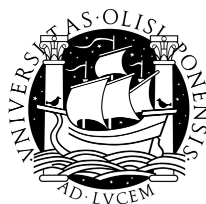
UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática



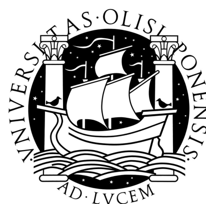
UM *MIDDLEWARE* PARA A INTERNET DAS COISAS

Bruno Alexandre Loureiro Valente

MESTRADO EM INFORMÁTICA

2011

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática



UM *MIDDLEWARE* PARA A INTERNET DAS COISAS

Bruno Alexandre Loureiro Valente

DISSERTAÇÃO

Projecto orientado pelo Prof. Doutor Francisco Cipriano da Cunha Martins
e co-orientado pela Prof[a]. Doutora Maria Dulce Pedroso Domingos

MESTRADO EM INFORMÁTICA

2011

Agradecimentos

Agradeço em primeiro lugar aos meus orientadores Professor Doutor Francisco Martins e Professora Doutora Dulce Domingos, pela oportunidade e pela confiança demonstrada ao longo deste ano de trabalho. Agradeço especialmente ao Professor Francisco, não só pelo apoio e dedicação neste último ano, mas por todos os momentos no decorrer destes últimos cinco(!). Nenhum outro Professor me orientou desde do primeiro até ao último ano, nem nenhum outro conseguiu entrar no lugar restrito dedicado aos amigos. Muito Obrigado Professor.

E o que seria eu sem a minha família? Nada. Queria portanto agradecer a quem mais fez por mim, aos meus pais e à minha namorada. Aos meus pais por me darem tudo o que preciso para crescer e ser quem sou, por me apoiarem quando preciso e por ralharem quando necessário. Obrigado! Obrigado também à minha namorada, talvez a pessoa que mais "sofreu" comigo nestes últimos cinco anos, dedicando-me mais à FCUL que a ela. Obrigado por me aturares, por me apoiares e por teres a paciência necessária para suportar a minha ausência. Espero que tenha valido o esforço, Obrigado :)

Quando pensamos em amigos, há pessoas que surgem na mente mais depressa que outras. Querendo agradecer a todas essas pessoas o quanto foi bom fazerem parte da minha vida nestes últimos anos, sinto a necessidade de agradecer em especial a quem mais me marcou: ao Miguel, com o qual *colidi* no segundo semestre do segundo ano (ficando como companheiro de armas), surgindo à posteriori a amizade, as discussões e as partilhas de ideais que espero que não fiquem por aqui; à Lúcia e ao Ruben (*LuhBens*, para que esta alcunha fique eterna), por fazerem parte não só das noitadas da FCUL, mas também daqueles momentos que são sempre bons de recordar e, por antecipação, pelos momentos que ainda virão. OBRIGADO!

Para quem me lembro quando penso em família e amigos, sem os quais eu, não seria eu.

Resumo

Depois da Internet tradicional (com comunicação programa-para-humano) e depois da Internet dos Serviços (com comunicação programa-para-programa), a *Internet das Coisas* é um novo paradigma de comunicação que pretende identificar e integrar o estado das *Coisas* do mundo real no mundo digital. Porém, as coisas são ubíquas e têm funções e estados intermitentes, o que coloca grandes desafios ao desenho e ao desenvolvimento de aplicações fiáveis que dependem do estado destes objectos. Esta dissertação tem como objectivo enfrentar estes desafios e propor uma plataforma do tipo *middleware* para interagir com os dispositivos das redes de coisas e os seus dados, tudo isto suportado pelo uso de serviços na *Web*. Desenhámos e implementámos uma plataforma genérica, onde os serviços representam funcionalidades das redes de dispositivos, oferecendo às aplicações cliente métodos dinâmicos para interacção com *hardware* heterogéneo. Usando serviços na *Web* beneficiamos de uma tecnologia interoperável, independente de plataformas e linguagens de programação e disponível na *Web*, atributos ideais para combinar sistemas heterogéneos, como a Internet das Coisas. Como suportamos diversos e numerosos serviços, propomos um método inovador de pesquisa baseado numa ontologia que relaciona as propriedades das redes de coisas com os serviços oferecidos, permitindo a pesquisa semântica dos serviços desejados através das suas características.

Palavras-chave: *Middleware*, Computação orientada-a-eventos, Internet das coisas, serviços na Web e disponibilização de sensores na *Web*.

Abstract

After the traditional Internet (with program-to-human communication), and after the Internet of Services (with program-to-program communication), the Internet of *Things* is a novel paradigm of communication, aiming at integrating the state of everyday things into the digital world. But things are everywhere, have different colours, come in different flavours, so, building reliable applications that depend on such things imposes great challenges and demands for new approaches to integrate heterogeneous devices smoothly. This master thesis faces these challenges and proposes a middleware framework to manage the things networks and their data, all this supported by Web services. We designed and implemented a generic platform, where services represent network features, allowing for high level applications to interact with heterogeneous hardware using dynamic methods. When we use Web services, we benefit from an interoperable technology, cross platform and independent from programming languages, and the most important, available on the Web. These attributes makes easier building heterogeneous systems, like the Internet of Things. As we support many services, we propose an innovative search method based on an ontology that relates the things networks with the available services, allowing semantics searches through their characteristics.

Keywords: Middleware, event-driven computing, Internet of things and services, Web services, and sensor Web enablement.

Conteúdo

Lista de Figuras **xiv**

Lista de Listagens **xv**

1 Introdução **1**
1.1 Motivação 1
1.2 Objectivos e Abordagem 4
1.3 Contribuições 7
1.4 Estrutura do Documento 8
1.5 Considerações Finais 8

2 Trabalho relacionado **9**
2.1 Arquitectura Orientada-a-Serviços 9
2.1.1 Dispositivos Orientados-a-Serviços 11
2.2 Serviços na *Web* 12
2.3 Serviços com Pontos de Entrada Genéricos 13
2.4 Canal para Composição de Serviços 14
2.5 Arquitectura Orientada-a-Eventos 15
2.6 Disponibilização de Sensores na *Web* 15
2.7 O Estado da Arte 18
2.8 Considerações Finais 21

3 O MuFFIN **23**
3.1 Arquitectura 23
3.2 Modelo de Dados 24
3.3 Abordagem 26
3.4 Serviços Disponibilizados 28
3.5 Programação de Redes e Gestão de Dados 34
3.6 Considerações Finais 36

4 Implementação **39**
4.1 Tecnologias Utilizadas 39

4.2 Tipos de Comunicação Suportadas 40
4.3 Os Módulos do *MuFFIN* 41
4.4 Pesquisa de Serviços Baseada em Ontologias 45
4.4.1 Entidades do OntoMuFFIN 46
4.4.2 Interacção com a OntoMuFFIN 47
4.5 Considerações Finais 47

5 Avaliação **49**
5.1 Ambiente de Testes 49
5.2 Testes Efectuados ao Módulo *ThingsGateway* 50
5.3 Testes Efectuados ao Módulo *DFN-Engine* 53
5.4 Testes Efectuados ao Módulo *SOS* 54
5.5 Considerações Finais 56

6 Conclusão **57**

A Interfaces dos Principais Módulos **59**
A.1 IThingsGateway 59
A.2 IWSGateway 60
A.3 ISubscriptions 60
A.4 ISOS 61
A.5 IDFN-Engine 62
A.6 ISOSFacade 62
A.7 IServicesFacade 63
A.8 IGatewaysFacade 63

B Documentos SWE **65**
B.1 Inserção de Observação 65
B.2 Leitura de Observações 66
B.2.1 Filtro através da identificação da observação 66
B.2.2 Filtro através da oferta (*offering*) 66
B.2.3 Filtro através de várias propriedades SWE 66

Abreviaturas **68**

Bibliografia **74**

Lista de Figuras

2.1 Diagrama de sequência da pesquisa e invocação de um serviço. 9
2.2 Protocolo para o registo, descoberta e invocação de serviços na *Web*. . . . 12
2.3 Diagrama de camadas da arquitectura OSGi. 14
2.4 Arquitectura genérica de um ESB. 14

3.1 Arquitectura do *middleware* através de um diagrama de camadas. 24
3.2 Diagrama entidade relacional simplificado da base de dados do *MuFFIN*. 25
3.3 Exemplo de *ponto de acesso* utilizado na comunicação com uma rede de dispositivos. 27
3.4 Instanciação do módulo *M1* para objectivos diferentes (*M1 i1* e *M1 i2*). . . 27
3.5 Diagrama de sequência do serviço *deployModule*. 28
3.6 Diagrama de sequência do serviço *instantiateService*. 28
3.7 Relação conceptual entre os filtros a sua representação na *Web*. 29
3.8 Diagrama de sequência do serviço *InstallGateway*. 29
3.9 Diagrama de sequência do serviço *insertObservation*. 30
3.10 Diagrama de sequência do serviço*getCapabilitites*. 30
3.11 Diagrama de sequência do serviço *readObservation*. 31
3.12 Diagrama de sequência do serviço *getModulesInfo*. 32
3.13 Diagrama de sequência do serviço *subscribeService*. 33
3.14 Diagrama de sequência do serviço *unsubscribeService*. 33
3.15 Diagrama de sequência do serviço *deployCode*. 34
3.16 Exemplo de cadeia de dependências gerada na camada de middleware. . . 36

4.1 Visão conceptual dos principais módulos instalados no Fuse ESB. 41
4.2 Diagrama das relações existentes entre as classes da OntoMuFFIN. 45

5.1 Gráfico de linha com os intervalos de tempo, por repetição, até a recepção das mensagens por parte do subscritor, com as definições por omissão da *ActiveMQ*. 51
5.2 Gráfico de linha com os intervalos de tempo, por repetição, até a recepção das mensagens por parte do subscritor, com o controlo de fluxo de mensagens desactivo. 52

5.3 Gráfico de linhas com a sobreposição dos intervalos de tempo com o controlo de fluxo de mensagens da *ActiveMQ* activo (a vermelho) e desactivo (a azul). 52
5.4 Gráfico de barras com os intervalos de tempo médios desde o envio até à recepção de mensagens, com um até cinco pontos de acesso a serem executados em simultâneo. 53
5.5 Gráfico de barras com os intervalos de tempo médios para que uma mensagem percorra uma cadeia de cinco, dez, vinte, cinquenta e cem filtros. . 54
5.6 Gráfico de linha com os intervalos de tempo, em cada mensagem, até que a mensagem percorra uma cadeia composta por cinco filtros. 54
5.7 Gráfico de barras com os intervalos de tempo médios despendidos na escrita e leitura de observações, com um até cinco clientes e, por fim, cem clientes. 55
5.8 Gráfico de barras com a proporção de tempo despendido na leitura e escrita de observações dependendo da origem das observações (XML ou Base de Dados). 56

A.1 Diagrama de camadas da comunicação entre módulos do MuFFIN. 59

Lista de Listagens

3.1 Documento XML com informação dos módulos existentes no *middleware*. 31
3.2 Um exemplo de documento XML para instanciar um filtro. 35
4.1 Configuração da integração dos serviços *Web* do MuFFIN no Fuse ESB. . 43
A.1 Interface do módulo *ThingsGateway*. 59
A.2 Interface do módulo *WS-Gateway*. 60
A.3 Interface do módulo *Subscriptions*. 60
A.4 Interface do módulo *SOS*. 61
A.5 Interface do módulo *DFN-Engine*. 62
A.6 Fachada do módulo *DataAccess* para o módulo *SOS*. 62
A.7 Fachada do módulo *DataAccess* para gestão de serviços utilizada pelos módulos *DFN-Engine* e *Subscriptions*. 63
A.8 Fachada do módulo *DataAccess* para o módulo *ThingsGateway*. 63
B.1 Exemplo de documento XML para inserção de observação SWE. 65
B.2 Leitura de observação através da sua identificação. 66
B.3 Leitura de observações através da propriedade *oferta*. 66
B.4 Leitura de observações através de várias propriedades. 66

Capítulo 1

Introdução

Este primeiro capítulo tem como objectivo apresentar a motivação e os objectivos do projecto que dá origem a esta dissertação. Apresentamos também as contribuições que resultaram do nosso trabalho, terminando com a listagem da estrutura dos restantes capítulos.

1.1 Motivação

A Internet das Coisas (IoT, do inglês *Internet of Things*) é um paradigma que tem como objectivo criar uma ponte entre acontecimentos do mundo real e as suas representações no mundo digital. O objectivo é integrar o estado das *Coisas* que constituem o nosso mundo em aplicações de *software*, beneficiando do contexto onde estão instaladas.

Existem vários métodos para capturar o estado das *Coisas*, desde simples códigos de barras para identificação de objectos, até tecnologias mais sofisticadas que envolvem rádio-frequência (RFID, do inglês *Radio-Frequency IDentification*), comunicação dentro de um raio de transmissão (NFC, do inglês *Near Field Communication*), ou até mesmo dispositivos mais complexos, como sensores, que vêm equipados com memória interna, navegação por satélite e, principalmente, com recursos de computação [33]. As redes de sensores têm especial interesse uma vez que permitem que Coisas, classificadas como *Objectos Inteligentes*, executem operações computacionais e interajam entre si.

As redes de sensores são actualmente um dos tópicos de investigação em destaque na ciência da computação e uma área de negócio em expansão fora do ambiente académico, tendo aplicações nas mais variadas áreas. Estas redes são compostas por um conjunto de dispositivos, chamados de nós, capazes de medir fenómenos físicos (como por exemplo, luminosidade, temperatura e humidade) do meio onde estão instalados e de comunicar entre si. O seu objectivo é recolher dados e encaminhá-los para um, ou mais, nós com características especiais, chamados de estações-base, servindo de ponte entre a rede e o mundo exterior. Em alguns casos os nós possuem maior capacidade de processamento, podendo também efectuar operações sobre os dados recolhidos.

Os avanços tecnológicos na electrónica e na comunicação sem fios permitiram que se desenvolvessem sensores com baixo custo de produção, multifuncionais e com maior rendimento energético. Este tipo de sensor é geralmente pequeno e é composto por quatro unidades básicas: unidades sensíveis às variações ambientais, unidade de processamento, unidade de rádio para comunicação sem fios e bateria [25]. Estes resultaram em redes de sensores sem fios (WSNs, do inglês *Wireless Sensor Networks*), uma especialização das redes de sensores, onde a comunicação entre dispositivos é feita através de rádio frequência e onde os nós dependem (geralmente) de uma bateria como fonte de energia.

As WSNs são utilizadas em aplicações de áreas tão distintas como a militar, a segurança física, o controlo de tráfego aéreo, a videovigilância, a automação de produção industrial ou a robótica distribuída. Em muitas destas aplicações, os sensores estão espalhados por áreas de grandes dimensões, de difícil acesso e fisicamente distantes dos utilizadores (*e.g.*, controlo florestal), dificultando o acesso aos dados recolhidos. Com a propagação desta tecnologia, surgiu a necessidade de melhorar o acesso e a gestão dos dados de forma remota, utilizando preferencialmente redes públicas e resilientes, como a Internet. Porém, a necessidade de controlar remotamente as WSNs contrasta com a heterogeneidade que caracteriza este tipo de rede, onde a variedade de dispositivos e linguagens de programação dificultam a criação de uma solução genérica.

O interesse em implementar aplicações de alto nível que processem os dados enviados pelos Objectos Inteligentes contrasta com o seu custo. Interagir com dispositivos que executam diferentes sistemas operativos (como por exemplo, os *TinyOS* [51] ou os *Contiki* [1]), ou que executam máquinas virtuais (como o exemplo da *Squawk* [20]), que processam linguagens de programação específicas (como por exemplo, o *nesC* [17]) e que usam diferentes protocolos de comunicação, torna difícil criar uma forma genérica que permita aplicações de alto nível lidarem com a complexidade destes sistemas.

A nossa motivação é a necessidade de abstrair a interacção entre as aplicações de alto nível e os objectos inteligentes, tornando transparente o protocolo de comunicação e outras idiossincrasias dos dispositivos, tal como restrições na capacidade destes para serem programados. Estas restrições podem ser causadas por limitações de *hardware* (*e.g.*, o sensor apenas tem a capacidade para executar primitivas simples), ou pode ser imposta pelo fornecedor dos dispositivos, não disponibilizando um método (ou informação) para a reprogramação do sensor. O nosso objectivo é fornecer um meio de interacção com os objectos inteligentes através do MuFFIN (do inglês, *Middleware Framework For the Internet of thiNgs*), uma camada de *middleware* onde, na perspectiva de uma aplicação, todos os dispositivos são reprogramáveis e apresentam-se através de uma interface comum.

A instalação de novo código nos dispositivos (reprogramação) depende da especificação do *hardware* fornecida pelo fabricante. Tanto quanto sabemos, não existem muitas plataformas que permitam a reprogramação remota dos dispositivos enquanto estes estão

em execução. Em Callas [22] é apresentada uma plataforma que suporta a reprogramação remota, e em tempo de execução, de nós de uma WSN. Neste trabalho elevamos essa ideia para o nível do *middleware*, implementando-o com a capacidade para reprogramar os dispositivos e os seus fluxos de dados, independentemente do suporte oferecido pelo fabricante.

Do ponto de vista das aplicações de alto nível, a instalação de código nos dispositivos de uma rede é feita independentemente da capacidade destes para serem reprogramados. Baseado na configuração dos dispositivos, o MuFFIN é capaz de instalar o código nestes ou, em alternativa, processar o comportamento do novo código ao nível da plataforma. Para esta última abordagem o nosso *middleware* inclui um mecanismo para processamento de fluxo de dados, que executa módulos enviados pelos clientes. Estes módulos processam os dados recebidos e são estruturados numa cadeia de fluxos de dados, agindo como filtros que processam os dados de acordo com as necessidades e especificação dos clientes.

Dentro deste enquadramento, a computação orientada-a-serviços é um paradigma emergente que tem vindo a alterar a forma como as aplicações são concebidas. Consiste numa arquitectura que utiliza serviços como base para o desenvolvimento de aplicações distribuídas que executam em ambientes heterogéneos, tal como as redes de sensores. Este tipo de arquitectura tem como objectivo minimizar dependências entre as componentes do sistema, aumentando a sua flexibilidade. Tendo em conta as características associadas a este paradigma, os serviços na *Web* têm-se revelado como uma tecnologia promissora. Estes fornecem uma base para o desenvolvimento de aplicações distribuídas, disponibilizando serviços através da Internet. Os serviços utilizam protocolos e padrões *Web* bem conhecidos, facilitando a combinação e interacção entre serviços.

Uma característica a ter em conta neste tipo de arquitectura é o tempo de resposta, fazendo com que os dados recolhidos estejam disponíveis o mais rapidamente possível. Portanto, é também nosso objectivo criar um sistema eficiente de comunicação entre as componentes do sistema e entre os serviços e as aplicações cliente. Para alcançar este objectivo é necessário combinar o paradigma orientada-a-serviços com o paradigma orientada-a-eventos, aproveitando as vantagens destes dois padrões.

Outra área problemática da construção de uma Internet das *Coisas* é a interoperabilidade entre vários centros de observação e os seus clientes. Seguimos a especificação *Sensor Web Enablement* (SWE) da *Open Geospatial Consortium*. Esta especificação define um conjunto de interfaces interoperáveis e a codificação de metadados que permite a integração, em tempo real, de redes de sensores heterogéneas. A nossa implementação respeita dois destes padrões: o Observações e Medições (O&M, do inglês *Observations and Measurements*) e o Serviços de Observação de Sensores (SOS, do inglês *Sensor Observations Service*), apresentados na Secção 2.6.

Outro aspectos que teremos em conta na implementação do MuFFIN é a quantidade e a diversidade de serviços que irão ser oferecidos, o que dificultará a descoberta e a utilização destes por parte das aplicações cliente da plataforma. Mais, dada a falta de resiliência deste tipo de dispositivos, pode acontecer que um serviço fique indisponível e que uma aplicação cliente possa querer encontrar alternativas com características semelhantes. Por exemplo, se um serviço que fornece a temperatura em Lisboa falhar, existindo outro disponível, o cliente pode querer subscrevê-lo enquanto o primeiro não fica disponível. Para que a flexibilidade nas subscrições seja possível, preparamos o nosso *middleware* de forma a este suportar a pesquisa semântica de serviços, definindo uma ontologia genérica que pode ser estendida por quem instala os módulos na nossa plataforma.

Em súmula, a motivação para o *middleware* que propomos baseia-se na dificuldade de interacção com redes de dispositivos inteligentes, distantes dos utilizadores, e na heterogeneidade que as caracteriza. Pretendemos melhorar a forma como os utilizadores interagem com os dados enviados pelas redes, podendo optar por um modelo síncrono ou assíncrono, abordando de uma forma inovadora a forma como estas podem ser reprogramadas.

No próxima secção apresentamos, detalhadamente, cada um dos objectivos desta dissertação.

1.2 Objectivos e Abordagem

O nosso objectivo é criar uma arquitectura que permita reprogramar redes de sensores e gerir os dados recolhidos por estas, combinando de forma transparente, redes logisticamente distantes.

De seguida são apresentados os objectivos concretos que dão origem a esta dissertação, os desafios que estes colocam e o método como abordamos cada um dos desafios.

1. **Manipular os dados recolhidos pela rede de dispositivos:** deverá permitir o acesso remoto aos dados provenientes do meio onde a rede está instalada.
   Desafios:

   - Criar uma arquitectura e um protocolo comum, capaz de interagir com diferentes redes.
   - A comunicação entre dispositivos das redes deverá ser transparente para o utilizador.

   Abordagem:

   - Criámos um *middleware* que abstrai a origem dos dados, sendo independente do tipo de rede e da localização dos seus nós. Os dados são enviados para a

nossa plataforma através de um conjunto de pontos de entrada que têm o conhecimento necessário à interacção com os dispositivos das redes. Estes pontos de entrada são instalados pelos gestores da plataforma, parametrizando-a e tornando-a independente de arquitecturas específicas de sensores.

2. **Gestão dos serviços de processamento de dados:** deverá apresentar uma interface que permita aos utilizadores criarem os seus próprios serviços e instalá-los no sistema para invocação ulterior.
   Desafios:
   - A interface do sistema terá de ser independente de qualquer rede.
   - Terá de ser criado um ambiente de execução, onde os serviços dos utilizadores possam ser instalados e disponibilizados como serviços na *Web*.

   Abordagem:
   - Permitimos que o utilizador crie os seus próprios serviços e que os instale no *middleware*. Estes serviços são vistos como filtros que podem ser combinados para processarem os dados recebidos das diferentes redes, disponibilizando o resultado num formato independente da rede e do processamento executado. Esta independência é suportada pela implementação de alguns dos padrões da especificação SWE.
3. **Compor os serviços de forma a criarem um sistema dinâmico de filtros:** deverá permitir que os utilizadores agrupem os serviços existentes, de forma a operar sobre os dados recolhidos pela rede.
   Desafio:
   - O ambiente de execução terá de permitir a agregação de serviços e permitir a criação de um fluxo de invocações dos mesmos.

   Abordagem:
   - As dependências entre invocações são criadas através de uma especificação que permite ao utilizador instanciar os serviços instalados e indicar a origem dos dados a processar. Desta forma é criada uma rede dinâmica de fluxos de dados, que processa os dados recebidos de acordo com as necessidades dos clientes.
4. **Subscrever os serviços e eventos disponíveis:** deverá permitir que os utilizadores invoquem os serviços existentes, podendo receber os resultados de forma síncrona ou assíncrona.

Desafios:

- Oferecer diferentes métodos de interacção, podendo o utilizador invocar os serviços de forma síncrona ou assíncrona.
- Disponibilizar a opção que possibilita aos utilizadores registarem-se num serviço e receber eventos relacionados com este.

Abordagem:

- A arquitectura que implementámos disponibiliza serviços na *Web* que possibilitam os dois tipos de comunicação acima referidos. Na comunicação síncrona o cliente especifica o intervalo temporal e logístico com os quais os dados das redes estão relacionadas, recebendo a resposta "de imediato". Para habilitar a comunicação assíncrona, possibilitamos a criação de representações dos serviços instalados como serviços na *Web*, permitindo que os clientes os subscrevam e que sejam notificados com os dados processados.

5. **Expor a plataforma como um conjunto de serviços na *Web*:** deverá permitir que as aplicações de alto nível acedam de forma transparente a redes de sensores heterogéneas, através de tecnologias bem conhecidas, como os serviços na *Web*.
Desafio:

- Agrupar redes de sensores logisticamente distantes de forma transparente para o utilizador.

Abordagem:

- A tipologia das redes é transparente para o utilizador graças aos pontos de entrada e aos serviços já descritos. Assim, o cliente consegue subscrever o serviço na *Web* relacionado com o serviço instalado, não sendo necessário o conhecimento do tipo de rede, nem do encaminhamento dos dados até ao *middleware*.

6. **Integrar instâncias do *middleware*:** deverá permitir que várias instâncias comuniquem entre si, partilhando os seus dados e serviços. Esta funcionalidade é particularmente útil quando um cliente necessita aceder a vários centros de observação espalhados pelo globo, sem que isso signifique várias ligações com os diferentes centros.
Desafio:

- Criar uma forma genérica de comunicação entre as instâncias.

Abordagem:

- Utilizar os serviços existentes em todas as instâncias, criando uma rede onde uma das instâncias é subscritora das restantes, ficando esta encarregue de orquestrar as invocações. Os pontos de acesso de cada uma das instâncias são especificados através de padrões *Web*.

7. **Permitir a pesquisa semântica de serviços:** deverá permitir que os utilizadores consigam dar semântica às suas necessidades e subscrever os serviços na *Web* que respeitem os seus requisitos.
Desafio:

- Criar uma ontologia genérica e extensível que permita catalogar os serviços existentes no sistema.

Abordagem:

- Utilizar a linguagem OWL (do inglês, *Web Ontology Language*) para definir a ontologia, esta linguagem permite agregar e utilizar diferentes ontologias, baseadas em recursos existentes na *Web*.

1.3 Contribuições

Os objectivos anteriormente descritos resultam num conjunto de contribuições que apresentamos de seguida:

Gestão genérica de dados: criámos um *middleware* que permite recolher, filtrar e devolver dados provenientes das redes de sensores. A nossa arquitectura é genérica graças à implementação dos padrões SOS e O&M da especificação SWE, permitindo que tanto utilizadores humanos, como aplicações de alto nível, consigam pesquisar e receber os dados que necessitam. Outro facto que contribui para que a nossa plataforma seja genérica é a utilização de serviços na *Web*, tecnologia independente de arquitecturas e linguagens de programação.

Reprogramação de dispositivos e fluxos de dados: possibilitamos a programação dos dispositivos das redes e dos seus fluxos de dados, através de abstracções em diferentes níveis. Abstraímos o *hardware* dos dispositivos que compõem as redes, abstraímos algoritmos de comunicação entre os nós e abstraímos a comunicação entre os dispositivos e o MuFFIN. Assim, permitimos que aplicações cliente programem fisicamente dispositivos heterogéneos, instalando o código directamente nestes, ou logicamente, através

da criação de uma rede de fluxo de dados. Esta rede é executada directamente no *middleware*, processando e filtrando os dados enviados pelos dispositivos, disponibilizando através de serviços na Web os dados que as aplicações cliente necessitam.

Pesquisa de serviços orientada ao utilizador: desenhámos e construímos uma ontologia com o objectivo de permitir ao utilizador dar semântica às suas pesquisas e desta forma conseguir procurar quais os serviços que satisfazem as suas necessidades, sem que tenha de lidar directamente com a totalidade dos serviços existentes.

1.4 Estrutura do Documento

Para além da introdução, esta dissertação encontra-se organizado da seguinte forma: o Capítulo 2 apresenta o trabalho relacionado, onde são apresentados os conceitos básicos sobre as tecnologias utilizadas e onde damos a conhecer o estado da arte onde este projecto se enquadra. No Capítulo 3 são discutidas as decisões acerca do desenho da nossa plataforma, tais como a arquitectura e as abordagens tomadas para cumprir com os requisitos apresentados. O Capítulo 4 detalha os pormenores da implementação deste projecto e ainda o que cada módulo da aplicação oferece aos restantes. No Capítulo 5 apresentamos os testes efectuados à nossa plataforma e os seus resultados. Por fim, apresentamos as nossas conclusões e ainda o trabalho futuro deste projecto.

1.5 Considerações Finais

Neste capítulo apresentámos a motivação, os objectivos e as contribuições do nosso trabalho. Apresentámos o MuFFIN como uma plataforma de suporte à gestão de redes de dispositivos inteligentes e dos seus dados. É objectivo do MuFFIN: (1) gerir os dados recolhidos pela rede de dispositivos através da criação um sistema dinâmico de filtros de dados; (2) oferecer um conjunto de serviços *Web* com os resultados do processamento dos seus filtros; (3) notificar, através da *Web*, aplicações de alto nível subscritoras dos serviços disponibilizados e (4) integrar várias instâncias da nossa plataforma com o objectivo de facilitar o acesso das aplicações cliente a dados oriundos de vários pontos do globo.

De seguida apresentamos o estado da arte das plataformas de gestão de redes de dispositivos inteligentes e introduzimos a arquitectura implementada e as tecnologias utilizadas no seu desenvolvimento.

Capítulo 2

Trabalho relacionado

Este capítulo apresenta as tecnologias utilizadas na especificação e implementação do MuFFIN, com o objectivo de contextualizar futuras referências. São também apresentados projectos relacionados com o nosso, apontando os aspectos em comum e ainda limitações que temos como objectivo mitigar.

2.1 Arquitectura Orientada-a-Serviços

A arquitectura de *software* orientada-a-serviços (SOA, do inglês *Service-Oriented Architecture*) permite a criação de serviços interoperáveis, podendo estes ser reutilizados e compartilhados entre sistemas.

Existem três intervenientes com papéis distintos numa arquitectura SOA: os *fornecedores*, os *consumidores* e o *registo* de serviços. Os fornecedores disponibilizam serviços, enquanto que os consumidores utilizam esses serviços. O registo é uma entidade localizada num endereço conhecido, onde estão registados os serviços disponíveis aos consumidores. O registo é o interveniente chave na descoberta dinâmica de serviços, pois mantém um catálogo dos serviços existentes e divulga os endereços para comunicação com os serviços. Os consumidores são os intervenientes no processo que procuram os serviços no registo para ulteriormente invocarem o serviço desejado.

A Figura 2.1 apresenta a sequência de invocações necessárias para que o consumidor comunique com o fornecedor do serviço.

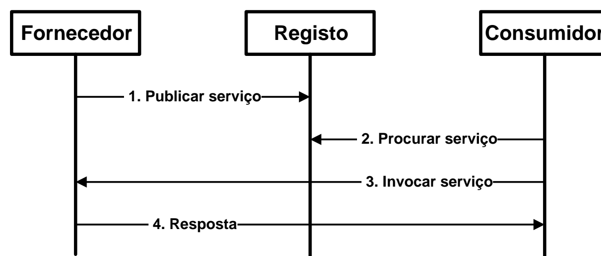


Figura 2.1: Diagrama de sequência da pesquisa e invocação de um serviço.

A arquitectura SOA tem as seguintes características [39]:

- **Descoberta e ligação dinâmica a serviços:** permite que um consumidor descubra, através de um conjunto de critérios, um serviço em tempo de execução. Numa fase ulterior à descoberta do serviço o consumidor invoca-o de forma a satisfazer as suas necessidades.

- **Modular:** um dos aspectos mais importantes da SOA é o seu conceito de modularidade. Um serviço pode ser composto por outros serviços e ainda suportar a ligação de vários consumidores, compondo assim um sistema complexo com dependências entre componentes.

- **Interoperabilidade:** permite que aplicações a utilizarem diferentes plataformas e linguagens comuniquem entre si. Cada serviço oferece uma interface que pode ser invocada através de um conector. Este conector oferece um protocolo e um tipo de dados genérico que ambos os intervenientes da comunicação conhecem, possibilitando a comunicação entre sistemas heterogéneos.

- **Independência entre módulos:** os módulos contêm um número limitado de dependências entre si, permitindo uma fácil reutilização de código e promovendo uma relação de independência entre fornecedores e consumidores de serviços.

- **Transparência da localização do serviço:** o consumidor não sabe onde se encontra o serviço até o localizar no registo. Desta forma, o local onde o serviço está instalado pode ser alterado sem prejudicar ou alterar a forma como o consumidor interage com o serviço. Esta característica facilita a implementação de um balanceador de carga, distribuindo os pedidos dos clientes por várias instâncias do serviço sem o conhecimento do cliente, aumentando o desempenho e a disponibilidade do serviço.

- **Composição de aplicações:** permite a construção de aplicações através da composição de vários serviços já existentes e previamente testados, optimizando o tempo de desenvolvimento já que o programador reutiliza código, em vez de construir aplicações de raiz. Este tipo de composição pode assumir três formas distintas:

  - Uma aplicação: é tipicamente uma montagem de serviços, componentes e outras aplicações, combinadas de forma a criar outra aplicação que satisfaça determinados requisitos;

  - Conjunto de serviços: serviços independentes entre si, combinados de forma a criar um sistema no qual o *todo* é maior que a *soma das parte*. Por exemplo, um serviço de autenticação de contas bancárias, um serviço de depósitos bancários e um serviço de verificação de saldo de uma conta bancária, podem

ser combinados numa camada de serviços mais alargada de forma transparente para o cliente, criando um serviço de gestão bancária;

  – Orquestração de serviços: ocorre quando uma única transacção tem impacto num ou mais serviços de forma estruturada. Este tipo de transacção é chamada de processo de negócio. Consiste num conjunto de passos em que cada um destes é uma invocação de um serviço. Se um dos serviços falha, todos os outros serviços já executados terão de recuperar o seu estado antes da execução inicial.

- **Auto-recuperação**: com a complexidade crescente das aplicações distribuídas, uma das características mais importantes da SOA é a recuperação de cada uma das componentes em caso de falha, sem a necessidade de intervenção humana.

2.1.1 Dispositivos Orientados-a-Serviços

A Arquitectura para Dispositivos Orientados-a-Serviços (SODA, do inglês *Service-Oriented Device Architecture*) [19] tem como objectivo permitir a ligação de dispositivos a uma SOA. Actualmente os serviços estão ligados entre si utilizando padrões para serviços na *Web*. A SODA permite que os clientes interajam com os dispositivos como se acedessem a outro serviço na *Web*. A finalidade é converter o *hardware* em *software* com uma interface bem definida, independente da linguagem de programação e da plataforma que executam.

Resumindo, a SODA:

- ocupa o espaço entre os sistemas que integram os dispositivos e as especificações dos fabricantes;

- segue o paradigma "Integrar uma vez, utilizar em todo o lado", focado em adaptadores de dispositivos;

- cria uma interface comum entre a infraestrutura e os diversos protocolos dos dispositivos;

A plataforma que implementámos é uma SOA composta por um conjunto de módulos que oferecem serviços entre si. Estes módulos são orquestrados de forma a disponibilizar um conjunto de serviços às aplicações de alto nível, nossas consumidoras, criando um sistema interoperável, modular, independente e transparente para os seus consumidores. Para além de uma SOA, a nossa plataforma é SODA, disponibilizando os meios necessários para integrar diversos dispositivos e apresentar as suas funcionalidades como serviços na *Web*.

2.2 Serviços na *Web*

Tendo em conta as características da SOA, os serviços na *Web* têm-se revelado como uma das mais promissoras ferramentas para a implementação de sistemas orientados-a-serviços. Este tipo de serviço fornece uma base para o desenvolvimento de aplicações distribuídas, disponíveis na Internet, utilizando protocolos e padrões *Web* para comunicação. Os serviços na *Web* permitem o desenvolvimento de aplicações independentes de plataformas ou linguagens de programação, graças à sua comunicação baseada em XML, possibilitando um alto grau de interoperabilidade.

As interfaces destes serviços são definidas através da linguagem WSDL (do inglês, *Web Service Description Language* [54]), baseada em XML, que especifica um contracto entre o fornecedor e o cliente do serviço, onde fica estabelecido que o fornecedor oferece um conjunto de operações caso o cliente efectue correctamente o pedido. O WSDL apresenta ao cliente os pontos de acesso ao serviço na *Web* e as funções que este pode invocar, sendo suportados argumentos de tipos primitivos e complexos, também especificados no WSDL.

A comunicação é efectuada através de SOAP (do inglês, *Simple Object Access Protocol* [52]), protocolo que utiliza XML para definir um padrão extensível para troca de mensagens. É possível fazer a analogia entre este protocolo e um envelope, onde o cabeçalho SOAP equivale aos dados do emissor-receptor e o corpo contém a informação em XML, análogo ao conteúdo do envelope.

A descoberta dos serviços na *Web* é efectuada através da especificação UDDI (do inglês, *Universal Description Discovery and Integration* [44]), que permite aos fornecedores registar os seus serviços e aos clientes pesquisar pelos serviços que satisfaçam as suas necessidades. Depois da descoberta do serviço, o UDDI envia para o cliente o WSDL necessário à comunicação. A Figura 2.2 mostra o protocolo de comunicação necessário à invocação de serviços na *Web*, desde o registo efectuado pelo fornecedor até à pesquisa e invocação por parte do cliente.

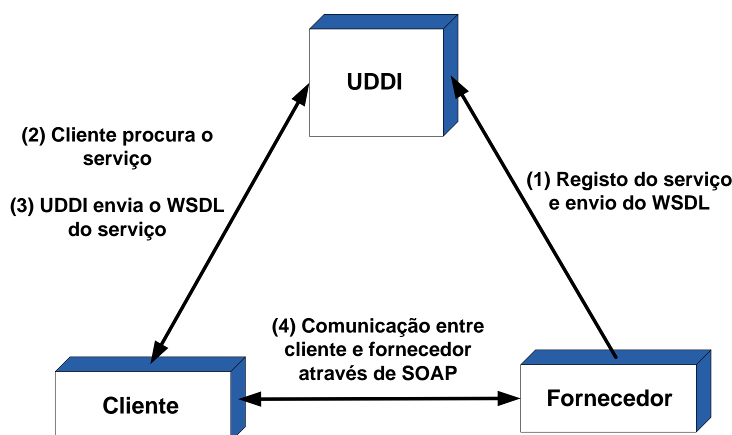


Figura 2.2: Protocolo para o registo, descoberta e invocação de serviços na *Web*.

Não havendo vantagens em utilizar os serviços na *Web* na comunicação local do *middleware* (*e.g.*, na comunicação entre componentes), pois este perderia rendimento no processamento constante de XML, utilizamos esta tecnologia na comunicação com os clientes da nossa plataforma. Assim, esta comunicação é independente das plataformas onde as aplicações cliente se encontram e das linguagens de programação com que estas são implementadas.

2.3 Serviços com Pontos de Entrada Genéricos

A Arquitectura para Serviços com Pontos de Entrada Genéricos (OSGi, do inglês *Open Services Gateway initiative framework* [45]) é uma plataforma de serviços modulares para a linguagem Java que implementa um modelo de componentes dinâmicas. As componentes podem ser geridas remotamente, sem que seja necessário reiniciá-las, graças à gestão do ciclo de vida (instalação, arranque, interrupção, etc.) através de interfaces bem definidas. As plataformas OSGi são implementadas em pequenas componentes, onde as dependências estão definidas em ficheiros de configuração bem estruturados. Estes ficheiros são carregados dinamicamente, permitindo a dissociação entre componentes. Como visível na Figura 2.3, a arquitectura conceptual destas plataformas possui:

- **Componentes:** conjunto de classes Java (.jar) com especificações extras no seu .MANIFEST.

- **Serviços:** camada que liga as componentes dinamicamente através de um modelo *publica-encontra-conecta* para interfaces e objectos java.

- **Registo de Serviços:** conjunto de interfaces para gestão de serviços que permite o registo e a pesquisa destes.

- **Ciclos de vida:** conjunto de interfaces para a gestão do ciclo de vida das várias componentes.

- **Módulos:** camada que define o encapsulamento e a declaração de dependências entre componentes.

- **Segurança:** camada que define as políticas de acesso das componentes aos recursos do ambiente de execução.

- **Ambiente de execução:** Define os métodos e as classes que estão disponíveis na plataforma.

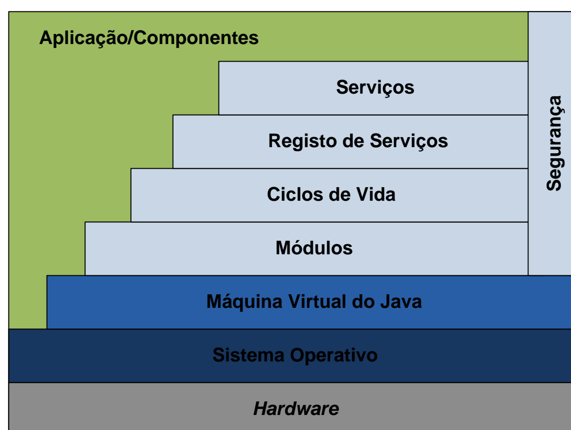


Figura 2.3: Diagrama de camadas da arquitectura OSGi.

2.4 Canal para Composição de Serviços

Numa SOA os serviços não podem ser pensados para um cliente específico, mas sim para um grupo de potenciais consumidores heterogéneos. Um dos métodos para compor aplicações SOA é utilizando um Canal para Composição de Serviços (ESB, do inglês *Enterprise Service Bus*). O ESB é uma plataforma OSGi que permite compor serviços heterogéneos e aplicações distribuídas através da transformação, comunicação e encaminhamento de pedidos entre serviços. Porém, o ESB não implementa uma SOA, o que oferece é uma plataforma capaz de agregar e adaptar os serviços existentes de forma a resolver pedidos específicos por parte do consumidor [24].

A Figura 2.4 apresenta, de forma genérica, as potencialidades de um ESB, onde são visíveis serviços heterogéneos a comunicarem entre si de forma transparente, utilizando um padrão de comunicação comum previamente acordado (em formato XML).

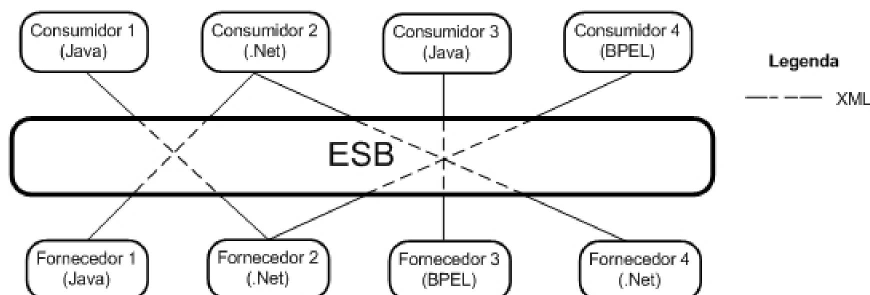


Figura 2.4: Arquitectura genérica de um ESB.

A nossa plataforma é uma SOA implementada com componentes OSGi instaladas num ESB. Assim, não apenas os módulos do *middleware* são independentes entre si, como a tecnologia OSGi permite uma melhor manutenção e reutilização destes. De um modo geral a plataforma não necessita ser reiniciada quando um módulo OSGi é actualizado, diminuindo as consequências das actualizações no normal funcionamento do sistema.

2.5 Arquitectura Orientada-a-Eventos

Numa SOA, a comunicação entre os serviços pode ser feita usando duas metodologias:

- **Comunicação síncrona:** onde o consumidor do serviço fica bloqueado enquanto o fornecedor processa o pedido efectuado e responde.
- **Comunicação assíncrona:** onde o consumidor não fica bloqueado durante o processamento do seu pedido. Esta segunda opção permite que o consumidor continue com o seu normal processamento enquanto não recebe resposta do serviço.

Quando a comunicação é feita através do método assíncrono o consumidor pode receber a resposta ao seu pedido de duas formas distintas. Num dos métodos, designado de espera activa, o consumidor questiona periodicamente o fornecedor para saber se o seu pedido já foi processado. Noutro método, designado por *chamada de retorno* (em inglês, *Callback*), o fornecedor desencadeia um evento quando o pedido é processado e entra em contacto com o consumidor para lhe enviar a resposta do pedido. Uma arquitectura orientada-a-eventos (EDA, do inglês *Event-Driven Architecture*) utiliza o segundo método, onde são desencadeados eventos quando ocorre, por exemplo, um evento agendado para uma determinada hora ou a chegada de uma determinada temperatura vinda da rede de sensores.

A EDA tem conquistado o seu espaço na investigação e em áreas aplicacionais, pois é vista simultaneamente como um método para optimizar a comunicação assíncrona e como um método para optimizar a arquitectura orientada-a-serviços. O normal mecanismo de comunicação da SOA é do tipo pedido/resposta síncrono, enquanto que a EDA permite a comunicação assíncrona [41].

Um exemplo de uma EDA é a arquitectura Publicador-Subscritor (PubSub) [34], onde existem publicadores de informação relacionados com tópicos e subscritores que se registam para receber essas informações. Esta arquitectura permite a criação de sistemas complexos, onde publicadores podem também ser subscritores de outros tópicos.

O sistema de filtros implementado na nossa plataforma segue o paradigma PubSub, existindo uma cadeia de dados com filtros que dependem dos dados de outros (sendo ao mesmo tempo publicadores e subscritores) e onde o resultado do processamento de cada filtro pode ser apresentado como um serviço na *Web*.

2.6 Disponibilização de Sensores na *Web*

Os padrões para Disponibilização de Sensores na *Web* (SWE, do inglês *Sensor Web Enablement*) [36] foram criados pela OGC (em inglês, Open Geospation Consortium) com o objectivo de desenvolver especificações que facilitem o acesso a dados provenientes de redes de sensores. O SWE fornece interoperabilidade entre redes heterogéneas, actuando

como um camada intermédia entre estas e aplicações que operam sobre os dados. Esta especificação foi pensada seguindo o fluxo: *descoberta*, *acesso*, *subscrição* e *notificação*, sendo composta pelos seguintes padrões:

Observações e Medições (O&M) [48] [49]: modelos e esquemas XML para codificar medidas e observações recebidas de sensores. Uma observação é definida como um acto de medição de uma propriedade ou fenómeno, com o objectivo de produzir uma estimativa desse valor. Uma observação contém os seguintes campos:

- **Tempo da recolha da amostra:** valor temporal do momento em que a medição foi feita.
- **Procedimento:** procedimento usado para gerar o resultado. Pode ser uma observação vinda de um dispositivo, o resultado de um algoritmo, de uma computação ou o resultado de uma cadeia de processamento complexa.
- **Fenómeno:** define a propriedade do ambiente que foi medida e as unidades dessa medição.
- **Características de interesse:** o objectivo da observação; o objecto do mundo real onde a observação foi feita.

O padrão O&M agrega observações que possuem características em comum; por exemplo, observações com *Características de interesse* semelhantes e que observem os mesmos *Fenómenos*, devem ser relacionados entre si através da mesma *Oferta*. *Oferta* é um método de correlação que define o que é oferecido pelo sistema, sendo esta a propriedade base para todas pesquisas de observações. A especificação SWE não restringe a forma como a correlação entre observações deve gerada, porém aconselha que os campos anteriormente descritos sejam utilizados como a sua base.

Serviço de Observação de Sensores (SOS) [10]: interfaces para serviços na *Web*, com o objectivo de solicitar, filtrar e devolver observações e informação sobre os dispositivos. O objectivo do SOS é oferecer o acesso a observações de dispositivos de forma padronizada e consistente, para todos os sistemas que dependam de sensores. As propriedades utilizadas para filtrar as observações permitem que o cliente especifique a hora, o local, o fenómeno observado e a característica de interesse, de forma a receber apenas as informações do seu interesse.

Linguagem para modelação de sensores (SensorML) [35]: a SensorML (do inglês, *Sensor Model Language*) especifica modelos e esquemas XML que descrevem os sensores de uma rede e as suas capacidades. O objectivo é uniformizar a forma como os dispositivos são classificados, tornando genérica a identificação e a utilização de cada tipo de dispositivo.

Serviço de Alertas de Sensores (SAS) [27]: interface para um serviço *Web* que permite a publicação e subscrição de alertas desencadeados por uma rede de sensores ou por sistemas de simulação. Este padrão especifica um sistema publicador-subscritor, que recebe os dados vindos dos dispositivos da rede e processa condições de forma a verificar se os dados cumprem os requisitos dos alertas registados, notificando posteriormente os seus subscritores. O SAS funciona como um sistema de registo onde os sensores se ligam e registam os alertas que podem enviar. O SAS, como mediador, publica todos os alertas (tópicos) que pode fornecer, permitindo subscrições de clientes. Sempre que um evento associado a um alerta ocorrer, o sensor respectivo irá comunicá-lo ao SAS, que depois notifica todos os subscritores do alerta, utilizando um serviço de mensagens genérico.

Serviço de Planeamento de Sensores (SPS) [26]: interface para um serviço *Web* genérico que permite ao utilizador personalizar pedidos a efectuar à rede. Esta interface auxilia a recepção dos dados de dispositivos das redes e sistemas relacionados. O SPS aceita diferentes configurações, criando planeamentos compostos pela leitura, processamento, arquivo e distribuição de dados provenientes de redes de sensores. O processo de planeamento necessita de duas interacções: (1) o cliente pede ao SPS a lista de parâmetros necessários para a especificação do planeamento, para que de seguida (2) forneça os argumentos necessários para a execução deste.

Serviço de notificações na *Web* (WNS) [28]: o WNS (do inglês, *Web Notification Services*) é uma interface para um serviço *Web* que permite a troca de mensagens assíncronas entre o cliente e o sistema SWE, suportando vários protocolos, tais como, *e-mail*, SMS e HTTP.

O WNS especifica um conjunto de operações obrigatórias à comunicação:

- ***getCapabilities:*** devolve as informações relacionadas com a comunicação, tais como, que protocolos de notificação são suportados e ainda que argumentos são obrigatórios ou facultativos.

- ***registerUser:*** permite o registo do cliente para futuras notificações, sendo necessário enviar o endereço para onde as notificações serão encaminhadas e o protocolo que será usado.

- ***doNotification:*** operação que desencadeia a notificação do cliente.

Linguagem para modelação de transdutores (TML) [50]: o TML (do inglês, *Transducer Model Language*) é uma aproximação contextual e um conjunto de especificações XML para permitir a transferência de dados das redes de sensores em tempo real. Os *Sensores* e os *Transmissores* podem ser caracterizados através do TML, passando a ser

utilizado o termo “transdutor” para ambos. Este padrão permite manipular dados recebidos estaticamente e em *streaming*, especificando várias operações que permitem gerir os dados que percorrem os canais de comunicação, bem como os seus intervenientes.

Este conjunto de especificações é especialmente útil numa plataforma como a nossa, onde pretendemos criar soluções genéricas e independentes de tipos e arquitecturas de dispositivos. Com a implementação de algumas destas especificações (O&M e SOS) fomos capazes de abstrair os nós das redes que a plataforma gere e ainda disponibilizar serviços na *Web* capazes de receber e enviar observações relacionadas com diferentes tipos de dados e variáveis do ambiente onde as redes estão instaladas.

2.7 O Estado da Arte

A orquestração de serviços relacionados com redes de dispositivos inteligentes tem sido um tópico de interesse na investigação, melhorado a forma como interagimos os dispositivos e os seus dados.

Xinhuai Tang *et al.* [56] apresentam um método para agregação dinâmica de serviços, criando uma arquitectura de partilha de dados entre sistemas de informação espalhados pelo globo, onde os serviços são orquestrados dinamicamente. Este método usa uma arquitectura orientada-a-serviços, eliminando limitações existentes em arquitecturas orientadas-a-componentes (CDA, do inglês *Component-Driven Architecture*). Numa CDA (*e.g.*, CORBA [40]) a interacção entre aplicações fica limitada a métodos não padronizados e a restrições da rede (*e.g.*, *firewalls*). A nossa solução implementa uma arquitectura sobre um ESB, usando XML como padrão comum de comunicação. O problema apresentado é semelhante ao encontrado nas redes de sensores, onde a heterogeneidade dificulta a integração das redes numa só camada de serviços. O nosso objectivo é utilizar uma plataforma semelhante à apresentada em [56], criando um sistema de agregação de serviços, onde utilizadores de redes de sensores podem organizar os serviços existentes e recolher dados de redes remotas.

A comunicação entre serviços heterogéneos é possível através de um protocolo comum, utilizando um formato padronizado, tal como o XML. Porém, esta tecnologia necessita de maiores recursos computacionais para processar a informação contida nos documentos. Choon-Sung Nam *et al.* [15] propõe um *middleware* com uma EDA que utiliza o paradigma do publicador-subscritor, permitindo que aplicações cliente subscrevam tópicos do seu interesse e que posteriormente recebam notificações quando é publicada informação relacionada com estes. Este tipo de comunicação reduz a quantidade de mensagens trocadas, em oposição às técnicas de espera activa usadas em cenários onde a comunicação é do tipo síncrona. A arquitectura apresentada é limitada a um conjunto estático de tópicos, não permitindo uma gestão modular do sistema, nem o acesso remoto através de serviços. A nossa plataforma implementa uma arquitectura semelhante à

apresentada, porém eliminámos as limitações relacionadas com a falta de modularidade através de uma rede dinâmica de filtros que é construída ao longo do tempo. Mais, tal como este sistema, a comunicação com a nossa plataforma é baseada em XML, portanto abordamos o problema da mesma forma, criando uma EDA que notifica os seus subscritores quando surgem eventos relacionados com a informação subscrita, diminuindo a quantidade de XML processado ao longo do tempo.

As plataformas do tipo *middleware* são muito utilizadas para agregar e gerir redes de sensores. Rakhi Motwani *et al.* [46] apresentam uma SOA para coordenar centros de observações e partilhar a informação gerida por estes através de serviços na *Web*. A motivação dos autores é que muitos destes centros estão geograficamente distantes e isolados, o que dificulta a partilha de informação. Os serviços criados por cada observatório respeitam as especificações do padrão SWE e são orquestrados através de ESBs existentes em cada um dos observatórios, comunicando entre si através da *Web*. A comunicação é transparente para o utilizador que apenas comunica com o seu próprio ESB. A nossa arquitectura cumpre estes requisitos, permitindo também que os utilizadores possam inserir novos serviços, processando de forma transparente o os dados recebidos de redes distantes.

Outra implementação com código livre dos padrões SWE é a aplicação *52North Sensor Web* [9]. Esta aplicação aparenta ter alcançado o seu estado de maturidade, e portanto tentámos utilizá-la como um módulo do nosso *middleware*. Porém, a integração de um sistema autónomo e complexo com as funcionalidades da nossa plataforma revelou-se difícil. A solução que seguimos foi utilizar parte desta especificação (*e.g.*, o modelo entidade relacional da base de dados) como base da nossa implementação do SWE.

João Santos [32] implementou um *middleware* que possibilita a interoperabilidade entre redes de sensores e aplicações cliente. A arquitectura deste *middleware* foi desenhada para ser SOA e EDA, expondo as suas funcionalidades através de serviços na *Web*. Para dar a possibilidade aos utilizadores da rede de operarem sobre os dados recebidos, é apresentada a sintaxe de uma linguagem criada para definir um sistema de filtros sobre os dados, possibilitando a interacção entre os serviços existentes. Esta abordagem tem em conta os padrões SWE, porém não os implementa, apresentando soluções que podem ser utilizadas de forma semelhante. Para além desta limitação, a solução apresentada não suporta a composição dinâmica de filtros na cadeia de fluxo de dados.

Catello Di Martino *et al.* [14] apresentam uma arquitectura configurável e adaptativa para processamento de redes de sensores baseada na especificação dos dispositivos da rede. Estas especificações resultam em filtros e focos de informação, onde os clientes se ligam para, de uma forma rápida, recolher dados ou receber notificações com os dados que necessitam.

A utilização de ontologias em redes de dispositivos inteligentes tem vindo a aumentar ao longo do tempo, melhorando a interacção com os dispositivos e os seus dados, havendo já vários casos de estudo. Sasikanth Avancha *et al.* [47] apresentam um sistema adaptável baseado em ontologias como uma solução para a mutabilidade dos ambientes onde as redes de sensores são instaladas. Os autores organizam os nós da rede em grupos e programam o sistema em dois passos distintos: no primeiro, cada nó fornece o estado actual da sua bateria e os valores lidos ao responsável do grupo; no segundo passo, cada responsável executa um algoritmo orientado a ontologias que determina o futuro estado da rede baseado nos valores recebidos. Este modelo calibra as redes de sensores, permitindo especificar as condições iniciais da rede e prever o comportamento destas em determinadas condições. O objectivo é respeitar os requisitos dos utilizadores especificados numa sintaxe baseada em *Web* semântica. A forma como integramos ontologias como a nossa plataforma é diferente, porém temos o mesmo objectivo: cumprir os requisitos do cliente em relação aos dados que estes necessitam. Enquanto que Sasikanth Avancha utiliza as ontologias para adaptar a rede às necessidades do cliente, nós utilizamos esta tecnologia para adaptar os pedidos dos clientes à realidade existente. Assim, caso um serviço falhe, é subscrito outro dinamicamente, desde que cumpra os mesmos requisitos especificados na ontologia.

Para além da vertente adaptativa, as ontologias fazem parte de sistemas que têm como objectivo aumentar a precisão das pesquisas dos dados recebidos das redes de dispositivos inteligentes. Mohamad Eid *et al.* [37] definiram e implementaram ontologias com o objectivo de apenas recolher as informações relevantes das redes de sensores. Estas ontologias definem representações e relações utilizadas pelo motor de pesquisa para processar o significado da informação. Os autores propuseram um protótipo de ontologia em duas camadas [38] que utiliza a IEEE *Suggested Upper Merged Ontology* (SUMO) [11] como uma definição base para os conceitos e associações e mais duas sub-ontologias: uma para os dados dos sensores e outra para a hierarquizar os sensores da rede. Desta forma, os autores pretendem utilizar a informação semântica para maximizar a precisão nas pesquisas em dados enviados pelos dispositivos das redes de sensores. O nosso *middleware* não suporta a utilização desta tecnologia para categorizar os dados das redes de sensores, porém é uma funcionalidade que pretendemos implementar num futuro próximo, recorrendo a parte da implementação apresentada como base.

David J. Russomanno *et al.* [18] descrevem a OntoSensor: uma abordagem para implementar ontologias que descrevem dispositivos de uma rede de sensores. A ontologia apresentada estende a SUMO e inclui a definição de conceitos e propriedades existentes na especificação SensorML. Com esta abordagem os autores pretendem classificar semanticamente os vários tipos de dispositivos existentes em redes de sensores, criando a ponte entre a especificação SensorML e a formalização semântica. A ontologia apresentada pode ser integrada como parte da nossa, pois o SensorML é uma especificação SWE

que descreve dispositivos das redes de sensores, sendo útil no nosso trabalho que aborda temas relacionados com as redes de dispositivos inteligentes e a sua heterogeneidade.

Jie Liu e Feng Zhao [30] descrevem uma arquitectura e um modelo de programação para uma plataforma orientada a serviços semânticos. Os autores argumentam que a chave para permitir o acesso escalável à informação das redes de sensores é a associação hierárquica dos dados através de uma ontologia. A abstracção que as ontologias oferecem permite a optimização de recursos na gestão e processamento dos dados. O objectivo é optimizar automaticamente o uso dos serviços existentes no sistema, respeitando as especificações dos utilizadores. A conjugação dos serviços resulta num grafo optimizado para satisfazer as necessidades dos clientes. Desta forma, os utilizadores não necessitam de interagir directamente com os nós da rede, necessitando apenas de interagir semanticamente com a informação da rede como um todo. Nós partilhamos da opinião dos autores e apresentamos a nossa plataforma como um sistema que possibilita a pesquisa semântica dos serviços existentes, permitindo abstrair não só a implementação do serviço mas também os recursos utilizados pelo mesmo.

2.8 Considerações Finais

Neste capítulo apresentámos as tecnologias utilizadas na especificação e implementação da nossa plataforma. O MuFFIN foi implementado como uma SOA/SODA, constituído por módulos independentes que fornecem serviços entre si e capaz de apresentar as funcionalidades de dispositivos como serviços na *Web*. O MuFFIN é instalado num servidor do tipo ESB, onde cada módulo que o compõe implementa a arquitectura OSGi. Assim, é possível gerir os módulos do *middleware* minimizando o impacto no seu funcionamento. Para que a nossa plataforma seja independente de dispositivos e de tipos de dados, respeitámos na sua implementação algumas das especificações SWE da OGC para disponibilização de sensores na *Web*. Desta forma apresentamos o MuFFIN como uma arquitectura genérica, capaz de comunicar com outras plataformas e com as aplicações cliente, através de uma especificação criada para o efeito.

No próximo capítulo apresentamos em pormenor a arquitectura do MuFFIN e os serviços que este disponibiliza às aplicações cliente.

Capítulo 3

O MuFFIN

Este capítulo apresenta as decisões de desenho, o modelo de dados e a arquitectura do nosso *middleware*, abordando as decisões tomadas para cumprir os objectivos com os quais nos comprometemos.

3.1 Arquitectura

Com o objectivo de criar um sistema modular, consistente e de fácil manutenção e utilização, baseámos o desenho da nossa aplicação numa SOA composta por sete módulos independentes, que disponibilizam serviços entre si. O sistema que criámos fornece também as propriedades de uma SODA, disponibilizando como serviços na *Web* as funcionalidades dos dispositivos que este gere. Esta decisão arquitectural foi tomada com a convicção de ser uma solução que respeita os requisitos apresentados anteriormente. A independência entre módulos é facilitada pela arquitectura OSGi do ESB, que escolhemos para executar a nossa aplicação, onde cada módulo disponibiliza uma interface que os restantes utilizam. Um módulo não depende do processamento específico dos restantes, mas do que a interface destes disponibiliza. Desta forma, o processamento de cada módulo é encapsulado nos serviços que este oferece. Se um módulo for trocado por outro que cumpra a mesma especificação, não oferecendo menos funcionalidades e não exigindo restrições adicionais, o MuFFIN continua com o seu processamento normal.

As Camadas Lógicas do MuFFIN A Figura 3.1 mostra o diagrama de camadas do nosso *middleware*, representando a comunicação entre componentes, onde é visível que disponibilizamos duas camadas de serviços para comunicação. Estas camadas são implementadas pelo módulo *ThingsGateway*, que suporta a comunicação com redes de objectos inteligentes (como por exemplo, redes de sensores) e pelo *WS-Gateway*, que implementa a comunicação com aplicações de alto nível através de serviços na *Web*. O nosso sistema foi desenhado para suportar dois tipo de comunicação: (1) comunicação síncrona nos serviços de pesquisa; e (2) comunicação assíncrona, em que o cliente subscreve

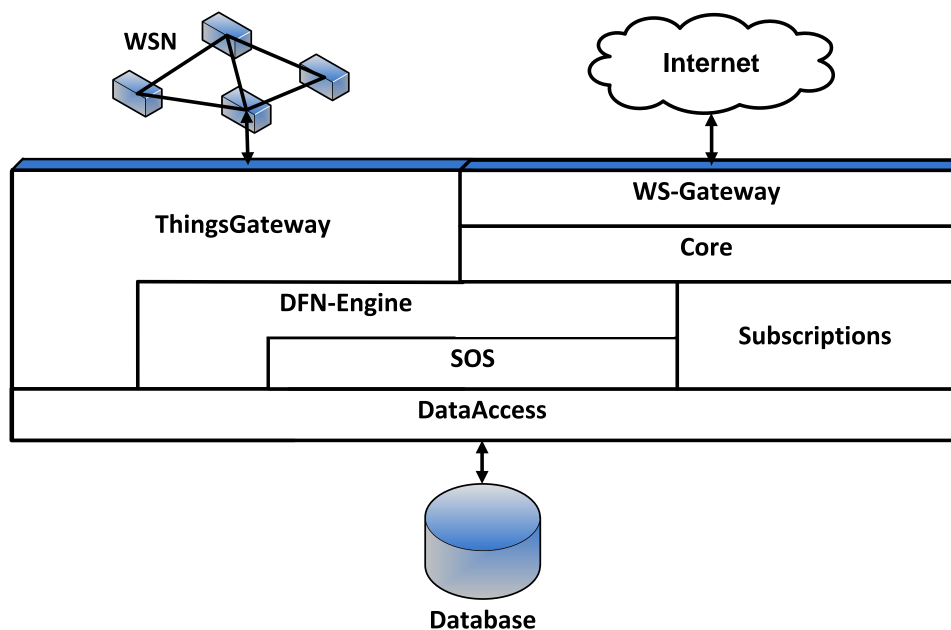


Figura 3.1: Arquitectura do *middleware* através de um diagrama de camadas.

um serviço e recebe notificações desencadeadas por eventos ocorridos no *middleware*. O módulo *Core* disponibiliza a implementação dos serviços *Web*, invocando as operações dos restantes módulos. O módulo *DFN-Engine* tem como responsabilidade gerir os filtros enviados pelas aplicações cliente e a cadeia de dependências entre estes. Este módulo instancia os filtros dos clientes e cria as ligações PubSub necessárias à sua comunicação. Os detalhes da instalação desta cadeia de dependências é apresentada na Secção 3.5. O módulo *Subscriptions* recebe as subscrições dos clientes, processa os pedidos e guarda a informação do subscritor, sendo também responsável pela publicação na *Web* das notificações para os clientes. O processamento das operações do padrão SOS e dos seus documentos XML são delegadas no módulo *SOS*. Os serviços disponibilizados neste módulo seguem as especificação da OGC, apresentadas em detalhe no Capítulo 4. Na base do nosso *middleware* encontra-se o módulo *DataAccess* que oferece aos restantes módulos um conjunto de fachadas para acesso aos dados da camada de persistência. As tabelas relacionadas com os objectos inteligentes são baseadas na especificação O&M.

3.2 Modelo de Dados

A Figura 3.2 apresenta a versão simplificada do modelo de base de dados da camada de persistência, contendo as entidades de maior relevância. Este modelo é inspirado na especificação O&M do padrão SWE, à qual acrescentamos as tabelas específicas do nosso *middleware*. Cada observação (tabela *observation*) guarda os dados relacionados com as leituras do ambiente onde a rede de dispositivos está instalada. A mesma observação pode registar vários fenómenos, sendo guardado apenas um valor para cada um destes. A

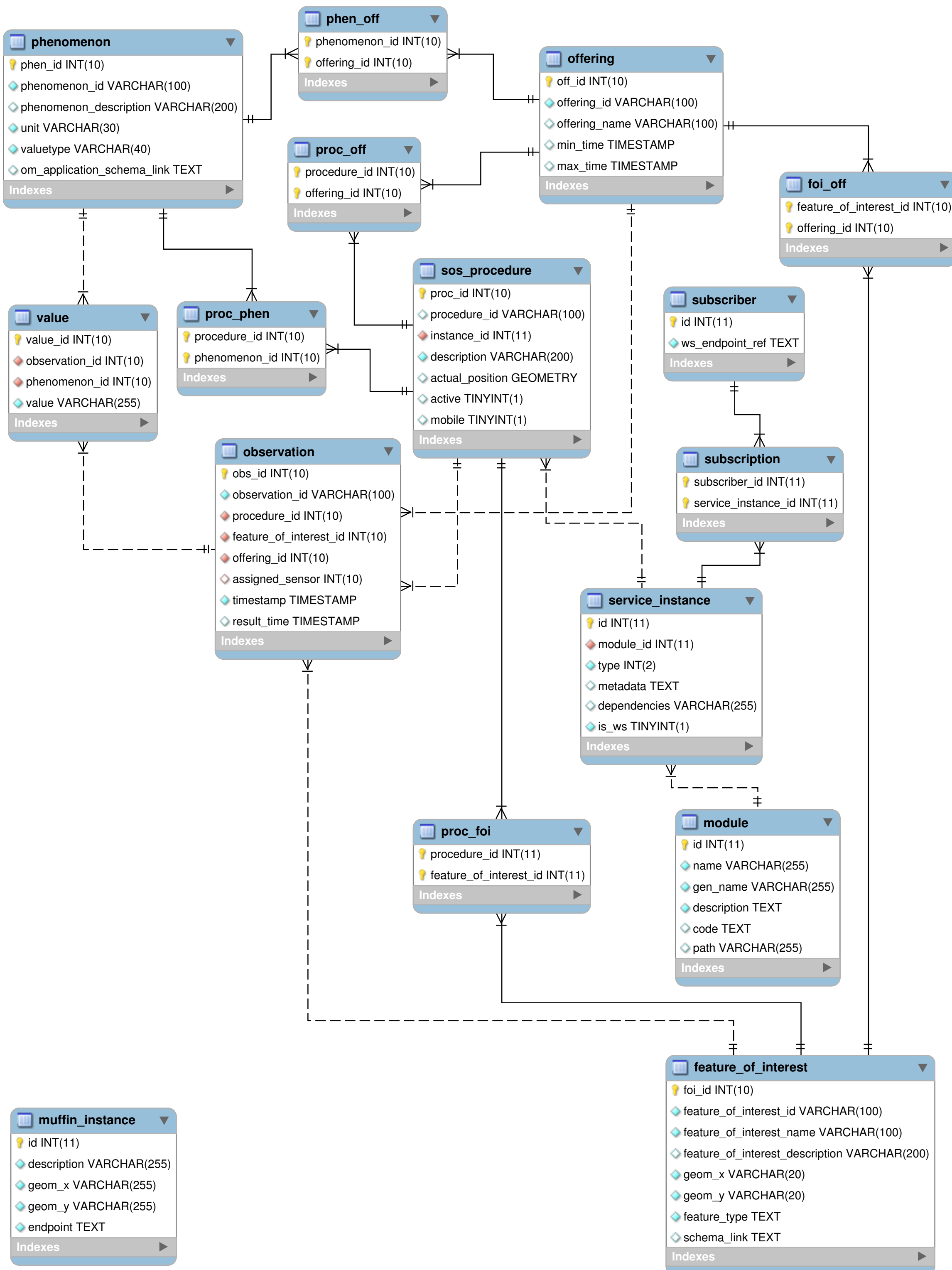


Figura 3.2: Diagrama entidade relacional simplificado da base de dados do *MuFFIN*.

relação (de muitos para muitos) entre a observação e a tabela *phenomenon* é feita através da tabela *value*, onde ficam realmente guardados os valores lidos do ambiente. Cada registo da tabela *observation* guarda a origem dos dados (através da tabela *sos_procedure*), a característica de interesse (tabela *feature_of_interess*) e em que oferta (tabela *offering*) se enquadra. Para além destas relações é também registada a hora a que a leitura foi efectuada. Os módulos enviados pelo cliente e os *gateways* instalados ficam registados na tabela *module*. Para cada instância destes é registado um novo serviço na tabela *service_instance*, onde é também guardado a especificação XML enviada pelo cliente para instanciar os filtros (campo *metadata*). As tabelas *subscriber* e *subscription* guardam, respectivamente, os dados do subscritor e a relação deste com o serviço subscrito. O MuFFIN está preparado para fazer parte de um sistema complexo composto por várias instâncias. A tabela *muffin_instance* guarda os dados de cada instância do nosso *middleware*, sendo especialmente útil o campo *endpoint*, a referência na *Web* utilizada na comunicação entre instâncias.

3.3 Abordagem

Um dos objectivos apresentados na Secção 1.2 é permitir que os clientes do nosso sistema manipulem dados recolhidos de diferentes redes de dispositivos inteligentes. Para tal, o nosso *middleware* não depende de um tipo específico de dispositivo, sendo parametrizado com pontos de acesso que funcionam como adaptadores para as várias redes. Estes pontos de acesso (*gateways*) respeitam uma interface com duas operações: (1) uma operação de arranque do *gateway*, para que este fique à escuta de mensagens da rede; (2) uma operação para envio de código, que permite reprogramar as funcionalidades dos dispositivos. A Figura 3.3 apresenta um *gateway* como o ponto de entrada e de saída de dados do *middleware* para a rede de dispositivos, sendo este executado no âmbito do módulo *ThingsGateway*.

Para cumprir o objectivo de programar dispositivos inteligentes através do processamento dos seus fluxos de dados, disponibilizamos uma interface que permite instalar pequenos módulos, que funcionam como filtros numa cadeia de manipulação de dados. A interface separa os conceitos de *instalação* e *instanciação*, permitindo: (1) instalar módulos provenientes de clientes e (2) instanciá-los e organizá-los consoante o objectivo do cliente. Desta forma os clientes podem instanciar o mesmo módulo com diferentes objectivos. A Figura 3.4 mostra um exemplo onde existe um módulo (instância *M1*) que publica de cinco em cinco minutos a média dos valores provenientes de uma rede de dispositivos. O módulo já foi instanciado uma vez para receber dados do *Gateway-1* (*M1 i1*), porém o cliente pode voltar a instanciá-lo (*M1 i2*), ligando-o a outra origem de dados (neste caso o *Gateway-2*).

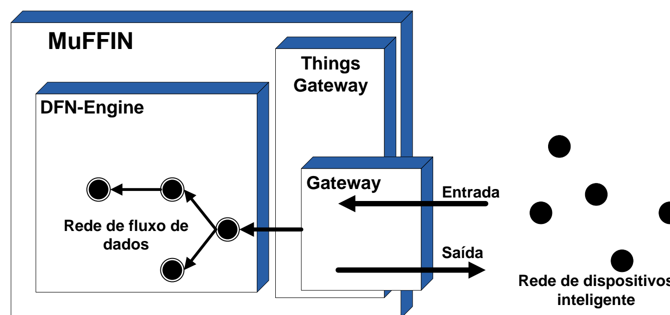


Figura 3.3: Exemplo de *ponto de acesso* utilizado na comunicação com uma rede de dispositivos.

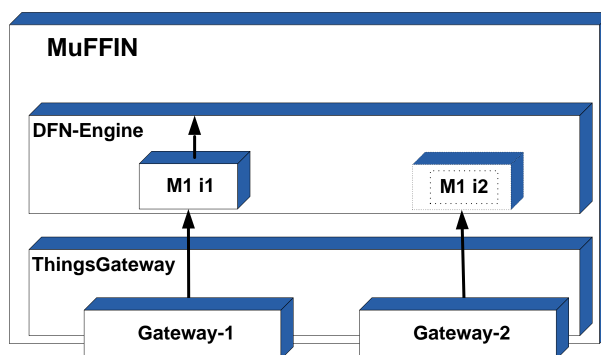


Figura 3.4: Instanciação do módulo *M1* para objectivos diferentes (*M1 i1* e *M1 i2*).

A nossa plataforma foi pensada de forma a permitir a interacção entre várias instâncias, de forma a disponibilizar aos clientes dados oriundos de diferentes plataformas instaladas em vários pontos do globo, reduzindo a troca de mensagens entre o cliente e os centros de observação. A interacção pode ser feita de duas formas: (1) acedendo independentemente a cada instância ou (2) acedendo a um centro de observação que age como orquestrador das invocações às restantes, caso (ideal) onde a comunicação entre instâncias é transparente para o cliente. A integração com outros tipos de plataformas é possível desde que estas suportem o padrão SOS, criando um sistema distribuído heterogéneo onde a comunicação com o nosso *middleware* é feita através de um *gateway* específico. Para que as instâncias do MuFFIN comuniquem entre si de forma a partilhar recursos e dados, é necessário configurar uma das instâncias com as referências na *Web* das restantes. Assim, a instância que guarda as referências é responsável por orquestrar as invocações às restantes, não sendo necessário criar serviços especiais para o efeito. Quando um cliente quer invocar determinado serviço terá de enviar o identificador da instância à qual se destina a invocação, sendo responsabilidade do orquestrador redireccionar o pedido, devolvendo a resposta ao cliente. A ligação entre as instâncias é feita dinamicamente através de pontos de acesso especificados através do padrão *WS-Addressing* [55], que padroniza os dados para referenciar ligações futuras.

3.4 Serviços Disponibilizados

De seguida apresentamos as operações disponibilizadas às aplicações de alto nível, nossas clientes, para que estas interajam com o MuFFIN:

***deployModule*:** permite que os clientes instalem os módulos que servirão como filtros na cadeia de manipulação de dados. A Figura 3.5 apresenta a sequência de invocações desta operação. Quando é recebido um pedido, o *WS-Gateway* invoca a implementação deste serviço no *Core*, de seguida o *DFN-Engine* regista o novo módulo na camada de persistência, ficando pronto para ser instanciado. O retorno do serviço é o identificador do módulo no sistema, útil para que o cliente possa de seguida instanciá-lo.

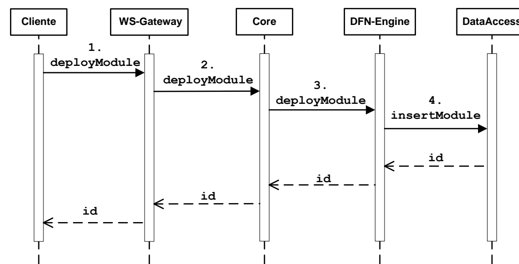


Figura 3.5: Diagrama de sequência do serviço *deployModule*.

***instantiateService* (Criação de serviços):** permite instanciar e orquestrar módulos, criando filtros que poderão ser disponibilizados como serviços na *Web*. A Figura 3.6 apresenta o fluxo de invocações desta operação. O módulo *DFN-Engine* é o responsável pelo processamento da especificação do cliente, pedindo os módulos a instanciar ao *DataAccess* e instanciando-os, criando as dependências necessárias para respeitar o pedido do cliente e para associar os novos filtros na cadeia já existente. Para finalizar, o *DFN-Engine* regista os novos filtros com a indicação se estes são disponibilizados como serviços na *Web*.

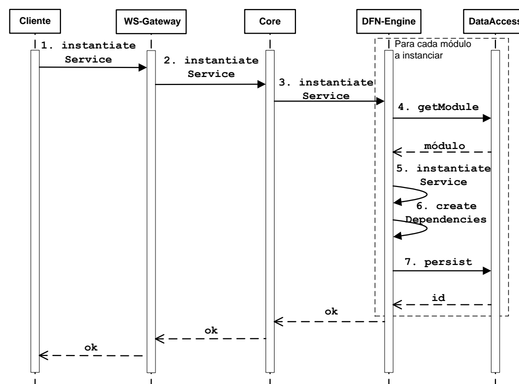


Figura 3.6: Diagrama de sequência do serviço *instantiateService*.

A Figura 3.7 mostra a relação conceptual entre os módulos e a sua representação como serviço na *Web*. No cenário apresentado, apenas os módulos 1, 2, e n estão disponíveis para subscrições *Web*. Os módulos 3 e 4 apenas computam acções intermédias que serão refinadas e disponibilizadas posteriormente. O módulo 4 exemplifica um ponto de extensão que pode ser refinado mais tarde.

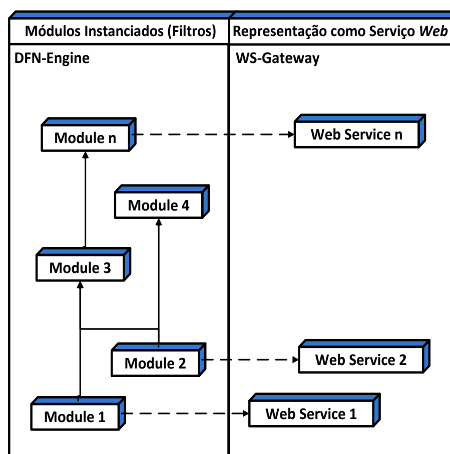


Figura 3.7: Relação conceptual entre os filtros a sua representação na *Web*.

***installGateway*:** permite configurar um novo ponto de acesso para uma rede de objectos inteligentes. Esta operação recebe o novo *gateway* e instala-o, ficando pronto para fazer parte da DFN como filtro base dos dados recebidos da nova rede. A Figura 3.8 mostra a sequência de invocações, onde é visível que o *ThingsGateway* é responsável por guardar o *gateway* na directoria correspondente e persistir a informação relacionada com o filtro, invocando o módulo *DataAccess*.

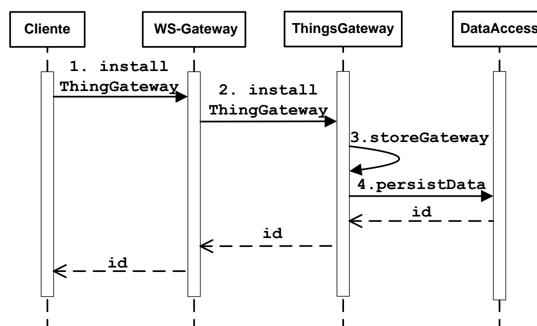


Figura 3.8: Diagrama de sequência do serviço *InstallGateway*.

***insertObservation*:** este serviço fornece às restantes plataformas a operação para inserirem as suas observações no nosso sistema, desde que estas respeitem a especificação SOS. A Figura 3.9 específica o fluxo de operações entre os módulos, onde é visível que o módulo *SOS* é responsável pelo processamento de documentos XML da especificação

SOS. O objectivo é recolher do documento XML os dados das observações e enviá-los para o *DataAccess* para que este os persista. Como implementámos a especificação SWE, a resposta deste serviço é um documento XML onde se encontra o identificador da observação inserida. Este método para recolha de observações tem como objectivo complementar o já existente, disponibilizado pelos *pontos de acesso* ligados às redes de dispositivos inteligentes.

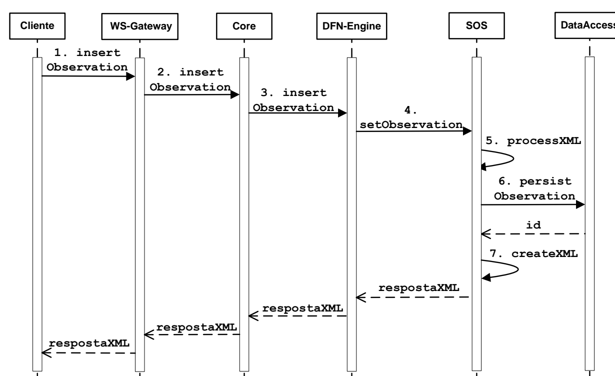


Figura 3.9: Diagrama de sequência do serviço *insertObservation*.

***getCapabilities*:** operação que respeita a especificação do método *GetCapabilities* do padrão SOS. Esta operação tem como objectivo apresentar aos clientes a lista de propriedades SWE que o MuFFIN processa, tais como o conjunto de *Fenómenos*, *Ofertas* e *Características de Interesse* das observações. A Figura 3.10 mostra a relação entre módulos no processamento da invocação: o módulo *SOS* recolhe as propriedades SWE através das *Ofertas* existentes na camada de persistência, acrescentando informação específica da configuração do *middleware*. Por fim, é gerado o documento XML que é devolvido ao cliente.

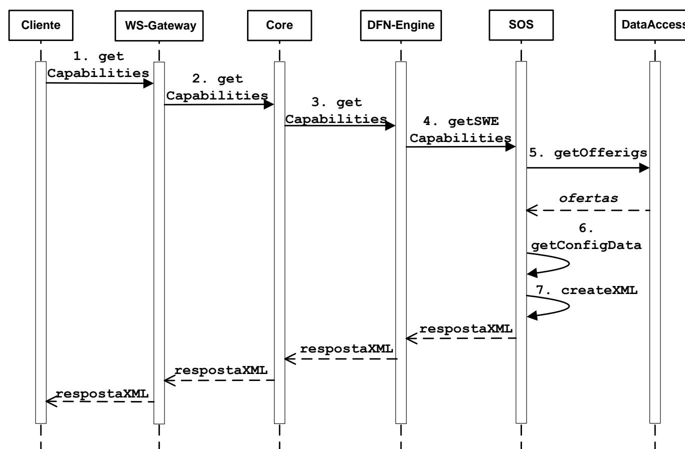


Figura 3.10: Diagrama de sequência do serviço*getCapabilitites*.

***readObservation*:** serviço síncrono especificado na operação *GetObservations* do padrão SOS, que devolve um conjunto de observações que respeitam as restrições enviadas pelo cliente. A Figura 3.11 apresenta o fluxo de invocações desta operação, desde a recepção das restrições do cliente no módulo *WS-Gateway*, até ao processamento destas por parte do módulo *SOS*. As restrições do cliente, em formato XML, são processadas e representadas numa pesquisa à camada de dados. O resultado é formado pelos dados das observações que serão enviados de volta para o cliente num documento XML que respeita a especificação SWE.

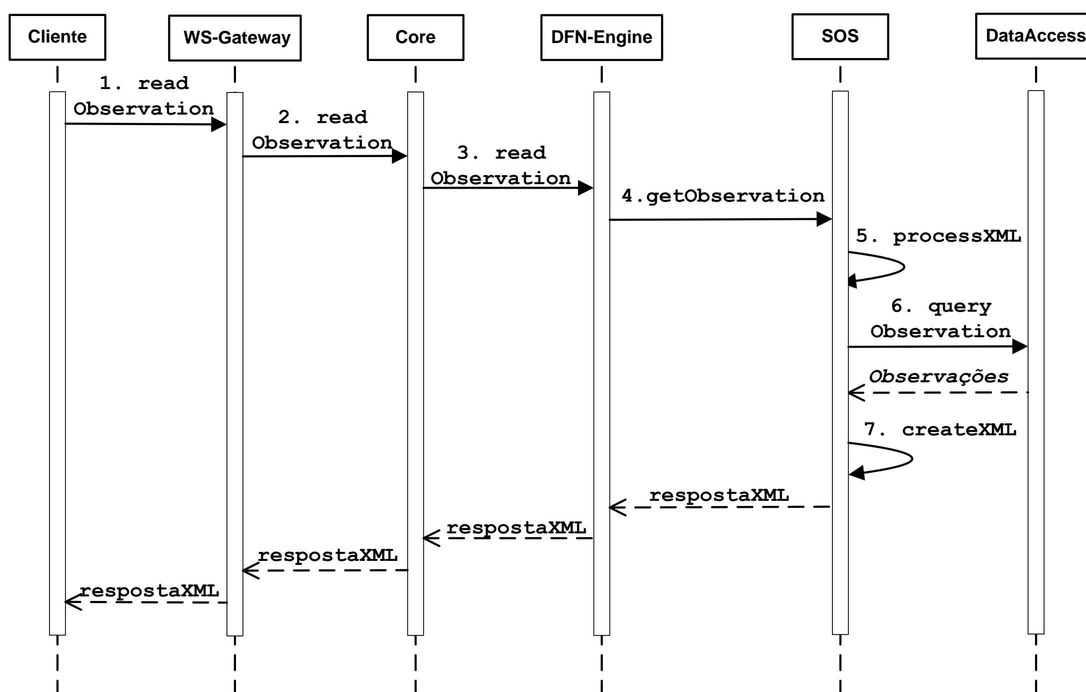


Figura 3.11: Diagrama de sequência do serviço *readObservation*.

***getModulesInfo*:** retorna as informações necessárias à interacção com os módulos, com os serviços e com as restantes instâncias do nosso *middleware*. A Listagem 3.1 apresenta um retorno possível desta operação, contendo: (1) as instâncias do *middleware*, no exemplo apenas uma com o identificador *lx1* (informação na linha 2); (2) os módulos instalados em cada instância (da linha 3 à linha 16), no exemplo com os identificadores *LisbonSink* e *M1*; (3) os serviços instanciados e as suas dependências (da linha 17 à linha 29), neste cenário o serviço *LisbonSink-i1* instanciado a partir do módulo *LisbonSink* (sem dependências) e *M1-i1* instanciado a partir do módulo *M1* (dependendo de *LisbonSink-i1*). A Figura 3.12 apresenta as invocações necessárias a esta operação. O módulo *DFN-Engine* é o responsável pela recolha dos dados relacionados com os módulos e serviços existentes para que este construa o XML de resposta que respeita a especificação SWE.

Listagem 3.1: Documento XML com informação dos módulos existentes no *middleware*.

```xml
<muffin-contents>
    <muffin-instance description="Lisbon Networks" id="lx1">
      <modules>
        <module id="LisbonSink">
          <name>Gateway-1</name>
          <description>
            Gateway para a rede de dispositivos de Lisboa
          </description>
        </module>
        <module id="M1">
          <name>M1</name>
          <description>
            Agrega 10 temperaturas e de seguida publica-as
          </description>
        </module>
      </modules>
      <services>
        <service isWS="false"
          moduleId="LisbonSink" id="LisbonSink-i1">
          <description/>
          <dependencies/>
        </service>
        <service isWS="true" moduleId="M1" id="M1-i1">
          <description>
            Processa os dados recebidos do LisbonSink
          </description>
          <dependencies>LisbonSink-i1</dependencies>
        </service>
      </services>
    </muffin-instance>
</muffin-contents>
```

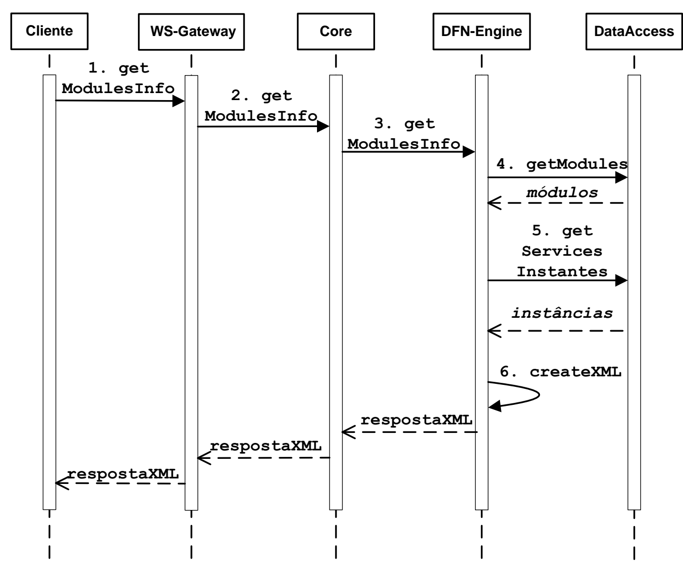


Figura 3.12: Diagrama de sequência do serviço *getModulesInfo*.

***subscribeService*:** regista a aplicação cliente como subscritor dos resultados dos filtros instanciados. O nosso sistema recebe o identificador do serviço a subscrever e a referência para o serviço na *Web* onde o cliente espera receber as notificações. Esta referência, especificada através do padrão *WS-Addressing*, serve também como identificador único do

subscritor, não sendo esperado que clientes diferentes aguardem respostas na mesma referência. A Figura 3.13 mostra como os dados da subscrição são processados: o módulo *subscription* é responsável por gerir a relação entre os subscritores e os serviços, invocando a operação *persistSubscription* para registar a subscrição. Quando o filtro subscrito processa informação, o módulo *DFN-Engine* gera um evento de forma a guardar o processamento resultante e (caso o filtro possua uma representação como serviço *Web*) notificar o módulo *subscriptions* para que os dados sejam publicados no subscritor interessado.

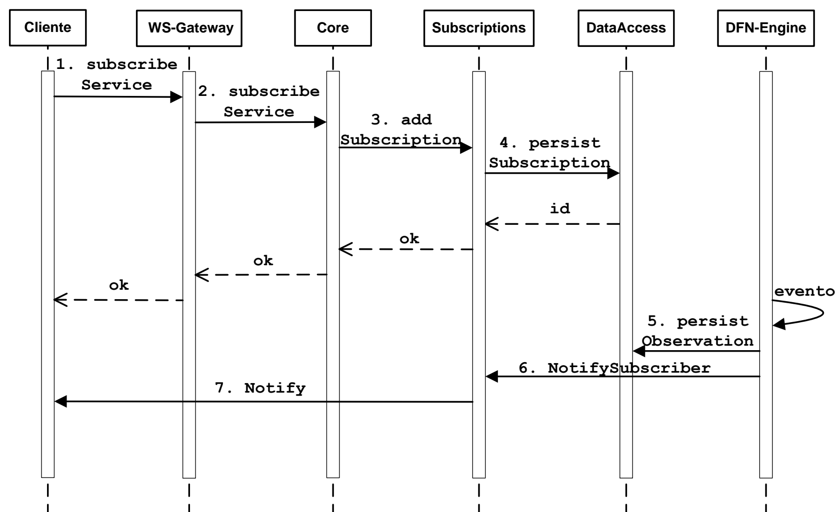


Figura 3.13: Diagrama de sequência do serviço *subscribeService*.

unsubscribeService: operação inversa da anterior. Permite que o cliente cancele a subscrição de um serviço, fornecendo os mesmos dados utilizados na subscrição. A Figura 3.14 apresenta o fluxo de invocações em tudo semelhante à operação de subscrição, terminando com a persistência do seu cancelamento.

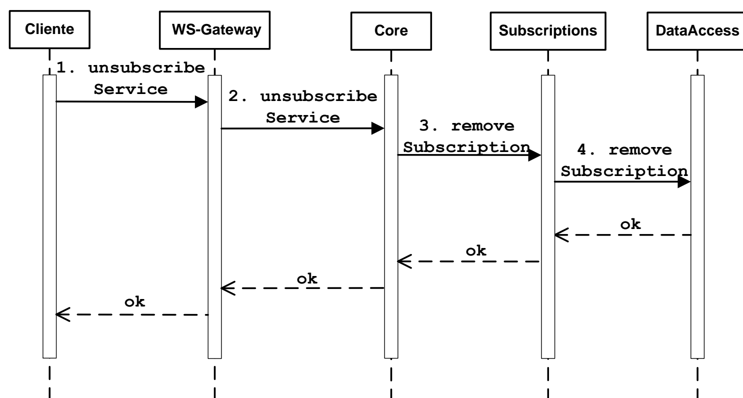


Figura 3.14: Diagrama de sequência do serviço *unsubscribeService*.

deployCode: permite às aplicações cliente instalarem fisicamente novas instruções nas redes de objectos inteligentes, recebendo como argumentos o identificador único do ponto de acesso responsável pela rede e o novo código a instalar. A Figura 3.15 ilustra o fluxo de execução onde é visível que o *ThingsGateway*, como gestor da comunicação com os dispositivos, é responsável por procurar o ponto de acesso à rede que se pretende reprogramar e por fim enviar o código para os dispositivos através deste.

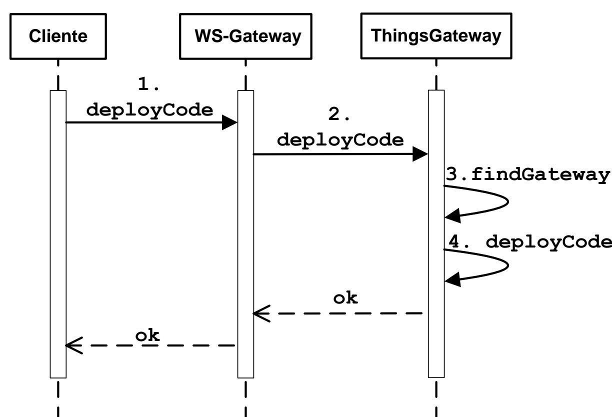


Figura 3.15: Diagrama de sequência do serviço *deployCode*.

A informação recebida e gerada nestas operações fica guardada na camada de dados do *middleware*, estando o sistema preparado para recuperar o seu estado em caso de falha. A recuperação é possível porque a cada arranque o sistema instancia em primeiro lugar os *pontos de acesso* (porque estes são a base da DFN) e de seguida os módulos, criando os filtros pela ordem em que estes foram instanciados pela primeira vez.

3.5 Programação de Redes e Gestão de Dados

Esta secção pormenoriza a forma como o nosso *middleware* gere os dados recebidos das redes correspondentes aos *gateways* instalados e como permite a reprogramação dos seus dispositivos.

O MuFFIN é configurável de forma a enviar novo código para os dispositivos da rede se estes o suportarem, caso contrário o código é instanciado localmente e é executado directamente na camada de *middleware*. Nesta última abordagem o cliente necessita de especificar que módulos pretende instanciar e as dependências entre estes. Esta especificação é feita através de um documento XML, onde se pode indicar que o resultado do processamento do filtro ficará disponível como serviço na *Web*. Quem instancia os filtros decide se quer expor o resultado na *Web*, ou em alternativa usar o resultado como processamento intermédio no fluxo de dados.

A Listagem 3.2 apresenta o documento XML utilizado para instanciar o módulo com o identificador *TemperaturaDiff* (linha 2). O utilizador decide expor o resultado na *Web* (através da opção *isWS* apresentado na linha 2) e especifica que os filtros *LisbonTemperature* e *PortoTemperature* são as dependências deste (da linha 7 à 10).

Listagem 3.2: Um exemplo de documento XML para instanciar um filtro.

```
1  <deploy>
2    <instance moduleId="TemperaturaDiff" isWS="true">
3      <description>
4        Recebe as temperaturas de Lisboa e do porto,
5        subtraindo-as e devolvendo o resultado.
6      </description>
7      <dependencies>
8        <dependency serviceId="LisbonTemperature"/>
9        <dependency serviceId="PortoTemperature"/>
10     </dependencies>
11   </instance>
12 </deploy>
```

Os módulos instanciados (filtros) e os eventos desencadeados criam uma cadeia de dependências, tal como uma rede publicador-subscritor, como ilustrado na Figura 3.16. Esta cadeia resulta numa Rede de Fluxo de Dados (DFN, do inglês *Data-Flow Network*), onde os dados em bruto, recebidos da rede de dispositivos, são processados para produzir a informação subscrita pelas aplicações de alto nível. Esta DFN é o método disponibilizado pelo *middleware* para reprogramar as redes, sem que seja necessário instalar código nos dispositivos, sendo apenas manipulados os dados que interessam aos clientes. Porém, os dois métodos de programação disponibilizados (DFN e envio de código para os dispositivos) não são mutuamente exclusivos, podendo os clientes do MuFFIN instalarem código nos nós da rede e ainda continuarem a utilizar a DFN no filtro dos dados recebidos.

A Figura 3.16 ilustra a interacção com duas redes de objectos inteligentes, acedidas através dos pontos de acesso *LisbonSink* e *PortoSink*, recebendo estes a informação em *bruto* directamente das redes. Sempre que estão disponíveis novos dados, estes pontos de acesso notificam os seus subscritores que processam os dados recebidos, guardam-os na camada de persistência e, por fim, notificam os seus subscritores. Este processamento termina quando já não houver filtros subscritos. No caso de um dos filtros possuir uma representação na *Web*, os subscritores remotos (aplicações cliente do sistema) serão também notificados. No exemplo apresentado, é visível uma DFN com dois pontos de acesso e quatro filtros: *LisbonSink* e *PortoSink*, que representam pontos de acesso a redes de dispositivos em Lisboa e no Porto; *LisbonTemperature* e *PortoTemperature*, filtros de dados que recebem vários tipos de observações e apenas encaminham as relacionadas com a temperatura; *LisbonHumidity*, filtro que apenas encaminha leituras de humidade; *TemperatureDiff*, que calcula a diferença entre leituras de temperatura recebidas de locais diferentes. O comportamento ilustrado é: o módulo *LisbonSink* notifica os módulos *LisbonTemperature* e *LisbonHumidity*. Por seu turno, *LisbonTemperature* irá informar *TemperatureDiff*, que, dependente do valor recebido do módulo *PortoTemperature*, irá computar a diferença entre as temperaturas e disponibilizá-las aos seus subscritores.

Se nenhum valor for disponibilizado pelo *PortoTemperature*, *TemperatureDiff* ficará suspenso até receber todos os dados que necessita. Como cada módulo guarda a informação processada, esta poderá ser acedida mais tarde através de pesquisas síncronas (serviço *readObservation*).

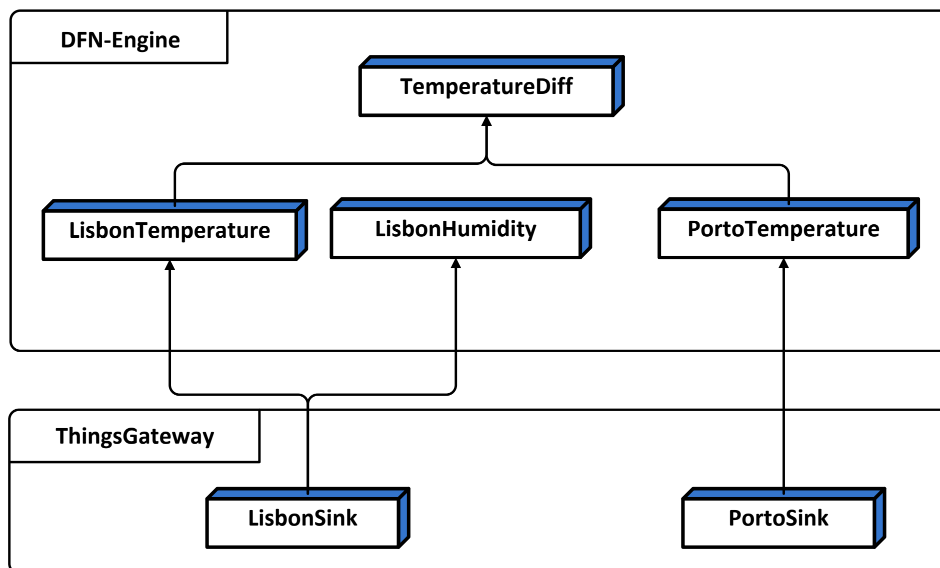


Figura 3.16: Exemplo de cadeia de dependências gerada na camada de middleware.

3.6 Considerações Finais

Neste capítulo pormenorizámos a arquitectura e o modelo de dados do MuFFIN. Apresentámos também os seus principais módulos: (1) *ThingsGateway*, para a comunicação com redes de dispositivos inteligentes; (2) *WS-Gateway*, para a comunicação com aplicações clientes através de serviços na *Web*; (3) *Core*, como orquestrador das invocações aos restantes módulos; (4) *DFN-Engine*, para gestão dos módulos instalados pelos clientes, utilizados na cadeia de manipulação de dados; (5) *SOS*, para processamento dos dados relacionados com a especificação SWE; (6) *Subscriptions*, para gestão das subscrições das aplicações cliente e notificação das mesmas através da *Web*; e por fim (7) *DataAccess*, para gestão da camada de persistência do nosso *middleware*.

Com os seus módulos, o MuFFIN disponibiliza: serviços na *Web* para instalar e instanciar módulos capazes de processar os dados das redes de sensores; serviços para instalar pontos de acesso com o objectivo de parametrizar a comunicação com diferentes tipos de redes; serviços para inserir e ler observações que respeitam a especificação SWE; e ainda serviços que permitem às aplicações cliente fazerem parte de uma rede PubSub na *Web* relacionada com os filtros de dados do MuFFIN.

O MuFFIN é capaz de reprogramar redes de dispositivos inteligentes através de dois métodos que se podem complementar. Caso os dispositivos suportem a instalação de

código, o nosso *middleware* disponibiliza serviços para o efeito; se não for possível, o MuFFIN suporta a programação dos fluxos de dados recebidos, através de uma DFN composta por filtros resultantes da instanciação de módulos enviados pelas aplicações cliente.

No capítulo que se segue são apresentados os detalhes dos módulos da nossa plataforma e dos padrões SWE que implementámos.

Capítulo 4

Implementação

Neste capítulo apresentamos em detalhe cada módulo do MuFFIN, dando ênfase às ferramentas mais importantes para a implementação e execução da plataforma. Introduzimos também exemplos de documentos XML, que respeitam o padrão SWE, e explicamos como possibilitamos a instalação de módulos que poderão ser instanciados como pontos de acesso e filtros no nosso *middleware*.

4.1 Tecnologias Utilizadas

A nossa plataforma foi implementada na linguagem de programação Java [42]. Esta decisão teve em conta as vantagens das linguagens orientadas a objectos, em especial o Java pelo suporte e documentação que possui e por ser independente da plataforma onde é executado, permitindo que o MuFFIN seja instalada nos mais diversos sistemas.

Escolhemos como servidor aplicacional o *Fuse Enterprise Service Bus* [23], baseado no *Apache ServiceMix* [7]. O Fuse ESB implementa as funcionalidades OSGi, permitindo uma completa e dinâmica separação dos componentes do nosso sistema. A vantagem deste servidor sobre os restantes ESBs é a documentação disponibilizada e o suporte existente no seu sítio na *Web*. Como o Fuse ESB é baseado no ESB da *Apache Software Foundation*, este integra várias ferramentas que nos foram úteis no decorrer da implementação. Entre estas tecnologias encontram-se:

- ***Apache ActiveMQ [3]*:** implementa o *Java Message Service* — padrão de comunicação para componentes J2EE (do inglês, *Java 2 Enterprise Edition*) — apresentando-se como um intermediário da comunicação através da gestão de filas de mensagens e de tópicos de interesse, utilizados para definir a comunicação entre filtros da cadeia de fluxo de dados e entre alguns módulos da plataforma. As filas são utilizadas na comunicação *um-para-um*, garantindo a entrega das mensagens, mesmo quando um dos intervenientes não está disponível no momento da transmissão. Os tópicos são utilizados numa comunicação PubSub (*um-para-muitos*), não garantindo (por omissão) a entrega das mensagens a todos os subscritores que

não estejam à escuta no momento do envio. Para garantir a entrega de mensagens, mesmo quando o subscritor não esta activo no momento, é necessário criar subscritores *duráveis*, num tipo de comunicação semelhante às filas, mas com uma gestão mais complexa. Na nossa plataforma as filas são utilizadas pelo módulo *DFN-Engine* para notificar o módulo *Subscriptions* aquando de novos eventos, sendo os tópicos utilizados na organização dos filtros da DFN.

- ***Apache XML Beans [8]*:** ferramenta que cria representações de esquemas XML em classes Java. Verificámos que esta ferramenta é mais versátil em relação às restantes opções testadas, entre estas o *Apache Axis2* [4] utilizado em bibliotecas Java para serviços na *Web*, sendo a única capaz de resolver as dependências dos esquemas dos padrões SWE.

- ***Apache CXF [5]*:** ferramenta que implementa a especificação JAX-WS [16] para construção de serviços na *Web* Java através da anotação das classes a apresentar como serviços. É através do CXF que disponibilizamos os serviços *Web*, delegando neste a criação dos ficheiros WSDL e a construção dos canais de comunicação utilizados.

Como o Fuse ESB suporta a instalação e manutenção de módulos através do *Apache Maven [6]*—ferramenta de *software* para gestão de projectos e automatização de processos de desenvolvimento—é possível instalar outras ferramentas a partir de repositórios na *Web*. Foi através de um repositório Maven local que instalámos os módulos que compõem a nossa plataforma, tornando o MuFFIN num sistema complexo, com módulos de várias fontes.

Foi através do Maven que instalámos a ferramenta *JBoss Hibernate [29]*, utilizada para o acesso e gestão dos dados contidos na camada de persistência. O *Hibernate* implementa o *Java Persistence API*, representando as entidades e as relações existentes na base de dados em classes Java, oferecendo uma camada de abstracção do modelo de domínio. Como esta ferramenta abstrai o tipo de base de dados utilizada, através das entidades Java e de uma linguagem para pesquisas própria (o HQL, do inglês Hibernate Query Language), é possível gerir a camada de persistência de forma transparente, podendo alterar o Sistema Gestor de Base de Dados (SGBD) sem ter de alterar a camada de negócio da nossa plataforma.

4.2 Tipos de Comunicação Suportadas

O nosso *middleware* suporta comunicação síncrona e assíncrona com o mundo (aplicações cliente), implementando também estas duas metodologias na comunicação entre algumas das suas componentes. Os objectivos de cada tipo de comunicação, no que diz respeito às aplicações cliente, são:

- **Comunicação síncrona:** permite que os clientes invoquem os serviços *Web* da plataforma de forma a obterem na resposta a informação desejada. Para minimizar a comunicação necessária entre o MuFFIN e as aplicações cliente, estes serviços retornam informação chave que pode ser utilizada nas invocações seguintes. Por exemplo, é importante que as aplicações de alto nível recebam na resposta o identificador do módulo que instalaram na nossa plataforma (serviço *deployModule*) para que de seguida consigam instanciar o filtro correspondente, ou que recebam as informações sobre os filtros existentes no *middleware* (serviço *getModulesInfo*). Mas este tipo de comunicação é essencial no serviço *readObservation*, permitindo a recepção das observações ocorridas no passado e que foram guardadas no *middleware*.

- **Comunicação assíncrona:** permite que as aplicações cliente do MuFFIN recebam notificações com dados relacionadas com as suas subscrições. Assim, as aplicações cliente invocam o serviço de subscrição (*subscribeService*) apenas uma vez, recebendo ao longo do tempo notificações de interesse, diminuindo a comunicação entre aplicações cliente e a nossa plataforma.

A comunicação entre os principais módulos do sistema segue também estes dois esquemas. Geralmente as respostas dos serviços dos módulos são síncronas, porém os módulos *DFN-Engine* e *Subscriptions* comunicam através de uma fila de mensagens, tirando partindo da gestão de mensagens da *ActiveMQ*. Assim, quando o *DFN-Engine* recebe notificações da DFN, verifica se o filtro que originou o evento possui representação na *Web* e, caso se verifique, redirecciona estes eventos para a fila de mensagens. A *ActiveMQ* fica encarregue de apresentar os eventos ao módulo *Subscriptions* quando este tem disponibilidade para os processar.

4.3 Os Módulos do *MuFFIN*

Estruturámos a nossa plataforma em módulos independentes que, como um todo, formam o *middleware* que estamos a apresentar. A Figura 4.1 ilustra a visão conceptual dos módulos que implementámos e que foram instalados no Fuse ESB. No Apêndice A apresentamos as interfaces dos módulos da nossa plataforma, sendo de seguida detalhados.

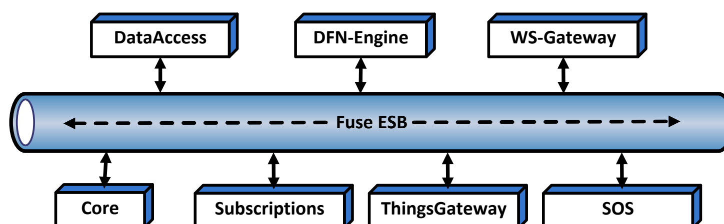


Figura 4.1: Visão conceptual dos principais módulos instalados no Fuse ESB.

ThingsGateway: permite o carregamento dinâmico de pontos de acesso que medeiam a comunicação entre os objectos inteligentes e a nossa plataforma. Estes pontos de acesso são a raiz da árvore da cadeia de fluxo de dados e são responsáveis por enviar o código para os dispositivos na sua reprogramação. Esta funcionalidade é possível porque o *middleware* recebe o código já compilado, facilitando o processo. Se o MuFFIN compilasse código, limitava a sua utilização às linguagens que suportava, deixando de ser uma plataforma genérica e tornando o *middleware* muito mais complexo.

O Apêndice A.1 apresenta a interface do *ThingsGateway*, ficando a operação *installThingGateway* encarregue de receber os novos pontos de acesso, de validar se estes têm as classes necessárias para serem executados e de registá-los na camada de persistência. Este módulo fica também responsável por iniciar, a cada arranque da plataforma, os pontos de acesso registados. A operação *deployCode* tem como objectivo instalar fisicamente o novo código nos dispositivos das redes que o *middleware* controla. O *ThingsGateway* tem uma estrutura de dados para gerir os pontos de acesso instalados no MuFFIN, tendo acesso directo às suas funcionalidades.

Nós possibilitamos a criação de novos pontos de acesso através de um biblioteca que implementámos para o efeito. Nesta biblioteca encontra-se a *AbstractThing*, que tem de ser estendida de forma a implementar o método *run*, invocado no arranque para que o ponto de acesso fique à escuta de mensagens enviadas pela rede de dispositivos; e o método *deployInNetwork*, que recebe como argumento o código já compilado e que é invocado pela plataforma na programação física dos dispositivos. É importante salientar que o método *installThingGateway* instala de imediato o ponto de acesso na plataforma, não sendo possível instanciar mais que uma vez o mesmo ponto de acesso. Esta decisão evita que duas instâncias iguais repetissem o envio dos dados para a DFN. Se ambos os pontos de acesso estivessem à escuta de mensagens oriundas da mesma rede, enviariam os mesmos dados duas vezes, repetindo o processamento de dados iguais.

WS-Gateway: responsável por apresentar as funcionalidades do *middleware* como serviços na *Web*. Respeitámos o protocolo para as notificações com serviços na *Web* (WS-N, do inglês *Web Service Notification* [57]) para notificar subscritores remotos. Quando uma aplicação de alto nível subscreve um serviço, deve enviar a referência para um ponto de acesso na *Web* onde as notificações podem ser publicadas. Quando um evento é desencadeado, a nossa *framework* liga-se dinamicamente ao ponto de acesso do cliente e envia a informação subscrita.

Os serviços *Web* do MuFFIN são implementados através do *Apache CXF*, anotando as classes Java, os seus métodos e os argumentos destes como @*WebService*, @*WebMethod* e @*WebParam*, respectivamente. Através da anotação @*WebService* definimos os argumentos *targetNamespace* (onde se encontra a definição das variáveis dos serviços), *serviceName* (o nome dado ao conjunto de serviços do *middleware*) e *portName* (o porto utili-

zado na comunicação SOAP). Utilizando a anotação @*WebMethod* definimos o nome das operações. Por fim, definimos os nomes dos argumentos das operações através do @*WebParam*. A informação contida nestas anotações é utilizada na geração da especificação WSDL, adicionando dados importantes para a criação dos clientes dos serviços do MuFFIN.

Na Listagem 4.1 apresentamos as configurações necessárias à integração dos serviços *Web* do MuFFIN no Fuse ESB. Cada módulo instalado no Fuse ESB necessita de um ficheiro de configuração, onde definimos as componentes (*beans*) utilizadas no seu processamento. Da linha 2 à linha 5 apresentamos as dependências do módulo CXF, a partir da linha 7 até à linha 10 configuramos os módulos *Core* e *ThingsGateway* como implementação dos serviços *Web*. Por fim, da linha 12 à linha 16 apresentamos as configurações específicas do JAX-WS, onde é feita a ligação entre a interface *Web* e a implementação existente nos módulos.

Listagem 4.1: Configuração da integração dos serviços *Web* do MuFFIN no Fuse ESB.

```
1  <beans>
2    <import resource="classpath:META-INF/cxf/cxf.xml" />
3    <import resource="classpath:META-INF/cxf/cxf-extension-http-jetty.xml" />
4    <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
5    <import resource="classpath:META-INF/cxf/cxf-extension-http-binding.xml" />
6
7    <bean class="MuFFIN_WS_Imple" id="muffin_WS_Imple">
8      <property name="core" ref="coreService" />
9      <property name="thingsGateway" ref="thingsGatewayService" />
10   </bean>
11
12   <jaxws:
13     endpoint id="MuFFIN_WS_EndPoint"
14     implementor="#muffin_WS_Imple"
15     address="http://localhost:9090/MuFFIN_WS"
16   />
17 </beans>
```

Core: responsável pela orquestração dos módulos utilizados na implementação dos serviços *Web* do MuFFIN. Este módulo depende directamente dos serviços prestados pelos módulos *Subscriptions* e *DFN-Engine*, redireccionando pedidos das aplicações de alto nível para estes módulos.

O módulo *Core* é também o orquestrador de pedidos entre instâncias do MuFFIN, variando a responsabilidade deste módulo de acordo com a responsabilidade da instância onde se encontra. Se uma das instâncias é definida como orquestradora, o seu *Core* possui uma estrutura de dados com as referências *Web* das restantes instâncias. Quando uma aplicação cliente deseja invocar serviços de uma determinada instância, terá de enviar o seu identificador que será utilizado na pesquisa da respectiva referência *Web*. O MuFFIN disponibiliza operações que são obrigatoriamente disseminadas por todas as instâncias (como por exemplo a *getModulesInfo*). Neste caso a instância orquestradora não só invoca as restantes, como ainda compila as várias respostas num só ficheiro XML, sendo este a resposta do serviço.

Subscriptions: gestor das subscrições dos serviços *Web* do MuFFIN, efectuadas pelas aplicações de alto nível, suas clientes. Este módulo é responsável por guardar os dados dos subscritores e notificá-los quando necessário. Para que a notificação dos subscritores seja possível, o módulo *Subscriptions* instancia as referências *Web* das aplicações cliente. Cada instanciação reflecte-se numa ligação *Web*, sendo uma operação que demora tempo variável consoante a ligação utilizada na comunicação entre o MuFFIN e os seus clientes. De forma a minimizar o tempo despendido em ligações *Web*, o *Subscriptions* possui uma *cache* que guarda as referências mais utilizadas. O tamanho da *cache* é configurável e, quando necessário, é eliminada a referência menos utilizada. Assim, este módulo primeiro procura na sua *cache* a referência desejada, caso esta não exista, é instanciada e guardada na *cache* para futuras utilizações.

DFN-Engine: responsável pela gestão da rede de fluxo de dados, utilizada para reprogramar os dados recebidos das redes de dispositivos. Este módulo recebe e instancia os módulos enviados pelas aplicações cliente, organizando a DFN através dos métodos de comunicação da *ActiveMQ*.

Os módulos recebidos implementam (à semelhança dos *pontos de acesso* do módulo *ThingsGateway*) o padrão *Command* [21], estendendo a classe *Service* existente na biblioteca que criámos para a utilização da DFN e rescrevendo o método *doAction*, para que este processe os dados recebidos. Este método é invocado quando um filtro é notificado com novos dados, tendo o processamento que terminar com a invocação do método *send* (existente na classe *Service*) para que os filtros seguintes sejam notificados e a DFN continue o seu processamento. A classe *Service* oferece também métodos que possibilitam o acesso às propriedades O&M das observações transferidas entre filtros. Entre estes o método *getFeatureOfInterest*, para aceder à *característica de interesse* da observação e o método *getRecord*, utilizado para aceder aos dados relacionados com os fenómenos lidos pelos dispositivos.

SOS: responsável pelo processamento dos documentos XML do padrão SOS através do *Apache XML Beans*. Este módulo possui os interpretadores XML necessários à implementação da especificação SOS. No Apêndice B disponibilizamos exemplos de documentos XML que respeitam esta especificação, onde é visível a utilização das propriedades apresentadas ao longo desta dissertação. O Apêndice B.1 é um exemplo de inserção de uma observação, enquanto que no Apêndice B.2 disponibilizamos exemplos relativos aos vários métodos para a leitura de observações: (1) através do identificador da observação (B.2.1); (2) através do identificador da *oferta* (B.2.2); e (3) através de um conjunto de filtros sobre várias propriedades (B.2.3).

DataAccess: disponibiliza as fachadas necessárias para o acesso à camada de persistência do *middleware*. Para isolar serviços, facilitando a manutenção de código, cada tipo de serviço utiliza uma interface diferente para o acesso à base de dados. Os Apêndices A.6, A.7 e A.8 apresentam, respectivamente, as interfaces utilizadas pelos módulos *SOS* (gestão de observações), *Subscriptions* e *DFN-Engine* (gestão de serviços) e *Things-Gateway* (gestão de *pontos de acesso*). Os dados são guardados numa base de dados MySQL [43] e o acesso é feito usando o *JBoss Hibernate*.

4.4 Pesquisa de Serviços Baseada em Ontologias

Esta secção apresenta o trabalho realizado na criação de uma ontologia com o objectivo de classificar os serviços existentes no nosso *middleware*, permitindo a sua pesquisa semântica. Apresentamos ainda a forma como os clientes poderão utilizar a OntoMuFFIN [13], melhorando a forma como estes interagem com o MuFFIN. A Figura 4.2 mostra o diagrama da OntoMuFFIN, contendo as entidades que compõem a ontologia e as relações entre estas.

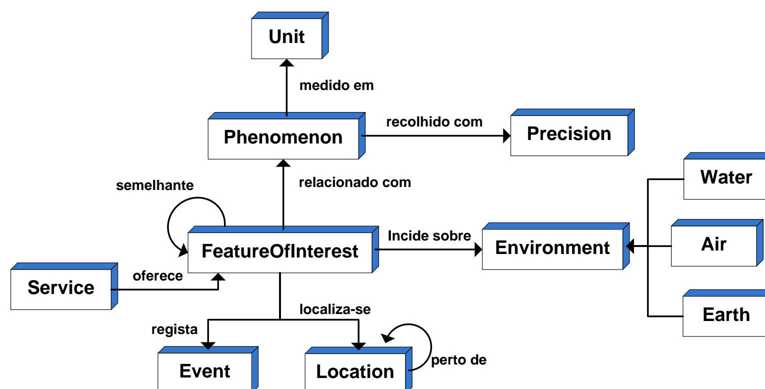


Figura 4.2: Diagrama das relações existentes entre as classes da OntoMuFFIN.

Baseámos a OntoMuFFIN na especificação O&M, onde os conceitos (entidades) estão identificados, criando as relações necessárias à pesquisa dos serviços e adicionando os pontos de extensão que permitem às aplicações cliente do MuFFIN acrescentar as suas próprias ontologias e definir semanticamente os serviços que instalam no *middleware*. Para tal, é necessário que, ao adicionarem um novo serviço, os clientes enviem um *URL* da sua ontologia, estendendo a OntoMuFFIN e caracterizando o serviço no sistema. Assim, a ontologia é enriquecida com mais um ponto de acesso que integrará a pesquisa do novo serviço no repositório.

O nosso objectivo é apresentar a OntoMuFFIN como a base para que os clientes consigam utilizar as suas próprias ontologias, estendendo a nossa, permitindo que as pesquisas dos serviços sejam feitas em todas as ontologias.

4.4.1 Entidades do OntoMuFFIN

O núcleo da OntoMuFFIN é a entidade *Feature Of Interest*, que relaciona conceitos com instâncias de serviços. De seguida apresentamos em pormenor as entidades da ontologia que propomos:

- ***Service***: representa as instâncias dos módulos instalados pelos clientes no *middleware*. Cada serviço está relacionado apenas com uma *feature of interest*, sendo esta o resultado do processamento interno do serviço.

- ***FeatureOfInterest***: específica as características de cada serviço. Esta entidade, e as suas relações, foram especificadas com base no padrão SWE.

- ***Phenomenon***: representa o fenómeno que foi observado pelos dispositivos das rede de objectos inteligentes.

- ***Precision***: representa a precisão do processamento dos dados do serviço. Diferentes serviços podem processar as mesmas *feature of interest*, porém o resultado pode ser mais preciso, ou eficiente, consoante o algoritmo usado. O objectivo desta entidade é permitir que o cliente especifique a precisão que necessita na subscrição do serviço.

- ***Unit***: unidade utilizada para descrever o fenómeno observado.

- ***Event***: uma *feature of interest* pode estar relacionada com um evento ou acontecimento específico, como por exemplo a passagem do furacão *Katrina* ao longo dos Estados Unidos. Esta entidade representa os eventos que podem ser pesquisados na OntoMuFFIN, permitindo relacionar os acontecimentos registados e as respectivas *feature of interest*.

- ***Location***: localização onde a *feature of interest* se verifica. Esta entidade utiliza uma ontologia geográfica (*e.g.*, Geo-Net-PT [31]), facilitando a pesquisa semântica de serviços através dos locais desejados.

- ***Environment***: representa o meio (Água, Ar e Terra) onde a *feature of interest* é registada, incorporando o meio onde os fenómenos são observados na pesquisa.

As relações de semelhança entre localizações são inferidas através da ontologia geográfica, enquanto que nas *feature of interests* a relação é especificada pelo cliente aquando da instalação do serviço. Estas relações facilitam a pesquisa de serviços relacionados com as necessidades dos clientes, mesmo quando não existem resultados exactos para as restrições na pesquisa.

4.4.2 Interacção com a OntoMuFFIN

As pesquisas à OntoMuFFIN podem ser efectuadas através de (1) pesquisa em navegador, onde se recebe os *URLs* para os dados das ontologias, através dos quais é possível *navegar* pelos recursos existentes, até encontrar o serviço desejado; ou (2) através de um motor de pesquisas SPARQL [53], de forma a procurar directamente os serviços que respeitam as restrições do cliente. Caso a pesquisa seja feita na ontologia do cliente, esta devolve o endereço através do qual o MuFFIN representa o *id* utilizado para subscrever o serviço.

4.5 Considerações Finais

Neste capítulo expusemos os pormenores de implementação de cada um dos módulos da nossa plataforma e apresentámos as ferramentas chave para a implementação de cada um destes: *Apache ActiveMQ*, para suportar a comunicação entre filtros da DFN e entre os módulos *DFN-Engine* e *Subscriptions*; *Apache XML Beans* para implementar os interpretadores de XML da especificação SOS; *Apache CXF* para implementar a comunicação com o MuFFIN através de serviços *Web*; e *JBoss Hibernate* para o acesso e abstracção da camada de persistência.

Apresentámos também a OntoMuFFIN, uma ontologia extensível para suportar a pesquisa semântica das representações *Web* dos filtros existentes na DFN do MuFFIN. As entidades desta ontologia são baseadas na especificação O&M com o objectivo de a tornar genérica e coerente com a camada de persistência do *middleware*.

No próximo capítulo apresentamos os resultados dos testes de desempenho efectuados ao MuFFIN, dando ênfase à capacidade de resposta aos pedidos efectuados pelas aplicações cliente e ainda à capacidade para suportar um grande número de filtros a serem executados no nosso *middleware*.

Capítulo 5

Avaliação

Este capítulo tem como objectivo apresentar os resultados dos testes de desempenho a que submetemos o MuFFIN, analisando a capacidade de resposta do *middleware* no processamento de dados e em invocações das aplicações cliente. A bateria de testes a que submetemos o MuFFIN tem também o objectivo de concluir os seus limites de processamento em relação ao número de filtros e de pontos de acesso que este suporta.

O desempenho é um factor importante na avaliação das aplicações, sendo essencial minimizar o tempo de resposta às invocações das aplicações de alto nível. Para estudarmos o desempenho da nossa plataforma, testámos os módulos responsáveis pelas funcionalidades críticas do MuFFIN: *ThingsGateway*, de forma a concluirmos a capacidade de resposta em relação ao fluxo de dados proveniente da rede de sensores; *DFN-Engine* para que seja possível verificar o desempenho da rede de fluxo de dados utilizada na manipulação dos dados provenientes dos dispositivos; e *SOS* para testarmos a capacidade de resposta às invocações síncronas relacionadas com os dados guardados pelo MuFFIN.

5.1 Ambiente de Testes

O objectivo foi testar exclusivamente o desempenho da plataforma. Assim, não incluímos nos resultados dos testes os tempos de processamento das mensagens SOAP trocadas entre as aplicações cliente e o MuFFIN, nem o tempo decorrido entre o envio e recepção destas. Como o nosso *middleware* disponibiliza os seus serviços através da Internet, é difícil concluirmos, para efeitos estatísticos, a consequência da variação de qualidade entre os fornecedores de serviços de Internet (ISP, do inglês *Internet Service Provider*) das redes utilizadas pelas aplicações cliente. Como o MuFFIN é independente de tipos de dispositivos, de tipologias de redes e do algoritmo de encaminhamento utilizado, ignoramos também o intervalo de tempo entre a leitura de uma variável do ambiente, por parte de um dispositivos, e a recepção desta por parte do *middleware*.

Para efeito de testes, o MuFFIN foi instalado numa máquina com as seguintes especificações:

Processador: Intel(R) Core(TM)2 Duo E8300 @ 2.83GHz;

Memória RAM: 3343MB;

Sistema Operativo: Linux, Ubuntu 10.10.

Não sendo uma máquina de última geração (processador de 2008), tem ainda os requisitos necessários para executar os testes que iremos apresentar. Porém, num cenário real de utilização, é aconselhada a utilização de um servidor dedicado, de forma a obter um desempenho optimizado no processamento de documentos XML e de grandes quantidades de dados, características comuns a este tipo de plataforma.

Como já referido, a comunicação entre os módulos *ThingsGateway* e *DFN-Engine* depende da *ActiveMQ*, portanto ao falarmos no desempenho da comunicação nestes dois módulos estamos a falar da capacidade de resposta desta ferramenta para gerir as filas e os tópicos criados pelo MuFFIN. Como base para os restantes testes, a primeira variável testada foi o tempo que demora uma mensagem a ser recebida pelo subscritor de um tópico. Para tal criámos um publicador que envia uma mensagem por segundo, recolhendo os tempos antes do envio e depois da recepção com uma precisão ao milissegundo. A amostra foi de 2000 mensagens, chegando à conclusão que, em média, passam 62 milissegundos até que uma mensagem chegue ao receptor. Porém, não é certo que num cenário real de utilização seja possível alcançar este tempo de resposta, a *ActiveMQ* implementa um controlo de fluxo de mensagens que impossibilita um publicador monopolizar o seu processamento [2], o que pode provocar um abrandamento do tempo de resposta se um determinado publicador enviar muitas mensagens num curto espaço de tempo, o que é plausível no funcionamento normal do MuFFIN.

De seguida as Secções 5.2, 5.3 e 5.4 apresentam, respectivamente, os resultados dos testes efectuados aos módulos *ThingsGateway*, *DFN-Engine* e *SOS*.

5.2 Testes Efectuados ao Módulo *ThingsGateway*

Este teste teve como objectivo quantificar o tempo que demora uma mensagem enviada por um *pontos de acesso* a ser recebida pela camada de dados e assim descobrir o intervalo de tempo desde que o MuFFIN recebe os dados dos dispositivos inteligentes, até que estes fiquem disponíveis para as aplicações cliente. Dependendo da configuração da rede de dispositivos que o MuFFIN gere, é possível que os pontos de acesso estejam constantemente a processar e a enviar dados para a DFN. Os ensaios efectuados testaram a capacidade de resposta e a escalabilidade do MuFFIN, estudando o número de pontos de acesso que este consegue processar.

Para o teste criámos pontos de acesso que enviam fluxos contínuos de cem mensagens, de forma a simular a constante entrada de dados no MuFFIN e alcançando os limites de processamento deste. Ao longo do ensaio controlámos os instantes antes do envio e

depois da recepção de cada uma destas. Para testarmos a escalabilidade, executámos o mesmo teste com vários (de um a cinco) pontos de acesso, enviando cada um destes uma amostra de cem mensagens em cada uma das cem repetições que compõem o teste. Por fim, calculámos a média do tempo que uma mensagem demora até chegar ao destino.

Para concluirmos quais as consequências do controlo de fluxo de mensagens no processamento do MuFFIN, executámos o teste com um ponto de acesso e com as definições por omissão da *ActiveMQ*. O gráfico da Figura 5.1 apresenta o intervalo de tempo médio, em cada repetição, desde o envio da mensagem por parte do ponto de acesso, até à recepção desta pelo módulo responsável pela gravação dos dados. O objectivo é apresentar uma linha temporal onde é visível o contínuo aumento do intervalo, causado pela *ActiveMQ*, em cada repetição.

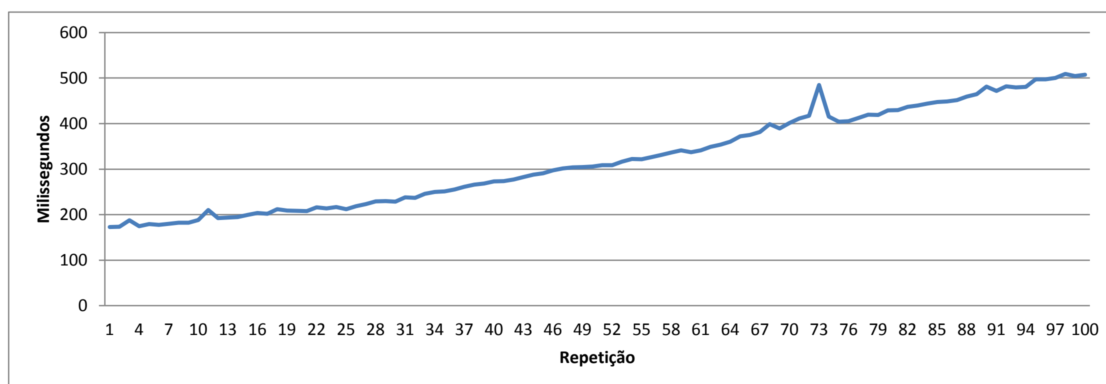


Figura 5.1: Gráfico de linha com os intervalos de tempo, por repetição, até a recepção das mensagens por parte do subscritor, com as definições por omissão da *ActiveMQ*.

A Figura 5.2 apresenta o mesmo teste com o controlo de fluxo de mensagens desligado, apresentando a Figura 5.3 a sobreposição das duas linhas de forma a dar uma visão global dos intervalos de tempo. É visível uma linha temporal onde o intervalo de tempo é estável (entre 120 e os 148 milissegundos), comparando com o do gráfico anterior. Como esperado, é possível concluir que o controlo efectuado pela *ActiveMQ* prejudica os tempos nas trocas de mensagens. Porém, este controlo é importante para que um publicador não *inunde* a comunicação, prejudicando os restantes intervenientes. A *ActiveMQ* permite aplicar o controlo de fluxo de dados de forma selectiva, controlando os publicadores através do seu identificador. Assim, decidimos não aplicar o controlo sobre as filas de mensagens (não prejudicando a comunicação entre os módulos *DFN-Engine* e *Subscriptions*) e sobre os tópicos que tenham como publicadores os pontos de acesso instalados no MuFFIN, controlando o fluxo de mensagens entre os restantes filtros da DFN, aumentando a memória atribuída a estes de forma a que a *ActiveMQ* não bloqueie os publicadores tão cedo como nas definições por omissão.

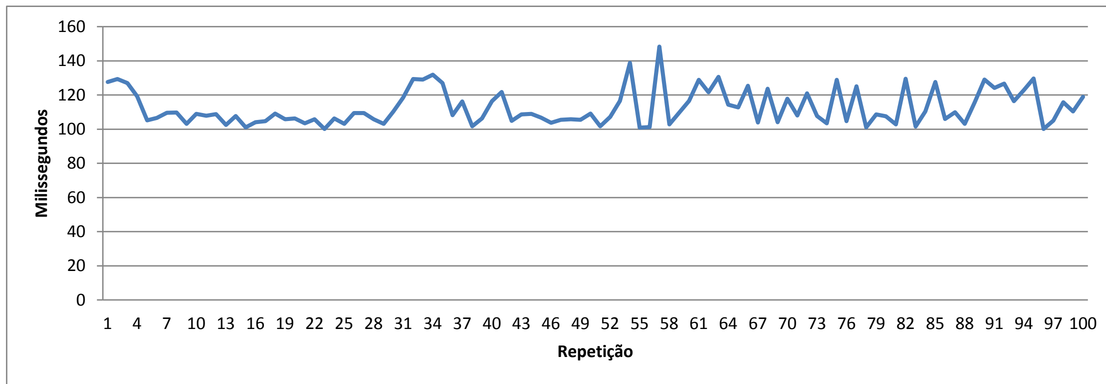


Figura 5.2: Gráfico de linha com os intervalos de tempo, por repetição, até a recepção das mensagens por parte do subscritor, com o controlo de fluxo de mensagens desactivo.

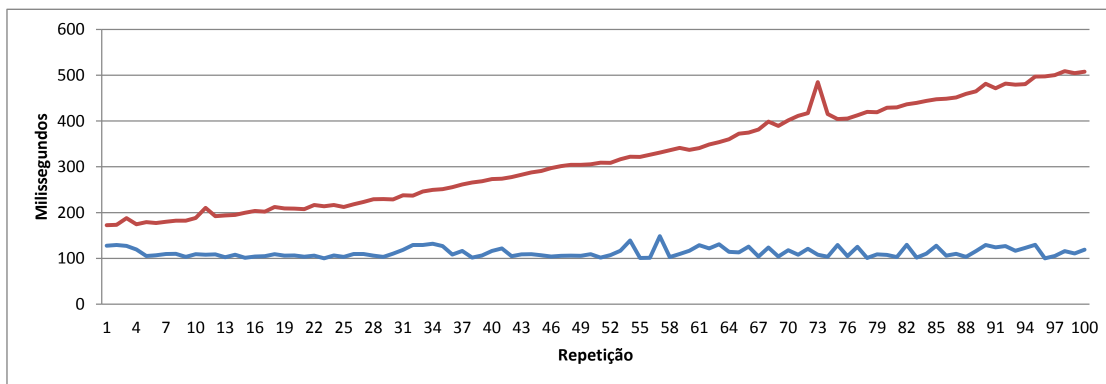


Figura 5.3: Gráfico de linhas com a sobreposição dos intervalos de tempo com o controlo de fluxo de mensagens da *ActiveMQ* activo (a vermelho) e desactivo (a azul).

Depois de configurarmos a *ActiveMQ*, executámos os restantes testes. O gráfico da Figura 5.4 apresenta os resultados para cada teste efectuado, onde é visível a duração média do intervalo de tempo, desde o envio da mensagem, até à recepção da mesma na comunicação entres os pontos de acesso e os seus subscritores. Relembramos que o intervalo de tempo médio até à recepção de uma mensagem é de 62 milissegundos, porém os testes demonstram que numa utilizam intensiva este tempo aumenta consoante o número de publicadores a serem executados em simultâneo. Estudando o gráfico apresentado, podemos concluir que o nosso *middleware* é escalável, existindo um aumento linear do tempo de espera até à recepção de mensagens em relação ao número de pontos de acesso a serem executados, nunca chegando a duplicar esse tempo em relação à adição de mais um ponto de acesso. Assim, o curto espaço de tempo que existe desde a recepção das observações por parte do MuFFIN, até que estas fiquem disponíveis às aplicações de alto nível, não prejudica o desempenho dos sistemas.

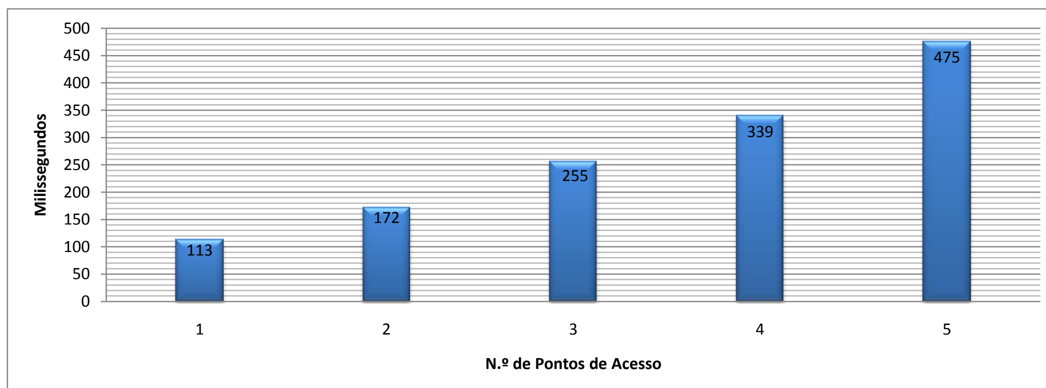


Figura 5.4: Gráfico de barras com os intervalos de tempo médios desde o envio até à recepção de mensagens, com um até cinco pontos de acesso a serem executados em simultâneo.

5.3 Testes Efectuados ao Módulo *DFN-Engine*

Esta bateria de testes teve como objectivo calcular o tempo necessário para que uma mensagem percorra uma DFN com diversos filtros, comparando os resultados tendo em conta as diferentes dimensões da cadeia e tentando alcançar os limites de processamento desta.

Para este teste criámos um ponto de acesso que envia mil mensagens por segundo, registando os instantes antes do envio por parte do ponto de acesso e depois da recepção desta por parte do último filtro da cadeia. O nosso objectivo foi sobrecarregar a DFN, testando a capacidade de resposta consoante o número de filtros que a compõem. Foram testadas DFNs com cinco, dez, vinte, cinquenta e cem filtros, cada um com complexidade de processamento O(1). A razão pela qual efectuámos o teste com filtros com processamento constante está relacionada com o facto de não queremos influenciar os resultados com o processamento do filtro, quantificando apenas os limites da capacidade de comunicação do módulo *DFN-Engine*.

O gráfico da Figura 5.5 apresenta os resultados deste teste, onde é visível o crescimento do intervalo de tempo necessário para que a mensagem percorra a cadeia de filtros, acompanhando a dimensão da mesma. Podemos afirmar que o desempenho da DFN escala, pois o tempo de processamento cresce linearmente consoante a sua dimensão. É importante relembrar que, ao contrário dos pontos de acesso, os filtros estão sobre o controlo do fluxo de mensagens, portanto a *ActiveMQ* abrandou o fluxo de cada um dos filtros (como é visível na Figura 5.6). Não consideramos este comportamento preocupante porque não esperamos fluxos contínuos de mil mensagens e também não são esperadas DFNs com mais de vinte filtros.

Como esperado, é visível que o desempenho da DFN depende da sua dimensão. Quantos mais filtros forem instalados, maior o intervalo de tempo até que a mensagem enviada pelo ponto de acesso alcance o último filtro. Porém, as aplicações de alto nível que subscrevem filtros intermédios são notificadas quando os dados passam pelo filtro subscrito,

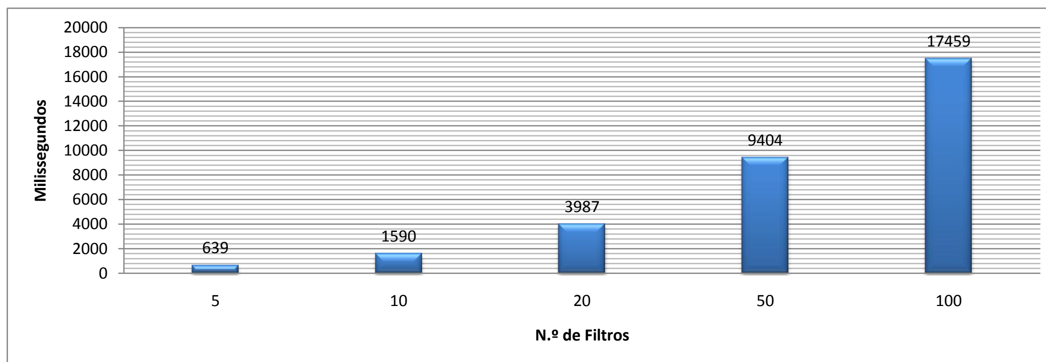


Figura 5.5: Gráfico de barras com os intervalos de tempo médios para que uma mensagem percorra uma cadeia de cinco, dez, vinte, cinquenta e cem filtros.

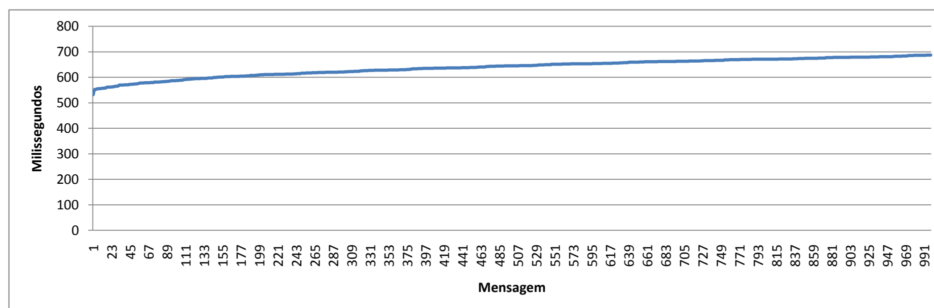


Figura 5.6: Gráfico de linha com os intervalos de tempo, em cada mensagem, até que a mensagem percorra uma cadeia composta por cinco filtros.

não sendo necessário aguardar que a mensagem alcance o fim da DFN.

Outra conclusão que surgiu deste ensaio está relacionada com o tamanho máximo da DFN, não implementámos o MuFFIN com o objectivo de suportar cadeias de cem filtros, sendo o intervalo de tempo resultante (aproximadamente 17 segundos) excessivo para sistemas que dependam do acesso rápido aos dados. Assim, especificamos que as cadeias com dimensões entre vinte e cinquenta filtros têm bom desempenhos, mesmo quando se trata de fluxos contínuos de mensagens.

5.4 Testes Efectuados ao Módulo *SOS*

Os testes efectuados ao módulo *SOS* tiveram como objectivo observar o desempenho da nossa implementação dos padrões *O&M* e *SOS* da especificação *SWE*. Este módulo é responsável pela gestão dos acessos síncronos às observações guardadas no MuFFIN, sendo importante que este processe os pedidos das aplicações cliente com a maior brevidade. Como o módulo SOS processa dados provenientes de documentos XML e da camada de persistência, recolhemos os intervalos de tempo em cada processamento de forma a concluirmos onde o MuFFIN tem maior e pior desempenho.

Para este teste criámos um cliente que invoca continuamente os serviços *Web* do nosso

middleware: *readObservation* e *insertObservation*; de forma a recolhermos exclusivamente os tempos de escrita e leitura dos dados de documentos XML e da camada de persistência no decorrer de mil invocações por serviço. Repetimos este teste com vários clientes (num intervalo de um a cinco) e por fim levámos o MuFFIN ao limite, repetindo o ensaio com cem clientes, comparando o crescimento dos intervalos de tempos.

O gráfico da Figura 5.7 apresenta os resultados dos testes em cada repetição. A vermelho observamos os intervalos de tempo relacionados com os acessos à Base de Dados (BD) do MuFFIN e a azul os intervalos de tempo relacionados com o processamento de documentos XML, significando a soma dos dois o intervalo de tempo total despendido no acesso aos dados. Podemos afirmar que o com poucos clientes a acederem em simultâneo ao MuFFIN, o tempo de resposta do módulo SOS não ultrapassa (em média) os 210 milissegundos nas escritas de observações e os quinze milissegundos nas leituras. Podemos ainda concluir que o módulo tem um desempenho escalável, pois aumentando vinte vezes o número de clientes, o tempo de resposta apenas aumenta aproximadamente 125% nas escritas de dados, mantendo-se estável nas leituras de observações.

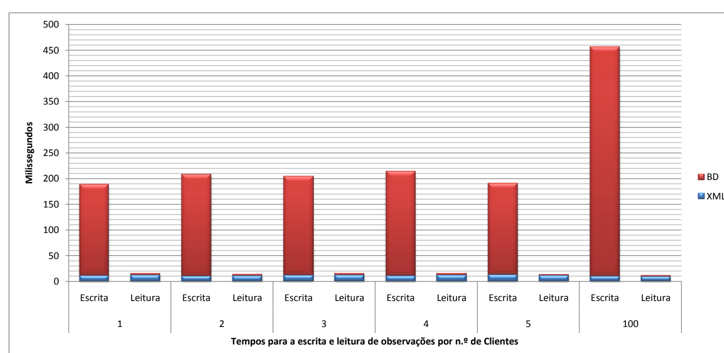


Figura 5.7: Gráfico de barras com os intervalos de tempo médios despendidos na escrita e leitura de observações, com um até cinco clientes e, por fim, cem clientes.

Através destes testes pudemos observar a percentagem de tempo despendido em cada tipo de acesso para cada origem das observações. A Figura 5.8 apresenta esta proporção, onde é possível observar que a escrita (construção) de documentos XML é bastante rápida (aproximadamente 6% do tempo total), em relação à leitura destes (restantes 94% do tempo total). Porém, nos acessos à base de dados passa-se o contrário, sendo as escritas que demoram mais tempo (88% do tempo total), em contraste com os restantes 12% despendidos nas leituras de observações. Seria de esperar que as escritas de dados fossem sempre mais lentas que as leituras, porém no caso dos documentos XML este resultado é explicado pelo facto de o módulo SOS transformar, num passo intermédio, os documentos XML recebidos em documentos XML mais simples, facilitando a implementação do nosso interpretador. Esta transformação não é efectuada na escrita do XML, verificando-se um tempo inferior, concluindo que optimizando a transformação, melhoramos o desempenho da leitura dos documentos.

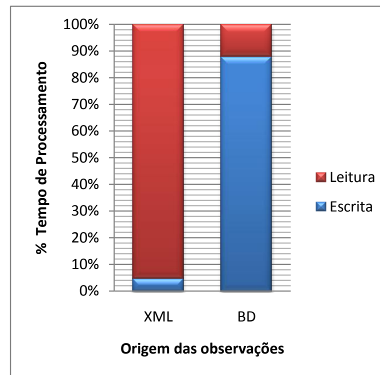


Figura 5.8: Gráfico de barras com a proporção de tempo despendido na leitura e escrita de observações dependendo da origem das observações (XML ou Base de Dados).

5.5 Considerações Finais

Neste capítulo apresentámos os resultados dos testes de desempenho suportados pelo MuFFIN. Foi possível concluir que os módulos que implementámos são escaláveis na capacidade de resposta a grandes quantidades de invocações e ainda descobrir o número de pontos de acesso e filtros que o MuFFIN suporta em situações de sobrecarga.

Estes testes serviram o objectivo de concluirmos as limitações do MuFFIN, observando que não é possível atingir um grande desempenho caso sejam instaladas cadeias com mais de cinquenta filtros na DFN. Foi possível também observar que o controlo de fluxo de dados da *ActiveMQ* prejudica a comunicação entre filtros, sendo possível minimizar o problema configurando a ferramenta de forma a não bloquear tão cedo os publicadores de dados.

Capítulo 6

Conclusão

A *Internet das Coisas* (IoT) é um novo paradigma que tem como objectivo mediar o espaço existente entre o mundo real e o mundo digital, através da integração do contexto do mundo, descrito pelo estado das *Coisas*, em aplicações de *software*. Um dos tópicos importantes na IoT é a forma como se pode gerir a heterogeneidade existente entre os dispositivos que representam as coisas. Tendo em conta este desafio, implementámos soluções genéricas, não dependendo de tipos ou arquitecturas específicas.

Ao longo desta dissertação apresentámos o MuFFIN, uma plataforma genérica que possibilita a gestão dos dados recebidos de redes de Coisas. Propusemos as nossas soluções para a programação dos objectos inteligentes e apresentámos a estrutura de uma ontologia que permite a pesquisa semântica dos serviços oferecidos.

Para possibilitar a gestão de diversos tipos de observações, a nossa plataforma oferece um conjunto de serviços *Web* que respeitam dois dos padrões da *Sensor Web Enablement* do *Open Geospatial Consortium*, nomeadamente o *Observations and Measurements*, para organizar e gerir os dados das redes de sensores e o *Sensor Observations Service*, para possibilitar o acesso genérico aos dados guardados no *middleware*.

A plataforma que propusemos oferece serviços na *Web* para a programação de dispositivos heterogéneos. A programação pode ser feita através da instalação física de código nos dispositivos, ou criando uma rede de fluxo de dados ao nível do *middleware*. A instalação de código nos dispositivos nem sempre é possível, dependendo da capacidade dos dispositivos para suportar novo código, ou mesmo da especificação dos seus fabricantes. Caso não seja possível enviar o código para os dispositivos, o *middleware* simula a sua programação, processando os dados recebidos dos dispositivos através de cadeias de manipulação de dados (DFN). Esta DFN é constituída por módulos enviados por aplicações de alto nível que, depois de instanciados, funcionam como filtros de dados que subscrevem e são subscritos por outros filtros, compondo uma cadeia complexa e dinâmica. Para possibilitar o acesso das aplicações de alto nível ao resultado da DFN, cada filtro instanciado pode ter uma representação como serviço na *Web*, permitindo que as aplicações cliente subscrevam o resultado do processamento dos filtros, recebendo

notificações com os dados. Caso seja necessário aceder a observações das redes (ou processamento dos filtros) ocorridos no passado, disponibilizamos um serviço síncrono de pesquisa que devolve os dados desejados.

Para suportar a pesquisa semântica de serviços e tornar a utilização destes mais eficaz, a nossa plataforma suporta a utilização de ontologias com o objectivo de dar semântica aos recursos do *middleware*. O objectivo é oferecer às aplicações cliente um serviço mais eficaz de pesquisa e tornar mais resiliente o acesso a redes de objectos inteligentes, dando ao *middleware* o "conhecimento" que lhe permite escolher serviços alternativos em caso de indisponibilidade dos serviços subscritos. Assim, é necessário estabelecer métodos eficazes de catalogação de conhecimento de serviços que facilitem a sua descoberta e subscrição.

Através da bateria de testes a que submetemos o nosso *middleware*, concluímos que a sua capacidade de resposta é satisfatória, mesmo em situações de processamento intensivo, possuindo um desempenho capaz de escalar em relação ao número de clientes, de filtros e de pontos de acesso que suporta. Para além das anteriores conclusões, descobrimos os limites de processamento da nossa plataforma e ainda descobrimos como optimizar o seu desempenho. Observámos que as cadeias existentes na DFN não devem ultrapassar os cinquenta filtros, perdendo rendimento a partir desta dimensão e que o controlo de fluxo de mensagens da *ActiveMQ* prejudica a comunicação entre publicadores e subscritores, partindo destes testes a decisão de configurar a ferramenta de forma a minimizar as consequências.

Como trabalho futuro será importante implementar o suporte para outras especifica ções *SWE*, oferecendo às aplicações cliente outras funcionalidades genéricas, importantes na gestão de dispositivos e dos seus dados. Desejamos ainda integrar a OntoMuFFIN, terminando o suporte da pesquisa semântica dos serviços do MuFFIN. Por fim, é nosso objectivo instalar o MuFFIN num caso de uso real, onde todas as suas funcionalidades serão testadas. O cenário envolve a colecção de variáveis do ambiente (temperatura e humidade), de quintas no arquipélago dos Açores, de forma a descobrir as condições exactas onde aparece um fungo no gado. Este fungo provoca sensibilidade aos raios solares e obriga a um tratamento dispendioso, resultando em distúrbios no normal funcionamento das quintas em questão. Desta forma o nosso projecto estará completo, passando por todas as fases da criação um sistema complexo, desde o desenho da arquitectura, até à sua utilização num cenário real.

Apêndice A

Interfaces dos Principais Módulos

Este apêndice apresenta as interfaces dos principais módulos da nossa plataforma (relembrados na Figura A.1), representando os serviços que cada um oferece aos restantes. Estas interfaces são fulcrais para o isolamento e independência entre os processamentos dos módulos.

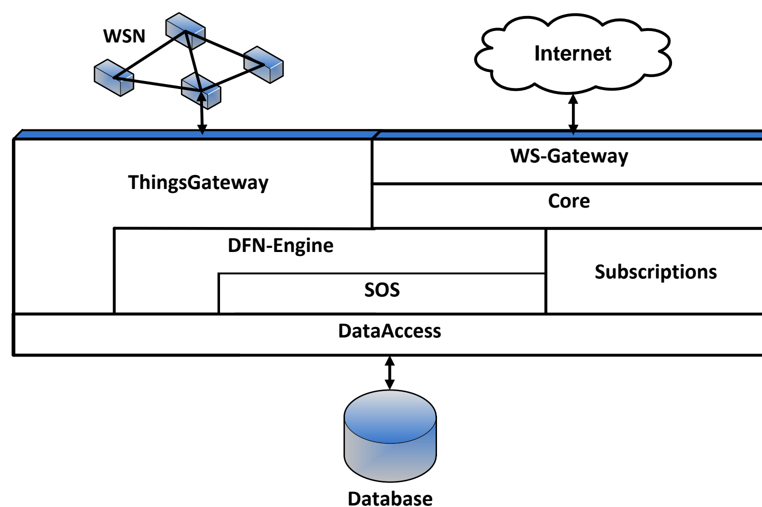


Figura A.1: Diagrama de camadas da comunicação entre módulos do MuFFIN.

A.1 IThingsGateway

Listagem A.1: Interface do módulo *ThingsGateway*.

```
public interface IThingsGateway {
  /**
   * Allows the deployment and installation of new Things Gateways into
   * middleware layer.
   *
   * @param name
   *           The name of the new Gateway.
   * @param description
   *           The description of the new gateway. This description will be
   *           presented as the gateway specification.
```

```
   * @param content
   *          The jar with the gateway content.
   * @return The gateway identification or null if the installation failed.
   */
  public Integer installThingGateway(String name, String description, byte[] content);

  /**
   * Allows the deployment of new code into the network things.
   *
   * @param thingGatewayID
   *          The identification of the gateway responsible for the deploy.
   *
   * @param content
   *          The new code to deploy.
   *
   * @return True if the new code was correctly deployed or False otherwise
   */
  public Boolean deployCode (Integer thingGatewayID, byte[] content);

}
```

A.2 IWSGateway

Listagem A.2: Interface do módulo *WS-Gateway*.

```
public interface IWSGateway{

  public int deployModule(int instanceId, String ame, String description, String code);

  public boolean instantiateService(int instanceId, String serviceMetada);

  public String insertObservation(String xml);

  public String readObservation(String xml);

  public boolean subscribeService(int instanceId, int serviceId, String callbackRef);

  public boolean unsubscribeService(int instanceId, int serviceId, String callbackRef);

  public String getModulesInfo();

  public String getCapabilities();

}
```

A.3 ISubscriptions

Listagem A.3: Interface do módulo *Subscriptions*.

```
public interface ISubscriptions {
  /**
   * Allows the subscription of a service through its identifier
   * @param serviceId
   *          The identification of the service to subscribe
   *
   * @param callbackWSDL
   *          The Web reference where the subscriber will receive the notifications.
   *          This reference will be used as subscriber identification.
   *
   * @return True if the service has been subscribed or False otherwise
   */
  public boolean addSubscription(int serviceId, String callbackWSDL);
```

```
  /**
    * Allows the unsubscription of a service.
    *
    * @param serviceId
    *           The identification of the service to unsubscribe
    *
    * @param callbackWSDL
    *           The Web reference to identify the subscriber.
    *
    * @return True if the service has been subscribed or False otherwise
    */
  public boolean removeSubscription(int serviceId, String callbackWSDL);

}
```

A.4 ISOS

Listagem A.4: Interface do módulo *SOS*.

```
public interface ISOS {
  /**
    * Returns the metadata information about the middleware and the provided methods.
    *
    * @return An XML file which respects the schema:
    *   [http://schemas.opengis.net/sos/1.0.0/sosGetCapabilities.xsd]
    */
  public String getSWECapabilities();

  /**
    * Stores the received observation and returns an XML file which respects the schema:
    *   [http://schemas.opengis.net/sos/1.0.0/sosInsert.xsd].
    *
    * @param xml
    *   The observation in XML format to be inserted.
    *
    * @return An XML file which respects the example in:
    *   [http://schemas.opengis.net/sos/1.0.0/examples/sosInsertObserationResponse.xml]
    */
  public String setObservation(String xml);

  /**
    * Returns the observations that respect the filters received as arguments in an
    * XML file. The received XML file has to respect the schema in
    *   [http://schemas.opengis.net/sos/1.0.0/sosGetObservation.xsd]
    *
    * @param xml
    *   The XML file with the search parameters.
    *
    * @return An XML file which respects the example
    *   [http://schemas.opengis.net/sos/1.0.0/examples/sosGetObservation1.xml]
    */
  public String getObservation(String xml);

  /**
    * Aggregates all observations in each received ObservationCollection and returns an
    * unique collection.
    *
    * @param obsevations
    *           List of ObservationCollections.
    *
    * @return The ObservationCollection with all received observations.
    */
  public String composeObservations(LinkedList<String> obsevations);
```

```
  /**
   * Stores the received observation data.
   *
   * @return The observation unique identification.
   */
  public String insertObservation(String sensor, String samplingTime, String resultTime,
                                  String procedure, String featureOfInterest,
                                  HashMap<String, String> dataRecord);

}
```

A.5 IDFN-Engine

Listagem A.5: Interface do módulo *DFN-Engine*.

```
public interface IDFN-Engine {

  public int deployModule(String moduleName, String description, String code);

  public boolean instantiateService(String serviceMetadata);

  public String insertObservation(String xml);

  public String readObservation(String xml);

  public String composeObservations(LinkedList<String> obsevations);

  public String getModulesInfo(int instanceId);

  public String getCapabilities();

  public String composeModulesInfo(LinkedList<String> modules);

}
```

A.6 ISOSFacade

Listagem A.6: Fachada do módulo *DataAccess* para o módulo *SOS*.

```
public interface ISOSFacade {

  public String setObservation(Observation observations);

  public Observation getObservation(String observationId);

  public Observation[] getObservation(String offering, ObservationTransferObject oto);

  public Phenomenon getPhenomenon(String phenId);

  public FeatureOfInterest getFeatureOfInterest(String foiId);

  public SOSProcedure getProcedure(int procedureId);

  public Offering getOffering(Timestamp timestamp, FeatureOfInterest foiObj);

  public Sensor getSensor(String sensor);

  public List<Offering> getOfferings();

}
```

A.7 IServicesFacade

Listagem A.7: Fachada do módulo *DataAccess* para gestão de serviços utilizada pelos módulos *DFN-Engine* e *Subscriptions*.

```
public interface IServicesFacade {

  public Integer insertModule(String moduleName, String genName,
                              String description, String code);

  public List<Module> getModules();

  public Module getModule(Integer id);

  public Integer insertServiceInstance(Integer moduleId, Integer type, Boolean ws,
                                       String metadata, String dependencies);

  public Boolean existsService(Integer id);

  public ServiceInstance getServiceInstancesById(Integer serviceId);

  public List<ServiceInstance> getServicesInstances();

  public Integer insertSOSProcedure(Integer serviceId, String description);

  public Boolean addSubscription(Integer serviceId, String subscriberEPRef);

  public Boolean removeSubscription(Integer serviceId, String subscriberEPRef);

  public MuffinInstance getMuffinInstance(int thisInstanceId);

}
```

A.8 IGatewaysFacade

Listagem A.8: Fachada do módulo *DataAccess* para o módulo *ThingsGateway*.

```
public interface IGatewaysFacade {

  public Integer getGatewayId(String genName);

  public Integer insertGatewayModule(String name, String genName,
                                     String description, String path);

  public Integer insertGatewayInstance(Integer moduleId, Integer type);

}
```

Apêndice B

Documentos SWE

De seguida apresentamos exemplos de documentos XML utilizados na comunicação entre sistemas que suportam a especificação SWE da OGC. Os objectivos destes documentos são a inserção e leitura de observações através das propriedades apresentadas nesta dissertação.

B.1 Inserção de Observação

Este documento insere uma observação lida pelo sensor *S::1278g21cx* (linha 2), às 12:21h do dia 12 de Julho de 2011 (especificado da linha 4 à linha 8). Os dados da observação foram processados pelo filtro *F:123123hn* (linha 9), estando estes relacionados com o fenómeno *urn:ogc:def:property:MyOrg:WindSpeed* (linha 10). A *característica de interesse* é a *urn:ogc:def:feature:OGC-SWE:3:transient* (especificada na linha 11). Os dados lidos e as suas unidades são apresentadas da linha 12 à linha 21.

Listagem B.1: Exemplo de documento XML para inserção de observação SWE.

```
1  <InsertObservation>
2    <AssignedSensorId>S::1278g21cx</AssignedSensorId>
3    <Observation>
4      <samplingTime>
5        <TimeInstant>
6          <timePosition>2011-07-12T12:21:00Z</timePosition>
7        </TimeInstant>
8      </samplingTime>
9      <procedure xlink:href="F:123123hn"/>
10     <observedProperty>urn:ogc:def:property:MyOrg:WindSpeed</observedProperty>
11     <featureOfInterest xlink:href="urn:ogc:def:feature:OGC-SWE:3:transient"/>
12     <result>
13       <SimpleDataRecord>
14         <field name="WindSpeed">
15           <Quantity definition="urn:ogc:def:property:MyOrg:WindSpeed">
16             <uom code="ppm"/>
17             <value>20.1</value>
18           </Quantity>
19         </field>
20       </SimpleDataRecord>
21     </result>
22   </Observation>
23 </InsertObservation>
```

B.2 Leitura de Observações

B.2.1 Filtro através da identificação da observação

O objectivo deste documento é filtrar as observações existentes, devolvendo apenas a observação com a identificação *urn:MyOrg:Observation:5678*, especificada na linha 2.

Listagem B.2: Leitura de observação através da sua identificação.

```
1 <GetObservationById>
2   <ObservationId>urn:MyOrg:Observation:5678</ObservationId>
3   <resultModel>om:Observation</resultModel>
4 </GetObservationById>
```

B.2.2 Filtro através da oferta (*offering*)

Com este documento é pretendido filtrar as observações de forma a devolver aquelas que tenham como oferta *urn:MyOrg:offering:3*, especificada através da propriedade *offering* na linha 2.

Listagem B.3: Leitura de observações através da propriedade *oferta*.

```
1 <GetObservation>
2   <offering>urn:MyOrg:offering:3</offering>
3   <responseFormat>text/xml; subtype=&quot;om/1.0.0&quot;</responseFormat>
4   <resultModel>om:Observation</resultModel>
5 </GetObservation>
```

B.2.3 Filtro através de várias propriedades SWE

Através deste documento só irão ser devolvidas observações que tenham ocorrido depois do dia 12 de Julho de 2011 às 17:20h (restrição especificada da linha 3 à linha 10), através do procedimento *urn:ogc:object:Sensor:MyOrg:12349* (linha 11) e que tenham lido fenómenos respeitantes a *urn:ogc:def:property:MyOrg:WindSpeed* (linha 12).

Listagem B.4: Leitura de observações através de várias propriedades.

```
1  <GetObservation>
2    <offering>urn:MyOrg:offering:3</offering>
3    <eventTime>
4      <ogc:TM_After>
5        <ogc:PropertyName>om:samplingTime</ogc:PropertyName>
6        <TimeInstant>
7          <timePosition>2011-07-12T17:20:00Z</timePosition>
8        </TimeInstant>
9      </ogc:TM_After>
10   </eventTime>
11   <procedure>urn:ogc:object:Sensor:MyOrg:12349</procedure>
12   <observedProperty>urn:ogc:def:property:MyOrg:WindSpeed</observedProperty>
13   <resultModel>om:Observation</resultModel>
14 </GetObservation>
```

Abreviaturas

CDA	Arquitectura Orientada-a-Componentes
EDA	Arquitectura Orientada-a-Eventos
ESB	Canal para Composição de Serviços
HQL	*Hibernate Query Language*
IoT	Internet das Coisas
ISP	*Internet Service Provider*
J2EE	*Java 2 Enterprise Edition*
MuFFIN	*Middleware Framework For the Internet of thiNgs*
NFC	*Near Field Communication*
O&M	Observações e Medições
OGC	*Open Geospatial Consortium*
OSGi	Serviços com Pontos de Entrada Genéricos
OWL	*Web Ontology Language*
PubSub	Padrão de Comunicação Publicador Subscritor
RFID	*Radio-Frequency IDentification*
SAS	Serviço de Alertas de Sensores
SensorML	Linguagem para Modelação de Sensores
SGBD	Sistema Gestor de Base de Dados
SOA	Arquitectura Orientada-a-Serviços
SOAP	*Simple Object Access Protocol*
SODA	Arquitectura para Dispositivos Orientados-a-Serviços
SOS	Serviço de Observação de Sensores
SPS	Serviço de Planeamento de Sensores
SUMO	*Suggested Upper Merged Ontology*
SWE	Habilitação de Sensores na Web

TML	Linguagem para Modelação de Transdutores
UDDI	*Universal Description Discovery and Integration*
WNS	Serviço de Notificações na *Web*
WSDL	*Web Service Description Language*
WSN	Redes de Sensores sem Fios

Bibliografia

[1] Adam Dunkels and Björn Grönvall and Thiemo Voigt. Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, pages 455–462, 2004.

[2] Apache Software Foundation. ActiveMQ - Producer Flow Control. `http://activemq.apache.org/producer-flow-control.html`. Recurso acedido no dia 25 de Julho de 2011.

[3] Apache Software Foundation. Apache ActiveMQ. `http://activemq.apache.org/`. Recurso acedido no dia 24 de Março de 2011.

[4] Apache Software Foundation. Apache Axis. `http://axis.apache.org/axis/`. Recurso acedido no dia 25 de Julho de 2011.

[5] Apache Software Foundation. Apache cxf: An open-source services framework. `http://cxf.apache.org/`. Recurso acedido no dia 25 de Julho de 2011.

[6] Apache Software Foundation. Apache Maven Project. `http://maven.apache.org/`. Recurso acedido no dia 25 de Julho de 2011.

[7] Apache Software Foundation. Apache ServiceMix 4.3. `http://servicemix.apache.org/`. Recurso acedido no dia 20 de Outubro de 2010.

[8] Apache Software Foundation. XML Beans project. `http://xmlbeans.apache.org/`. Recurso acedido no dia 15 Julho de 2011.

[9] Arne Broering and Theodor Foerster and Simon Jirka and Carsten Priess. Sensor Bus: An Intermediary Layer for Linking Geosensor Networks and the Sensor Web. In *Proceedings of the 1st International Conference on Computing for Geospatial Research and Applications (COM. Geo)*, pages 21–30, 2010.

[10] Arthur Na and Mark Priest. Sensor Observation Service. `http://portal.opengeospatial.org/files/?artifact_id=26667`, 2007. Recurso acedido no dia 20 Março de 2011.

[11] Articulate Software. Suggested Upper Merged Ontology Project. `http://www.ontologyportal.org/`. Recurso acedido no dia 3 de Julho de 2011.

[12] Bruno Valente and Francisco Martins. A Middleware Framework for the Internet of Things. In *Proceedings of The Third International Conference on Advances in Future Internet (AFIN) (to appear)*, 2011.

[13] Bruno Valente and Francisco Martins. OntoMufFIN: descoberta de serviços baseada em ontologias. In *Proceedings do terceiro simpósio português de informática (INForum)*, 2011.

[14] Catello Di Martino and Gabriele D'Avino and Alessandro Testa. iCAAS: An Interoperable and Configurable Architecture for Accessing Sensor Networks. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, 1(2):30–45, 2010.

[15] Choon-Sung Nam and Hee-Jin Jeong and Dong-Ryeol Shin. Design of wireless sensor networks middleware using the publish/subscribe paradigm. In *Proceedings of IEEE International Conference on Service Operations and Logistics, and Informatics (IEEE/SOLI)*, pages 559–563, 2008.

[16] Comunidade GlassFish, projecto Metro. JAX-WS Reference Implementation. `http://jax-ws.java.net/`. Recurso acedido no dia 25 de Julho de 2011.

[17] David Gay and Philip Levis and Robert von Behren and Matt Welsh and Eric Brewer and David Culler. The nesC Language: A Holistic Approach to Network Embedded Systems. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 1–11, 2003.

[18] David J. Russomanno and Cartik R. Kothari and Omoju A. Thomas. Building a sensor ontology: A practical approach leveraging ISO and OGC models. In *Proceedings of The International Conference on Artificial Intelligence (ICAI)*, pages 637–643, 2005.

[19] Deugd, S. and Carroll, R. and Kelly, K.E. and Millett, B. and Ricker, J. SODA: Service Oriented Device Architecture. *IEEE Pervasive Computing*, 5(3):94–96, 2006.

[20] Doug Simon and Cristina Cifuentes and Dave Cleal and John Daniels and Derek White. Java on the Bare Metal of Wireless Sensor Devices – The Squawk Java Virtual Machine. In *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, pages 78–88, 2006.

[21] Erich Gamma and Richard Helm and Ralph Johnson and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[22] Francisco Martins and Luís Lopes and João Barros. Towards the Safe Programming of Wireless Sensor Networks. In *Proceedings of Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES)*, volume 17 of *EPTCS*, pages 49–62, 2010.

[23] FuseSource Open Source Community. Fuse ESB / Apache ServiceMix 4.3. http://fusesource.com/products/enterprise-servicemix/. Recurso acedido no dia 24 de Março de 2011.

[24] Hyun Jung La and Jeong Seop Bae and Soo Ho Chang and Soo Dong Kim. Practical methods for adapting services using enterprise service bus. In *Proceedings of the Seventh International Conference on Web Engineering (ICWE)*, pages 53–58, 2007.

[25] Ian F. Akyildiz and Weilian Su and Yogesh Sankarasubramaniam and Erdal Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[26] Ingo Simonis. OpenGIS Sensor Planning Service - Implementation Specification. http://portal.opengeospatial.org/files/?artifact_id=13922, 2005. Recurso acedido no dia 20 Março de 2011.

[27] Ingo Simonis. OGC Sensor Alert Service Candidate - Implementation Specification. http://portal.opengeospatial.org/files/?artifact_id=15588, 2006. Recurso acedido no dia 20 Março de 2011.

[28] Ingo Simonis and Johannes Echterhoff. Draft OpenGIS Web Notification Service - Implementation Specification. http://portal.opengeospatial.org/files/?artifact_id=18776, 2006. Recurso acedido no dia 20 Março de 2011.

[29] JBoss Community. Hibernate - JBoss Community. http://www.hibernate.org/. Recurso acedido no dia 25 de Julho de 2011.

[30] Jie Liu and Feng Zhao. Towards semantic services for sensor-rich information systems. In *Proceedings of Second International Conference on Broadband Networks (BroadNets)*, pages 967–974, 2005.

[31] João D. Ferreira and David Batista and Francisco Couto and Mário J. Silva. The Geo-Net-PT/Yahoo! GeoPlanet (TM) concordance. Technical report, University of Lisbon, Faculty of Sciences, LASIGE, 2010.

[32] João Santos. Um *middleware* para acesso e gestão de redes de sensores em ambientes Web. Master's thesis, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, 2009.

[33] Luigi Atzori and Antonio Iera and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[34] Microsoft Patterns & Practices. Publish/Subscribe. http://msdn.microsoft.com/en-us/library/ff649664.aspx. Recurso acedido no dia 25 de Julho de 2011.

[35] Mike Botts. OpenGIS Sensor Model Language - Implementation Specification. http://portal.opengeospatial.org/files/?artifact_id=13879, 2006. Recurso acedido no dia 20 Março de 2011.

[36] Mike Botts and George Percivall and Carl Reed and John Davidson. OGC Sensor Web Enablement: Overview And High-Level Architecture. http://portal.opengeospatial.org/files/?artifact_id=25562, 2007. Recurso acedido no dia 20 Março de 2011.

[37] Mohamad Eid and Ramiro Liscano and Abdulmotaleb El Saddik. A Novel Ontology for Sensor Networks Data. In *Proceedings of IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA)*, pages 75–79, 2006.

[38] Mohamad Eid and Ramiro Liscano and Abdulmotaleb El Saddik. A Universal Ontology for Sensor Networks Data. In *Proceedings of IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA)*, pages 59–62, 2007.

[39] Mohammad H. Valipour and Bavar Amirzafari and Khashayar N. Maleki and Negin Daneshpour. A brief survey of software architecture concepts and service oriented architecture. In *Proceedings of Second IEEE International Conference on International Computer Science and Information Technology (ICCSIT)*, pages 34–38, 2009.

[40] Object Management Group. The Common Object Request Broker Architecture (CORBA). http://www.corba.org/. Recurso acedido no dia 25 de Julho de 2011.

[41] Olga Levina and Vladimir Stantchev. Realizing Event-Driven SOA. In *Proceedings of Fourth International Conference on Internet and Web Applications and Services (ICIW)*, pages 37–42, 2009.

[42] Oracle Corporation. Java Programming Language. http://java.com/. Recurso acedido no dia 24 de Setembro de 2011.

[43] Oracle Corporation. MySQL relational database management system. http://www.mysql.com/. Recurso acedido no dia 24 de Setembro de 2011.

[44] Organization for the Advancement of Structured Information Standards (OASIS). OASIS UDDI Specification TC. http://www.uddi.org/pubs/uddi_v3.htm/. Recurso acedido no dia 3 de Julho de 2011.

[45] OSGi Alliance. OSGi - The Dynamic Module System for Java. http://www.osgi.org/. Recurso acedido no dia 3 de Julho de 2011.

[46] Rakhi Motwani and Mukesh Motwani and Frederick Harris Jr. and Sergiu Dascalu. Towards a scalable and interoperable global environmental sensor network using Service-Oriented Architecture. In *Proceedings of the Sixth International Conference on Intelligent Sensors, Sensor Networks, and Information Processing (ISSNIP)*, pages 151–156, 2010.

[47] Sasikanth Avancha and Chintan Patel and Anupam Joshi. Ontology-driven Adaptive Sensor Networks. In *Proceedings of The First International Conference on Mobile and Ubiquitous Systems (MobiQuitous)*, pages 194–202, 2004.

[48] Simon Cox. Observations and Measurements Part 1 - Observation schema. http://portal.opengeospatial.org/files/?artifact_id=22466, 2007. Recurso acedido no dia 20 Março de 2011.

[49] Simon Cox. Observations and Measurements Part 2 - Sampling Features. http://portal.opengeospatial.org/files/?artifact_id=22467, 2007. Recurso acedido no dia 20 Março de 2011.

[50] Steve Havens. OpenGIS Transducer Markup Language - Implementation Specification. http://portal.opengeospatial.org/files/?artifact_id=14282, 2006. Recurso acedido no dia 20 Março de 2011.

[51] TinyOS Alliance. The TinyOS Documentation Project. http://www.tinyos.net/. Recurso acedido no dia 3 de Julho de 2011.

[52] World Wide Web Consortium (W3C). SOAP Version 1.2 Specification Assertions and Test Collection (Second Edition) Recommendation Errata. http://www.w3.org/2007/04/REC-soap12-testcollection-20070427-errata.html/. Recurso acedido no dia 3 de Julho de 2011.

[53] World Wide Web Consortium (W3C). SPARQL, Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/. Recurso acedido no dia 3 de Julho de 2011.

[54] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl/. Recurso acedido no dia 3 de Julho de 2011.

[55] World Wide Web Consortium (W3C). WS-Addressing 1.0. `http://www.w3.org/2005/08/addressing/`. Recurso acedido no dia 12 de Julho de 2011.

[56] Xinhuai Tang and Sizhe Sun and Xiaozhou Yuan and Delai Chen. Automated Web Service Composition System on Enterprise Service Bus. In *Proceedings of Third IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, pages 9–13, 2009.

[57] Yi Huang and John D. Gannon. A comparative study of Web services-based event notification specifications. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 8–14, 2006.