troca de estado de operação. O campo `networkId` contém o identificador do recurso Network. O campo `totalSensors` informa o número total de recursos Sensor englobados pelo recurso Collector. O campo `info` é destinado armazenar informações complementares definidas no módulo ***Collector***.

5.2.1.3 Sensor

Os dispositivos da rede de sensores são representados pelo do recurso Sensor no módulo ***SensorNet***. O recurso Sensor é o mais complexo dos três, com dados sensoriais, informa-

Tabela 5.10: Campos do recurso Collector

Tipo	Campo	Descrição
`String`	`id`	Identificador do módulo ***Collector***
`String`	`state`	Estado de operação do coletor
`Long`	`lastModified`	Data da última atualização do recurso
`String`	`networkId`	Identificador da rede de sensores
`Integer`	`totalSensors`	Total de sensores do coletor
`Map<String, String>`	`info`	Informações extras sobre o coletor

ções sobre os níveis dos recursos fungíveis e as informações extras atribuídas ao dispositivo. A Tabela 5.11 descreve todos os campos para o recurso. O campo `id` recebe o identificador do dispositivo sensor, o mesmo utilizado na rede de sensores. O campo `state` é a representação do estado de operação, como explicado anteriormente no recurso Network. O campo `lastModified` armazena a data da última atualização do recurso Sensor, seja essa mudança uma alteração de estado de operação, seja uma atualização nos valores de sensoriamento ou recursos fungíveis. O campo `networkId` é destinado para o identificador do recurso Network. O campo `collectorId` é o identificador do recurso Collector. O campo `info` é destinado a armazenar as informações complementares do dispositivo sensor, informações estas que são enviadas no pacote oriundo do Collector. O campo `values` é destinado a comportar uma lista de tuplas com os valores dos sensores(componentes eletrônicos) do dispositivo. Um dispositivo pode realizar sensoriamento diverso, como temperatura, luminosidade entre outros. A tupla é referente a um valor de sensoriamento, que está apresentada na Seção 5.1.1.1. O campo `captureTimestampInMillis` é o instante que os valores de sensoriamento foram capturados pelo dispositivo. O campo `capturePrecisionInNano` é o dado de precisão em nano segundos para o tempo de captura. O campo `acquisitionTimestampInMillis` representa o momento em que os dados foram coletados pelo módulo ***Collector***. O campo `storageTimestampInMillis` é o tempo de armazenamento do dado pelo módulo ***SensorNet***. O campo `consumables` comporta uma lista de tuplas com informações sobre os recursos fungíveis do dispositivo, como exemplo de recursos fungíveis é possível citar bateria, combustível entre outros. A tupla referente ao recurso fungível está apresentada na Seção 5.1.1.2.

5.2.2 Funcionamento do módulo

Existem algumas características(requisitos funcionais) de necessária implementação para o módulo ***SensorNet***. Essas características visam estabelecer diretrizes sólidas para diversas implementações atuarem de modo aproximado, sem grandes diferenças em suas

Tabela 5.11: Campos do recurso Sensor

Tipo	Campo	Descrição
`String`	`id`	Identificador do dispositivo sensor na rede de sensores
`String`	`state`	Estado de operação do sensor
`Integer`	`lastModified`	Data da última atualização do recurso Sensor
`String`	`networkId`	Identificador do coletor
`String`	`collectorId`	Identificador do coletor
`Map<String, String>`	`info`	Informações extras sobre o sensor
`List<Value>`	`values`	Valores de sensoriamento
`Long`	`captureTimestampInMillis`	Instante de captura dos valores na rede de sensores
`Integer`	`capturePrecisionInNano`	Precisão em nano segundos da captura dos valores
`Long`	`acquisitionTimestampInMillis`	Instante da aquisição dos valores pelo coletor
`Long`	`storageTimestampInMillis`	Instante do armazenamento do valor pelo sensornet
`List<Comsumable>`	`consumables`	Nível dos recursos fungíveis do dispositivo sensor

operações. Os requisitos para o módulo ***SensorNet*** são: armazenar apenas dados sensoriais mais recentes, inferir o estado de operação dos recursos Network, Collector e Sensor, realizar a difusão de informações quando ocorrerem atualização dos recursos Network, Collector e Sensor e emitir alertas para condições que necessitem de atenção, como o nível dos recursos fungíveis, as alterações no estado de operação dos recursos e novos recursos encontrados.

5.2.2.1 Armazenamento de dados sensoriais mais recentes

O módulo ***SensorNet*** deve estar restrito a manter as amostras mais recentemente capturadas pelos dispositivos sensoriais. Situações onde os dados são iguais ou mais antigos devem ser tratadas de forma criteriosa para não ocorrer uma atribuição de valores desatualizados ao recurso Sensor. Um recurso Sensor existente possui informações sobre o tempo de captura sensorial, o campo `captureTimestampInMillis` em conjunto com o campo `capturePrecisionInNano`, e os valores de sensoriamento. Os campos de tempo podem ser utilizados como referências para definir se uma informação é a mais recente, é igual ou é uma nova amostra, oriunda de uma captura mais recente do dispositivo da rede de sensores. É importante salientar que os dados enviados pelo módulo ***Collector*** possuem também os campos `captureTimestampInMillis` e `capturePrecisionInNano`.

5.2.2.2 Inferência do estado de operação de um recurso

É sabido, de fato, que um dispositivo físico pode variar seu estado de operação de pleno funcionamento, passando, em alguns casos, por um estado de funcionalidade parcial, até se tornar inapto para cumprir as atribuições sensoriais. Para descrever tais estados definimos, na Seção 5.1.1.7, os estados de operação: NEW, INACTIVE, UPDATED, REACTIVATED e MALFUNCTION. Como o módulo ***SensorNet*** deve manter um diagnóstico correto sobre a rede de sensores, ele deve ser capaz de inferir alguns estados de operação independentemente da perda de comunicação entre as partes. No caso do recurso Sensor, o recebimento de amostras pode ser interrompido. Já um módulo ***Collector*** está suscetível a interrupções na comunicação com o OSIRIS. Uma rede de sensores pode entrar em pane, sem enviar qualquer amostra sensorial ao ***SensorNet***. Para tais situações, a inferência deve ser aplicada a fim de refletir, aos recursos internos do módulo via estados de operação, a anormalidade funcionamento da rede de sensores e, assim, informar corretamente para interessados externos que algo no sistema de monitoramento não está funcionando corretamente.

5.2.2.3 Difusão de atualização dos recursos

O módulo ***SensorNet*** deve trabalhar sobre o paradigma de menor esforço dos interessados externos em obter atualizações dos recursos Network, Collector e Sensor. Para atingir tal característica, na ocorrência de uma atualização sobre um recurso, o módulo deve publicar uma representação atualizada, contendo todos os dados do recurso para o grupo de mensagens `omcp://update.messagegroup.osiris/`. Cada interessado inscrito no grupo de mensagens receberá uma mensagem com todos os dados atualizados do recurso, sem precisar requisitar tais informações ao módulo. A seguir, são listados os endereços para a publicação das representações dos recursos do módulo ***SensorNet***:

- Network: `omcp://update.messagegroup.osiris/sensornet/{Network id}/`
- Collector: `omcp://update.messagegroup.osiris/sensornet/{Network id}/collector/{Collector id}/`
- Sensor: `omcp://update.messagegroup.osiris/sensornet/{Network id}/collector/{Collector id}/sensor/{Sensor id}/`

5.2.2.4 Emissão de alertas de notificação

O ***SensorNet*** emite alertas para notificar alterações relevantes em sua estrutura interna. Os alertas podem ocorrer por conta de mudança de estados de operação dos recursos(Network, Collector ou Sensor) ou por conta dos níveis dos recursos fungíveis. Todos os alertas de notificação devem ser emitidos seguindo o padrão definido pelo OSIRIS, o qual é apresentado a seguir.

Suporte para notificações do OSIRIS Para realizar notificações no OSIRIS, definimos uma mensagem de notificação e um grupo de mensagens para realizar a publicação da mensagem. O grupo de mensagens dedicado para notificações é o `omcp://notification.messagegroup.osiris/{criticidade}/`. Já a mensagem de notificação deve conter o nível de criticidade e informações inerentes ao recurso que emitiu a notificação.

As notificações são classificadas em seis níveis, descritos na Tabela 5.12: VERBOSE para mensagens com informações extras e de pouca importância; DEBUG para informações de debug; INFO para informações relevantes de qualquer natureza; WARNING para

os avisos; CRITICAL para informações críticas; ERROR para informações relacionadas a erros ocorridos nos módulos.

Tabela 5.12: Definição dos níveis de criticidade para as notificações

Campo	Descrição
VERBOSE	Notificações de *log* verboso
DEBUG	Notificações de *debug*
INFO	Notificações de informações
WARNING	Notificações de aviso
CRITICAL	Notificações de ocorrências críticas
ERROR	Notificações de erro

A estrutura da mensagem de notificação está apresentada na Tabela 5.13. O campo `uri` contém o endereço do recurso notificado. O campo `origin` é o campo dedicado para o nome do módulo notificante. O campo `title` contém um título para a mensagem notificação. O campo `message` contém uma mensagem de notificação. O campo `level` determina o nível de criticidade da notificação.

Tabela 5.13: Campos da mensagem de notificação

Tipo	Campo	Descrição
`String`	`level`	Nível do alerta
`String`	`uri`	Endereçamento do recurso
`String`	`title`	Título para o alerta
`String`	`message`	Mensagem para o alerta
`String`	`origin`	Módulo de origem do alerta

Alertas de troca de estado de operação Alertas são emitidos utilizando a estrutura de notificação do OSIRIS. Estes alertas informam aos interessados externos sobre alterações no estado de operação dos recursos(Network, Collector e Sensor). Os estados de operação NEW, INACTIVE, REACTIVATED e MALFUNCTION são considerados anormais, pois representam mudança de significativa atenção do operador do sistema de monitoramento. Os estados de operação NEW e REACIVATED são notificados com o nível de criticidade INFO e os estados INACTIVE e MALFUNCTION são notificados com o nível CRITICAL. A criticidade INFO informa uma alteração de impacto positivo ou nulo para o sistema. Já o nível CRITICAL exige atuação imediata do operador do sistema de monitoramento, pois informa situações de anormalidade que podem interromper a atividade de monitoramento.

Alertas para recursos fungíveis O módulo ***Collector*** pode enviar os níveis dos recursos fungíveis de um dispositivo sensor em conjunto com expressões condicionais para realizar verificações sobre estes. As expressões condicionais são compostas de valores limites, sendo capazes de informar se um recurso fungível está ou não com níveis aceitáveis. O módulo ***SensorNet*** deve utilizar as expressões condicionais para verificar continuamente os níveis dos recursos fungíveis e emitir alertas de nível CRITICAL em caso de valores inaceitáveis. Alguns exemplos de notificação são a carga de uma bateria chegando ao fim e quantidade de combustível na reserva.

5.2.3 Interação com o módulo

O ***SensorNet*** deve ser capaz de atender requisições e respondê-las. Para isto, ele deve disponibilizar recursos e operações OMCP para interagir com o mundo exterior. Seu endereço OMCP é o `omcp://sensornet.osiris/`. O módulo deve disponibilizar operações de consultas e remoção para os seus recursos Network, Collector e Sensor.

Na Tabela 5.14 mostramos a listagem completa de recursos e operações OMCP. O módulo ***SensorNet*** deve estar habilitado para receber consultas que retornam uma coleção ou apenas uma única representação do recurso, quando passado o identificador. Esta ação é realizada pela operação GET. Para a remoção de recursos, a ação se dá através do método DELETE, definindo qual recurso será excluído pelo identificador(`id`). É importante lembrar que o recurso Network possui Collectors e Sensors, por conta disso, remover um recurso Network impacta em remover todos os demais recursos agregados. A remoção do recurso Collector também remove os recursos Sensor agregados.

Definimos ainda que a capacidade de manter o histórico dos dados de sensoriamento e dos níveis de recursos fungíveis é uma característica relevante, pois isto abre espaço para estudar o comportamento dos dispositivos sensores no que se diz respeito ao consumo dos recursos fungíveis em diferentes cenários de utilização. Em nossa definição, cada dispositivo sensor deve possuir seu histórico de valores, sendo o histórico formado por um conjunto de itens chamados revisão(*revision*). Cada revisão armazena os valores(sensoriamento e os níveis dos recursos fungíveis) de um momento no tempo. Para requisitar as revisões de um dispositivo é necessário realizar uma requisição de operação GET a um determinado recurso Sensor, seguido da palavra "revision".

As revisões são armazenadas com critério temporal e para a requisição é possível utilizar o tempo como meio de obter um determinado conjunto de revisões. Para isto, foram definidos três parâmetros para a consulta de revisões sobre um recurso Sensor,

Tabela 5.14: Lista dos recursos e operações OMCP do módulo SensorNet

Operação	Recursos	Descrição
GET	`/`	Retorna todos os recursos de Network existentes
GET	`/{network id}/`	Retorna um recurso Network de acordo com seu identificador
DELETE	`/{network id}/`	Remove um recurso Network de acordo com o identificador
GET	`/{network id}/collector/`	Retorna todos os recursos de Collector existentes em uma rede
GET	`/{network id}/collector/{collector id}/`	Retorna um recurso Collector de acordo com seu identificador e sua rede
DELETE	`/{network id}/collector/{collector id}/`	Remove um recurso Collector de acordo com o identificador
GET	`/{network id}/collector/{collector id}/sensor/`	Retorna todos os recursos de Sensor existentes em um coletor da rede
GET	`/{network id}/collector/{collector id}/sensor/{sensor id}/`	Retorna um recurso Sensor de acordo com seu identificador, seu coletor e sua rede
DELETE	`/{network id}/collector/{collector id}/sensor/{sensor id}/`	Remove um recurso Sensor de acordo com o identificador
GET	`/{network id}/collector/{collector id}/sensor/{sensor id}/revisions/`	Retorna dados de histórico de um recurso Sensor. Aceita parâmetros
GET	`/{network id}/sensor/`	Retorna todos os recursos de Sensor existentes em uma rede
GET	`/{network id}/sensor/{sensor id}/`	Retorna um recurso Sensor de acordo com seu identificador e sua rede

como descrito na Tabela 5.15. O parâmetro `from` define uma data inicial em que uma revisão foi criada. O parâmetro `to` é utilizado para definir uma data final, ou data limite, para o retorno dos registros. A utilização conjunta dos parâmetros `from` e `to` determina a busca de revisões dentro de um período de tempo. O parâmetro `from`, quando utilizado sozinho, define buscas de revisões compreendendo um período de um ponto no tempo para os mais antigos. O parâmetro `to`, quando utilizado sozinho, define buscas sobre um período de um ponto no tempo até o momento presente. O parâmetro `limit` define a quantidade de revisões retornadas na busca. Caso nenhum parâmetro de tempo(*i.e* `from` ou `to`) seja passado, os resultados incluirão as revisões das mais recentes para as mais antigas. Os valores passados nos parâmetros `from` e `to` são informações de data e hora e o valor do parâmetro `limit` é um número inteiro. Para exemplificar, uma consulta para `omcp://sensornet.osiris/{Network id}/collector/{Collector id}/sensor/{Sensor id}/revisions?from=06/25/2015&to=06/26/2015&limit=10` retorna as revisões criadas de 00:00 de 25/06/2015 até 23:59 99 de 26/06/2015, com um limite de 10 registros no total. Os formatos do dado de tempo e hora passado nos parâmetros `from` e `to` devem ser definidos pela implementação do módulo ***SensorNet***.

Tabela 5.15: Definição de Query Strings para a requisição das revisões

Parâmetro	Descrição
`from`	Parâmetro para definir uma data inicial
`to`	Parâmetro para definir uma data final
`limit`	Parâmetro para definir o total de revisões retornadas

5.3 Function

Os módulos ***Function*** são recursos independentes para processar dados sensoriais. Cada módulo ***Function*** é especializado em algum tipo de processamento, trabalhando sobre um conjunto de dados específico para a geração de resultados. A especialidade do módulo ***Function*** é determinada pela implementação, a qual deve atender aos requisitos do sistema de monitoramento. A arquitetura de base prevê a utilização de diversos módulos ***Function***, ampliando a capacidade de processamento do sistema de monitoramento.

5.3.1 Recursos do módulo Function

Definimos para o módulo ***Function*** alguns recursos para que ele possa operar e interagir com outros componentes. Os recursos são: uma representação da interface do módulo,

um objeto de requisição e um objeto de resposta.

5.3.1.1 Interface

Para atender aos pedidos de processamento de dados, um conjunto de valores deve ser passado para o módulo. Devido a característica do módulo ***Function*** ser um processador de dados especializado, cada módulo tem seu próprio conjunto de dados de entrada e saída. Para informar o tipo do dado do parâmetro é utilizado uma tupla, apresentada na Tabela 5.16, contendo um nome, para identificar o parâmetro, o tipo de valor(*e.g* `number`, `text` ou `logic`), a unidade(*e.g* Celsius, volt, ampere, entre outros) e um sinalizador de coleção(*i.e* similar a um vetor) ou valor único.

Tabela 5.16: Tupla para determinar o tipo de parâmetro, *Param*

Tipo	Campo	Descrição
`String`	`name`	Nome do parâmetro
`String`	`type`	Tipo do dado
`String`	`unit`	Unidade do dado
`Boolean`	`isCollection`	*Flag* para informar se o parâmetro é um vetor ou não

A Tabela 5.17 apresenta a estrutura completa utilizada pelo módulo ***Function*** para informar aos interessados externos o seu conjunto de dados de entrada e saída, o recurso Interface. A estrutura contém o nome do módulo ou funcionalidade, a descrição da funcionalidade, o endereço OMCP do módulo, uma listagem com os modos de operação para requisição suportados, uma lista para os parâmetros de requisição e outra lista para informar os parâmetros de resposta. Os modos de requisição, Tabela 5.18, suportados podem ser de dois tipos: requisição síncrona, quando é realizada uma operação de imediata e bloqueante, e requisição assíncrona, onde é submetida uma requisição de processamento para ser executada em algum momento futuro pelo módulo ***Function***.

Tabela 5.17: Campos do recurso Interface

Tipo	Campo	Descrição
`String`	`name`	Nome do módulo relacionando ao processamento
`String`	`description`	Descrição do módulo em relação ao processamento
`String`	`address`	Endereço do módulo ***Function***
`List<String>`	`operations`	Lista de operações para requisição `sync` ou `async`
`List<Param>`	`requestParams`	Parâmetros de requisição
`List<Param>`	`responseParams`	Parâmetros de resposta

Tabela 5.18: Definição para os modos de operação do módulo Function

Valor	Descrição
`sync`	Identificação para requisição de execução imediata
`async`	Identificação para requisição com agendamento de execução

5.3.1.2 RequestObject

O recurso RequestObject é o pacote constituído de um ou mais valores a serem transmitidos e um endereço de resposta. Cada valor do recurso RequestObject é estruturado em forma de tupla, como descrito na Tabela 5.19, contendo o nome do parâmetro e o valor. O campo `name` contém o identificador do parâmetro e o campo `value` armazena o dado referente ao valor do parâmetro. Se o valor for uma coleção, é repassada uma lista de valores ao invés de um único valor. A referência de quais tipos de dados devem ser enviados ao módulo ***Function*** está no recurso Interface, nos parâmetros de requisição, como descrito anteriormente.

Tabela 5.19: Tupla de valor parametrizado, *Value*

Tipo	Campo	Descrição
`String`	`name`	Nome do argumento
`String/List<String>`	`value`	Valor sensorial, único ou vetor

A Tabela 5.20 apresenta os campos do recurso RequestObject. O campo `responseTo` armazena um endereço OMCP para a resposta do processamento e o campo `values` contém um conjunto de tuplas com os valores para processamento.

Tabela 5.20: Campos do recurso RequestObject

Tipo	Campo	Descrição
`String`	`responseTo`	Endereço OMCP para a resposta
`List<Value>`	`values`	Lista de valores sensoriais

O pacote é enviado por completo em requisições assíncronas. Porém, quando a requisição é síncrona, os valores devem ser transformados em *Query Strings* e repassados no endereço OMCP, sendo cada parâmetro convertido para um par de chave-valor, como será explicado na Seção 5.3.3.

5.3.1.3 ResponseObject

O recurso ResponseObject é o pacote que armazena o resultado do processamento para o envio ao requisitante, seja a requisição síncrona ou assíncrona. O recurso deve estar

em consonância com os parâmetros de resposta informados pelo recurso Interface. A Tabela 5.21 demonstra os dados contidos no recurso ResponseObject.

Tabela 5.21: Campos do recurso ResponseObject

Tipo	Campo	Descrição
`Long`	`requestTimestampInMillis`	Tempo do instante da requisição
`Long`	`responseTimestampInMillis`	Instante do término do processamento
`List<Value>`	`values`	Lista de valores com resultados

O ResponseObject inclui o tempo do instante da requisição e o tempo do instante quando o processamento dos dados foi finalizado. O conjunto dos valores de resposta é uma lista contendo uma ou mais tuplas, como apresentada na Tabela 5.19, com os valores resultantes do processamento.

5.3.2 Funcionamento do módulo

A especificação do módulo ***Function*** define os requisitos para serem implementados na construção de cada novo módulo. Os requisitos são: atender requisições síncronas e assíncronas. A requisição síncrona deve ser utilizada para operações simples, sem grande custo computacional para o processamento. Já a requisição assíncrona foi pensada como uma alternativa para tratar com processamentos mais complexos, realizados por um espaço de tempo consideravelmente maior que a requisição síncrona. Nós não definimos uma referência de tempo ideal para a requisição síncrona, ficando por conta da implementação a escolha do valor mais adequado para sua necessidade.

As requisições síncronas devem ser realizadas utilizando a operação GET do OMCP, com a passagem de parâmetros como argumentos no endereço requisitado, em pares de chave-valor. Por exemplo, o endereço `omcp://sum.function.osiris/?arg1=1&arg2=[1,2,3,4]` contém como argumentos o `arg1` é o `arg2`. O `arg1` é o primeiro argumento, de valor único, e o `arg2` é o segundo argumento, o qual contém um vetor. Uma requisição síncrona deve ainda ser uma operação bloqueante, recebendo o cliente uma representação do recurso ResponseObject como resposta de requisição bem sucedida.

As requisições assíncronas ocorrem por meio da operação POST do OMCP. Os dados da requisição são constituídos por uma representação do recurso RequestObject(Tabela 5.19), a qual deve ser enviada no corpo da mensagem OMCP. No momento que o módulo recebe a requisição, retorna a resposta com o código CREATED, informando o endereço do item em seu armazenamento interno, por exemplo: `omcp://sum.function.osiris/spool/5432`. Devido à característica de requisição assíncrona ser indeterminada em relação ao tempo

de geração do resultado, o módulo deve prover recursos para armazenar a requisição até que o processamento se realize. Após o fim do processamento, o módulo ***Function*** envia resultado ao requisitante utilizando o endereço(`responseTo`) do recurso RequestObject.

5.3.3 Interação com o módulo Function

Para que um módulo ***Function*** seja encontrável dentro da estrutura do OSIRIS, ele deve definir um endereçamento OMCP como, por exemplo, `omcp://myfunction.function.osiris/`. Além do endereçamento, é necessário disponibilizar os seguintes recursos OMCP descritos na Tabela 5.22. O recurso raiz(/) deve ser utilizado para as requisições síncronas e assíncronas. O recurso `/interface/` deve ser disponibilizado para a obtenção da interface de operação do módulo, como descrito na Tabela 5.20. O recurso `/spool/` deve ser utilizado quando existir a necessidade de disponibilizar informações sobre o armazenamento interno. E o `/spool/{item id}` pode ser utilizado para remover um item(*i.e* a requisição) do armazenamento interno.

Tabela 5.22: Lista dos recursos e operações OMCP do módulo Function

Operação	Recursos	Descrição
GET	/	Recurso raiz para requisição síncrona
POST	/	Recurso raiz para requisição assíncrona
GET	`/interface/`	Endereço para interface do módulo ***Function***
GET	`/spool/`	Retorna informações sobre a fila de processamento
DELETE	`/spool/{item id}`	Remove um item do armazenamento interno

5.4 VirtualSensorNet

O módulo ***VirtualSensorNet*** é o responsável por estabelecer uma camada de abstração entre os interessados externos(*i.e* consumidores de dados de sensoriamento) e os dispositivos presentes na rede de sensores. Esta camada de abstração desacopla os dados de suas fontes geradoras, possibilitando o foco do processo de monitoramento apenas nos dados de sensoriamento, sem se importar com o tipo de hardware utilizado. O ***VirtualSensorNet*** possui ainda a capacidade de realizar processamentos sobre um conjunto de dados sensoriais, contidos no próprio módulo, quando auxiliado por módulos ***Function***. Excluindo as especialidades, o módulo ***VirtualSensorNet*** trabalha de forma similar ao módulo ***SensorNet***, pois ele emite alertas, difunde informações sobre as atualizações de recursos e deve conter recursos e operações OMCP para a interação com o mundo externo.

5.4.1 Camada de abstração

A camada de abstração atua selecionando os dados de sensoriamento em detrimento de outras informações específicas do dispositivo, informações estas que não influenciam o resultado final. Em geral, a abordagem consiste em separar os dados das respectivas fontes geradoras, sendo a fonte geradora um dispositivo físico(*i.e* dispositivo sensor) ou um software dedicado em gerar dados. Escolhemos esta abordagem para atingir um baixo impacto na substituição dos dispositivos da rede de sensores sobre a disponibilização de dados sensoriais para interessados externos. Pretendemos que o máximo esforço gerado por uma substituição seja uma troca de identificador(*i.e* o qual identifica a fonte geradora) e uma conversão de base do dado sensorial.

5.4.1.1 Processo de abstração de dados

Citar abstração de dados[referência]. Processo da abstração possibilita processar uma informação de modo a remover detalhes e manter apenas as características relevantes para o processo que irá utilizá-la. Optamos por manter os dados sensoriais em um formato padronizado, sem detalhes de fonte geradora, possibilitando qualquer consumidor trabalhar com um tipo único de dado de sensoriamento. A seguir, apresentamos a visão da abstração de dados proposta.

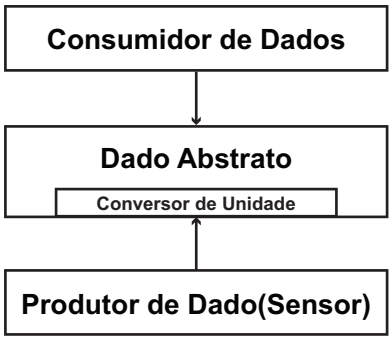

Figura 5.3: Níveis do processo de abstração de dados

Na Figura 5.3 são apresentados os três níveis do processo de abstração de dados. O nível mais superior representa o consumidor de dados e está vinculado ao nível do meio. O nível do meio é a representação do dado abstrato, sem detalhes inerentes à fonte geradora. Por sua vez, o nível do meio está diretamente vinculado ao produtor de dados(*i.e* a fonte geradora), o nível mais baixo, o qual pode ser um dispositivo físico existente em uma rede sensora ou um processamento de dados. É possível observar que o nível do meio conecta o produtor ao consumidor de dados. Ainda no nível do meio, incluímos um conversor de

base para atuar no momento da substituição do produtor de dados, pois, em determinada situação, pode ocorrer a troca de produtores que trabalham com valores similares mas em bases diferentes. Como exemplo, dois dispositivos sensores de temperatura, emitindo valores em Fahrenheit e outro em Celsius. Para a substituição de um pelo outro, uma conversão de base se faz necessária, a fim manter os dados consistentes no nível do meio. Por outro lado, o consumidor de dados será inconsciente da substituição dos produtores de dados, dispensando manutenção em suas rotinas internas por conta da mudança. Em nossa abordagem, o nível do meio é chamado de sensor virtual.

5.4.1.2 O sensor virtual

O sensor virtual é uma unidade sensora que comporta dados sensoriais abstraídos de informações inerentes à fonte geradora. Esta unidade recebe um fluxo de entrada de dados de uma fonte geradora e armazena apenas os dados sensoriais. Sem a preocupação com detalhes específicos das diferentes fontes geradoras utilizadas no processo de monitoramento, é possível criar sistemas de monitoramento que sofram impacto reduzido com mudanças em seus dispositivos sensores. A seguir um exemplo da abordagem do sensor virtual.

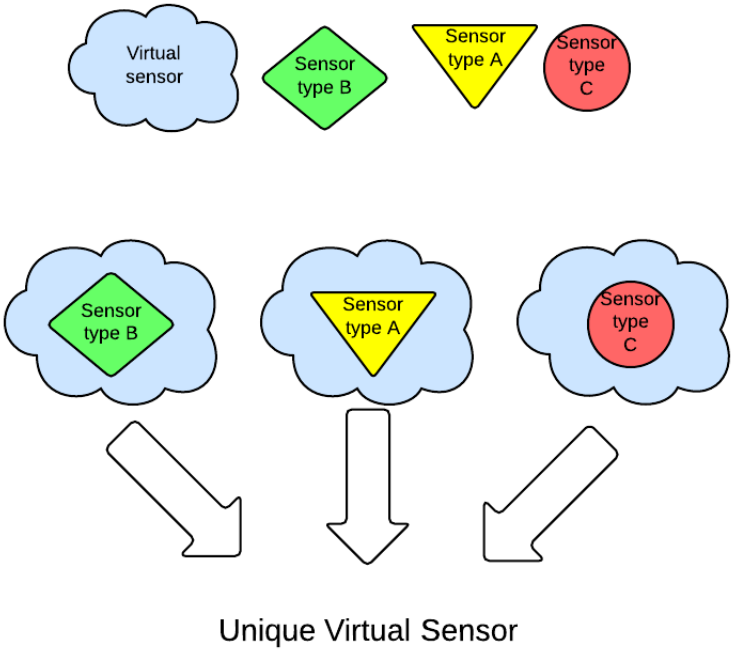

Figura 5.4: Ilustração da abordagem do sensor virtual

A Figura 5.4 ilustra a abordagem do sensor virtual, sendo este representado em forma de nuvem. O sensor virtual pode englobar diversos outros dispositivos sensoriais, resultando em uma unidade sensorial final de natureza única. Ele ainda possibilita a troca de fontes geradoras sem problemas de dados inconsistentes.

5.4.2 Recursos do módulo VirtualSensorNet

Igualmente ao módulo ***SensorNet***, o módulo ***VirtualSensorNet*** trabalha disponibilizando recursos computacionais e emitindo alertas em caso de alterações nos sensores virtuais. As alterações dos sensores virtuais podem ser mudanças de estados de operação ou atualizações dos dados de sensoriamento. O módulo ***VirtualSensorNet*** também recebe dados oriundos dos módulos ***Collector*** existentes, pela inscrição no grupo de mensagens `omcp://collector.messagegroup.osiris/`. O principal recurso do ***VirtualSensorNet*** é o VSensor, relacionado a abordagem do sensor virtual. Os demais recursos do módulo são: o DataType para o tipo de dados, o Converter para a conversão de base e o Function como recurso destinado a criar uma representação interna do módulo ***Function***.

5.4.2.1 DataType

O recurso DataType define o tipo dos dados sensoriais. Este recurso foi definido para auxiliar nas operações de comparação de valores sensores, por exemplo, a conversão de bases. A Tabela 5.23 descreve os campos do recurso DataType. O campo `id` contém o identificador único do recurso. O campo `displayName` armazena o nome de exibição do recurso. O campo `type` define o tipo de dado declarado(*e.g* `number`, `logic` ou `text`), conforme definido na Tabela 5.2. O campo `unit` é a unidade do dado declarado(*e.g* Celsius, volt, ampere). O campo `symbol` contém o símbolo relacionado a unidade do dado declarado(*e.g* Hz, V, A). O campo `usedBy` informa a quantidade de uso do DataType por outros recursos.

Tabela 5.23: Campos do recurso DataType

Tipo	Campo	Descrição
`Long`	`id`	Identificador do recurso DataType
`String`	`displayName`	Nome de exibição
`String`	`type`	Tipo do dado
`String`	`unit`	Unidade do dado
`String`	`symbol`	Símbolo do dado
`Long`	`usedBy`	Quantos outros recursos estão utilizando o DataType

5.4.2.2 Converter

O recurso Converter é o mecanismo utilizado para realizar a conversão de base, como explicado na Seção 5.4.1.1. Este recurso possui duas referências internas de tipos de dados(*i.e* dois recursos DataType) e uma expressão de conversão de base. Os dois tipos de

dados informam o tipo de dado de entrada e o tipo de dado de saída, gerado pela conversão. Já expressão de conversão é uma equação que opera sobre o valor de entrada, com base original(*i.e* o tipo original), para resultar em um valor de saída, com base modificada. A Tabela 5.24 descreve os campos do recurso Converter. O campo `id` contém o identificador único do recurso; o campo `displayName` armazena o nome de exibição do recurso; `expression` contém a expressão utilizada para a conversão de base; `inputDataTypeId` é reservado para o identificador do recurso DataType de entrada; `outputDataTypeId` é reservado para o identificador do recurso DataType de saída; por fim, o campo `usedBy`, que informa a quantidade de uso do Converter por outros recursos.

Tabela 5.24: Campos do recurso Converter

Tipo	Campo	Descrição
`Long`	`id`	Identificador do recurso Converter
`String`	`displayName`	Nome de exibição
`String`	`expression`	Expressão de conversão, em Javascript
`Long`	`inputDataTypeId`	Identificador do recurso DataType do dado de entrada
`Long`	`outputDataTypeId`	Identificador do recurso DataType do dado de saída
`Long`	`usedBy`	Quantos outros recursos estão utilizando o Converter

Definimos que a expressão utilizada no recurso Converter deve ser escrita em linguagem Javascript. A expressão de conversão deve ser implementada para operar sobre uma variável denominada `value`. A variável `value` contém o valor de entrada e deve receber o valor de saída. A seguir, apresentamos o Código 5.1 , com um exemplo de expressão para converter a temperatura em Celsius para Fahrenheit. É possível notar que variável `value` está sendo operada e, em seguida, recebe o resultado desta operação como novo valor.

Código 5.1: Exemplo de código para expressão de conversão

```
value = value * 9/5 + 32;
```

5.4.2.3 Function

O recurso Function é uma abstração interna do módulo ***VirtualSensorNet*** para representar o módulo ***Function***, que atua criando um objeto representativo para o recurso Interface, apresentado na Seção 5.3.1.1 do módulo ***Function***. O recurso ***Function*** auxilia na ação de processamento de dados, pois ele armazena as informações inerentes ao recurso Interface do módulo ***Function***. Essas informações possibilitam determinar corretamente os parâmetros para envio e os parâmetros para receber o resultado do processamento, bem como informar o modo de operação do módulo ***Function***(*i.e* síncrono

ou assíncrono).

A Tabela 5.25 descreve os campos do recurso Function. É possível reparar que o recurso Function contém os mesmos dados do recurso Interface, Tabela 5.17, com o acréscimo do campo `id`. O campo `id` contém o identificador do recurso Function.

Tabela 5.25: Campos do recurso Function

Tipo	Campo	Descrição
`Long`	`id`	Identificador do recurso Function
`String`	`name`	Nome do módulo relacionando ao processamento
`String`	`description`	Descrição do módulo em relação ao processamento
`String`	`address`	Endereço do módulo ***Function***
`List<String>`	`operations`	Lista de operações para requisição `sync` ou `async`
`List<Param>`	`requestParams`	Parâmetros de requisição
`List<Param>`	`responseParams`	Parâmetros de resposta

5.4.2.4 VSensor

Aqui, apresentamos as definições para a implementação da abordagem dos sensores virtuais, via recurso VSensor. O recurso VSensor disponibiliza os dados de sensoriamento em forma de tupla(*i.e* apelidada de Value), com formato similar a tupla de sensoriamento apresentada na Seção 5.1.1.1 relacionada ao módulo Collector. A tupla de dados sensoriais se encontra apresentada na Tabela 5.26. A diferença entre a tupla referente ao módulo ***Collector***(Tabela 5.1) e esta, referente ao recurso VSensor, é o campo `id`.

Tabela 5.26: Tupla para dados de sensoriamento do recurso VSensor, *Value*

Tipo	Campo	Descrição
`Long`	`id`	Identificador do valor
`String`	`name`	Nome do tipo de sensoriamento
`String`	`type`	Tipo do dado do campo valor
`String`	`value`	Valor
`String`	`unit`	Unidade do valor
`String`	`symbol`	Símbolo da unidade do valor

A Tabela 5.27 descreve todos os campos do recurso VSensor. O campo `id` contém o identificador único do recurso VSensor; o campo `label` é reservado para a adição de um rótulo informativo ao recurso VSensor; o campo `state` contém o estado de operação do recurso (*e.g* NEW, INACTIVE, UPDATED, REACTIVATED e MALFUNCTION), como explicado na Seção 5.1.1.7; o campo `creationTimestampInMillis` representa o instante de criação ou captura do dado sensorial; o campo `creationPrecisionInNano` é

o complemento para atribuir ao campo `creationTimestampInMillis` a precisão de nano segundos; o campo `acquisitionTimestampInMillis` é o instante de aquisição do dado sensorial pelo módulo ***Collector*** ou instante do término da requisição para um módulo Function; o campo `storageTimestampInMillis` contém o instante de armazenamento do dado sensorial pelo ***VirtualSensorNet***. Também há o campo `creationInterval`, para o intervalo de captura sensorial de um sensor da rede de sensores ou o intervalo geração de dados sensoriais pelo módulo Function; o campo `creationIntervalTimeUnit`, para a unidade de tempo do campo `creationInterval`; o campo `lastModified`, para a data da última atualização do recurso VSensor; o campo `values`, para uma lista de tupla para dados de sensoriamento, como descrito na Tabela 5.26; por fim, o campo `sensorType`, para o tipo de especialização do VSensor, sendo para esta versão os valores possíveis: LINK, COMPOSITE ou BLENDING.

Definimos que o VSensor é um recurso abstrato, de alto nível, devendo ser utilizado apenas para consumo de dados. Por conta dessa característica, conceitualizamos três especializações operacionais do recurso VSensor. Tais especializações têm o objetivo de capturar dados oriundos dos módulos ***Collector***, a composição de diversos recursos VSensor para um único VSensor e o processamento de dados de diversos VSensor, auxiliado por um módulo ***Function***. Como mostrado na Figura 5.5, os tipos são: Link, para criar uma ligação com os sensores da rede de sensores(*i.e* os dados enviados pelos módulos ***Collector***); Composite, responsável por criar um novo VSensor a partir de *Values* de outros VSensor; Blending, o qual realiza processamentos, com auxílio dos módulos ***Function***, sobre um conjunto de *Values* oriundos de diversos VSensors.

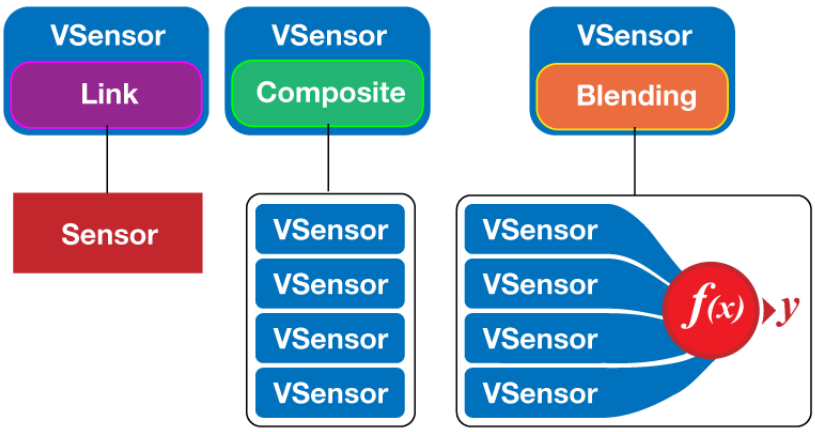

Figura 5.5: Especializações do VSensor

O recurso VSensor não pode ser criado diretamente, pois depende da definição da origem de aquisição dos dados sensoriais. As especializações diferem entre si a forma de

Tabela 5.27: Campos do recurso VSensor

Tipo	Campo	Descrição
`Long`	`id`	Identificador do recurso VSensor
`String`	`label`	Rótulo informativo do VSensor
`String`	`state`	Estado de operação do recurso
`Long`	`creationTimestampInMillis`	Instante da criação ou captura em milissegundos
`Integer`	`creationPrecisionInNano`	Precisão do instante de criação ou captura em nano segundos
`Long`	`acquisitionTimestampInMillis`	Instante da aquisição da amostra pelo módulo Collector ou o instante do término da requisição a um módulo Function, em milissegundos.
`Long`	`storageTimestampInMillis`	Instante do armazenamento do valor pelo VirtualSensorNet
`Long`	`creationInterval`	Valor do intervalo de criação ou captura dos valores
`String`	`creationIntervalTimeUnit`	Unidade de tempo do intervalo de criação ou captura
`Long`	`lastModified`	Data da última atualização do recurso VSensor
`List<Value>`	`values`	Valores de sensoriamento
`String`	`sensorType`	Tipo de VSensorLINK, COMPOSITE ou BLENDING

aquisição dos dados. Outra característica inerente à criação são as tuplas para dados de sensoriamento(*Value*) do recurso VSensor, pois cada *Value* irá armazenar dados de um tipo de sensoriamento. Para dizer quais *Values* um recurso VSensor possui, definimos um elemento de configuração sensorial chamado de *Field*. Cada elemento *Field* reflete um *Value*, definindo as informações inerentes ao dado sensorial. A Tabela 5.28 apresenta os campos do elemento *Field*. A estrutura contém um nome, um recurso DataType para definir o tipo de dado sensorial do *Value*, um recurso Converter em caso de necessidade de conversão, a informação sobre a existência de dados já armazenados, o identificador do VSensor proprietário do elemento *Field*, o número total de outros VSensors agregados e dependentes do elemento *Field*.

Tabela 5.28: Campos do elemento *Field*

Tipo	Campo	Descrição
`Long`	`id`	Identificador do campo
`String`	`name`	Nome do campo
`Long`	`dataTypeId`	Identificador de um recurso DataType
`Long`	`converterId`	Identificador do conversor adicionado
`Boolean`	`initialized`	Sinalização de existência de valores sensoriais no campo
`Long`	`sourceId`	Identificador do VSensor proprietário do campo
`Integer`	`aggregates`	Total de agregados utilizando este campo
`Integer`	`dependents`	Total de dependentes utilizando este campo

Uma outra atribuição das especializações é definir o estado de operação do recurso VSensor, que, neste caso, fica por conta da implementação de cada especialização.

Link: O Link é a especialização do recurso VSensor que atua capturando dados sensoriais enviadas pelos módulos ***Collector***. Esta especialização é a responsável direta por abstrair os dados sensoriais dos dispositivos da rede de sensores. Cada Link se conecta a um dispositivo da rede de sensores através da identificação rede-coletor-sensor. Do mesmo modo que o módulo ***SensorNet*** mapeia os dispositivos de uma rede de sensores através de seu identificador, o Link é capaz de adquirir os dados de sensoriamento de um dispositivo físico. Uma unidade da rede de sensores pode realizar monitoramento diverso, como temperatura e luminosidade no mesmo dispositivo. O elemento *Field* é utilizado para informar quais os dados de sensoriamento o Link deve armazenar através do tipo de dado(DataType) definido. Os campos do Link estão apresentados na Tabela 5.29. O Link possui os campos `id` e `label`, que são similares aos campos descritos na Tabela 5.27, pra o VSensor. O campo `fields` contém uma lista elementos *Field*, elementos apresentados na Tabela 5.28. Os campos `networkId`, `collectorId` e `sensorId` armazenam a identificação

rede-coletor-sensor, necessária para criar o vínculo a um dispositivo da rede de sensores.

Tabela 5.29: Campos do recurso VSensor-Link

Tipo	Campo	Descrição
`Long`	`id`	Identificador do VSensor-Link
`String`	`label`	Rótulo informativo do VSensor
`String`	`sensorId`	Identificador do sensor enviado pelo módulo ***Collector***
`String`	`collectorId`	Identificador do coletor enviado pelo módulo ***Collector***
`String`	`networkId`	Identificador da rede enviado pelo módulo ***Collector***
`List<Field>`	`fields`	Lista de elementos *Field* do VSensor

Por conta do Link, o módulo ***VirtualSensorNet*** deve manter-se vinculado ao grupo de mensagens `omcp://collector.messagegroup.osiris/` para o recebimento das amostras sensoriais enviadas pelos módulos ***Collector***.

Composite: A especialização Composite tem o objetivo de agregar diversos elementos *Field* de outros VSensors na criação de um novo VSensor. A ideia desta especialização é criar um sensor virtual complexo, a partir de diversos outros sensores virtuais mais simplificados. A Tabela 5.30 descreve os campos do recurso Composite. O campo `fields` contém uma lista elementos *Field*, sendo que, a operação sobre este campo é realizada apenas para agregar ou desagregar elementos *Field* de outros recursos VSensor. Os campos `id` e `label` também existem no Composite.

Tabela 5.30: Campos do recurso VSensor-Composite

Tipo	Campo	Descrição
`Long`	`id`	Identificador do VSensor-Composite
`String`	`label`	Rótulo informativo do VSensor
`List<Field>`	`fields`	Lista de campos, de outros VSensors, para a composição

Blending: O Blending é a especialização processadora de dados sensoriais. Esta especialização trabalha em conjunto com o módulo ***Function***, enviando dados para o mesmo processar e armazenar o resultado do processamento nos elementos *Field*. O Blending define seu conjunto de elementos *Field*, os quais serão o repositório dos dados. O recurso Function é utilizado no Blending para definir quais parâmetros devem ser passados para o módulo ***Function*** para o processamento. Os valores dos parâmetros passados são extraídos de elementos *Field* de outros recursos VSensor, elementos que são mapeados pelo Blending. O Blending deve possibilitar, ainda, definir se a chamada ao módulo ***Function*** será síncrona ou assíncrona e, também, bem como o intervalo de tempo entre as chamadas.

A Tabela 5.31 descreve os campos do Blending. O campo `fields` contém uma lista de elementos *Field*. O campo `requestParams` mantém um mapa com identificadores dos parâmetros relacionados para elementos *Field* de outros VSensors. Este mapeamento consiste nos dados para serem processados. O campo `functionId` contém o identificador do recurso Function. Já o campo `responseParams` contém a relação entre os parâmetros recebidos do módulo ***Function*** no resultado do processamento e, os elementos, *Field*, do próprio Blending. As chamadas para o módulo ***Function*** são realizadas em função de um intervalo de tempo em milissegundos, o campo `callIntervalInMillis`. O campo `callMode` define como as chamadas devem ser realizadas, síncrona ou assincronamente.

Tabela 5.31: Campos do recurso VSensor-Blending

Tipo	Campo	Descrição
`Long`	`id`	Identificador do VSensor-Blending
`String`	`label`	Rótulo informativo do VSensor
`Long`	`functionId`	Identificador do recurso Function
`String`	`callMode`	Modo de requisição (`sync` ou `async`)
`Long`	`callIntervalInMillis`	Intervalo de requisição
`List<Field>`	`fields`	Lista de elementos *Field* do VSensor
`Map<Field, Param>`	`requestParams`	Parâmetros para envio na requisição
`Map<Field, Param>`	`responseParams`	Parâmetros para armazenar respostas

5.4.3 Demais funcionalidades do VirtualSensorNet

Da mesma maneira que o módulo ***SensorNet***, o módulo ***VirtualSensorNet*** tem, como requisito, armazenar apenas dados sensoriais mais recentes, realizar a difusão de informações (quando ocorrerem atualizações nos valores dos recursos VSensor) e emitir alertas de alterações no estado de operação dos recursos e dos novos elementos criados. Estas características são, portanto, requisitos funcionais do módulo, e valem como diretrizes para minimizar as grandes diferenças de operações entre as diversas implementações.

5.4.3.1 Armazenamento de dados sensoriais mais recentes

O requisito de armazenamento de dados sensoriais é que o recurso VSensor esteja sempre com o valor mais atualizado, considerando momento da geração(ou captura) da amostra. Para tal verificação, o VSensor conta com os parâmetros `creationTimestampInMillis` e `creationPrecisionInNano`, os quais determinam quando o dado sensorial foi de fato computado. Para implementar o recurso VSensor-Link é necessário realizar a comparação com os dados sensoriais oriundos do módulo ***Collector***. Os dados do coletor possuem os cam-

pos `captureTimestampInMillis` e `capturePrecisionInNano`, os quais são equivalentes aos parâmetros `creationTimestampInMillis` e `creationPrecisionInNano`, respectivamente. Já os dados obtidos através do processamento realizado pelo módulo ***Function***, via VSensor-Blending, contém o campo `responseTimestampInMillis`, o qual pode ser comparado com o parâmetro `creationTimestampInMillis`.

5.4.3.2 Difusão de informações

O módulo ***VirtualSensorNet*** possui a atribuição de informar as atualizações do recurso VSensor a todos os consumidores externos interessados. Para tal ação, o módulo ***VirtualSensorNet*** deve enviar uma representação a cada atualização do recurso VSensor para o grupo de mensagens `omcp://update.messagegroup.osiris/`. Seguindo o mesmo exemplo do módulo ***SensorNet***, a informação divulgada deve ser classificada seguindo as palavras-chave de módulo-recurso. A seguir, o modelo de endereçamento do recurso para a publicação:

- VSensor: `omcp://update.messagegroup/virtualsensornet/vsensor/{vsensor id}`

Como os recursos VSensor-Link, VSensor-Composite e VSensor-Blending são especializações do VSensor, todas as modificações inerentes a estes recursos devem ser difundidas como atualizações do recurso VSensor.

5.4.3.3 Emissão de alerta para notificações

A emissão dos alertas de notificação do módulo ***VirtualSensorNet*** também utiliza o suporte para notificação do OSIRIS(Seção 5.2.2.4) como referência de estrutura de mensagens e grupo de mensagens para a publicação das notificações. No contexto do módulo ***VirtualSensorNet***, a mensagem de notificação segue com informações sobre o estado de operação do recurso VSensor, emitida para os estados de operação NEW, INACTIVE, REACTIVATED e MALFUNCTION. Os estados de operação NEW e REACIVATED são enviados com o nível de criticidade INFO, e os estados INACTIVE e MALFUNCTION são notificados com a criticidade CRITICAL.

5.4.4 Interação com o módulo

O módulo ***VirtualSensorNet*** deve ser capaz de interagir com mundo exterior. Portanto, deve estar habilitado a receber requisições e respondê-las como o módulo ***SensorNet***. A forma de se dispor para interação é via disponibilização de recursos e operações OMCP, sendo encontrável através do endereço `omcp://virtualsensornet.osiris/`. O ***VirtualSensorNet*** possibilita a gerência dos recursos DataType, Conveter, Function, VSensor, VSensor-Link, VSensor-Composite e o VSensor-Blending, por meio de operações de consulta, inclusão, modificação e remoção.

Na Tabela 5.32 é mostrada a listagem completa das operações e recursos disponibilizados pelo módulo ***VirtualSensorNet***. O VSensor é um recurso de apenas leitura, sendo possível realizar consultas para retornar todos os VSensors existentes no módulo, `/vsensor/`, ou um VSensor em particular, `/vsensor/{vsensor id}/`, utilizando um identificador. Para realizar a consulta de histórico de valores de sensoriamento de um VSensor, deve-se utilizar o recurso `revisions`, `/vsensor/{vsensor id}/revisions/`. A lógica de consulta das revisões segue o mesmo critério utilizado no módulo ***SensorNet***, aceitando os mesmos parâmetros (`to`, `from` e `limit`) para filtrar as buscas. Para os demais recursos existentes no módulo ***VirtualSensorNet***, DataType, Converter, Function, VSensor-Link, VSensor-Composite e VSensor-Blending, o conjunto de operações disponíveis compreendem a busca de todos os itens, busca de um item específico quando passado um identificador, criação de recursos pela operação POST, alteração de um recurso pela operação PUT e remoção de um recurso pela operação DELETE.

5.5 Service e External

Quando falamos em sistemas de monitoramento, é possível pensar nas mais diversas aplicações, porém não é possível prever a necessidade exata dos recursos necessários para o desenvolvimento de um sistema de monitoramento mais exotérico. Por conta disso, o OSIRIS contém em sua convenção os nomes para endereçamentos de módulos de serviço(***Service***) e módulos externos(***External***). Por conta disso, novos módulos criados para atender as necessidades de um novo sistema de monitoramento podem ser endereçados como módulos ***Service*** ou módulos ***External***.

A diferença entre estes dois tipos de módulos é que o módulo ***Service*** deve ser utilizado como um recurso *lightweight*, destinado a apenas receber mensagens dos grupos de mensagens. O módulo ***Service*** pode ser utilizado para realizar atualização de tela sobre

Tabela 5.32: Lista dos recursos e operações OMCP do módulo VirtualSensorNet

Operação	Recursos	Descrição
GET	/datatype/	Retorna todos os recursos de DataType existentes
POST	/datatype/	Cria um recurso DataType
GET	/datatype/{datatype id}/	Retorna um recurso DataType de acordo com seu identificador
PUT	/datatype/{datatype id}/	Altera um recurso DataType de acordo com o identificador
DELETE	/datatype/{datatype id}/	Remove um recurso DataType de acordo com o identificador
GET	/converter/	Retorna todos os recursos de Converter existentes
POST	/converter/	Cria um recurso Converter
GET	/converter/{converter id}/	Retorna um recurso Converter de acordo com seu identificador
PUT	/converter/{converter id}/	Altera um recurso Converter de acordo com o identificador
DELETE	/converter/{converter id}/	Remove um recurso Converter de acordo com o identificador
GET	/function/	Retorna todos os recursos de Function existentes
POST	/function/	Cria um recurso Function
GET	/function/{function id}/	Retorna um recurso Function de acordo com seu identificador
PUT	/function/{function id}/	Altera um recurso Function de acordo com o identificador
DELETE	/function/{function id}/	Remove um recurso Function de acordo com o identificador
GET	/vsensor/	Retorna todos os recursos de VSensor existentes
GET	/vsensor/{vsensor id}/	Retorna um recurso VSensor de acordo com seu identificador
GET	/vsensor/{vsensor id}/revisions/	Retorna dados de histórico de um recurso VSensor (Aceita parâmetros)
GET	/link/	Retorna todos os recursos de VSensor-Link existentes
POST	/link/	Cria um recurso VSensor-Link
GET	/link/{link id}/	Retorna um recurso VSensor-Link de acordo com seu identificador
PUT	/link/{link id}/	Altera um recurso VSensor-Link de acordo com o identificador
DELETE	/link/{link id}/	Remove um recurso VSensor-Link de acordo com o identificador
GET	/composite/	Retorna todos os recursos de VSensor-Composite existentes
POST	/composite/	Cria um recurso VSensor-Composite
GET	/composite/{composite id}/	Retorna um recurso VSensor-Composite de acordo com seu identificador
PUT	/composite/{composite id}/	Altera um recurso VSensor-Composite de acordo com o identificador
DELETE	/composite/{composite id}/	Remove um recurso VSensor-Composite de acordo com o identificador
GET	/blending/	Retorna todos os recursos de VSensor-Blending existentes
POST	/blending/	Cria um recurso VSensor-Blending
GET	/blending/{blending id}/	Retorna um recurso VSensor-Blending de acordo com seu identificador
PUT	/blending/{blending id}/	Altera um recurso VSensor-Blending de acordo com o identificador
DELETE	/blending/{blending id}/	Remove um recurso VSensor-Blending de acordo com o identificador

novas medidas, como um *daemon* que recebe valores e armazena estes dados em algum tipo de repositório, entre outras funcionalidades. Já os módulos ***External*** são módulos mais robustos, que podem realizar diversas tarefas, requisições, processamentos, entre outras funcionalidades que venham a ser necessárias ao sistema de monitoramento.

Capítulo 6

Implementação do OSIRIS

Neste capítulo abordamos a implementação do OSIRIS, falando sobre as tecnologias utilizadas e as soluções para a implementação da arquitetura base do framework utilizando a linguagem Java. Iniciamos falando sobre a camada de comunicação e o protocolo OMCP, seguindo da API do OSIRIS, a qual contém as definições do framework, e encerramos com uma visão da implementação dos módulos.

O capítulo está organizado em: a)Seção 6.1: com detalhes sobre a implementação da camada de comunicação; b)Seção 6.2: apresentando a API do OSIRIS, com detalhes sobre o local das definições; c)Seção 6.3: finalizando com a explicação da implementação dos módulos nativos.

6.1 Implementação da Camada de Comunicação

Como já citado, a camada de comunicação precisava ser um meio facilitador e não um obstáculo para a adição e remoção de módulos. Devido essa decisão, buscamos construir a camada de comunicação livre de influências restritivas, aceitando módulos escritos em qualquer linguagem de programação. Optamos por utilizar o RabbitMQ [29], um middleware de comunicação orientado a mensagens para sistemas distribuídos, como apresentado na Seção 2.4 do Capítulo 2. O RabbitMQ contém características que possibilitam alimentar, de forma satisfatória, sistemas de monitoramento de dados amostrais. A seguir, explicamos a implementação do protocolo OMCP sobre o RabbitMQ.

6.1.1 OMCP sobre RabbitMQ

Para utilizar o OMCP sobre o RabbitMQ, trabalhamos com middleware para atender três requisitos básicos do OMCP: realizar requisições(comunicação síncrona), enviar mensagens de notificação (comunicação assíncrona) e dar suporte à criação dos grupos de mensagens. Ainda com o RabbitMQ, atingimos o suporte necessário ao OMCP para endereçamento e resolução de nomes, devido as filas e os *exchanges* serem recursos endereçáveis dentro do middleware, possibilitando o vínculo a um endereço de recurso OMCP.

O RabbitMQ possui a habilidade natural de trabalhar com publish-subscribe por meio da utilização de *exchanges* em conjunto com as filas, como apresentado na Figura 6.1. Para implementar os grupos de mensagens com a característica de separação de informação por temas e subtemas, foi escolhido utilizar *exchanges* com o algoritmo de roteamento *TOPIC*. Um *TOPIC exchange* é capaz de rotear mensagens para uma ou mais filas, baseado no casamento entre a *routing key* com o *binding key*. A *routing key* é uma estrutura composta de palavras delimitadas por pontos, que, anexada à mensagem enviada, especifica características relacionadas à mesma. O *binding key*, um elemento do *binding*, segue a mesma sintaxe, porém, trata-se de uma estrutura para declarar o interesse da fila sobre quais mensagens deseja receber. Na camada de comunicação, os interessados devem se conectar ao grupo de mensagens informando temas e subtemas, os quais são convertidos para uma *binding key*. Já uma mensagem publicada pode possuir temas e subtemas, que se convertem em uma *rounting key*.

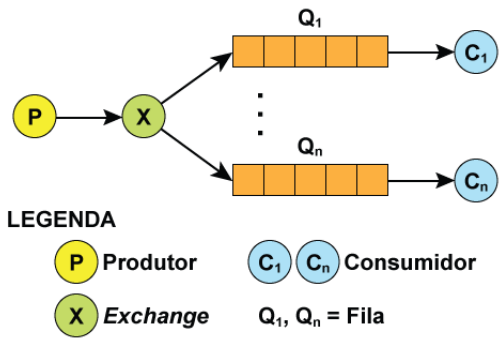

Figura 6.1: Pubilsh-subscribe com o RabbitMQ

Para realizar requisições sobre a estrutura do RabbitMQ, utilizamos a abordagem descrita na Figura 6.2. A figura mostra a abordagem para a comunicação síncrona entre dois pontos, denominados requisitante e requisitado. Cada ponto possui um produtor, uma fila e um consumidor para receber os dados da fila. A fila é a referência do ponto no RabbitMQ, contendo um endereçamento passível de ser mapeado para um endereço

de recurso OMCP. O elemento consumidor e o produtor são componentes embarcados na aplicação, que para este caso, são a aplicação requisitante e a aplicação requisitada. A comunicação entre o requisitante e o requisitado se dá pelo produtor do requisitante enviando uma mensagem de requisição para a fila (*F1*) do requisitado. Quando a fila *F1* recebe a mensagem, imediatamente a repassa para o consumidor da aplicação requisitada. A ação de resposta realiza um processo semelhante: o requisitado envia uma mensagem de reposta para a fila (*F2*) do requisitante, que a repassa imediatamente ao consumidor. É importante destacar neste ponto que o requisitante conhece previamente o endereçamento da fila (*F1*) do requisitado. Para o requisitado conhecer a fila (*F2*) do requisitante, a mensagem de requisição carrega a informação com o endereço da fila (*F2*), para que a mensagem de resposta possa ser corretamente enviada ao destinatário.

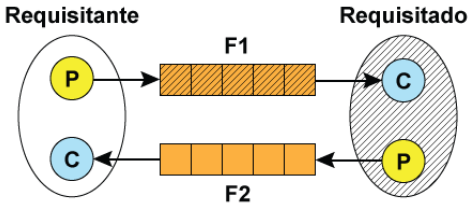

Figura 6.2: Abordagem cliente-servidor sobre o RabbitMQ

6.1.2 Componentes cliente e servidor

Para trabalhar efetivamente como componentes de um módulo do OSIRIS, criamos um componente cliente, Client, e um componente servidor, Server. O componente Server pode responder requisições e receber notificações e o componente Client pode enviar notificações como realizar requisições aos componentes Server.

A Figura 6.3 apresenta o esquema da comunicação dos componentes Client e Server. O Client, quando realiza requisições, comunicação síncrona, ele cria uma fila temporária para receber a resposta enviada pelo Server. Por outro lado, quando emite notificações, comunicação assíncrona, seja para um *exchange* (representando o grupo de mensagens) ou seja diretamente para um Server, o Client não precisa esperar resposta, abstendo-se de criar uma fila temporária.

Disponibilizamos no pacote `br.uff.labtempo.omcp`, que pode ser obtido neste endereço `https://github.com/labtempo/osiris/tree/master/OMCP`, a implementação dos componentes de comunicação Client, Server e Service, sendo o Service uma variação simplificada do Server, com capacidade de inscrever-se em grupos de mensagens e receber apenas mensagens de notificação. Todos os componentes criados para funcionar com

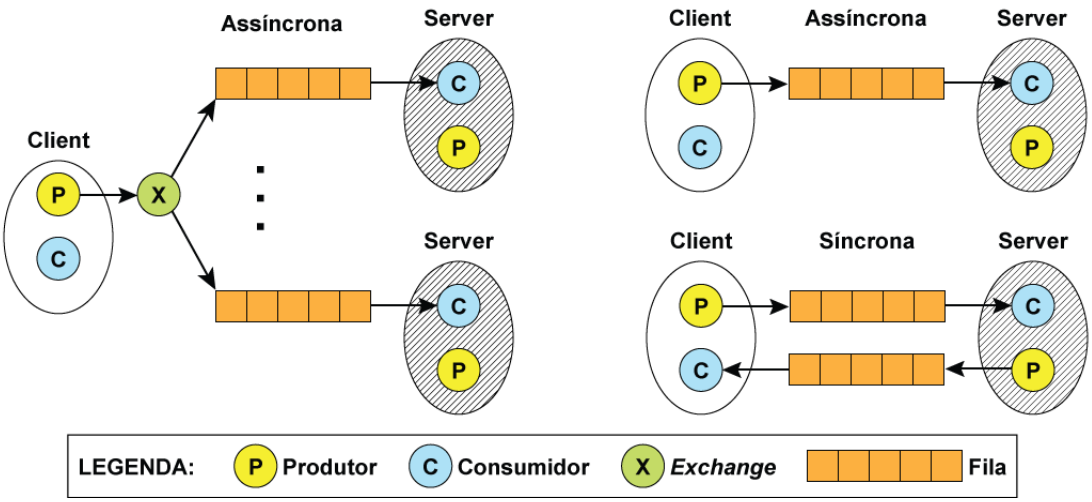

Figura 6.3: Abordagem de comunicação síncrona e assíncrona no RabbitMQ

RabbitMQ utilizam um conversor de endereços para transformar o domínio de uma URL OMCP para o endereçamento de filas e *exchanges*. Os componentes operam também com JSON [4][14] como padrão para serializar os objetos a serem transmitidos e recebidos. O componente Client implementa todas as operações(GET, POST, PUT, DELETE e NOTIFY).

Antes de qualquer comunicação síncrona, o Client verifica a existência da fila relacionada ao Server. Em seguida, verifica se existe um consumidor para esta fila. Em caso positivo, envia a requisição. Na comunicação síncrona, o Client espera por um tempo pela resposta. Se a resposta não chegar dentro deste tempo, ele desiste da ação e assume que ela não foi executada. O Server, por sua vez, envia a resposta como uma transação atômica após executar a requisição. O RabbitMQ oferece o suporte para realização de transações atômicas. Quando o Client utiliza a operação NOTIFY, comunicação assíncrona, as mensagens enviadas são marcadas como persistentes e podem ser armazenadas internamente no RabbitMQ até serem consumidas. Por outro lado, as mensagens de requisição têm um tempo de vida curto, devem ser repassadas ao consumidor ou imediatamente descartadas.

O componente Server implementado contém uma fila, mas também tem a capacidade de se inscrever em grupos de mensagens. A fila relacionada a um componente Server é persistente, o que significa que, em caso da aplicação servidora ser desligada por eventuais causas, a fila continuará ativa no RabbitMQ para receber mensagens de notificação. Os dados da fila ficarão persistidos para serem consumidos quando o Server retornar ao funcionamento normal. É permitido apenas um consumidor para cada fila relacionada ao Server nesta implementação.

6.2 API do OSIRIS

Disponibilizamos no pacote `br.uff.labtempo.osiris` as definições do OSIRIS em forma de API para uso dos módulos, tanto os nativos quanto os novos. A API é a base para criar módulos por conter todas as definições estabelecidas na arquitetura do OSIRIS. O código pode ser obtido neste endereço `https://github.com/labtempo/osiris/tree/master/Osiris`. A API é composta por objetos de transferência ou objetos TO (*Transfer Object*) [1] e as definições para estados de operação dos recursos de sensoriamento, para tipos de valores, para os operadores lógicos e para os modos de operação dos módulos ***Function***. Ela contém ainda uma listagem com endereços dos recursos OMCP, um controle abstrato com funcionalidades pré-implementadas e um pacote de utilidades complementares. A API do OSIRIS inclui ainda a implementação do OMCP sobre o RabbitMQ, pelo pacote `br.uff.labtempo.omcp`, com os componentes `Client`, `Server` e `Service`.

6.2.1 Controle abstrato e pacote de utilidades

A API disponibiliza a classe abstrata `Controller`, a qual foi preparada para trabalhar com os componentes Server e Service do OMCP. A classe `Controller` implementa os padrões de projeto *strategy* e *chain of responsibility* [11], possibilitando encadear múltiplos controles(classes herdeiras da classe `Controller`) de modo a aplicar uma filtragem de mensagens de acordo com o recurso requisitado. A classe `Controller` contém ainda métodos para tratar extrair informações de nome e id dos endereços OMCP.

Já o pacote complementar da API do OSIRIS, consiste em um conjunto de componentes pré-implementados para serem utilizados pelos módulos de acordo com a necessidade. O pacote está disponível em `br.uff.labtempo.osiris.utils` e contém uma implementação para pool de requisições, chamada `RequestPool`; um componente para realizar difusão de informações de emissões de alertas de notificação, o `Announcement`; um agendador de tarefas, chamado `Scheduling`; um processador de datas em linguagem natural, denominado `NlpDateParser` e, finalizando, uma implementação para a persistência de dados em lote com JPA2, chamada de `BatchPersistence`.

6.2.2 Objetos de transferência

Os objetos TO são estruturas preparadas para a transferência de dados entre os módulos do OSIRIS. O objeto TO armazena dados de forma estruturada para uma posterior seri-

alização pelo componente OMCP `Client` e `Server`. Os objetos TO abrangem a definição dos módulos ***Collector***, ***SensorNet***, ***VirtualSensorNet*** e ***Function***, apresentados no Capítulo 4, pois estes necessitam de dados específicos inerentes as suas funcionalidades

6.2.2.1 Collector TO

Para transferir os dados coletados é disponibilizado o objeto TO no pacote `br.uff.labtempo.osiris.to.collector`. Na Tabela 6.1 estão descritas as classes existentes no pacote, que são estas: `NetworkCoTo`, comportando os dados definidos pelo container **network**; `CollectorCoTo`, contendo os atributos do container **collector**; `SensorCoTo`, comportando todas as informações do container **sensor**; `SampleCoTo`, comportando as classes `NetworkCoTo`, `CollectorCoTo` e `SensorCoTo`, criando um container único para envio dos dados. Esta é a implementação direta dos campos definidos na Tabela 5.8 da Seção 5.1.2 do módulo ***Collector***.

Tabela 6.1: Classes To para o módulo Collector

Objeto TO		**Container**	**Tabela**
`SampleCoTo`	`NetworkCoTo`	**network**	Tabela 5.8
	`CollectorCoTo`	**collector**	
	`SensorCoTo`	**sensor**	

6.2.2.2 SensorNet TO

A API do OSIRIS implementa os recursos apresentados na Seção 5.2.1 do módulo ***SensorNet*** em forma de classes, como apresentadas na Tabela 6.2. A classe `NetworkSnTo` engloba os dados do recurso Network, a classe `CollectorSnTo` compreende os atributos do recurso Collector e a classe `SensorSnTo` comporta os dados do recurso Sensor. A API ainda define um objeto TO para as revisões com classe `RevisionSnTo`, compreendendo os dados temporais e os valores sensoriais em conjunto com os níveis dos recursos fungíveis. As classes estão dispostas no pacote `br.uff.labtempo.osiris.to.sensornet`.

Tabela 6.2: Classes To para o módulo SensorNet

Objeto TO	**Recurso**	**Tabela**
`NetworkSnTo`	Network	Tabela 5.9
`CollectorSnTo`	Collector	Tabela 5.10
`SensorSnTo`	Sensor	Tabela 5.11
`RevisionSnTo`	Revision	

6.2.2.3 VirtualSensorNet TO

Para a transferência de dados do módulo ***VirtualSensorNet***, foram implementadas as classes descritas na Tabela 6.3 para os recursos apresentados na Seção 5.4.2. As classes `DataTypeVsnTo`, `ConverterVsnTo`, `FunctionVsnTo`, `VirtualSensorVsnTo`, `LinkVsnTo`, `CompositeVsnTo` e `BlendingVsnTo` são correspondentes aos recursos DataType, Converter, Function, VSensor, VSensorLink, VSensorComposite e VSensorBlending, respectivamente. Para as revisões, a classe `RevisionVsnTo` foi implementada comportar as informações temporais e os dados sensoriais. As classes de transferência estão disponíveis em `br.uff.labtempo.osiris.to.virtualsensornet`.

Tabela 6.3: Classes TO para o módulo VirtualSensorNet

Objeto TO	Recurso	Tabela
`DataTypeVsnTo`	DataType	Tabela 5.23
`ConverterVsnTo`	Converter	Tabela 5.24
`FunctionVsnTo`	Function	Tabela 5.25
`VirtualSensorVsnTo`	VSensor	Tabela 5.27
`LinkVsnTo`	VSensorLink	Tabela 5.29
`CompositeVsnTo`	VSensorComposite	Tabela 5.30
`BlendingVsnTo`	VSensorBlending	Tabela 5.31
`RevisionVsnTo`	Revision	

6.2.2.4 Function TO

Para os módulos ***Function*** estão disponibilizadas as classes de transferência em `br.uff.labtempo.osiris.to.function`. O módulo ***Function*** conta com três classes, descritas na Tabela 6.4, para realizar a transferência de informações. São elas: `InterfaceFnTo`, para comportar as informações sobre o recurso Interface, `ResquestFnTo` para realizar requisições, enviado os dados a serem processados e `ResponseFnTo`, a classe que tem o propósito de comportar os dados com o resultado do processamento. As classes TO são mapeamentos dos recursos apresentados na Seção 5.3.1. Em adicional está disponibilizada a classe `ParameterizedRequestFn` para a conversão dos dados contidos no objeto `ResquestFnTo` para parâmetros de endereço.

Tabela 6.4: Classes TO para o módulo Function

Objeto TO	Recurso	Tabela
`InterfaceFnTo`	Interface	Tabela 5.17
`ResquestFnTo`	RequestObject	Tabela 5.20
`ResponseFnTo`	ResponseObject	Tabela 5.21

6.2.2.5 Objeto TO para notificações

O módulo ***SensorNet*** e ***VirtualSensorNet*** precisam realizar notificações. Para esta ação foi definida uma mensagem de notificação e os níveis de criticidade na Seção 5.2.2.4. Para suportar as notificações definidas para o OSIRIS, a API disponibiliza em `br.uff.labtempo.osiris.to.notification` a classe `Notification`, que implementa todas as definições, os campos da mensagem e os níveis de criticidade. A Tabela 6.5 estabelece as ligações entre as classes e as tabelas com as definições.

Tabela 6.5: Objeto TO para a mensagem de notificação

Objeto TO	Tabela
`Notification`	Tabela 5.12 e 5.13

6.2.3 Definições do OSIRIS

Todas as definições computacionalmente relevantes para a implementação dos módulos foram mapeadas para enumerados. A Tabela 6.6 apresenta a lista completa de definições mapeadas para enumerados, compreendendo um conjunto de dados definidos nos Capítulos 4 e 5. Todos os enumerados se encontram no pacote `br.uff.labtempo.osiris.to.common.definitions`.

Tabela 6.6: Lista de enumerados da API com as definições do OSIRIS

Enumerados	Tabela
`State`	Tabela 5.7
`LogicalOperator`	Tabela 5.5
`ValueType`	Tabela 5.2
`FunctionOperation`	Tabela 5.18
`Path`	Tabela 4.4
	Tabela 4.5
	Tabela 4.6
	Tabela 5.14
	Tabela 5.15
	Tabela 5.32
	Tabela 5.22

No Capítulo 4 foram definidos os conjuntos dos estados de operação(NEW, INACTIVE, UPDATED, REACTIVATED e MALFUNCTION), dos operadores lógicos(`==`, `!=`, `>`, `<`, `>=` e `<=`) e dos tipos de valores(`number`, `logic` e `text`). Todas estas definições foram mapeados para os enumerados `State`, para o estado de operação, `LogicalOperator`, para os operadores lógicos, e `ValueType`, para determinar os tipos de valores possíveis de um

dado sensorial. Já os modos de operação do módulo ***Function***(`sync` e `async`) foram mapeados para o enumerado `FunctionOperation`.

O enumerado `Path` inclui as informações sobre definições do Capítulo 4, englobando os domínios dos módulos e do grupo de mensagens e o endereçamento dos recursos nativos. Do Capítulo 5, mantém informações sobre todos os nomes de recursos dos módulos ***SenorNet***, ***VirtualSensorNet*** e ***Function*** e os parâmetros de *Query Strings* para a busca de revisões. O enumerado `Path` conta ainda com uma listagem completa de recursos OMCP dos módulos ***SensorNet***, ***VirtualSensorNet*** e ***Function*** com caracteres coringas(*i.e wildcards*). Essa listagem é utilizada com a implementação do `Controller` para a extração de informações do endereço OMCP, como *name* e *id*.

6.3 Implementação dos módulos

Nesta seção abordamos a implementação dos módulos nativos do OSIRIS, o ***SensorNet*** e o ***VirtualSensorNet***. Estes módulos têm, em comum, a sua arquitetura no padrão MVC [18] e a persistência, utilizando um banco de dados PostgreSQL [27] em conjunto com o Hibernate [13]. Construímos estes módulos com a linguagem Java, com a tecnologia Java Platform Standard Edition ou JavaSE [23] para funcionar em desktops. Para referência de implementação dos módulos ***Collector***, ***Function*** e ***Service***, disponibilizamos os códigos-fonte em `https://github.com/labtempo/osiris`. Para obter o binário da API do OSIRIS e dos módulos ***SensorNet*** e ***VirtualSensorNet*** basta acessar `https://github.com/labtempo/osiris-binaries`.

6.3.1 SensorNet

Esta implementação do módulo ***SensorNet*** visou dar suporte à criação das especificações para o módulo, bem como avaliar o projeto arquitetural. Aqui, iremos abordar a solução utilizada para implementar os requisitos apresentados no Capítulo 5, os quais implicam no armazenamento de dados sensoriais mais recentes, a inferência de estado de operação de um recurso, a difusão de informações e a emissão de alertas de notificação. Em adicional, abordamos brevemente sobre a arquitetura escolhida para implementar o módulo ***SensorNet***.

6.3.1.1 Detalhes da implementação

A implementação do módulo ***SensorNet*** em Java foi realizada utilizando a arquitetura MVC, a qual conta com uma camada de controle, o modelo e a visão (view).

A camada de controle é a responsável por habilitar a interação do módulo com o mundo exterior, recebendo as requisições GET e DELETE para os recursos Network, Collector e Sensor. Outra atribuição da camada de controle é receber os dados sensoriais enviados pelos módulos ***Collector***, as com operação NOTIFY. A camada de controle realiza mediação com o modelo e a view. Os controles presentes no módulo foram criados com base no controle abstrato `Controller`, disponível em `br.uff.labtempo.osiris.omcp`. Aplicamos ainda um pool de requisições para tratar com o grande volume de mensagens oriundas dos módulos ***Collector***, para ordenar as mensagens com dados sensoriais. O pool de requisições está disponível em `br.uff.labtempo.osiris.utils.requestpool`.

O modelo, apresentado na Figura 6.4 pelo diagrama de classes de domínio, contém as classes utilizadas para implementar os requisitos do módulo ***SensorNet***. As classes `Network`, `Collector` e `Sensor` são as representações diretas para os recursos Network, Collector e Sensor respectivamente. As três classes herdam diretamente da classe abstrata `Model`. A classe abstrata `Model` foi utilizada para agrupar elementos comuns às três classes, que neste caso são o horário de criação, as informações extras e o estado de operação. Implementamos a classe `Field` para alocar os dados sensoriais e a classe `Consumable` para manter os níveis dos recursos fungíveis e as expressões condicionais de verificação destes níveis. A classe `Revision` desempenha o papel de armazenar valores sensoriais e os níveis dos recursos fungíveis para criar um histórico de valores para futuras buscas. O enumerado `ModelState` é a implementação dos estados de operação, incluindo a lógica de transição entre os estados, definida no Seção 5.1.1.7. Já os enumerados `ValueType` e `LogicalOperator` são elementos do pacote `br.uff.labtempo.osiris.to.common.definitions`, da API do OSIRIS.

Para realizar a persistência do modelo, nós utilizamos JPA2 [24] com o Hibernate em conjunto com o banco de dados Postgresql. Utilizamos a persistência de dados em lote devido ao volume de dados chegados dos módulos ***Collector***. Utilizando este modo de persistência, foi possível contornar alguns problemas em persistir um grande volume de dados sensoriais sem erros de abertura de conexões com o banco de dados pelo Hibernate. O componente utilizado para a persistência em lote está disponível em `br.uff.labtempo.osiris.utils.persistence.jpa.batch`.

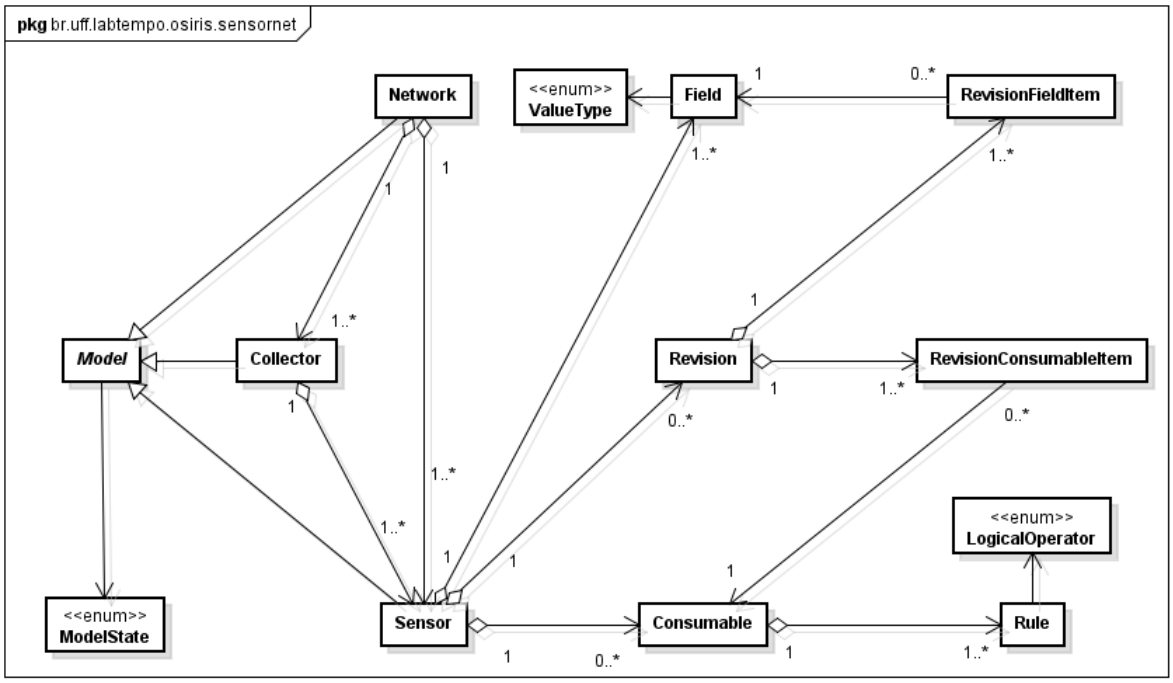

Figura 6.4: Diagrama de classes do modelo do SensorNet

6.3.1.2 Critério de armazenamento de dados sensoriais

Para esta implementação, resolvemos descartar dados sensoriais, oriundos do módulo ***Collector***, com o tempo de captura inferior ou igual ao último dado sensorial persistido no módulo ***SensorNet***, via objeto `Sensor`. A implementação, com esse tipo de política de armazenamento, foi possível pela utilização dos parâmetros `captureTimestampInMillis` em conjunto com o parâmetro `capturePrecisionInNano`, o qual informa o momento em nano segundos da captura. Nenhuma verificação nos valores é realizada, resultando em persistência de todos os dados, independente da mudança dos valores de sensoriamento.

6.3.1.3 Inferência do estado de operação do recurso

Para implementar a inferência do estado de operação dos recursos, optamos por usar uma funcionalidade periódica, utilizando um agendador de tarefas construído sobre o componente `Scheduler`, presente em `br.uff.labtempo.osiris.utils.scheduling`. O agendador de tarefas possibilitou realizar uma verificação temporal sobre os objetos `Sensor`, para verificar se o mesmo está atualizado dentro do tempo previsto, considerando a informação do intervalo de geração de dados disponibilizada pelo módulo ***Collector***. Caso o sensor perca o prazo, ele é marcado como inativo. Para inferir o estado de operação do `Collector`, optamos por verificar se todos os recursos agregados a este recurso `Sensor`, estão desativados. Se todos os `Collector` de uma rede de sensores se tornarem inativos,

o objeto `Network`, que agrega tais recursos, também refletirá o estado de inatividade. Em caso de um `Sensor` operar sobre o estado MALFUNCTION, toda a hierarquia(`Collector` e `Network`) refletem o estado de MALFUNCTION.

6.3.1.4 Difusão de informações e emissão de alertas de notificação

Para a difusão de dados, aplicamos a difusão dos objetos `NetworkSnTo`, `CollectorSnTo` e o `SensorSnTo` logo após os dados serem recebidos e persistidos. O objeto `SensorSnTo` é o mais comum de ser difundido, devido ás constantes atualizações de dados sensoriais recebidas da rede de sensores. Os objetos `NetworkSnTo` e `CollectorSnTo` são emitidos em uma frequência menor que o objeto `SensorSnTo`, devido às atualizações ocorrerem em um período de tempo maior. Utilizamos o componente `Announcement`, disponibilizado em `br.uff.labtempo.osiris.utils.announcement`, para auxiliar no envio dos dados para o grupo de mensagens `omcp://update.messagegroup.osiris/`. Ainda com o auxílio do componente Announcement, é realizada a emissão dos alertas de notificações para o mundo exterior, via publicação no grupo de mensagens `omcp://notification.messagegroup.osiris/`, informando os sensores desativados, reativados, em mau funcionamento e novos, bem como os alertas nos níveis dos recursos fungíveis.

6.3.2 VirtualSensorNet

A construção do módulo ***VirtualSensorNet*** consistiu em implementar uma camada abstração de dados, baseada nos sensores virtuais, e criar as ramificações, capazes de compor e processar dados de diversos sensores virtuais. Os requisitos adicionais do módulo compreendem o armazenamento de dados sensoriais mais recentes, a difusão de informações e a emissão de alertas de notificação.

6.3.2.1 Detalhes da implementação

Para a implementação do módulo ***VirtualSensorNet*** utilizamos recursos similares aos recursos utilizados no módulo ***SensorNet***. Dentre as similaridades esta a arquitetura MVC.

A camada de controle é a responsável pela interação do módulo ***VirtualSensorNet*** com o exterior. Foram construídos controles para os recursos DataType, Converter, Function, VSensor, VSensor-Link, VSensor-Composite e VSensor-Blending. Todos os controles foram criados com base sobre a classe `Controller`(`br.uff.labtempo.osiris.omcp`).

Para receber os dados de sensorioamento enviados pelos módulos Collector, foi criado um controle à parte chamado `NotifyController`. Utilizamos no `NotifyController` o componente RequestPool(`br.uff.labtempo.osiris.utils.requestpool`) para tratar com o grande volume de mensagens recebido. Os recursos DataType, Converter, Function, VSensor-Link, VSensor-Composite e VSensor-Blending possuem operações de criação, busca, modificação e remoção via operações POST, GET, PUT e DELETE, respectivamente.

O modelo, na Figura 6.5, apresenta o digrama de classes de domínio do módulo ***VirtualSensorNet***. Este modelo contém pontos similares com o modelo do módulo ***SensorNet***, como a classe abstrata `Model`, com dados sobre o estado de operação, informações e tempos criação e armazenamento. A abordagem de manter histórico de dados sensoriais também é realizada por uma classe `Revision`, porém, aqui, implementada de forma diferente, para comportar apenas dados sensoriais, sendo persistido todos os dados, mesmo sendo valores iguais. Seguindo a definição para os sensores virtuais, foi implementada uma classe abstrata, chamada `VirtualSensor`, com três herdeiros: o `Link`, o `Composite` e o `Blending`, representando respectivamente os recursos VSensor-Link, VSensor-Composite e o VSensor-Blending. A classe `Field` armazena os dados de configuração para armazenamento dos valores sensoriais. A classe `Function` é a responsável por mapear um módulo ***Function*** dentro do módulo ***VirtualSensorNet***. Os parâmetros de entrada e saída do módulo ***Function*** são mapeados para a classe `FunctionParam`. O enumerado `ModelState` e `ValueType` são elementos do pacote `br.uff.labtempo.osiris.to.common.definitions`, da API do OSIRIS. O enumerado `VirtualSensorType` contém os valores LINK, COMPOSITE, BLENDING, para ser utilizado no objeto TO `VirtualSensorVsnTo`.

A view do módulo ***VirtualSensorNet*** é formada pelos objetos TO apresentados na Tabela 6.3, `DataTypeVsnTo`, `ConverterVsnTo`, `FunctionVsnTo`, `VirtualSensorVsnTo`, `LinkVsnTo`, `CompositeVsnTo`, `BlendingVsnTo` e `RevisionVsnTo`.

A persistência é realizada utilizando JPA2 com o Hibernate e notificações são realizadas pelo componente `Announcement` para informar as mudanças de estados de operação no sensores virtuais. Um detalhe importante é que o módulo ***VirtualSensorNet*** utiliza os estados de operação inferidos pelo ***SensorNet*** para definir o estado de operação dos seus sensores virtuais.

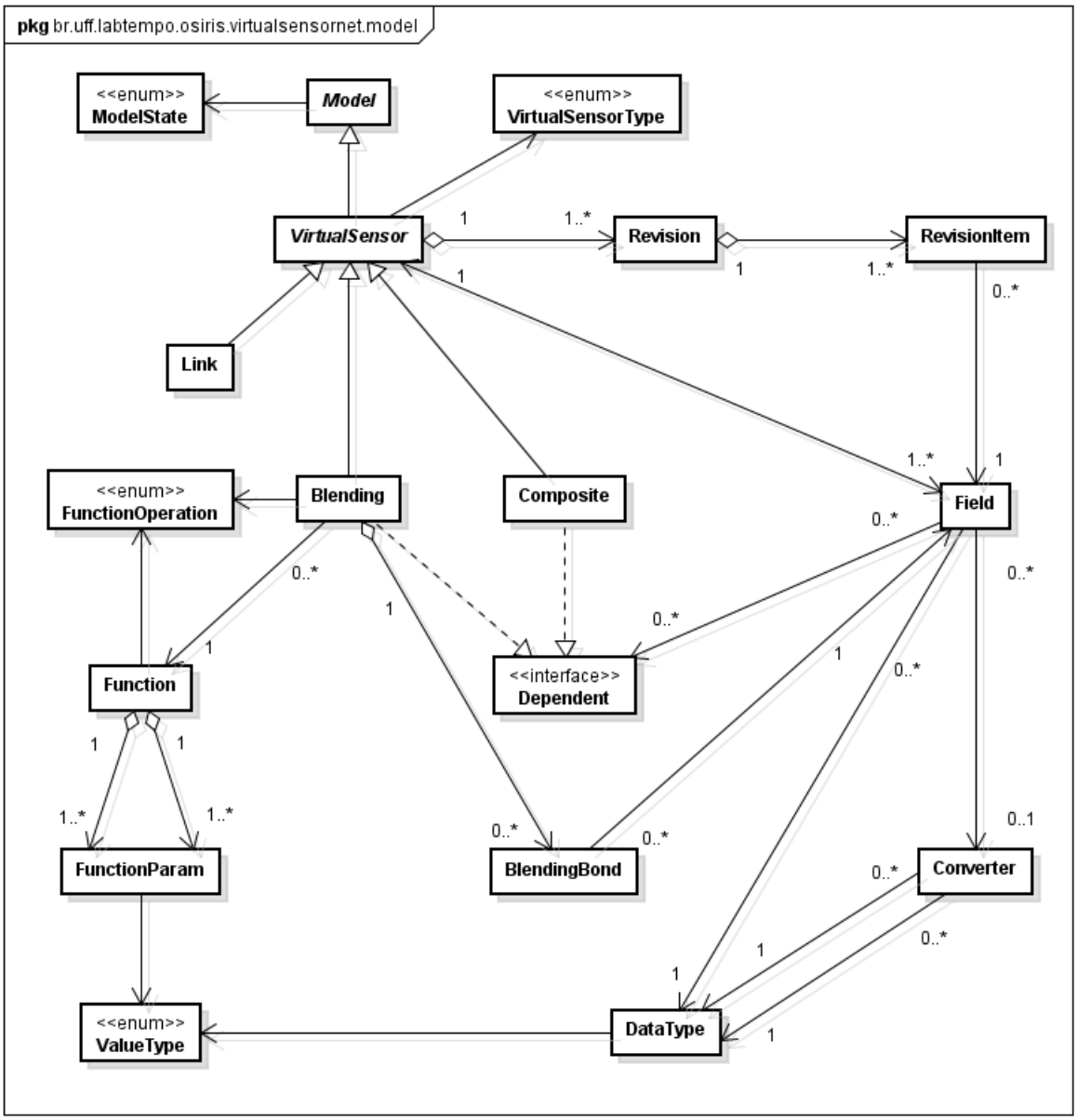

Figura 6.5: Diagrama de classes do modelo do VirtualSensorNet

6.3.2.2 Camada de abstração

A camada de abstração foi implementada com base na classe abstrata `VirtualSensor`. A classe `Field` foi implementada para comportar os valores sensoriais. A classe `Field` está diretamente relacionada à classe abstrata `VirtualSensor`, pois todo o sensor virtual deve conter campos sensoriais. O modo como os objetos `Field` são criados para um sensor virtual é definido pelas três extensões da classe `VirtualSensor`, as classe `Link`, `Composite` e `Blending`.

Link A classe do `Link` é a representante do recurso VSensor-Link, apresentado na especificação. Ela é mais simples, como pode ser visualizada no modelo, contendo a tripla de identificadores rede-coletor-sensor. A tripla de identificadores estabelece o vínculo(por isso o nome link) com um dispositivo existente na rede de sensores. Quando o módulo ***Collector*** envia os dados coletados, o `Link` consegue separar o que deseja armazenar. Neste ponto é que a criação de um objeto `Field` entra em ação. Optamos por utilizar tais objetos como pontos de referências para determinar quais valores de uma amostra sensorial devem ser armazenados, pois se uma mostra envia dados sobre temperatura e luminosidade, é perfeitamente possível criar um `Link` para manter os valores de temperatura e outro para os valores de luminosidade, bastando criar os objetos `Field` com o nome de tais medidas. É persistente mencionar que cada objeto `Field` contém um `DataType` e pode conter ainda um `Converter`, para, por exemplo, converter dados sensoriais de temperatura de Fahrenheit para Celsius.

Composite A implementação do `Composite` contém certa complexidade na questão de assimilar objetos `Field` de outros sensores virtuais. O `Composite` "pega emprestado" os objetos `Field` de outros sensores virtuais, os quais são considerados como próprios na perspectiva do `VirtualSensor`. O objeto `Field` tem ciência de qual `VirtualSensor` ele pertence e quais os `VirtualSensor` ele está agregado. Optamos por implementar o `Composite` refletindo as atualizações de seus objetos `Field` agregados, recebendo os valores sensoriais e o estado de operação do sensor virtual dono do `Field`.

Blending A implementação do `Blending` é a mais complexa, pois ele necessita se relacionar com os módulos ***Function***, conhecendo parâmetros de entrada e saída, bem como seus próprios campos para onde os resultados computacionais devem ser armazenados. O `Blending` é um sensor virtual que possui um conjunto próprio de objetos `Field`, porém ele pode agregar objetos `Field` de outros sensores virtuais. Seus objetos `Field` são o destino dos resultados do processamento executado pelo módulo ***Function***, enquanto os objetos `Field` de outros sensores virtuais são os argumentos passados para a função remota(módulo ***Function***). A classe responsável por manter esta configuração é a `BlendingBond`. A `BlendingBond` realiza o mapeamento entre `Field` e os parâmetros relacionados ao módulo ***Function***, parâmetros os quais estão contidos no objeto `Function`. Optamos nesta implementação do `Blending` por realizar requisições periódicas ao módulo ***Function***, com um intervalo de tempo definido no próprio virtual sensor. Para realizar as chamadas periódicas utilizamos o componente `Scheduler` como agendador de tarefas.

6.3.2.3 Critério de armazenamento de dados sensoriais

Os sensores virtuais `Link` e `Blending` descartam dados sensoriais recebidos mais antigos que os valores já persistidos pelo módulo. O `Link` utiliza os dados do módulo ***Collector*** e compara os parâmetros `captureTimestampInMillis` e `capturePrecisionInNano` com os parâmetros `creationTimestampInMillis` e `creationPrecisionInNano`, respectivamente equivalentes. Já o `Blending` compara apenas `responseTimestampInMillis`, da resposta do módulo ***Function***, com o parâmetro do `VirtualSensor` `creationTimestampInMillis`.

6.3.2.4 Inferência do estado de operação do recurso

Nesta implementação do módulo ***VirtualSensorNet*** optamos por realizar a inferência do estado de operação do recurso VSensor com o auxílio do módulo ***SensorNet***. Para realizar esta tarefa, o módulo ***VirtualSensorNet*** foi inscrito no grupo de mensagens `omcp://update.messagegroup.osiris/sensornet/` para capturar as atualizações dos recursos Sensor do ***SensorNet***. A cada atualização recebida, uma comparação com o `Link` é realizada com o propósito de verificar o estado de operação inferida pelo ***SensorNet***. O `Composite` e o `Blending` têm seus estados de operação baseados no estado de operação do `Link`.

6.3.2.5 Difusão de informações e emissão de alertas de notificação

Para a difusão de dados de atualização foi aplicada ao objeto `VirtualSensor`, via sua representação para transferência, um objeto `VirtualSensorVsnTo`. Foi utilizado o componente `Announcement` para o envio de dados para o grupo de mensagens `omcp://update.messagegroup.osiris/`. Com o mesmo componente é realizada a emissão dos alertas de notificações para o mundo exterior, pelo grupo de mensagens `omcp://notification.messagegroup.osiris/`, informando os sensores virtuais desativados, reativados e novos.

Capítulo 7

Prova de Conceito

Neste capítulo descreveremos como fazer uso do OSIRIS para construção de um novo sistema de monitoramento. Usaremos como exemplo a criação de um sistema de monitoramento térmico com redes de sensores sem fio, chamado TMON.

O capítulo está organizado em: a)Seção 7.1: explicando sucintamente alguns aspectos do TMON; b)Seção 7.2: explanando a abordagem utilizada para implementar o TMON com o OSIRIS; c)Seção 7.3: abordando aspectos da configuração do OSIRIS; d)Seção 7.4: explicando brevemente como criar um módulo ***Collector***; e)Seção 7.5: abordando o consumo de dados dos módulos ***SensorNet*** e ***VirtualSensorNet***; f)Seção 7.6: explicando como configurar o módulo ***VirtualSensorNet***; g)Seção 7.7: abordando a remoção de recursos dos módulos ***SensorNet*** e ***VirtualSensorNet***; h)Seção 7.8: explicando brevemente como criar um módulo ***Function***; i)Seção 7.9: finalizando com uma lista de códigos já implementados para serem utilizados como referência de novas implementações.

7.1 TMON

O TMON [37] é um sistema de monitoramento termoenergético para centrais de processamentos de dados operando através de uma rede de sensores sem fio. A rede de sensores é formada por dispositivo wireless, alimentados por baterias, dotados de sensores para a coleta da temperatura e luminosidade ambiente. Os dados são roteados entre os próprios dispositivos até chegarem ao coletor, seguindo para a camada de exibição. A camada de exibição armazena os dados para posterior apresentação em interface Web. As informações apresentadas são sobre os dispositivos, com os valores sobre a temperatura, luminosidade e nível de bateria. Outras informações exibidas são a média da temperatura capturada pelos dispositivos e topologia da rede de sensores.

7.2 A implementação do TMON com o OSIRIS

Para um sistema de monitoramento centrado nos dispositivos, as informações do módulo ***SensorNet*** são suficientes, pois o ***SensorNet*** disponibiliza dados relacionados à rede de sensores e seus dispositivos, incluindo estados de funcionamento e valores sensoriais. Quando o foco do sistema são os dados sensoriais, deve ser utilizado o módulo ***VirtualSensorNet***. O ***VirtualSensorNet*** é capaz de desacoplar os dados sensoriais dos dispositivos físicos e ainda atuar, em conjunto com o módulo ***Function***, para realizar o processamento de dados.

O TMON tem foco nos sensores e exibe o estado de cada dispositivo em sua interface Web, com adicional exibição da média de temperatura e a topologia de rede. A implementação do TMON com o OSIRIS conta com um coletor e os módulos ***SensorNet***, ***VirtualSensorNet*** e ***Function***. O módulo ***SensorNet*** atua como fonte de dados dos dispositivos da rede de sensores, com informações sobre temperatura, luminosidade, bateria e a topologia de rede. O módulo ***VirtualSensorNet*** atua, em conjunto com um módulo ***Function***, para realizar o cálculo da média de temperatura. Depois de tratados pelo OSIRIS, os dados seguem para a camada de exibição do TMON, a qual será reutilizada.

7.3 Preparando a infraestrutura do framework

A atual implementação do OSIRIS depende do RabbitMQ, PostgreSQL e Java Runtime 7 ou superior. Todos estes softwares estão disponíveis em seus respectivos sites sem qualquer custo de aquisição. O RabbitMQ e o PostgreSQL possuem necessidades de configuração:

- **RabbitMQ:** definir um usuário e senha, e obter o número do IP da máquina na qual será instalado. Por exemplo, o equipamento com o RabbitMQ contém o IP `192.168.0.7`, com usuário `admin` e senha `admin`.

- **PostgreSQL:** necessita dos dados de IP, usuário e senha, além da criação de duas tabelas, uma para o módulo ***SensorNet*** e outra para o módulo ***VirtualSensorNet***. Nos exemplos que utilizaremos neste capítulo, criamos as tabelas `OsirisSN` para o ***SensorNet*** e `OsirisVSN` para o ***VirtualSensorNet***. O equipamento utilizado contém o endereço IP `192.168.0.7`, com nome de usuário `postgres` e senha `postgres` para o banco de dados.

Da parte do OSIRIS, a dependência se dá pela aquisição dos arquivos da API (`Osiris-xxxxxxxx.jar`) e dos binários (`SensorNet-xxxxxxxx.jar` e `VirtualSensorNet-xxxxxxxx.jar`) referentes aos módulos ***SensorNet*** e ***VirtualSensorNet***. A API é necessária para implementar os pontos de coleta e o consumo de dados, disponibilizada no formato `.jar`, pronta para ser inclusa como uma biblioteca em um projeto Java. Tanto a API quanto os binários podem ser obtidos neste endereço `https://github.com/labtempo/osiris-binaries`.

Cada módulo deve ser depositado em uma pasta dedicada junto com seu arquivo `config.properties`. O `config.properties` é o arquivo de configuração que armazena as informações para o acesso ao RabbitMQ e para o PostgreSQL, como mostrado no Código 7.1.

Código 7.1: Arquivo config.properties

```
postgres.server.db=OsirisSN
postgres.server.ip=192.168.0.7
postgres.server.port=5432
postgres.user.name=postgres
postgres.user.pass=postgres
rabbitmq.server.ip=192.168.0.7
rabbitmq.user.name=admin
rabbitmq.user.pass=admin
```

O exemplo contém informações relacionadas ao banco de dados, como IP, porta, usuário, senha e o nome do banco criado para o módulo ***SensorNet***. As informações relacionadas ao RabbitMQ são IP, usuário e senha. Depois de efetuadas as configurações, o módulo pode ser executado com o seguinte comando:

Código 7.2: Comando para executar os módulos SensorNet e VirtualSensorNet

```
java -jar {nome_do_modulo}.jar
```

Quando o módulo ***SensorNet*** ou ***VirtualSensorNet*** é inicializado, três grupos de mensagens são disponibilizados. São eles: `omcp://collector.messagegroup.osiris/` para dados coletados; `omcp://update.messagegroup.osiris/` para atualizações de recursos e `omcp://notification.messagegroup.osiris/` para mensagens de notificação.

7.4 Criando um módulo Collector

O módulo ***Collector*** é o responsável por adquirir os dados oriundos da rede de sensores. Em seguida, ele trata os dados adquiridos para o formato do framework, finalizando com a submissão dos dados formatados para o framework. Para a submissão, é necessário utilizar as classes `SampleCoTo`, `NetworkCoTo`, `CollectorCoTo` e `SensorCoTo`, as quais estão presentes na API, no pacote `br.uff.labtempo.osiris.to.collector`. O `SampleCoTo` é o objeto utilizado para submeter os dados ao framework, sendo enviado para o grupo de mensagens `omcp://collector.messagegroup.osiris/`.

7.4.1 Captura dos dados da rede de sensores

Cada rede de sensores tem suas particularidades na interface de comunicação. A questão mandatória é que a lógica de captura esteja encapsulada no módulo ***Collector***. No caso do TMON, a implementação do módulo contém uma interface para escutar a porta USB e receber os dados de monitoramento. A cada amostra sensorial adquirida da rede de sensores é realizada a criação do objeto `SensorCoTo`. O objeto `SensorCoTo` em conjunto com os objetos `NetworkCoTo` e `CollectorCoTo` são encapsulados no objeto `SampleCoTo`.

7.4.1.1 Dados dos dispositivos do TMON

A cada captura realizada por um dispositivo na rede de sensores sem fio gera uma tupla similar a esta `<5678, 2015-11-11 15:47:37.136511, 1, 25.229326, 237, 3.338598, 0, 10, iris>`, contendo nove tipos de informações, como mostrados na Tabela 7.1.

Tabela 7.1: Dados coletados de um dispositivo da rede de sensores do TMON

Nome	Valor	Descrição
`mote`	`45678`	Identificador do dispositivo
`time`	`2015-11-11 15:47:37.136511`	Momento de captura
`sendCount`	`1`	Número de envios totais do dispositivo
`readingTemperature`	`25.229.326`	Valor da temperatura em Celsius
`readingLight`	`237`	Valor da luminosidade
`readingVoltage`	`3.338.598`	Valor da tensão da bateria
`parent`	`0`	Identificador do dispositivo pai na rede
`metric`	`10`	Métrica de envio
`moteModel`	`iris`	Modelo do dispositivo

Os dados da tupla são recebidos pelo módulo ***Collector*** e convertidos para um objeto `SensorCoTo`. O intervalo de captura dos dispositivos é realizado no intervalo de seis segundos. Nas seções seguintes, as variáveis serão utilizadas para destacar a utilização no código.

7.4.1.2 Configurando o objeto SensorCoTo

A configuração do objeto `SensorCoTo` é iniciada pelo construtor, com a passagem do identificador do dispositivo(`mote`) e o momento de captura(`time`), como mostrado no Código 7.3.

Código 7.3: Criação do objeto SensorCoTo

```
SensorCoTo sensor = new SensorCoTo(mote, time);
```

Observe que o construtor aceita apenas timestamp no formato `long`), então o momento de captura (`time`) foi convertido para `long`. A segunda etapa é a adição dos valores de temperatura (`readingTemperature`), luminosidade (`readingLight`) e tensão da bateria (`readingVoltage`) para o objeto `SensorCoTo`. O Código 7.4 demonstra essa operação.

Código 7.4: Adição dos valores de sensoriamento para o objeto SensorCoTo

```
sensor.addValue("temperature", readingTemperature, "celsius", "C°");
sensor.addValue("light", readingLight, "candela", "C");
sensor.addValue("voltage", readingVoltage, "volt", "V");
```

Foram adicionados os nomes `temperature`, `light` e `voltage` para os valores, possibilitando uma identificação do valor quando o dado for consumido. Os quatro dados restantes (`sendCount`, `parent`, `metric` e `moteModel`) não possuem um local dedicado no objeto `SensorCoTo`. Por conta disso, eles são dispostos como informações extras do objeto. O Código 7.5 apresenta a adição de informações.

Código 7.5: Adição de informações para o objeto SensorCoTo

```
sensor.addInfo("sendCount", sendCount);
sensor.addInfo("parent", parent);
sensor.addInfo("metric", metric);
sensor.addInfo("moteModel", moteModel);
```

Do mesmo modo que os valores recebem nomes, as informações extras também compartilham da mesma abordagem para que os dados possam ser identificados no momento do consumo. Vale destacar que o identificador do dispositivo pai na rede(`parent`) é a

informação sobre o relacionamento entre os nós que possibilitará conhecer a topologia da rede de sensores sem fio do TMON.

7.4.1.3 Configurando recursos fungíveis para o objeto SensorCoTo

Para o TMON este recurso não foi utilizado, porém o nível de bateria dos dispositivos pode ser aplicado com um recurso fungível do objeto `SensorCoTo`. O Código 7.6 exemplifica a adição de um recurso fungível.

Código 7.6: Adição de recurso fungível para o objeto SensorCoTo

```
double minVoltage = 1.6;
double maxVoltage = 3.0;
double current = (readingVoltage - minVoltage);
double max = (maxVoltage - minVoltage);
int currentPercent = (int) (current * 100 / max);
if (currentPercent > 100) {
   currentPercent = 100;
}
sensor.addConsumable("battery", currentPercent);
```

Os valores para os níveis dos recursos fungíveis são repassados como porcentagens, portanto são valores inteiros de `0` a `100`, sendo `0` o menor valor e `100` o maior valor. Como o valor da tensão da bateria(`readingVoltage`) é um valor real, foi necessário realizar a conversão para porcentagem, sendo o valor máximo da tensão definido em 3,0v e o valor mínimo em 1,6v. O recurso fungível contém um nome para o vínculo com uma regra de verificação. O Código 7.7 exemplifica como adicionar uma regra para a verificação ao recurso fungível.

Código 7.7: Inclusão de regra para a verificação do recurso fungível ao objeto SensorCoTo

```
sensor.addConsumableRule("Bateria baixa", "battery", LogicalOperator.
   LESS_THAN, 60, "Bateria baixa!");
```

É possível criar várias regras para a verificação para um recurso fungível. No exemplo mostrado, criamos a regra para verificar se o nível da bateria está abaixo dos 60%. A regra contém um título, um nome de recurso fungível que será verificado, o operador lógico(Tabela 5.5 da Seção 5.1.1.3), o valor para verificação e uma mensagem. Quando o módulo ***SensorNet*** recebe esta informação ele verifica o nível do recurso fungível e envia um alerta com o conteúdo da mensagem definida.

7.4.1.4 Configurando o objeto CollectorCoTo

O segundo objeto é o `CollectorCoTo` que representa o módulo `Collector`. O `CollectorCoTo` recebe como parâmetros do seu construtor o identificador do módulo ***Collector*** e o intervalo de captura dos dispositivos na rede de sensores. O Código 7.8 exemplifica a criação do objeto `CollectorCoTo`.

Código 7.8: Criação do objeto CollectorCoTo

```
CollectorCoTo collector = new CollectorCoTo("labtempo", 6, TimeUnit.
   SECONDS);
collector.addInfo("descricao", "laboratorio");
```

Para o TMON, foi criado um módulo ***Collector*** com o nome `labtempo`, com o tempo de seis segundos como intervalo de captura dos dispositivos e a informação sobre onde especificamente o módulo ***Collector*** está alocado, que no caso é o laboratório do labtempo.

7.4.1.5 Configurando o objeto NetworkCoTo

A criação do objeto `NetworkCoTo` exige apenas a identificação da rede de sensores, pois o OSIRIS suporta diversas rede, como, também, diversos coletores. Podem coexistir diversas redes quando redes heterogêneas operam para um mesmo propósito ou estão separadas fisicamente. O Código 7.9 demonstra a criação do objeto `NetworkCoTo` para o TMON.

Código 7.9: Criação do objeto NetworkCoTo

```
NetworkCoTo network = new NetworkCoTo("tmon");
network.addInfo("domain", "br.uff.labtempo");
network.addInfo("type", "wireless");
network.addInfo("OS", "TinyOS");
```

O nome da rede foi denominado `tmon` e as informações sobre o domínio, o tipo de rede e o sistema operacional dos dispositivos foram adicionados como informações da rede, ou seja, informações do objeto `NetworkCoTo`.

7.4.2 Submissão de dados

Para o envio de dados é necessário utilizar o objeto `SampleCoTo`, que encapsula os objetos `NetworkCoTo`, `CollectorCoTo` e `SensorCoTo`. O Código 7.10 demonstra o processo.

Código 7.10: Criação do objeto SampleCoTo

```
SampleCoTo sample = new SampleCoTo(network, collector, sensor);
```

O envio do objeto `SampleCoTo` é realizado pelo componente OMCP Client. Existe um componente OMCP Client disponível através da classe `RabbitClient` no pacote `br.uff.labtempo.omcp.client.rabbitmq`. O Código 7.11 descreve como instanciar o OMCP Client baseado no RabbitMQ.

Código 7.11: Exemplo de instância do OMCP Client baseado no RabbitMQ

```
OmcpClient client = new RabbitClient("192.168.0.7", "admin", "admin");
```

O construtor do `RabbitClient` recebe as credenciais de acesso relacionadas ao RabbitMQ, IP, usuário e senha. O envio do objeto `SampleCoTo` para o grupo de mensagens `omcp://collector.messagegroup.osiris/` está descrito no Código 7.12.

Código 7.12: Realizando o envio de objetos via NOTIFY do OMCP Client

```
client.doNofity("omcp://collector.messagegroup.osiris/", sample);
```

7.5 Consumindo dados do SensorNet e VirtualSensorNet

O módulo ***SensorNet*** e o ***VirtualSensorNet*** disponibilizam dados de duas maneiras: requisições de consultas ou inscrição no grupo de mensagens `omcp://update.messagegroup.osiris/`.

7.5.1 Requisições

Consultas aos módulos ***SensorNet*** e ***VirtualSensorNet*** são realizadas via requisições utilizando método `doGet()` do OMCP Client. O retorno do método `doGet()` é um objeto `Response` encapsulando os dados de resposta. Apresentamos dois exemplos de requisição, sendo um de um único objeto e, o outro, de uma coleção de objetos (um vetor).

O Código 7.13 demonstra uma requisição: recuperar a informação da rede do TMON mapeada pelo módulo ***SensorNet***, via endereço `omcp://sensornet.osiris/tmon`.

Código 7.13: Requisição para consulta do recurso Network ao SensorNet

```
Response response = client.doGet("omcp://sensornet.osiris/tmon/");
if (response.getStatusCode() == StatusCode.OK) {
   NetworkSnTo network = response.getContent(NetworkSnTo.class);
```

```
    System.out.println(network.getId());
    for (Map.Entry<String, String> entrySet : info.entrySet()) {
        String chave = entrySet.getKey();
        String valor = entrySet.getValue();
        System.out.println(chave +": "+valor);
    }
}
```

A requisição GET retorna um objeto `Response` que, em caso de requisição bem sucedida, o armazena um objeto `NetworkSnTo`. O objeto `NetworkSnTo` é uma representação do recurso Network, presente no ***SensorNet***, que mapeia a rede no TMON. Através do método `getContent()`, o objeto `Response` possibilita a recuperação do objeto `NetworkSnTo`. O método exige um tipo de classe como parâmetro que, neste caso, é o objeto `NetworkSnTo.class` para desserializar corretamente os dados transmitidos em JSON para objeto Java. Depois de obtido o objeto `NetworkSnTo`, relativo à rede do TMON, é realizada a impressão dos dados do objeto, gerando o resultado a seguir(Código 7.14).

Código 7.14: Resultado da impressão dos dados do objeto NetworkSnTo

```
tmon
OS: TinyOS
domain: br.uff.labtempo
type: wireless
```

O Código 7.15 demonstra uma requisição para recuperar informação de todas as redes mapeadas pelo módulo ***SensorNet***.

Código 7.15: Requisição para a busca de todos os recursos Network do SensorNet

```
Response response = client.doGet("omcp://sensornet.osiris/");
if (response.getStatusCode() == StatusCode.OK) {
    NetworkSnTo[] objs = response.getContent(NetworkSnTo[].class);
    ...
}
```

É possível observar que a requisição recebe uma coleção de objetos `NetworkSnTo`, necessitando o tipo de objeto `NetworkSnTo[].class` para a desserialização correta.

7.5.2 Mensagens dos grupos de mensagens

O módulo ***SensorNet*** e o ***VirtualSensorNet*** enviam atualizações de seus recursos para o grupo de mensagens `omcp://update.messagegroup.osiris/` e publicam mensagens de

notificação para o grupo de mensagens `omcp://notification.messagegroup.osiris/`. A obtenção de informações dos grupos de mensagens é realizada através do OMCP Service. O OMCP Service é um serviço que tem a capacidade de se inscrever nos grupos de mensagens para obter informações publicadas no grupo. O Código 7.16 mostra a utilização do `RabbitService`, uma implementação do OMCP Service, disponível no pacote `br.uff.labtempo.omcp.client.rabbitmq`.

Código 7.16: Exemplo de instância do OMCP Service baseado no RabbitMQ

```
OmcpService omcpService = new RabbitService("192.168.0.7", "admin", "
   admin");
omcpService.addReference("omcp://update.messagegroup.osiris/#");
omcpService.setHandler((EventHandler)handler);
omcpService.start();
```

O `RabbitService` recebe, em seu construtor, IP, usuário e senha para acesso ao RabbitMQ. A inscrição no grupo de mensagens é realizada através do método `addReference()`, passando o grupo como parâmetro. O OMCP Service recebe um objeto que implementa a interface `EventHandler` para manipular as informações recebidas. A interface `EventHandler` está disponível no pacote `br.uff.labtempo.omcp.service`. O OMCP Service inicia as atividades quando o método `start()` é executado. Incluímos na Tabela 7.3, da Seção 7.9, os endereços para diversos exemplos de utilização da API, incluindo o OMCP Service.

7.6 Configurando o módulo VirtualSensorNet

O ***VirtualSensorNet*** é um módulo dependente de configuração. É necessário criar os recursos DataType e Function manualmente, como também os sensores virtuais Link, Composite e Blending. A criação é realizada por meio de requisições OMCP ao módulo e os objetos utilizados para esta finalidade estão disponíveis no pacote `br.uff.labtempo.osiris.to.virtualsensornet` da API.

7.6.1 Criando sensores virtuais do tipo Link

O Link é um recurso do ***VirtualSensorNet*** relacionado a um dispositivo da rede de sensores. Ele capta os valores de sensoriamento de um dispositivo e os incorpora como seus próprios valores. Para criar um Link é necessário obter os identificadores de um dispositivo (rede, coletor e sensor) e o identificador de um recurso DataType. O DataType

é um recurso do ***VirtualSensorNet*** que define um tipo de dado. Consulte a Seção 5.4.2.1 do Capítulo 5 para obter mais informações sobre o recurso Datatype. Para o Link, o DataType precisa estar relacionado com os dados do dispositivo escolhido para ocorrer a aquisição de valores sensoriais. A seguir, o Código 7.17 para a criação do DataType para os dados de temperatura do TMON.

Código 7.17: Criação do recurso DataType no VirtualSensorNet

```
String name = "temperature";
ValueType type = ValueType.NUMBER;
String unit = "celsius";
String symbol = "C°";
DataTypeVsnTo dataType = new DataTypeVsnTo(name, type, unit, symbol);
Response response = client.doPost("omcp://virtualsensornet.osiris/
   datatype/", dataType);
if (response.getStatusCode() == StatusCode.CREATED) {
   response = client.doGet(response.getLocation());
   dataType = response.getContent(DataTypeVsnTo.class);
   ...
}
```

O DataType é criado através de um objeto `DataTypeVsnTo`, enviado por uma requisição POST OMCP para o endereço `omcp://virtualsensornet.osiris/datatype/`. O objeto `DataTypeVsnTo` recebe um nome, um tipo de valor(Tabela 5.2 da Seção 5.1.1.1), uma unidade de medida e um símbolo relacionado à unidade. Em uma requisição POST sucedida, o endereço do novo recurso criado é enviado como reposta e pode ser obtido via método `getLocation()` do objeto Response. Este endereço possui um identificador único do novo recurso DataType. Depois de obtido o identificador, é possível criar um novo recurso Link no módulo ***VirtualSensorNet***, como descrito no Código 7.18.

Código 7.18: Criação do objeto LinkVsnTo

```
LinkVsnTo link = new LinkVsnTo("11", "labtempo", "tmon");
link.createField("temperature", dataType.getId());
```

O objeto `LinkVsnTo` recebe os identificadores de sensor, coletor e rede. Para TMON, escolhemos o dispositivo da rede de sensores de número "`11`". Os identificadores do coletor e rede são os definidos anteriormente no `NetworkCoTo` e no `CollectorCoTo`. O método `createField()` cria um Field para armazenar os valores sensoriais enviados pelo dispositivo, via `SampleCoTo`. Cada Field contém um nome e um DataType, sendo o DataType definido pelo identificador. O Link utiliza o recurso DataType para realizar

a triagem de valores que, no caso do TMON, são apenas os valores de temperatura na unidade `Celsius`. A criação do Link no módulo ***VirtualSensorNet*** se dá de modo similar ao processo utilizado para o DataType, por requisição POST, passando um objeto `LinkVsnTo`. A criação está descrita no Código 7.31.

Código 7.19: Requisição para criação do recurso Link no VirtualSensorNet

```
Response response = client.doPost("omcp://virtualsensornet.osiris/link/"
   , link);
if (r.getStatusCode() == StatusCode.CREATED) {
   response = client.doGet(r.getLocation());
   link = response.getContent(LinkVsnTo.class);
   ...
}
```

O Link, depois de criado, torna-se mais um sensor virtual disponível no endereço `omcp://virtualsensornet.osiris/vsensor/` e pode ser consultado através de uma requisição GET. O Código 7.31 demonstra a requisição.

Código 7.20: Requisição para consulta ao recurso Link no VirtualSensorNet

```
Response response = client.doGet("omcp://virtualsensornet.osiris/vsensor
   /");
if (r.getStatusCode() == StatusCode.OK) {
   VirtualSensorVsnTo[] virtualSensors = Response.getContent(
      VirtualSensorVsnTo[].class);
   ...
}
```

O valor de um Field criado para o Link pode ser acessado com o Código 7.21. Os dados são impressos na tela apenas para exemplificar a manipulação do objeto `ValueVsnTo`, que contém os valores sensoriais para o sensor virtual identificado pelo id `1`. A Tabela 5.32 da Seção 5.4.4 descreve a lista de recursos e operações para o módulo ***VirtualSensorNet***, descrevendo a regra de acesso aos recursos.

Código 7.21: Requisição para consulta ao recurso VSensor no VirtualSensorNet

```
Response response = client.doGet("omcp://virtualsensornet.osiris/vsensor
   /1");
if (r.getStatusCode() == StatusCode.OK) {
   VirtualSensorVsnTo virtualSensor = Response.getContent(
      VirtualSensorVsnTo.class);
   List<ValueVsnTo> values = virtualSensor.getValuesTo();
   for (ValueVsnTo value : values) {
```

```
        System.out.println("nome: "+ value.getName());
        System.out.println("valor: "+ value.getValue());
        System.out.println("unidade: "+ value.getUnit());
        System.out.println("simbolo: "+ value.getSymbol());
        System.out.println("identificador: "+ value.getId());
    }
}
```

O campo Value contém um nome, um valor, uma unidade, um símbolo e um identificador. O identificador está relacionado ao Field do Link, pois cada Field é único em si. O identificador encontra sua utilização nos casos de composição, com o Composite, ou na seleção de dados para processamento, com o Blending.

7.6.2 Criando sensores virtuais do tipo Composite

O Composite é o sensor virtual para compor dados de outros sensores virtuais, o qual se dá pelos Fields de outros sensores virtuais. Para criar um Composite é necessário o objeto `CompositeVsnTo` e os identificadores dos Fields a serem vinculados, Código 7.22.

Código 7.22: Criação do objeto CompositeVsnTo

```
CompositeVsnTo composite = new CompositeVsnTo();
composite.bindToField(1);
...
composite.bindToField(n);//n Fields
```

O método `bindToField()` recebe o identificador de um Field de outro sensor virtual para estabelecer um vínculo ao novo Composite. Na seção 7.6.1 foi explicado como obter o identificador do objeto `ValueVsnTo`. Este é o identificador do Field que deve ser passado para o método `bindToField()`. Como cada Field tem um identificador único dentre todos os virtuais sensores, apenas indicar o `Id` do Field já informa implicitamente a qual virtual sensor este Field pertence. Repare que no exemplo existe a possibilidade de vincular diversos Fields para um Composite. A criação do recurso Composite no módulo ***VirtualSensorNet*** é realizada por uma requisição POST, enviando um objeto `CompositeVsnTo`, de acordo com o Código 7.23.

Código 7.23: Requisição para criação do recurso Composite no VirtualSensorNet

```
Response response = client.doPost("omcp://virtualsensornet.osiris/
    composite/", composite);
if (r.getStatusCode() == StatusCode.CREATED) {
```

```
    response = client.doGet(r.getLocation());
    composite = response.getContent(CompositeVsnTo.class);
    ...
}
```

7.6.3 Criando sensores virtuais do tipo Blending

O Blending é o sensor virtual capacitado a realizar processamentos sobre um conjunto de valores oriundos de diversos outros sensores virtuais. O Blending é o responsável por definir os dados que serão processados e armazenar o resultado do processamento como seus próprios valores sensoriais. O processamento dos valores sensoriais é delegado, pelo Blending, para um módulo ***Function***. Para realizar o processo de criar um Blending com funcionalidade de processamento é necessário que haja uma etapa de criação do recurso no módulo ***VirtualSensorNet*** e a atualização do recurso para adicionar os parâmetros e a função. As duas próximas seções mostram como criar um sensor virtual Blending e como associar a este sensor uma função e seus parâmetros de cálculo, respectivamente.

7.6.3.1 Criação do recurso

A criação do recurso Blending no módulo ***VirtualSensorNet*** é similar à criação do Link, realizada com auxílio de um objeto `BlendingVsnTo`, como descrito no Código 7.24.

Código 7.24: Criação do objeto BlendingVsnTo

```
BlendingVsnTo blending = new BlendingVsnTo();
blending.createField("temperature", dataType.getId());
```

No TMON é preciso realizar o cálculo da média da temperatura e armazenar os resultados do processamento. O resultado é uma nova temperatura com unidade em Celsius. Por conta disso, foi definido "`temperature`" como nome do Field e um DataType para comportar temperatura em Celsius, o mesmo DataType utilizado no Link. O Blending é criado no módulo ***VirtualSensorNet*** através de uma requisição POST para o endereço `omcp://virtualsensornet.osiris/blending/`, como descrito no Código 7.25.

Código 7.25: Requisição para criação do recurso Blending no VirtualSensorNet

```
Response response = client.doPost("omcp://virtualsensornet.osiris/
    blending/", blending);
if (response.getStatusCode() == StatusCode.CREATED) {
    response = client.doGet(response.getLocation());
```

```
    blending = response.getContent(BlendingVsnTo.class);
    ...
}
```

Depois de criado o Blending é necessário escolher quais valores serão processados e qual módulo ***Function*** irá realizar este processamento (Veja como criar módulos ***Function*** na Seção 7.8).

7.6.3.2 Definição dos parâmetros de processamento

Antes de configurar o Blending, para realizar o processamento, é necessário criar um recurso Function no módulo ***VirtualSensorNet*** relacionado a um módulo ***Function***. A criação do recurso Function se dá pelo objeto `FunctionVsnTo`, o qual recebe em seu construtor um objeto `InterfaceFnTo`, disponível no pacote `br.uff.labtempo.osiris.to.function`. Por outro lado, o objeto `InterfaceFnTo` é obtido de um módulo ***Function*** em operação.

No TMON foi criado o módulo ***Function*** para o cálculo da média de temperatura, o `omcp://average.function.osiris/`. Para obter um objeto `InterfaceFnTo` deve ser realizada uma requisição GET para o recurso `/interface/` do módulo ***Function***, como mostrado no Código 7.26.

Código 7.26: Requisição para obter o objeto InterfaceFnTo do módulo Function

```
String address = "omcp://average.function.osiris/interface/";
Response response = client.doGet(address);
InterfaceFnTo interface;
if (response.getStatusCode() == StatusCode.OK) {
    interface = response.getContent(InterfaceFnTo.class);
    ...
}
```

O objeto `InterfaceFnTo`, obtido do módulo ***Function*** de cálculo da média de temperatura, contém os seguintes dados descritos na Tabela 7.2:

Os dados contém o nome do módulo ***Function***, a descrição, o nome do único parâmetro de requisição, o nome do único parâmetro para o retorno do resultado e o endereço OMCP do módulo. Alguns destes serão utilizados para a configuração do Blending, como informado mais adiante. Obtido o objeto `InterfaceFnTo`, o objeto `FunctionVsnTo` pode ser criado e enviado para criar um recurso Function no módulo ***VirtualSensorNet***. O Código 7.27 demonstra essa etapa.

Tabela 7.2: Dados do objeto InterfaceFnTo

Descrição	Valor
nome do módulo	`average.function`
descrição	`módulo para calcular a média de temperatura`
parâmetro de requisição	`temperatures`
parâmetro de resposta	`average`
endereço do módulo	`omcp://average.function.osiris/`

Código 7.27: Requisição para criação do recurso Function no VirtualSensorNet

```
FunctionVsnTo averageFunction = new FunctionVsnTo(interface);
response = client.doPost("omcp://virtualsensornet.osiris/function/",
   function);
if (response.getStatusCode() == StatusCode.CREATED) {
   response = client.doGet(response.getLocation());
   averageFunction = response.getContent(FunctionVsnTo.class);
   ...
}
```

A criação do recurso Function é efetuada por uma requisição POST, para o endereço `omcp://virtualsensornet.osiris/function/`. Após essa etapa, é possível definir a função para calcular a média da temperatura no objeto `BlendingVsnTo`. O Código 7.28 demonstra essa operação.

Código 7.28: Atualização do objeto BlendingVsnTo com dados do recurso Function

```
blending.setFunction(averageFunction);
blending.setCallMode(FunctionOperation.SYNCHRONOUS);
blending.setCallIntervalInMillis(6000);//6 segundos
```

O método `setFunction()` recebe o `FunctionVsnTo` que, neste caso, contém os dados do módulo ***Function*** de cálculo de média. Em seguida, é definido um modo de requisição de processamento para síncrono(a Tabela 5.18 define os modos) e o intervalo de tempo de requisições para seis segundos. Depois de definido os dados relacionados ao processamento de dados, é necessário definir os valores que serão processados, bem como o Field destinado para armazenar os resultados do processamento. O Código 7.29 demonstra a definição dos valores a serem processados.

Código 7.29: Requisição para buscar todos os recurso VSensor do VirtualSensorNet

```
Response response = client.doGet("omcp://virtualsensornet.osiris/vsensor
   /");
if (r.getStatusCode() == StatusCode.OK) {
```

```
    VirtualSensorVsnTo[] virtualSensors = Response.getContent(
        VirtualSensorVsnTo[].class);
    for (ValueVsnTo virtualSensor: virtualSensors) {
        List<ValueVsnTo> values = virtualSensor.getValuesTo();
        for (ValueVsnTo value : values) {
            if (value.getName().equals("temperature")) {
                blending.addRequestParam(value.getId(), "temperatures");
                break;
            }
        }
    }
}
```

Para exemplificação, foi utilizado um objeto `VSensorVsnTo` para obter o identificador do Field de temperatura de um sensor virtual, processo explicado na Seção 7.6.1. O método `addRequestParam()` recebe um `FieldId` e o nome do parâmetro de requisição, que neste caso é "`temperatures`", seguindo o nome que foi definido na Tabela 7.2 para o parâmetro de requisição. O Código 7.30 demonstra a escolha do Field para armazenar os valores de média de temperatura calculados.

Código 7.30: Atualização do objeto BlendingVsnTo para definir os parâmetros de resposta

```
List<? extends FieldTo> fields = blending.getFields();
FieldTo field =fields.get(0);
blending.addResponseParam(field.getId(),"average");
```

O método `addResponseParam()` recebe um identificador de um Field do próprio Blending e o nome do parâmetro de resposta, "`average`". Uma vez completa a configuração do objeto `BlendingVsnTo`, é necessário executar o processo de atualização do recurso no módulo ***VirtualSensorNet***, via requisição PUT, para o endereço recurso, enviando o objeto `BlendingVsnTo` modificado. O Código 7.31 demonstra essa operação.

Código 7.31: Requisição para atualização do recurso Blending no VirtualSensorNet

```
String address = "omcp://virtualsensornet/blending/" + blending.getId()
    + "/";
Response response = client.doPut(address, blending);
if (response.getStatusCode() == StatusCode.OK) {
    response = client.doGet(address);
    blending = response.getContent(BlendingVsnTo.class);
    ...
}
```

7.7 Removendo recursos do SensorNet e VirtualSensorNet

A remoção de recursos nos módulos ***SensorNet*** e ***VirtualSensorNet*** ocorre por meio de requisições DELETE OMCP.

Código 7.32: Exemplo de requisição para remoção de recursos no VirtualSensorNet

```
response = client.doDelete("omcp://virtualsensornet.osiris/link/1/");
if (response.getStatusCode() == StatusCode.OK) {
   System.out.println("DELETED: " + "omcp://virtualsensornet.osiris/link
      /1/");
}
```

O Código 7.32 apresenta uma requisição para a remoção do Link com o número de identificação 1. O processo remoção é o mesmo para todos os recursos do módulo ***SensorNet*** e ***VirtualSensorNet***, salvo quando um recurso a ser removido contém dependentes e não pode ser efetuada a operação.

7.8 Criando um módulo Function

O módulo ***Function*** processa um conjunto de valores enviados pelo Blending, do módulo *VirtualSensorNet*, e retorna o resultado do processamento para o requisitante. O módulo *Function* pode ser criado para diversos propósitos de processamento de dados, utilizando o componente OMCP Server para tratar as requisições de processamento e enviar as respostas. A implementação do OMCP Server, baseada no RabbitMQ, está disponível no pacote `br.uff.labtempo.omcp.server.rabbitmq`, com o nome `RabbitServer`. O Código 7.33 ilustra o uso do OMCP Server para construir o módulo ***Function***, que realiza o processamento de média de temperatura.

Código 7.33: Exemplo de instância do OMCP Server baseado no RabbitMQ

```
OmcpServer server = new RabbitServer("average.function", "192.168.0.7",
   "admin", "admin");
server.setHandler((RequestHandler)new Handler());
server.start();
```

O OMCP Server recebe o nome `average.function`, o qual será utilizado como endereço OMCP, sendo neste caso `omcp://average.function.osiris/`. O nome para a criação de módulos deve seguir a convenção descrita na Seção 4.4.1 do Capítulo 4. O

OMCP Server recebe um objeto do tipo `RequestHandler`, do pacote `br.uff.labtempo.omcp.server`, para manipular as requisições. A classe abstrata `Controller` implementa a interface `RequestHandler`, com adicional de uma estrutura pronta para tratar requisições através de ferramentas para verificação de URLs dos objetos `Request`, via métodos `match()` e `extractParams()`. A classe abstrata `Controller` está disponível no pacote `br.uff.labtempo.osiris.omcp` da API. O Código 7.34 a seguir ilustra o uso da classe `Controller`. Por falta de espaço, mostramos apenas como implementar a resposta ao GET na raiz. A implementação completa do controle para o módulo(average) ***Function***, do TMON, está disponível em `https://github.com/labtempo/osiris-tmon/blob/master/TmonAvgFunction/src/main/java/br/uff/labtempo/tmon/tmonavgfunction/controller/MainController.java`.

Código 7.34: Exemplo de implementação da classe abstrata Controller

```
public class Handler extends Controller {
  @Override
  public Response process(Request request) throws
      MethodNotAllowedException, NotFoundException,
      InternalServerErrorException, NotImplementedException,
      BadRequestException {
    String contentType = request.getContentType();
    if (match(request.getResource(), Path.RESOURCE_FUNCTION_REQUEST.
        toString())) {
      Map<String, String> urlParams = extractParams(request.getResource
          (),Path.RESOURCE_FUNCTION_REQUEST.toString());
      switch (request.getMethod()) {
        case GET:
          ParameterizedRequestFn prf = new ParameterizedRequestFn(
              urlParams);
          RequestFnTo requestFnTo= prf.getRequestFnTo();
          Double[] values = toArray(requestFnTo);
          ResponseFnTo responseFnTo= calculateAverage(values);
          Response response = new ResponseBuilder().ok(responseFnTo,
              contentType).build();
          return response;
        ...
      }
    }
  }
}
```

A extensão da classe `Controller` exige a implementação do método `process()` para poder receber os objetos `Request`. A lista completa de todos os recursos e operações OMCP de necessária implementação para módulo ***Function*** estão descritas na Tabela 5.22 da Seção 5.3.3. O módulo ***Function*** precisa disponibilizar a interface com seus parâmetros e os modos de operação.

A interface é transmitida via objeto ***InterfaceFnTo***, disponível no pacote `br.uff.labtempo.osiris.to.function` da API. No TMON, o módulo(average) ***Function*** contém os valores apresentados na Tabela 7.1. O Código 7.35 ilustra a criação do objeto `InterfaceFnTo`.

Código 7.35: Criação do objeto InterfaceFnTo

```
List<FunctionOperation> operations = new ArrayList<>();
operations.add(FunctionOperation.SYNCHRONOUS);

List<ParamFnTo> requestParam = new ArrayList<>();
requestParam.add(new ParamFnTo("temperatures", ValueType.NUMBER, true));

List<ParamFnTo> responseParam = new ArrayList<>();
responseParam.add(new ParamFnTo("average", ValueType.NUMBER));

InterfaceFnTo interface = new InterfaceFnTo("average.function", "modulo
   para calcular a media de temperatura", "omcp://average.function.
   osiris/", operations, requestParam, responseParam);
```

Repare que o objeto `InterfaceFnTo` recebe o nome do módulo, uma descrição, o endereço OMCP e três listas. A primeira lista contém as operações suportadas pelo módulo, que no caso do módulo(average) ***Function*** é apenas a operação síncrona. As duas listas restantes informam os parâmetros para a requisição e resposta.

7.9 Lista com endereços de códigos implementados

Estamos disponibilizando diversos códigos como forma de melhorar o entendimento apresentado nesse capítulo. A Tabela 7.3 apresenta a lista com diversos endereços.

Disponibilizamos códigos relativos aos módulos ***Collector***, ***SensorNet***, ***VirtualSensorNet***, ***Function*** e ***Service***. Para o módulo ***Collector*** estão disponíveis um exemplo e o código do coletor para o TMON. Para os módulos ***SensorNet*** e ***VirtualSensorNet*** disponibilizamos diversos exemplos com códigos para a manipulação de

Tabela 7.3: Lista de códigos disponíveis para a consulta

Módulo	Descrição	Endereço
Collector	Exemplo de implementação do coletor	`https://github.com/labtempo/osiris/tree/master/x-examples/Collector`
	Coletor implementado para o TMON	`https://github.com/labtempo/osiris-tmon/tree/master/TmonCollector`
SensorNet	Exemplo de consulta de revisões	`https://github.com/labtempo/osiris/tree/master/x-examples/ServerTest/src/main/java/client/sn`
VirtualSensorNet	Diversos exemplos de operações	`https://github.com/labtempo/osiris/tree/master/x-examples/ServerTest/src/main/java/client/vsn`
Function	Módulo Function para soma	`https://github.com/labtempo/osiris/tree/master/x-examples/Sum`
	Exemplo de consulta ao módulo Function de soma	`https://github.com/labtempo/osiris/tree/master/x-examples/ServerTest/src/main/java/function`
	Módulo Function para média do TMON	`https://github.com/labtempo/osiris-tmon/tree/master/TmonAvgFunction`
Service	Exemplos de consumo de grupos de mensagens	`https://github.com/labtempo/osiris/tree/master/x-examples/ServerTest/src/main/java/service`
	Implementação do Service para o TMON	`https://github.com/labtempo/osiris-tmon/tree/master/TmonManager`

recursos de ambos os módulos. Disponibilizamos dois módulos ***Function***: um para a soma e, o outro, para o cálculo de média, sendo este último utilizado no TMON. Como exemplos de utilização do módulo ***Service***, disponibilizamos código para o consumo de dados dos grupos de mensagens e o código da abordagem utilizada no TMON.

Capítulo 8

Experimentos

No Capítulo 7, foi verificada a eficácia do OSIRIS, com a implementação do TMON utilizando o framework. A prova de conceito demonstrou que a arquitetura proposta é capaz de resolver os problemas de **processamento de dados**, **a disponibilização de dados para múltiplas aplicações concorrentes** e **gerência da rede de sensores**.

Neste capítulo, apresentamos os experimentos realizados para avaliar a eficiência da proposta arquitetural do OSIRIS. Realizamos os experimentos sobre implementação em Java do OSIRIS, descrita no Capítulo 6. Os resultados obtidos com o experimento servirão como uma referência para comparações futuras, seja ela relacionada a uma melhoria na arquitetura, uma implementação utilizando uma linguagem diferente ou mesmo uma comparação entre trabalhos relacionados. Buscamos responder as seguintes perguntas sobre eficiência da arquitetura:

- Qual é a latência do framework?
- Qual é a vazão de dados por segundo?
- Como cresce o *overhead* do framework?
- Existe *overhead* de quantidade de coletores?
- Qual é o *overhead* de composição?
- Qual é o limite no número de composições?
- Qual é o número de composições extrapola a capacidade do framework?

Com estes experimentos, buscamos também verificar a resolução dos seguintes problemas: **a coleta distribuída**, devido a utilização de diversos módulos ***Collector*** para

enviar dados simultaneamente; **a disponibilização de dados para múltiplas aplicações concorrentes**, pelo envio de um dado sintético para ser persistido e disponibilizado pelo módulo ***VirtualSensorNet***; por fim, a capacidade de lidar com **redes de sensores com dispositivos heterogêneos**, pela utilização de um sensor virtual Link para obter os dados de sensoriamento.

O capítulo está organizado em: a)Seção 8.1: apresentando as métricas utilizadas na avaliação; b)Seção 8.2: descrevendo o ambiente de avaliação; c)Seção 8.3: finalizando com a apresentação do resultado dos experimentos.

8.1 Métricas de avaliação do framework

Foram utilizadas métricas de avaliação para medir o tempo em diversos pontos do "circuito" do framework. A Figura 8.1 ilustra o encadeamento de processos, considerando os pontos de A, B, C e D distribuídos sobre os módulos ***Collector***, ***VirtualSensorNet*** e ***Service***. O ponto A representa o envio da amostra para o framework. O ponto B representa a etapa de busca dos recursos existentes no banco de dados para serem atualizados com os dados da amostra sensorial recebida. O ponto C representa a operação de persistência de dados no banco de dados, em relação ao recurso atualizado. O ponto D é o destino final, ilustrando o ponto de recepção da representação do recurso atualizado pelo módulo ***Service***.

Na imagem, ainda é possível notar as métricas *SendToStoringLatency*, *StoringLatency*, *FetchingLatency*, *EndToEndLatency* e *StoringToReceiptLatency*. A métrica *SendToStoringLatency* está relacionada ao tempo entre o momento do envio da amostra sensorial até o instante antes da ação de persistência de dados. A métrica *FetchingLatency* é relacionada ao tempo de busca de um recuso existente no banco de dados. A métrica *StoringLatency* está relacionada ao tempo gasto para persistir a atualização do recurso no banco de dados. A métrica *EndToEndLatency* está relacionada ao intervalo entre o envio da amostra sensorial ao framework e ao momento da recepção da representação do recurso atualizado pelo módulo ***Service***. A métrica *StoringToReceiptLatency* está relacionada ao intervalo antes da persistência de dados até o momento da recepção da representação do recurso atualizado pelo módulo ***Service***. O somatório das métricas *SendToStoringLatency* e *StoringToReceiptLatency* é equivalente a métrica *EndToEndLatency*.

É relevante observar que a ligação entre o módulo ***Collector*** e o ***VirtualSensorNet*** é realizada pelo RabbitMQ, via publicação de amostras sensoriais para o grupo de

mensagens `omcp://collector.messagegroup.osiris/`. A ligação entre o módulo ***VirtualSensorNet*** e o ***Service*** se dá também pelo RabbitMQ, via o grupo de mensagens `omcp://update.messagegroup.osiris/`, grupo destinado a receber as atualizações dos recursos do ***SensorNet*** e ***VirtualSensorNet***. Ainda para que as amostras sensoriais enviadas gerassem reação no módulo ***Service***, foi criado o recurso VSensor-Link, com os identificadores relacionados a amostra sensorial e um campo(Field) para armazenar os valores sensoriais.

Para atribuir os valores temporais para as métricas, foi utilizado os campos do recurso VSensor, apresentado na Seção 5.4.2.4(Tabela 5.27) do Capítulo 5. Foram utilizados os campos `creationTimestampInMillis`, `storageTimestampInMillis`, `fetchingTimeInMillis` e `packingTimestampInMillis`, em conjunto com o instante do recebimento da representação do recurso pelo módulo ***Service***. Os campos `fetchingTimeInMillis` e `packingTimestampInMillis` foram adicionados ao `VirtualSensorVsnTo` exclusivamente por conta dos experimentos, para informar, respectivamente, o tempo gasto na busca do VSensor-Link no banco de dados e o momento da geração da representação do recurso VSensor, o objeto `VirtualSensorVsnTo`.

A métrica *EndToEndLatency* recebe a subtração do instante do recebimento da representação do recurso pelo campo `creationTimestampInMillis`. A métrica *StoringToReceiptLatency* recebe a subtração do instante do recebimento da representação do recurso pelo campo `storageTimestampInMillis`. A métrica *StoringLatency* recebe a subtração do campo `packingTimestampInMillis` pelo campo `storageTimestampInMillis`. A métrica *FetchingLatency* recebe o valor do campo `fetchingTimeInMillis`. A métrica *SendToStoringLatency* recebe a subtração do campo `storageTimestampInMillis` pelo campo `creationTimestampInMillis`.

8.2 Configuração do ambiente de experimentações

Para a realização dos experimentos foram utilizados dois computadores, que chamaremos de `Oraculo` e `Phenom` daqui para frente.

O `Oraculo` é um equipamento configurado com um processador Intel i7 2600k, placa mãe Z68X-UD4-B3 e com 16GB de memória. O `Oraculo` opera com Windows 10 Pro e foi o responsável por alocar o módulo ***VirtualSensorNet***.

O `Phenom` é um equipamento com um processador AMD Phenom x4 9750, placa mãe GA-MA770-S3, com 8GB de memória e um disco SATA II(Seagate Barracuda ST3250820AS

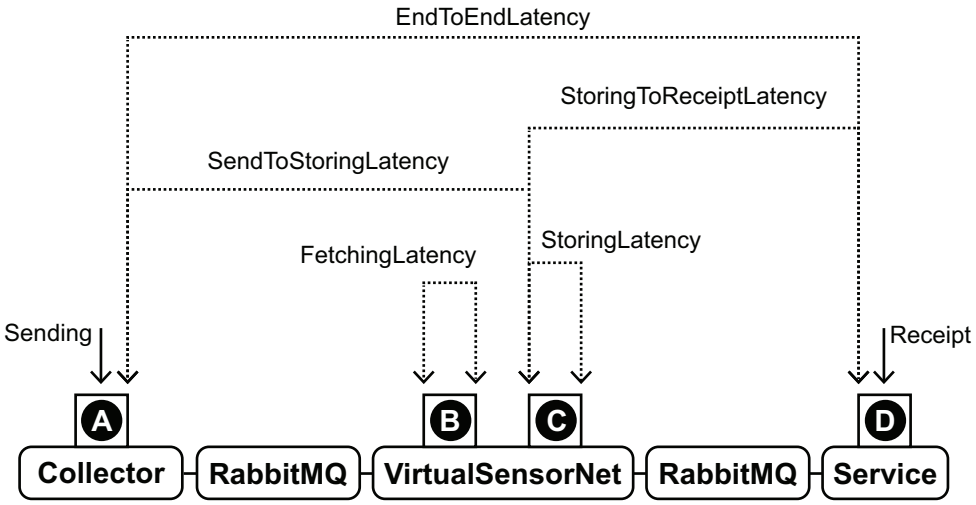

Figura 8.1: Circuito do OSIRIS utilizado nos experimentos

250GB 7200 RPM 8MB Cache SATA 3.0Gb/s). Configurado com o Windows 7, ficou encarregado de executar o RabbitMQ versão 3.5.4 e o PostgreSQL versão 9.4.4.

A conexão entre as duas máquinas foi realizada por uma ligação direta, de placa de rede a placa de rede, sendo o Oraculo e o Phenom possuidores de placas Gigabit, modelo Realtek RTL8111E chip (10/100/1000 Mbit) [28].

Utilizamos a suíte de testes TLaucher, disponível em `https://github.com/labtempo/osiris/tree/master/x-experiments/TLauncher`, criada para realizar os experimentos. Utilizamos dados sintéticos para o experimento, definindo a tupla do dado Value, Seção 5.1.1.1 no Capítulo 5, com os seguintes dados: `<“message”,“number”,“1”,“identifier”,“ID”>`, onde o valor `“1”` representa o identificador da mensagem que, em alguns casos, é incrementado para diferenciar os envios.

8.3 Resultados

Os resultados apresentados nesta seção foram obtidos com os seguintes experimentos: Latência de uma única transmissão (Seção 8.3.1), medindo o tempo de transmissão para um único envio de amostra sensorial; latência para transmissões simultâneas (Seção 8.3.2), avaliando o tempo transmissão para envio simultâneo de amostras sensoriais; latência para transmissões simultâneas com conexão dedicada (Seção 8.3.3), avaliando o tempo de transmissão para envio simultâneo de amostras sensoriais com o emprego de uma conexão com RabbitMQ, dedicada para cada envio de amostra sensorial; comportamento da composição de sensores virtuais (Seção 8.3.4), observando o custo na composição de sensores virtuais em relação a latência de propagação da informação com recursos Composite

aninhados; por fim, a latência para envio de amostras sensoriais por múltiplos módulos ***Collector***(Seção 8.3.5), avaliando a capacidade dos módulos de trabalharem distribuidamente e a latência de transmissão na abordagem distribuída.

8.3.1 Latência para uma transmissão

O experimento inicial consistiu em transmitir uma única amostra sensorial para observar a latência existente no framework. O experimento foi repetido 100 vezes, sem envios simultâneos, com um intervalo de dois segundos entre cada envio. A Figura 8.2 apresenta o gráfico *boxplot* com o resultado do experimento.

O boxplot (gráfico de caixa) é um gráfico utilizado para avaliar a distribuição empírica do dados. O boxplot é formado pelo primeiro e terceiro quartil e pela mediana. As hastes inferiores e superiores se estendem, respectivamente, do quartil inferior até o menor valor não inferior ao limite inferior e do quartil superior até o maior valor não superior ao limite superior. Os pontos fora destes limites são considerados valores discrepantes (outliers) e são denotados por um círculo ($\circ$). Os limites são calculados da forma abaixo:

Limite inferior: $\max\{\min(\text{dados}); Q_1 - 1,5(Q_3 - Q_1)\}$.

Limite superior: $\min\{\max(\text{dados}); Q_3 + 1,5(Q_3 - Q_1)\}$.

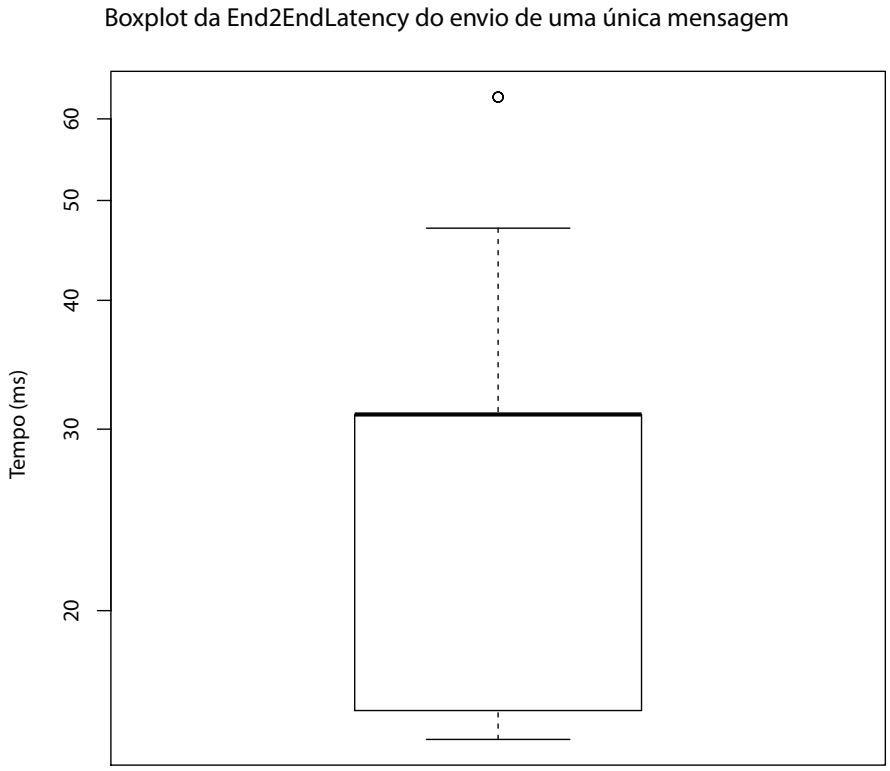

Figura 8.2: Resultado para o experimento de medir o tempo para uma transmissão

O gráfico da Figura 8.2, com o eixo y plotado em escala logarítmica, apresenta a mediana da latência na casa dos 30 milissegundos. Aproximadamente 30% dos valores da métrica *EndToEndLatency* se encontram abaixo dos 20 milissegundos, e 75% dos valores são menores que 35 milissegundos. O maior tempo *EndToEndLatency* ficou acima dos 60ms, mas é um caso isolado.

8.3.2 Latência para transmissões simultâneas

O experimento consistiu em realizar o envio simultâneo de amostras sensoriais. A taxa de envio iniciou-se em 10 amostras sensoriais, ampliando esse valor para 100 e, posteriormente, para 300 amostras enviadas. Todos os envios foram realizados por uma única conexão com o RabbitMQ. Para fins de análise estatística, o experimento foi repetido 100 vezes, com intervalo de dois segundos entre cada experimento, para os valores de 10, 100 e 300, totalizando 300 experimentos com 41.000 transmissões fim a fim. Não aumentamos mais o número de mensagens simultâneas sendo enviadas porque o valor da métrica *EndToEndLatency* excedeu um segundo. A Figura 8.3 apresenta o gráfico barplot da latência mínima, máxima e média para o envio simultâneo de 10, 100 e 300 amostras sensoriais. A barra da latência média apresenta o intervalo de confiança de 95%.

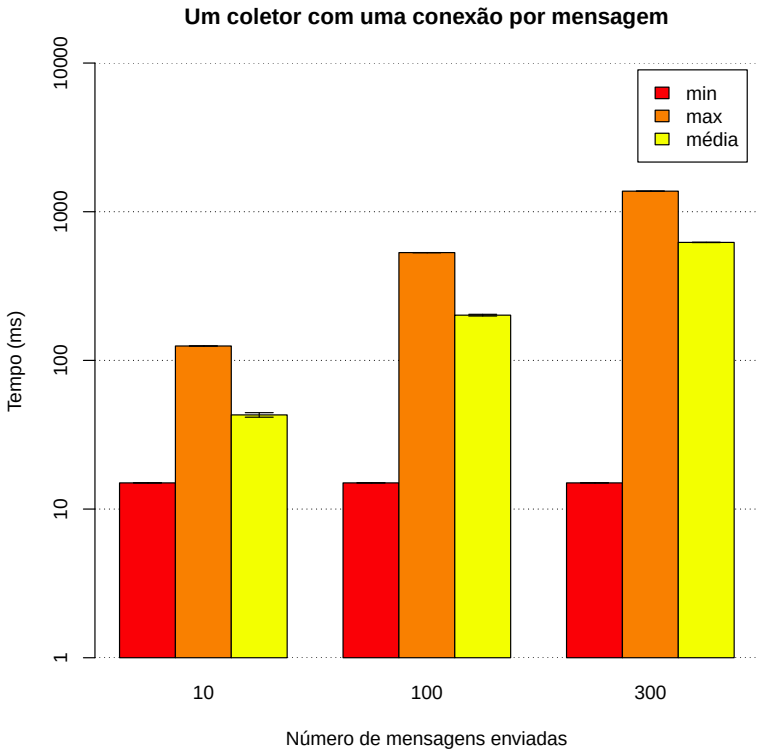

Figura 8.3: Resultado para o experimento de transmissões simultâneas

É possível observar no gráfico, com o eixo y plotado em escala logarítmica, um aumento do *EndToEndLatency* médio e máximo em relação ao aumento no envio de amostras, sendo que, com 300 amostras, o tempo máximo ultrapassou um segundo. Já o *EndToEndLatency* mínimo se manteve próximo a 10 milissegundos em todos os casos. Utilizando uma conexão para realizar transmissões, o experimento aponta a vazão do framework em torno de 300 mensagens por segundo, tanto no pior caso quanto no caso médio.

8.3.3 Latência para transmissões simultâneas com conexão dedicada

Este experimento consistiu também em realizar o envio simultâneo de amostras sensoriais, porém diferindo do experimento de latência para transmissões simultâneas, Seção 8.3.2, por utilizar uma conexão com o RabbitMQ dedicada para cada envio de amostra sensorial. Foram realizados envios em diferentes quantidades de amostras sensoriais, iniciando com o envio 10, passando pela quantidade 100, 300, 1000 e encerrando com o envio de 10.000 amostras. Cada experimento foi repetido 100 vezes, com dois segundos de intervalo entre cada experimento, gerando um total de 500 experimentos com 1.141.000 transmissões fim a fim, ou seja, 1.141.000 amostras sensoriais enviadas e representações do recurso recebidos. O critério de parada foi também o valor da métrica *EndToEndLatency* atingir um segundo. A Figura 8.4apresenta o gráfico barplot da latência mínima, máxima e média para o envio simultâneo de 10, 100, 300, 1.000 e 10.000 amostras sensoriais, utilizando a abordagem de conexão dedicada para cada envio. A barra da latência média apresenta o intervalo de confiança de 95%.

Para o experimento com conexões dedicadas, é possível observar, no gráfico, com o eixo y plotado em escala logarítmica, que o valores médios apresentados pela métrica *EndToEndLatency* foram próximos aos valores mínimos da mesma métrica apresentados no experimento anterior. Curiosamente, os valores mínimos deste experimento foram 0 para o envio de 100, 300, 1.000 e 10.000 amostras sensoriais. Não sabemos ao certo o motivo de ocorrer valores 0 para a métrica *EndToEndLatency*, pois os registros foram persistidos corretamente, inclusive com o valor sensorial correto, apenas com variações de tempo na casa de nano segundos. Este comportamento anormal é incoerente com o resultado apresentado no primeiro experimento, pois o teste de latência não apresentou valores 0.

Este experimento mostrou um ganho médio significativo no envio de amostras sensori-

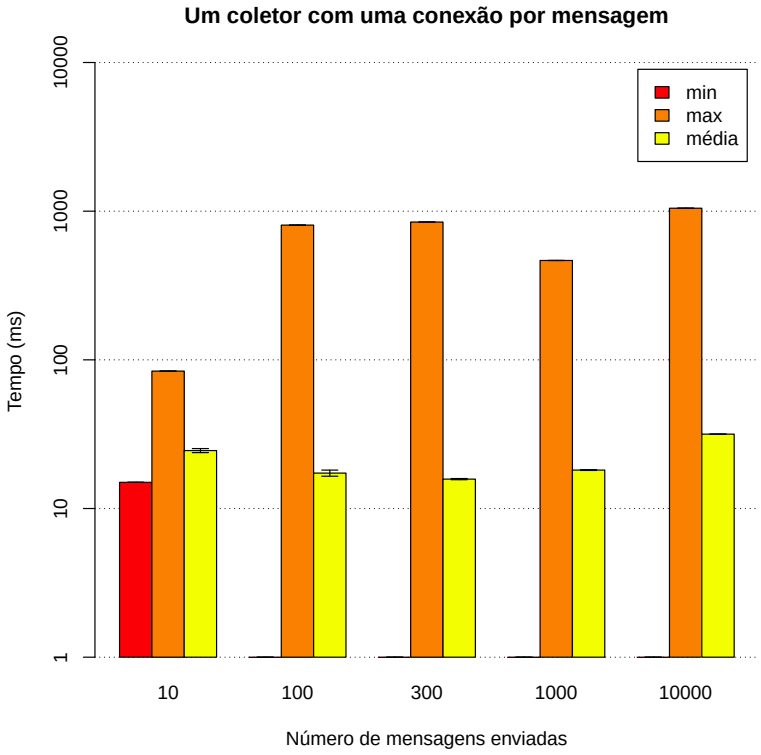

Figura 8.4: Resultado para o experimento de transmissões simultâneas, com conexão dedicada

ais, pois mesmo enviando 10.000 amostras, o valor médio da métrica *EndToEndLatency* ficou equiparado ao valor mínimo da mesma métrica do segundo experimento(Seção 8.3.1). Portanto, concluímos que utilizar conexões dedicadas pode gerar um aumento significativo na quantidade de amostras enviadas em relação a enviar todas as amostras por uma única conexão, o que pode ser aplicado na construção de módulos ***Collector*** para aumentar a capacidade no envio de amostras. Ainda neste experimento é possível responder a pergunta sobre a vazão do framework, a qual é apontada na avaliação na casa de 10.000 mensagens por segundo, isto no pior caso.

Este resultado demonstra que a arquitetura de OSIRIS suporta redes de sensores com taxa transferência de até 10.000 amostras por segundo. Considerando aplicações comerciais de rede de sensores sem fio, que tendem a usar frequência de amostragem da ordem de alguns minutos para reduzir tráfego de dados e, portanto, economizar energia [22, 33, 37], este rendimento é suficiente para atender redes de sensores sem fio com mais de um milhão de nós.

8.3.4 Comportamento da composição de sensores virtuais

Para este experimento enviamos uma amostra sensorial e verificamos a diferença entre o tempo de envio e o tempo do recebimento da representação do recurso VSensor, sendo este relacionado ao recurso VSensor-Composite.

Cabe lembrar que a implementação do OSIRIS sob experimentação contém um recurso VSensor-Composite operando em função das atualizações recebidas dos seus recursos VSensor agregados. Esta configuração resulta em um recurso VSensor-Composite difundindo atualizações dos recursos agregados como atualizações próprias, visto que, indiretamente, seus valores são modificados.

Para configurar a composição, foi criado um recurso VSensor-Link, mesmo VSensor-Link utilizado nos outros experimentos. O VSensor-Link foi criado com um único campo sensorial, *Field*. Em seguida, 100 unidades do VSensor-Composite foram criadas com referência para o *Field* do VSensor-Link. Efetuada a configuração, para cada amostra sensorial enviada com correspondência para o VSensor-Link, 101 representações de recurso seriam recebidas no módulo ***Service***, sendo uma representação para o VSensor-Link e 100 para os VSensor-Composite.

O experimento contou com o envio de 100 amostras sensoriais, ocasionando o recebimento de 10.100 representações de recurso pelo módulo ***Service***. A Figura 8.5 apresenta o gráfico com o resultado da latência mínima, máxima e média para cada nível de aninhamento do VSensor-Composite.

O experimento para avaliar a composição de sensores virtuais indica que o recurso de composição está suscetível a atrasos à medida que o aninhamento entre os VSensor-Composite aumenta. A média da métrica *EndToEndLatency* é crescente, variando entre 10 a 400ms no tempo. Como a latência média está próxima da mínima, concluímos que os valores máximos são casos isolados. O VSensor-Composite é um recurso útil, porém deve ser utilizado com cautela para não onerar sistema de monitoramento com atrasos desnecessários, como aponta o experimento.

8.3.5 Latência para envio de amostras sensoriais por múltiplos módulos Collector

Neste experimento, avaliamos a capacidade do módulo ***Collector*** operar distribuidamente. Utilizamos um módulo ***Collector*** implementado apenas para enviar uma mensagem e encerrar sua operação. Realizamos o experimento com 10, 20, 30, 40 e 50 instâncias

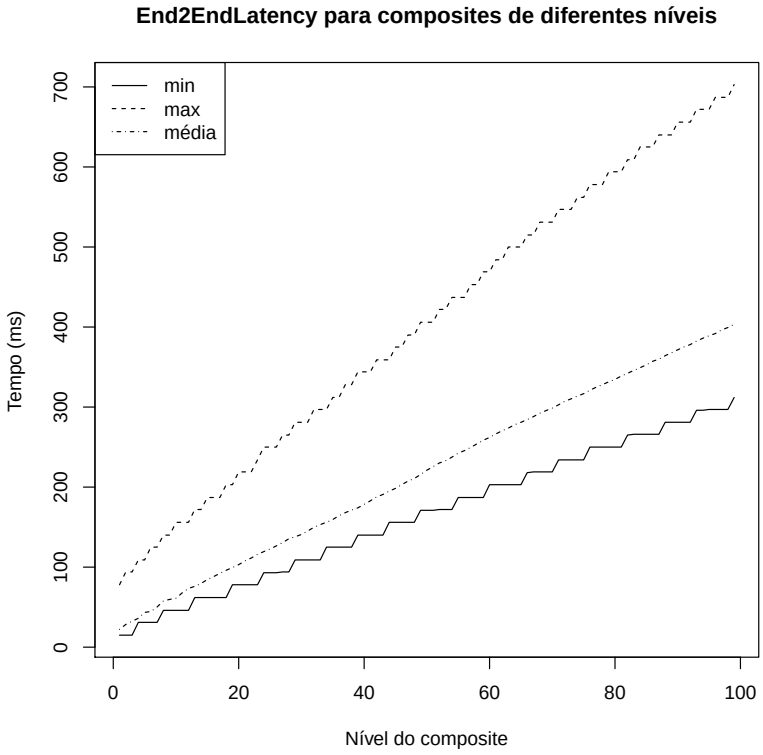

Figura 8.5: Resultado para o experimento de composição de sensores virtuais

do módulo executando concorrentemente. Para cada quantidade de instâncias do módulo foram realizadas 100 repetições, gerando um total de 500 experimentos, com 5.100 envios de amostras sensoriais. A Figura 8.6 apresenta o gráfico barplot da latência mínima, máxima e média para 10, 20, 30, 40 e 50 instâncias do módulo ***Collector***. A barra da latência média apresenta o intervalo de confiança de 95%.

A aparência do gráfico apresentado contém similaridades com o gráfico do experimento da Seção 8.3.2, o segundo experimento, com o comportamento dos valores mínimos estáveis e os valores médios e máximos em crescimento constante. Ainda no gráfico, plotado com o eixo y em escala logarítmica, é mostrado um aumento considerável nos valores *EndToEndLatency* deste experimento para os outros, pois o menor tempo de envio está em aproximadamente 100ms. Acreditamos que este acréscimo aproximado de 70ms, considerando a média da latência do experimento da Seção 8.3.1, seja por conta de iniciar o módulo executando-o diretamente pelo *prompt* de comando, porém, para conclusões definitivas, cabe aqui maiores investigações. O experimento com 30 módulos ***Collector*** apresenta o valor de transmissão em torno de um segundo. Analisando o resultado por outra perspectiva, o experimento demonstra que o módulo ***Collector*** é distribuível, comprovando nossas expectativas arquiteturais.

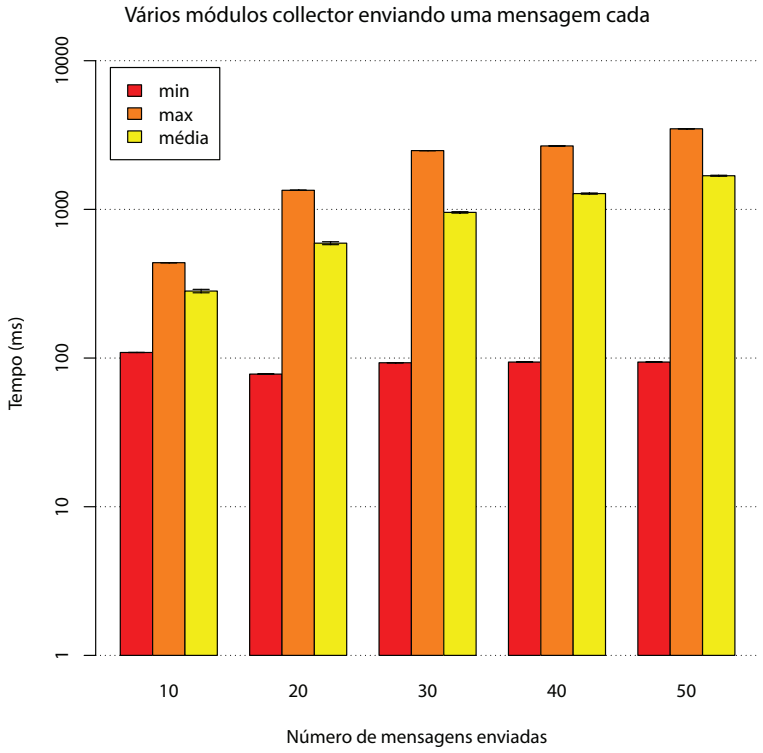

Figura 8.6: Resultado para o experimento de vários módulos Collector

Capítulo 9

Conclusão e Trabalhos Futuros

Este capítulo apresenta as considerações finais e as principais contribuições desta dissertação, assim como trabalhos futuros.

9.1 Considerações finais

Este trabalho teve início quando surgiram ideias de dar continuidade ao desenvolvimento do TMON [37], realizando melhorias na apresentação dos dados sensoriais para o usuário. Foram vislumbradas possibilidades de aplicação para o TMON além do sensoriamento térmico. A partir deste ponto, nos deparamos com o problema de substituir os sensores termo-luminosos por outros, porém, para isto ocorrer, modificações deveriam ser realizadas na camada de apresentação, no armazenamento de dados e na camada aquisição. Diante de tais fatos, chegamos a um entendimento de que seria proveitoso existir uma solução generalista para construir sistemas de monitoramento baseados em redes de sensores. Iniciamos a busca para definir uma arquitetura de base com desejo de obter um software capaz de suportar a adição de novos tipos de sensores, aplicados por meio do processo de extensão e customização, e apto para ser utilizado além do monitoramento térmico. Na etapa de concepção da arquitetura, nos deparamos com o problema da falta de padronização dos dados sensoriais, pois cada rede de sensores utiliza um formato de dados que mais lhe convém. Decidimos que o framework utilizaria um formato de dados padrão, com a unidade e o simbolo definidos no módulo ***Collector***. Uma vez definidos símbolo e unidade do dado, o dado tornou-se pronto para ser exibido na camada de apresentação.

Após esse período, chegamos ao problema da camada de comunicação, onde os módulos deveriam ser capazes de enviar mensagens síncronas e assíncronas, para realizar

consultas de histórico de valores e difundir informações para um ou muitos módulos destinatários. A ideia da comunicação acoplada para uma linguagem de programação não nos agradava, então buscamos um protocolo menos restritivo, desenvolvendo o OMCP. Definida a comunicação, desenvolvemos o módulo ***SensorNet*** para monitorar a rede de sensores, que tem características similares ao módulo existente no TMON.

Não queríamos que uma aplicação permanecesse em constante modificação para acomodar mudanças ocorridas pela troca de dispositivos em uma rede de sensores. Com isso, buscamos a solução de abstrair o hardware, aproveitando apenas os dados de sensoriamento. Inspirados pela arquitetura do Android [12], que atua como um framework que abstrai os detalhes do hardware para oferecer uma interface única para a utilização, criamos o conceito do sensor virtual. A ideia do sensor virtual é trabalhar apenas com dados de sensoriamento, abstraindo detalhes inerentes aos dispositivos presentes na rede de sensores. Uma vez concebido o sensor virtual, idealizamos especializações para compor e processar dados de sensoriamento de outros sensores virtuais, consolidando a arquitetura para o módulo ***VirtualSensorNet*** e o ***Function***. Por fim, devido à demanda de consumo de mensagens assíncronas, idealizamos o módulo ***Service***.

9.2 Contribuições

Neste trabalho apresentamos OSIRIS, um framework para a construção de sistemas de monitoramento baseado em redes de sensores, capaz de lidar os problemas apresentados na Seção 1.2 do Capítulo 1, da coleta de dados distribuída, disponibilização de dados para múltiplas aplicações concorrentes, processamento de dados, redes de sensores com dispositivos heterogêneos e gerência da rede de sensores.

Para resolver o problema da coleta distribuída, aplicamos uma arquitetura com suporte para inúmeros módulos ***Collector*** simultâneos, possibilitando implantar múltiplos pontos de coleta em uma única rede de sensores. Os experimentos apresentados no Capítulo 8 constataram que a abordagem arquitetural proposta é capaz de lidar com até 10.000 amostras por segundo.

Para resolver o problema da disponibilização de dados para múltiplas aplicações concorrentes, utilizamos os módulos ***SensorNet*** e ***VirtualSensorNet*** com o armazenamento dos dados de sensoriamento, evitando consulta para a rede de sensores. Utilizamos também um formato de dado padrão, comportando um valor, uma unidade e um símbolo para a representação do dado em qualquer camada de apresentação. Para interoperar

com outros sistemas, desenvolvemos o protocolo OMCP para regulamentar a troca de mensagens utilizadas na comunicação.

Já para resolver o problema do processamentos de dados, desenvolvemos a abordagem dos sensores virtuais para processar dados de sensoriamento, realizando a combinação ou a fusão dos dados de sensoriamento. Para processar dados de sensoriamento, o sensor virtual opera em conjunto com o módulo ***Function***, o qual pode ser construído para as mais diversas finalidades, incluindo seleção de valores para aplicar a tolerância à falha.

Para lidar com o problema das redes de sensores com dispositivos heterogêneos, utilizamos do formato de dados padronizado e a abstração sensor virtual para possibilitar a independência hardware. Com o sensor virtual, a fonte do dado torna-se oculta, possibilitando alterar uma fonte(no sensor virtual) sem impacto para a aplicação. Utilizando módulos ***Collector*** dedicados para cada rede existente e o módulo ***SensorNet***, lidamos com o problema de utilizar diversas redes de sensores ao mesmo tempo.

Por último, para resolver o problema da gerência da rede de sensores, utilizamos uma representação em software do estado atual da rede física e seus sensores: o módulo ***SensorNet***. O módulo ***SensorNet*** realiza a notificação sobre o estado e os níveis dos recursos fungíveis(bateria, combustível, entre outros) dos dispositivos existentes na rede de sensores, sendo capaz de identificar falhas do ponto de coleta, bem como falhas em toda rede de sensores. Com o conhecimento do nó com defeito e a causa variando entre os recursos fungíveis e outros problemas, possibilitamos uma certa precisão para a manutenção da rede de sensores.

Em adicional, consideramos a criação do OMCP como contribuição, pois é um protocolo de comunicação que regulamenta a troca de mensagens entre os módulos do OSIRIS, realizando comunicação síncrona e assíncrona. O OMCP aplica o estilo REST e trabalha com o publish-subscribe, possuindo um baixo acoplamento na comunicação, características que podem ser úteis para outros trabalhos.

9.3 Trabalhos futuros

O trabalho proposto aqui possui diversas funcionalidades, que foram desenvolvidas para que se chegasse ao resultado de um framework funcional. Entretanto, a forma como estão implementadas não é a mais eficiente, carecendo de mais observações a respeito das demandas resultantes. Por outro lado, surgiram novas necessidades como a composição de sensores virtuais eficiente. Todas as funcionalidades identificadas estão descritas nesta

seção para que, futuramente, sejam estudas e possam ser implementadas, resultando em uma solução mais robusta, ampliando seu escopo de atuação.

9.3.1 Módulos distribuídos

Atualmente os módulos ***SensorNet*** e ***VirtualSensorNet*** trabalham de forma centralizada, agregando todas as informações de todos os ***Collectors*** e atendendo à todas as aplicações. Espera-se, no futuro, evoluir a arquitetura do OSIRIS, para descentralizar o ***SensorNet*** e o ***VirtualSensorNet***, partilhando dados sensoriais entre diferentes instâncias, a fim de atender um raio maior de monitoramento.

9.3.2 Camada de comunicação

A camada de comunicação utilizada hoje é funcional e atende aos requisitos de comunicação do OSIRIS. Um bom campo de estudo para o futuro é tornar a comunicação mais eficiente, realizando uma análise minuciosa sobre seu funcionamento. Outro ponto sobre a camada de comunicação é retornar para a comunidade a implementação do REST sobre o RabbitMQ, pois antes do desenvolvimento do trabalho, foi pesquisado sobre a existência deste recurso, mas nada foi encontrado.

9.3.3 Composição de sensores virtuais mais eficiente

A composição de sensores virtuais tem um custo considerável e é uma abordagem para ser utilizada com cautela. Atualmente, o nosso interesse é apenas mostrar que a composição de dados é um conceito funcional, digno de atenção entre a comunidade. Fica como sugestão avaliar o conceito de composição de sensores virtuais para criar abordagens mais eficientes de serem utilizadas.

9.3.4 Armazenamento das revisões mais eficiente

As revisões são recursos necessários para a aquisição do histórico de sensoriamento e análise de diversos parâmetros em um ambiente de monitoramento. Temos em mente que o armazenamento das revisões pode ser melhorado para consumir menos espaço, com persistência apenas dos valores que sofrem alterações.

9.3.5 Criação de uma interface de administração

Um trabalho de extrema importância para o OSIRIS é um painel para gerenciar o framework. Como apresentado neste trabalho, alguns módulos disponibilizam recursos e operações OMCP para interagir com o mundo exterior, seja configurando o módulo, seja visualizando dados de sensoriamento. Vislumbramos para o futuro a criação de um painel administrativo, sendo este um módulo com uma interface amigável para operar, via operações OMCP, os módulos ***SensorNet*** , ***Function*** e ***VirtualSensorNet***. As operações podem abranger a criação de recursos VSensor-Links, remoção de recurso e a visualização dos níveis dos recursos fungíveis, bem como dados de sensoriamento.

9.3.6 Criação da API em outras linguagens

A especificação do framework foi desenvolvida para trabalhar com módulos em diferentes linguagens. Neste trabalho, utilizamos a linguagem Java, mas, para o futuro, desejamos ampliar o suporte para Python, C# e outras mais que forem necessárias. Por conta disso, é mister implementar as definições do OSIRIS para APIs em diferentes linguagens.

Referências

[1] Alur, D.; Malks, D.; Crupi, J.; Booch, G.; Fowler, M. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*, 2 ed. Sun Microsystems, Inc., Mountain View, CA, USA, 2003.

[2] Arduino. Arduino - home. `https://www.arduino.cc/`, 2015. Accessed: 2015-08-19.

[3] BeagleBoard. Beagleboard.org - bone. `http://beagleboard.org/bone`, 2015. Accessed: 2015-08-19.

[4] Crockford, D. The application/json media type for javascript object notation (json). RFC 4627, IETF, 7 2006.

[5] Di Francesco, M.; Das, S. K.; Anastasi, G. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Trans. Sen. Netw. 8*, 1 (Aug. 2011), 7:1–7:31.

[6] Durrant-Whyte, H.; Henderson, T. Multisensor data fusion. In *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer Berlin Heidelberg, 2008, pp. 585–610.

[7] Erl, T. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[8] Eugster, P. T.; Felber, P. A.; Guerraoui, R.; Kermarrec, A.-M. The many faces of publish/subscribe. *ACM Comput. Surv. 35*, 2 (June 2003), 114–131.

[9] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. Hypertext transfer protocol – http/1.1, 1999.

[10] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese de Doutorado, 2000. AAI9980887.

[11] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[12] Google. Android. `https://www.android.com/`, 2015. Accessed: 2015-08-21.

[13] Hibernate. Hibernate orm. `http://hibernate.org/orm/`, 2015. Accessed: 2015-08-19.

[14] JSON. Json. `http://json.org/`, 2015. Accessed: 2015-08-19.

[15] Kabadayi, S.; Pridgen, A.; Julien, C. Virtual sensors: Abstracting data from physical sensors. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks* (Washington, DC, USA, 2006), WOWMOM '06, IEEE Computer Society, pp. 587–592.

[16] Kabadayi, S.; Pridgen, A.; Julien, C. Virtual sensors: abstracting data from physical sensors. In *World of Wireless, Mobile and Multimedia Networks, 2006. WoWMoM 2006. International Symposium on a* (2006), pp. 6 pp.–592.

[17] Kansal, A.; Nath, S.; Liu, J.; Zhao, F. SenseWeb: an infrastructure for shared sensing. *IEEE MultiMedia 14*, 4 (2007), 8–13.

[18] Krasner, G. E.; Pope, S. T., et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming 1*, 3 (1988), 26–49.

[19] Lu, J. *Senhome: A Convenient and Inexpensive Sensing System for Improving the Energy Efficiency of Heating, Cooling, and Lighting in Homes.* Tese de Doutorado, Charlottesville, VA, USA, 2011. AAI3501812.

[20] Madria, S.; Kumar, V.; Dalvi, R. Sensor cloud: A cloud of virtual sensors. *Software, IEEE 31*, 2 (Mar 2014), 70–77.

[21] Massie, M. L.; Chun, B. N.; Culler, D. E. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing 30*, 7 (2004), 817–840.

[22] Networks, D. Wireless sensor networks make it possible to predict precious water supplies. Tech. rep., Linear Technology Corporation, 1630 McCarthy Blvd. Milpitas, CA 95035-7417.

[23] ORACLE. Home: Java platform, standard edition (java se) 8 release 8. `https://docs.oracle.com/javase/8/`, 2015. Accessed: 2015-08-19.

[24] ORACLE. Java persistence api. `http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html`, 2015. Accessed: 2015-08-20.

[25] PECHOTO, M. M.; UEYAMA, J.; de ALBUQUERQUE, J. P. E-noé: Rede de sensores sem fio para monitorar rios urbanos. In *Congresso Brasileiro Sobre Desastres Naturais* (2012).

[26] Pivotal. AMQP 0-9-1 model explained. `https://www.rabbitmq.com/tutorials/amqp-concepts.html`, 2014. Accessed: 2014-12-04.

[27] PostgreSQL. Postgresql: The world's most advanced open source database. `http://www.postgresql.org/`, 2015. Accessed: 2015-08-19.

[28] Realtek. Realtek. `http://www.realtek.com.tw/products/productsView.aspx?Langid=1&PFid=5&Level=5&Conn=4&ProdID=239`, 2015. Accessed: 2015-08-21.

[29] Russell, J.; Cohn, R. *Rabbitmq.* Book on Demand, 2012.

[30] Sanchez, A.; Blanc, S.; Yuste, P.; Serrano, J. J. Rfid based acoustic wake-up system for underwater sensor networks. In *Proceedings of the 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems* (Washington, DC, USA, 2011), MASS '11, IEEE Computer Society, pp. 873–878.

[31] Solarwinds. Systems management live product demo - application summary. `http://systems.demo.solarwinds.com/Orion/Apm/Summary.aspx/`, 2015. Accessed: 2015-11-25.

[32] Taylor, R. N.; Medvidovic, N.; Dashofy, E. M. *Software Architecture: Foundations, Theory, and Practice.* Wiley Publishing, 2009.

[33] Vigilent; Networks, D. Close the loop on energy management at the california franchise tax board. Tech. rep., Linear Technology Corporation, 1630 McCarthy Blvd. Milpitas, CA 95035-7417.

[34] Vinoski, S. Advanced message queuing protocol. *IEEE Internet Computing 10*, 6 (Nov. 2006), 87–89.

[35] Yanjun, Z.; Ruikun, G.; Li, D. Cems monitoring system of epa based on web. In *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on* (May 2010), vol. 1, pp. 914–917.

[36] Yi, W.; Liu, L. Monitoring schumann resonance and other electromagnetic precursors of an earthquake with a virtual mimo wireless sensor network. In *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief* (New York, NY, USA, 2011), ACWR '11, ACM, pp. 273–276.

[37] Zanatta, G.; Bottari, G. D.; Guerra, R.; Leite, J. C. B. Building a WSN infrastructure with COTS components for the thermal monitoring of datacenters. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014* (2014), pp. 1443–1448.