INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

ALSMon — Sistema para monitorizar ALS

VASILE GRIGORAS

(Licenciado)

Trabalho Final de Mestrado para obtenção do grau de Mestre
em Engenharia Informática e de Computadores

Orientadores : Doutor Nuno Miguel Soares Datia
Doutora Matilde Pós-de-Mina Pato

Júri:

Presidente: Doutor Manuel Martins Barata

Vogais: Doutora Cátia Luísa Santana Calisto Pesquita
Doutor Nuno Miguel Soares Datia

Setembro, 2015

Agradecimentos

Expresso aqui os meus sinceros agradecimentos:

Primeiramente aos meus orientadores, o Doutor Nuno Datia e a Doutora Matilde Pós-de-Mina Pato, pela disponibilidade, sugestões, partilha de conhecimento, e o total apoio que me deram durante todas as fases da dissertação. A sua orientação precisa foi essencial para a concretização deste trabalho.

A todos os docentes do ISEL, pela qualidade profissional que demonstraram, e todo o conhecimento de alto nível que me transmitiram. Este conhecimento permitiu-me aprofundar e organizar melhor o que já tinha e adquirir novas competências nas áreas que me interessavam.

À minha família, pelo incentivo, carrinho e o apoio incondicional.

Muito obrigado a todos!

Resumo

A doença esclerose lateral amiotrófica (ELA) é uma doença neuro-degenerativa, progressiva e rara, que danifica as células nervosas do cérebro e da medula, inclusive os nervos motores.

A monitorização, realizada por intermédio de múltiplos exames, origina informação com dimensão e características próprias. Torna-se uma componente fundamental para a observação, compreensão, conhecimento e a previsão da evolução da doença. Assim é muito importante disponibilizar aos profissionais de saúde informação adequada tão cedo quanto possível por forma a atuar no imediato.

O objetivo desta dissertação é produzir um sistema capaz de receber, transformar, guardar e disponibilizar para análise sinais fisiológicos. A solução proposta garante a segurança dos dados, utiliza ferramentas *open source* e é constituída por dois módulos. O primeiro módulo permite a receção, através de um serviço e uma aplicação *web*, de dados provenientes de dispositivos externos. Além disso, realiza o processamento dos dados recebidos duma forma automatizada, guardando-os num *data warehouse*, que possibilita a análise histórica dos dados, contextualizados por um conjunto de descritores. No segundo módulo, a informação do *data warehouse* é disponibilizada em formato padrão, por intermédio de duas APIs, OData e XMLA, que podem ser utilizadas por diferentes tipos de aplicações cliente.

Esta solução vai permitir uma melhor tomada de decisão dos profissionais de saúde, sobre os procedimentos médicos a adotar, graças à disponibilização de mecanismos de receção, armazenamento e exploração dos dados. A informação recebida e disponibilizada vai ser mostrada por uma aplicação *web* sob forma de gráficos e modelos de simulação, permitindo uma análise no tempo de métricas de desempenho relevantes para o problema em causa.

Palavras-chave: Esclerose Lateral Amiotrófica, *Data Warehouse*, ETL, OLAP, *Web Services*

Abstract

Amyotrophic lateral sclerosis (ALS) is a rare and progressive neuro-degenerative disease which damages the nerve cells of the brain and the medulla, including the motor nerves.

Monitoring, carried out by a multitude of tests, yields information with particular dimension and characteristics. It plays a fundamental role in the observation, understanding, knowledge and prediction of the evolution of the disease. It is therefore important to provide health professionals with accurate information as soon as possible, so they act immediately.

The objective of this dissertation is to produce a system capable to receive, transform, store and retrieve of physiological signals for analysis. The proposed solution ensures the security of the data, using open source tools and comprises two modules. The first module allows receiving, by a web service and an application, data from external devices. Moreover, it processes the received data of an automated mode, storing them in a data warehouse, which enables the historical data analysis, contextual by set of descriptors. In the second module, the information of the data warehouse is made accessible in standard format, through two APIs, OData and XMLA - which can be used by different types of client applications.

This solution will enable better decision - health professional decision on medical procedures to adopt, through the provision of reception mechanisms, storage and treatment of the data. The received and available information will be displayed by a web application in the form of graphics and simulation models, allowing analysis in the relevant performance metrics time to the problem in question.

Keywords: Amyotrophic Lateral Sclerosis, Data Warehouse, ETL, OLAP, Web Services

Abreviaturas e símbolos

ALS *Amyotrophic Lateral Sclerosis*

ALSMon Sistema para monitorizar *ALS*

API *Application Programming Interface*

AtomPub *Atom Publishing Protocol*

AUC *Area Under Curve*

BI *Business Intelligence*

DAL *Data Access Layer*

DMart *Data Mart*

DTD *Document Type Definition*

DW *Data Warehouse*

ELA Esclerose Lateral Amiotrófica

EMG Eletromiografia

ETL *Extraction, Transformation and Loading*

FK *Foreign Key*

HAT *Home Automated Telemanegement*

IDE *Integrated Development Environment*

JAR *Java ARchive*

JAX-RS *Java API for RESTful Web Services*

JAX-WS *Java API for XML Web Services*

JPA *Java Persistence API*

json *JavaScript Object Notation*

JVM *Java Virtual Machine*

MDX *Multidimensional Expressions*

MTOM *Message Transmission Optimization Mechanism*

OData *Open Data Protocol*

OLAP *Online Analytical Processing*

OLTP *Online Transaction Processing*

PAD *Pentaho Aggregation Designer*

PDI *Pentaho Data Integration*

PK *Primary Key*

POJO *Plain Old Java Objects*

POM *Project Object Model*

PSW *Pentaho Schema Workbench*

QoS *Quality of Service*

REST *Representational State Transfer*

SMTP *Simple Mail Transfer Protocol*

SOAP *Simple Object Access Protocol*

SSL *Secure Sockets Layer*

SwA *SOAP with Attachments*

URI *Uniform Resource Identifier*

WAR *Web Application ARchive*

WS-i *Web Services interoperability*

WSDL *Web Service Definition Language*

XMLA *XML for Analysis*

Índice

Lista de Figuras

Lista de Tabelas

Listagens

1

Introdução

Nos últimos anos, a telemedicina vem sendo aplicada mais frequentemente e permite monitorizar remotamente o estado clínico de um doente na sua própria casa. A telemedicina é a utilização de tecnologias de informação e a comunicação para fornecer serviços de cuidados de saúde aos indivíduos que estão a alguma distância do prestador de cuidados. Não se tratando de uma única tecnologia, a telemedicina é parte de um processo mais amplo ou uma cadeia de cuidados. Cadeia que pode melhorar a qualidade e a eficiência do estado de saúde [22, 41, 42]. Também se espera, que a equidade e a igualdade da distribuição de serviços aumenta, porque a acessibilidade dos serviços de saúde, especialmente em zonas remotas, é melhorada. Sood *et al.* [43] define que a "telemedicina é um subconjunto de telesaúde, usa redes de comunicação para a entrega de serviços de saúde e educação médica a partir de uma localização geográfica para outra, principalmente para resolver desafios como a distribuição desigual e escassez de recursos de infraestruturas e humanas." (tradução livre).

Nos doentes crónicos, a telemedicina permite que paciente e médico partilhem a tomada de decisão e os resultados clínicos, melhorando a continuidade dos cuidados e da eficiência através da deteção remota à distância [25, 44].

A crescente evolução das telecomunicações e tecnologias de informação tem proporcionado uma base sólida para a telemedicina como ferramenta viável, confiável e de uso fácil. Além disso, a Comunidade Europeia considera a utilização de um sistema *Home Automated Telemanegement* (HAT) para vigilância de doenças

crónicas um motivo de preocupação principal e, a "visão para a Europa 2020" apela à implementação de acordos de colaboração de telemedicina [45].

Neste quadro, pode-se considerar a monitorização de doentes com esclerose lateral amiotrófica (ELA) ou outras doenças onde se verifica uma degradação da parte muscular e motora, reduzindo a qualidade de vida e/ou provocando a morte.

A ELA, ou doença de Lou Gehrig, é uma doença neurodegenerativa progressiva e rara, causada por uma degeneração dos neurónios motores do córtex cerebral, as células do sistema nervoso central, Figura 1.1, que controlam os movimentos voluntários dos músculos. Cerca de 5 a 10% dos casos são familiares e 85 a 90% são para outros casos [17].

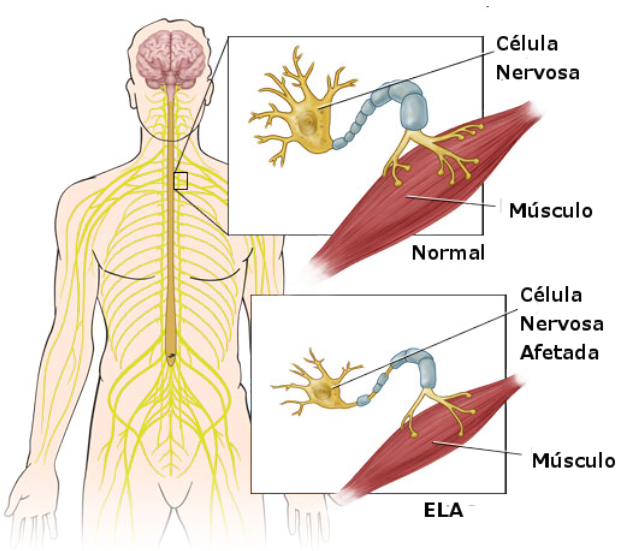
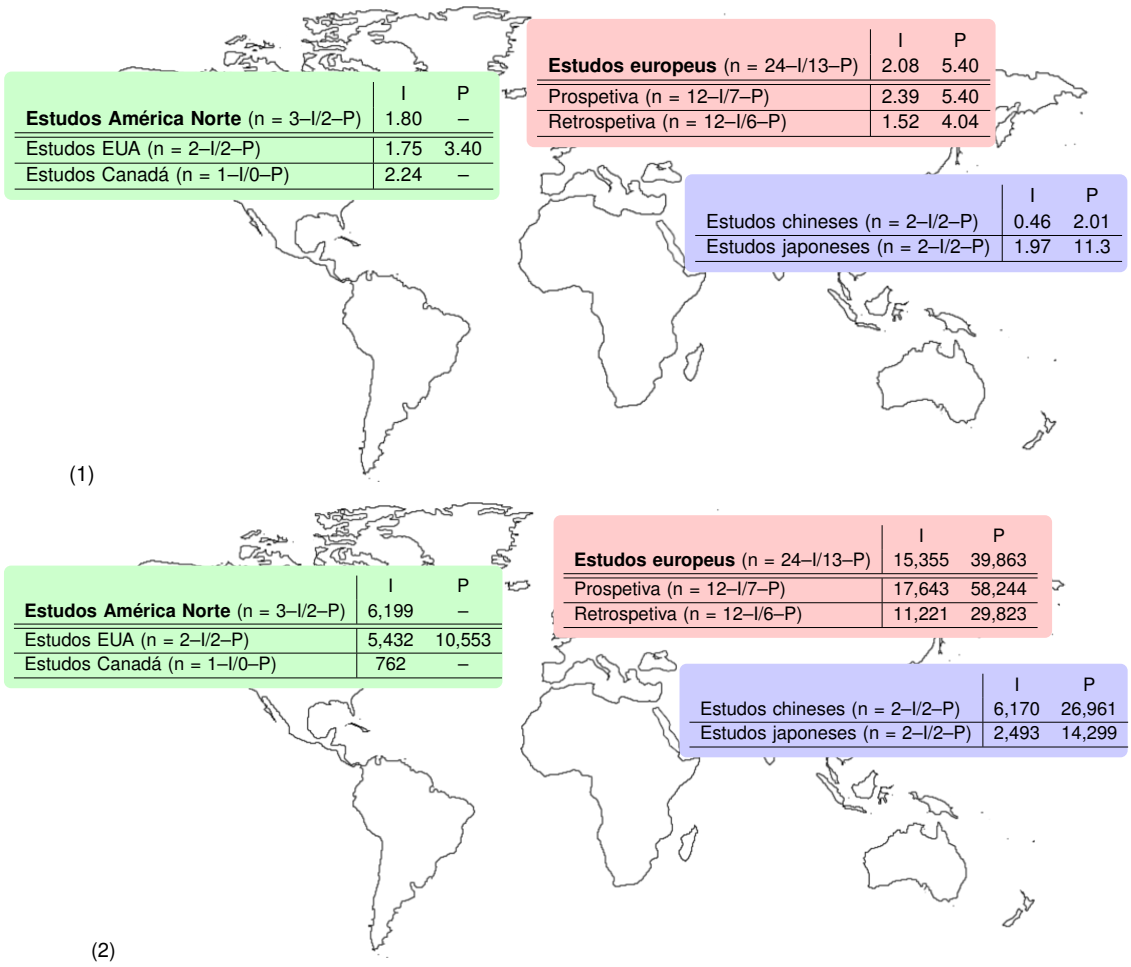


Figura 1.1: Células do nervo muscular: normal e ELA
(Adaptado da página sobre Neurological Disorders, The University of Chicago Medicine)

Apesar se tratar de uma doença rara, tem-se verificado uma incidência crescente da ELA nos últimos anos, fruto da melhor assistência médica à população em geral e da melhor acuidade no diagnóstico por parte dos profissionais de saúde. No entanto, não se pode excluir um aumento real da incidência. Os últimos estudos publicados em [21] mostram que na Europa, ver Figura 1.2, a taxa média de incidência é de $2.08/100,000$ habitantes-ano (IQR[1] $1.47-2.43$) correspondendo a uma

[1]IQR - amplitude inter-quartis

estimativa de 15, 355 casos (10, 852 – 17, 938); apenas 5 – 10% dos pacientes sobrevivem além dos 10 anos. Segundo a *International Alliance of ALS/MND Associations*[2], a incidência da ELA é de 2.0/100, 000 do total da população, enquanto que a prevalência é de cerca de 6.0/100, 000 da população total. A doença atinge igualmente ambos os sexos, sendo mais frequente nos homens (3.0/100, 000 habitantes-ano) do que entre as mulheres (2.4/100, 000 habitantes-ano). A ocorrência da doença diminui rapidamente depois de 80 anos de idade [36].



Total da população europeia (2010): 738, 199, 000.
Total da população nos EUA e Canadá (2010): 344, 401, 000; população nos EUA (2010): 310, 384, 000; população no Canadá (2010): 34, 017, 000.
População China (2010): 1, 341, 335, 000; População Japão (2010): 126, 536, 000.

Figura 1.2: Dados estatísticos (1) e estimativas (2) de casos com ELA a partir de um estudo realizado por Chiò *et al.* [21]. Valores de incidência (I) e prevalência (P), por 100, 000 habitantes-ano para os estudos acima referidos.

A sobrevida média da ELA está descrita como sendo entre 3 a 5 anos, mais baixa nas formas de início bulbar (aproximadamente 3 anos) que na medular, possivelmente pelo início mais tardio da doença [24].

[2] http://www.alsmndalliance.org/, consultado em Março 2015

O diagnóstico é baseado na história clínica, exame físico e no resultado de exames complementares [24]. Não existe um exame que diagnostique a ELA. Os exames mais importantes são a tomografia computadorizada, ressonância nuclear magnética e exames laboratoriais. Estes auxiliam a excluir outras doenças que apresentam sintomas semelhantes. A avaliação da função respiratória é essencial para a monitorização da progressão da doença e na determinação da necessidade de adaptação de ventilação não-invasiva, terapêutica não farmacológica que aumenta a sobrevida e a qualidade de vida destes doentes. A monitorização inclui a realização de eletroencefalografia, electrooculografia, eletrocardiografia e eletromiografia (EMG) com registo mentoniano, a avaliação do fluxo aéreo (ao nível do nariz e boca) e do esforço ventilatório, o registo da SpO2[3] por oximetria de pulso noturna e a avaliação da posição corporal e de movimentos dos membros (com possibilidade de registo por EMG).

A monitorização num doente com ELA torna-se uma componente fundamental para a observação, compreensão, conhecimento, previsão da evolução da doença e, em tempo real, agir de uma forma mais adequada e personalizada. Os sinais e sintomas podem, numa fase inicial, serem diferentes de doente para doente, assim como a evolução da doença é diferente e imprevisível em cada caso.

1.1 Enquadramento

Esta dissertação enquadra-se nas áreas de aplicações médicas, nomeadamente recolha, análise, visualização e processamento automático de informação. Foi realizado no ISEL e resulta de uma colaboração com uma equipa médica do Hospital de Santa Maria (HSM) e do Instituto de Medicina Molecular (IMM). Faz parte de um projeto multidisciplinar que envolve outros campos da engenharia e decorrem em paralelo no ISEL. Deste modo, foi desenvolvido fazendo uso de:

1. Um modelo de dados indicado para a recolha de registos da atividade elétrica depois de contração voluntária, em ambulatório, nos músculos flexor radial do carpo, tibial anterior e esternocleidomaistodeu dos pacientes.
2. Um serviço e uma aplicação web que fazem a receção dos dados de forma segura; um *data warehouse* onde os dados são verificados, processados e gravados autonomamente através do processo ETL; duas APIs, OData e XMLA, que permitem a aplicação web cliente realizar interrogações sobre os dados recolhidos garantindo a segurança dos mesmos.

[3]Saturação do oxigénio no sangue

3. Uma aplicação web cliente que permite ao utilizador, após autenticação, visualizar a ficha de cada paciente e selecionar filtros interativos visualizando os resultados instantaneamente.

Cada um dos pontos cima enumerados é realizado por diferentes pessoas, dada a especificidade e objetivos de cada trabalho. O meu contributo deste projeto multidisciplinar faz a ponte entre a receção dos registos e a visualização destes. Passa pela criação e a implementação de todos os artefactos definidos no ponto dois.

1.2 Motivação

A melhor maneira de combater uma doença é ter o maior conhecimento possível sobre a mesma.

O aumento de uso de ferramentas informáticas na área da saúde, com a elevada automatização dos processos de monitorização, fornece-nos uma grande quantidade de informação a ser analisada e processada.

A informação recolhida nesses exames origina bases de dados médicas com características únicas que as distinguem das demais e que tornam o processamento para obtenção do conhecimento num desafio.

A dimensão dos dados deriva do extenso número de exames produzidos e do número de características que têm que ser analisadas em cada um deles. Por si só, um único exame pode gerar inúmeras regiões de interesse a serem classificadas. Do ponto de vista informático estas questões levantam alguns desafios. É necessário usar diversas técnicas que permitem lidar com cada uma destas características por forma a minimizar os problemas que lhes estão associados.

1.3 Objetivos

Neste trabalho pretende-se produzir um sistema capaz de guardar e analisar sinais fisiológicos — Sistema para monitorizar ALS[4] (ALSMon). A solução disponibiliza mecanismos de recolha, exploração e análise de dados, permitindo aos profissionais de saúde uma melhor tomada de decisão sobre os procedimentos médicos a adotar. O sistema é constituído por dois módulos. O primeiro módulo

[4] ALS, sigla inglesa de Esclerose Lateral Amiotrófica

permite a receção, através de um serviço e uma aplicação *web*, de dados provenientes de dispositivos externos, e.g. monitor de *Holter* aplicado a EMG (ou, EMG ambulatória) e dos pacientes. Além disso, realiza o processamento dos dados recebidos, guardando-os num *data warehouse*. O segundo módulo funciona como um sistema de apoio à decisão, mostrando, por intermédio de duas APIs, OData e XMLA, e através de uma aplicação *web*, a informação recolhida sob forma de gráficos e modelos de simulação, permitindo uma análise no tempo de métricas de desempenho relevantes para o problema em causa.

O trabalho foi desenvolvido recorrendo a ferramentas *open source*.

O projeto foi dividido em seis etapas principais:

1. compreender o contexto, do ponto vista informático e médico,
2. compreender os dados disponíveis,
3. desenvolver o serviço e a aplicação *web* para permitir a receção dos dados fisiológicos e dos pacientes,
4. desenhar a arquitetura e desenvolver o módulo de *data warehouse*,
5. desenhar a arquitetura e desenvolver o módulo de análise (APIs OData e XMLA) e,
6. avaliação do sistema.

Com a realização das etapas apresentadas, espera-se que seja implementado um sistema que permite a identificação de padrões de incidência e modelos preditivos que possa trazer informação útil e contribuir para a formulação de um novo conhecimento nesta área.

1.4 Organização do documento

A dissertação está estruturada em seis capítulos. O presente capítulo faz o enquadramento dos temas abordados nesta dissertação. É feita uma pequena exposição de conceitos relativos à temática abordada, a motivação, a importância e a descrição do problema.

No Capítulo 2 são introduzidos alguns conceitos teóricos e apresentada uma síntese conceptual da revisão da literatura. Começa-se com a definição e o histórico do conceito de um *data warehouse*, os requisitos, a abordagem e a estrutura. Continuamos com servidores OLAP, definição, tipos de servidores e as operações que podem ser realizadas usando os cubos multidimensionais. O conceito de normalização nas bases de dados, os modelos dimensionais e a sua estrutura, os

esquemas mais comuns e diferenças entre elas. Terminamos com os metadados e a sua importância nas aplicações OLAP e, a descrição do processo de ETL.

Nos Capítulos 3 e 4 são apresentadas a implementação do serviço e da aplicação *web* necessárias, a receção dos dados, e a arquitetura e a implementação da solução encontrada para o primeiro módulo: um *data warehouse* onde os dados são verificados, processados e gravados.

No Capítulo 5 encontramos a arquitetura e a implementação da solução para o módulo de análise: duas APIs, OData e XMLA, que permitem o acesso à informação do *data warehouse* e que serão usadas por uma aplicação de análise que mostra a informação recolhida sob forma de gráficos e modelos de simulação.

Por fim, no Capítulo 6 são apresentadas as conclusões e apontadas linhas para trabalho futuro.

2

Conceitos fundamentais e trabalho relacionado

Nesta secção são apresentados alguns estudos desenvolvidos no âmbito da criação de um serviço de receção de dados, um *data warehouse* através do processo de extração, transformação e carregamento dos dados e, duas APIs, OData e XMLA, utilizadas para a disponibilização dos dados do *data warehouse*. Será utilizada uma forma de revisão de literatura baseada em síntese conceptual de modo a resumir o conhecimento conceptual sobre as questões abordadas.

2.1 *Data Warehouse*

O *data warehouse* (DW) surgiu como conceito académico na década de 80, quando investigadores da IBM [12], Barry Devlin e Paul Murphy, desenvolveram o "*business data warehouse*". Este conceito destinava-se a fornecer um modelo de arquitetura para o fluxo de dados de sistemas operacionais para os ambientes que suportavam a decisão.

Inmon usou o termo pela primeira vez em 1992 [31], e define um DW como um repositório de dados orientados por assunto, integrados, não voláteis e estruturados temporalmente que suportam os processos de tomada de decisão dos gestores. As principais vantagens da utilização de um DW, em vez de um repositório tradicional, são as seguintes:

- **Orientados aos assuntos**: Focados nos aspetos de negócio e excluindo os dados que não são relevantes no processo de tomada de decisão;
- **Integrados**: Os dados são provenientes de diversas fontes com os mais diversos formatos. Eles são selecionados, integrados e posteriormente armazenados no DW. Esta consistência é obtida usando técnicas de limpeza e transformação;
- **Não voláteis**: Os dados inseridos não devem ser modificados ou removidos.
- **Estruturados temporalmente**: Um DW mantém a informação atual e os dados históricos. Este conceito de estrutura temporal permite detetar padrões e relações a longo prazo para auxiliar a tomada de decisão. O tempo é uma dimensão essencial que todos os DW devem suportar.

Requisitos do *data warehouse*

Segundo Kimball [33], um DW deve:

- **Tornar a informação facilmente acessível**. O conteúdo do DW deve ser intuitivo e de fácil entendimento não apenas para o programador mas também, e principalmente, para o utilizador que toma as decisões;
- **Apresentar informações consistentes**. Informação consistente significa informação de alta qualidade: todos os dados são relevantes, precisos e completos;
- **Um DW tem de ser adaptativo e tolerante à mudança**. As alterações no DW devem ser suaves, não quebrando a compatibilidade com dados e aplicações existentes;
- **Proteger e tornar a informação segura**. O DW deve ter um controlo de segurança efetivo para os dados confidenciais;
- **Auxiliar no processo de tomada de decisão e ser aceito pela comunidade empresarial**. De nada adianta construir uma solução elegante mas que não é utilizada, por exemplo por não fornecer os indicadores necessários à tomada de decisão.

Abordagens na implementação

Um dos maiores desafios em sistemas de DW é como planear a sua construção. Duas abordagens são normalmente aceites na implementação destes sistemas: uma do tipo *top-down* e, outra do tipo *bottom-up*.

A primeira abordagem, do tipo **top-down**, divide-se em duas etapas [31]. A primeira etapa consiste na definição do esquema global do DW e a segunda baseia-se na implementação de Data Marts (DMart) de acordo com as necessidades e características das várias unidades de negócio da organização. A vantagem neste tipo de implementação é ter um modelo integrador de todos os assuntos. As desvantagens são o longo tempo de implementação exigido, a alta taxa de risco de insucesso, a criação de expectativas em relação ao ambiente a ser construído, já que a implementação e a obtenção de resultados é demorada.

Na segunda abordagem, do tipo **bottom-up**, as partes do sistema vão sendo construídos iterativamente [33]. O objetivo passa por construir esquemas individuais, de cada DMart, tendo em consideração as necessidades de cada unidade de negócio. Assim os projetos serão menores, focando áreas ou assuntos específicos. O seu desenvolvimento pode ser feito de forma independente, pois os requisitos foram previamente definidos de forma global ao sistema. A estrutura do DW será implementada de forma incremental, conforme vão sendo realizados os DMarts. A grande vantagem é a rápida implementação, a agilidade na apresentação dos resultados, diminuindo a taxa de risco de insucesso.

Esta arquitetura vem-se tornando mais popular dada as abordagens do tipo *top-down* serem caras e politicamente difíceis de serem definidas, necessitando de mais tempo para implementação, investimento e não apresentar um retorno rápido [26].

Estrutura do *data warehouse*

Como vimos no parágrafo anterior, a implementação de tipo **bottom-up** é mais popular, tendo vantagens em comparação com o outro tipo de implementação, **top-down**. Assim iremos focar a atenção neste tipo de implementação. A estrutura genérica de um DW [33] é apresentada na Figura 2.1.

Como neste tipo de implementação a estrutura do DW será implementada de forma incremental, conforme vão sendo realizados os DMarts, para realizar a integração e a coesão destes DMarts no DW é utilizado a arquitetura de "bus". O *data warehouse bus* é a parte da arquitetura do Kimball que permite que a soma dos DMarts seja vista verdadeiramente como um todo integrado — o DW. Assim todos os DMarts têm que ter factos definidos duma forma coerente e têm que utilizar dimensões concordantes (a sua semântica mantêm-se em todos os DMarts que a utiliza) [16].

Os elementos que compõem a estrutura são:

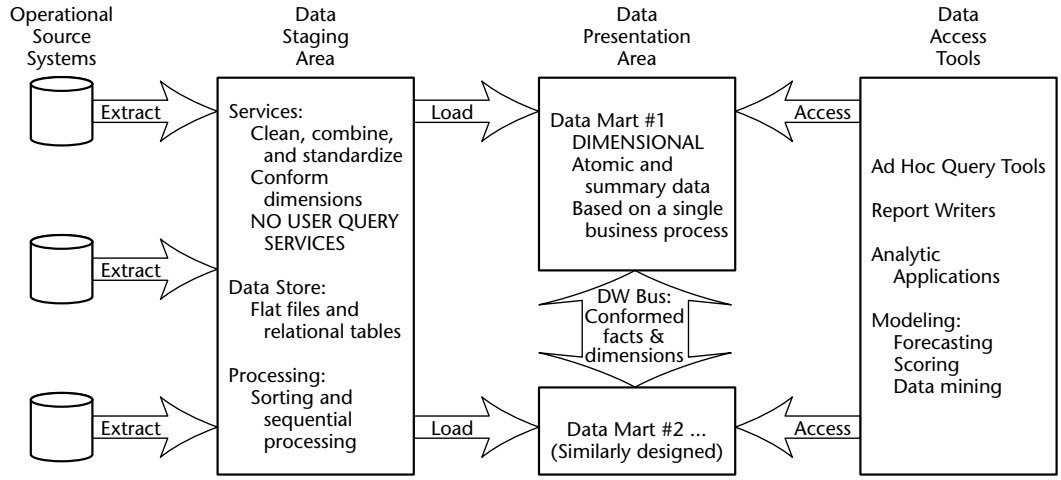


Figura 2.1: Estrutura de um *data warehouse* [33]

Data Sources: Locais de onde extraímos os dados que serão utilizados para a análise dentro do sistema de DW. Uma solução de DW consiste em promover a integração de dados a partir de diversas fontes, quer sejam internas ou externas à organização, em diversos formatos: relacional, csv, excel, entre outros.

Data Staging Area: Parte do DW responsável pelo armazenamento e execução de um conjunto de processos normalmente denominados como ETL – extração, transformação e carregamento dos dados. Encontra-se entre os sistemas operacionais e a área de apresentação. É uma atividade *back room*, não muito visível ao utilizador final.

Data Presentation Area: É onde os dados estão organizados, armazenados e disponíveis para responder às consultas num formato dimensional. Referimo-nos à área de apresentação como uma série de DMarts orientados ao processos de negócio e integrados através do DW Bus. Enquanto o DW usa dados de toda a organização, os chamados DMarts têm objetivo idêntico, mas em geral tratam apenas um departamento ou processo de negócio.

2.2 Servidores OLAP

Os servidores de processamento analítico (servidores OLAP) são uma tecnologia de *Business Intelligence* (BI) utilizada para explorar um DW e para apoiar as organizações nas análises efetuadas. Um DW é baseado num modelo de dados relacional ou multidimensional e utilizando o cubo, possibilita uma vista multidimensional sobre os dados.

O cubo é um modelo multidimensional, constituído principalmente por dimensões e métricas. Permite analisar a informação necessária à tomada de decisão, as métricas, sobre várias perspetivas, através das dimensões definidas. Os atributos das dimensões são em geral organizados em hierarquias que permitem o acesso à informação com maior ou menor detalhe. Quando são realizadas consultas em níveis hierárquicos menos detalhados, a informação que é apresentada é assente em valores agregados.

Todos os servidores OLAP utilizam a mesma técnica para otimizar o desempenho – através de pré-agregações. Pré-agregações são resumos pré-calculados dos dados detalhados. Este recurso melhora o tempo de resposta da consulta, preparando as respostas antes que as perguntas sejam feitas. Por exemplo, quando uma tabela de factos contiver centenas de milhares de linhas, com granularidade por dia, uma consulta solicitando os valores mensais dum determinado facto pode levar um longo tempo para responder. Isso porque todas as linhas da tabela têm de ser percorridas e calculada a resposta no momento da consulta. No entanto, a resposta pode ser quase imediata se os dados necessários para responder a essa consulta forem pré-calculados.

O cálculo das pré-agregações ocorre durante o processamento do cubo e elas são a base dos servidores OLAP para ter os tempos de resposta rápidos. No entanto, como estas pre-agregações ocupam espaço e levam tempo de processamento, a escolha das agregações a considerar tem de ser feita utilizando heurísticas, considerando custo vs. benefício.

Os servidores OLAP permitem analisar informação a partir dos dados, geralmente dum DW ou DMart, através da criação dos cubos multidimensionais. Enquanto um DW é utilizado para armazenar a informação, os servidores OLAP são utilizados para recuperá-la e apresentar os dados através de diversas técnicas e operações de visualização [23].

Estes servidores podem armazenar os dados sobre uma das seguintes três arquiteturas [34]:

- **OLAP multidimensional (MOLAP)** — Neste modo de armazenamento uma cópia dos dados de origem do DW, junto com as suas agregações armazenam-se numa estrutura multidimensional — referido normalmente por cubo multidimensional. Este tipo de arquitetura apresenta como principal vantagem o seu excelente desempenho. Tem como desvantagens a limitação do espaço ocupado e na atualização da informação é pouco eficiente,

uma vez que carregar nova informação pode implicar o total reprocessamento do cubo, tendo assim uma latência alta.

- **OLAP relacional (ROLAP)** — Não é mais do que uma vista multidimensional, com toda a sua funcionalidade, assente sobre um motor relacional. O trabalho do motor ROLAP é traduzir as interrogações em instruções SQL cujo resultado, posteriormente, é mostrado aos utilizadores com o nível de sofisticação presente na ferramenta OLAP. A capacidade de armazenamento é superior ao MOLAP, embora com um menor desempenho. Normalmente a utilização de ROLAP vs. MOLAP tem a ver com o peso do armazenamento vs. desempenho. [20].
- **OLAP híbrido (HOLAP)** — Este tipo combina e integra os anteriores, tirando o melhor dos dois, nomeadamente o desempenho das interrogações do MOLAP e a capacidade de armazenamento e tratamento de dados transacionais do ROLAP. Da mesma forma que o MOLAP, o HOLAP armazena as agregações numa estrutura multidimensional e os dados detalhados numa base de dados relacional, da mesma forma que no armazenamento ROLAP. Tem uma arquitetura de compromisso, que normalmente, nunca consegue obter os mesmos desempenhos que as arquiteturas anteriores. Como, cada vez mais, as ferramentas de OLAP apresentam soluções que permitem uma gestão dos dados de acordo com as necessidades, podendo, num mesmo cubo, ter diferentes tipos de armazenamento. A arquitetura HOLAP é pouco utilizada.

A comparação entre estes três tipos de servidores OLAP é esquematizada na Figura 2.2, mostrando onde ficam armazenados cada um dos componentes.

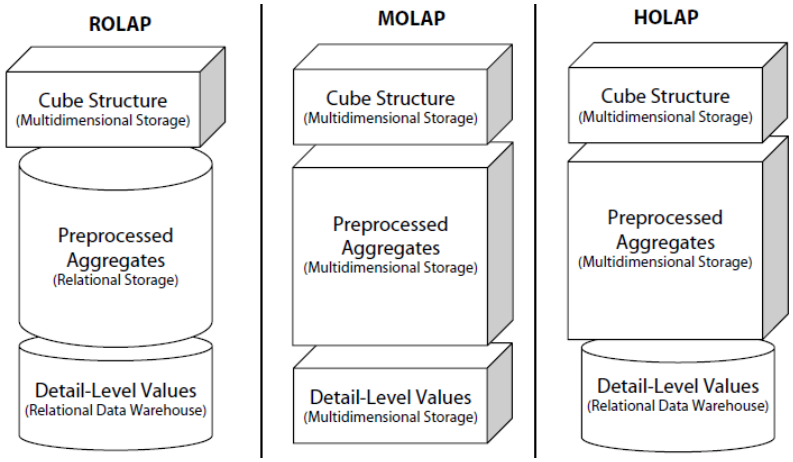


Figura 2.2: Servidores OLAP: relacional, multidimensional e híbrido (da esquerda para a direita) [23]

Independentemente do tipo de armazenamento escolhido, a possibilidade de análise de cada um é igual. As análises recorrem a hierarquias de conceitos para incluir ou retirar detalhes, alterando o nível de agregação.

A Figura 2.3 representa o exemplo de um cubo tridimensional (considerando como dimensões o paciente, o músculo e o tempo) pelas quais a área sobre a curva (AUC) pode ser analisada. A AUC é uma medida de qualidade do desempenho de um modelo, numa relação direta e como tal serve de comparação entre diferentes modelos.

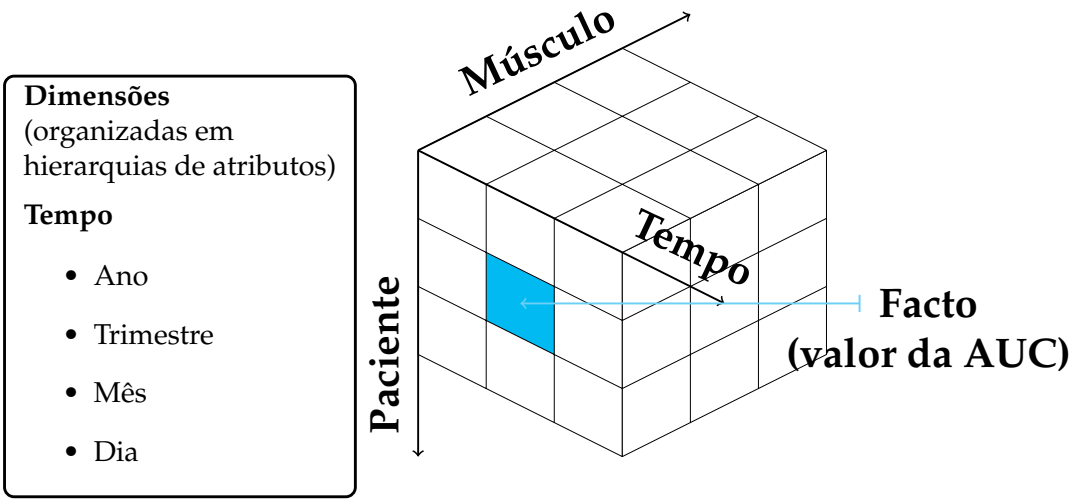


Figura 2.3: Exemplo do cubo de três dimensões

Conforme indicado por Barbieri [14], as principais operações para manusear e visualizar um cubo OLAP são:

- ***Drill down***: aumentar o nível de detalhe, navegando ao longo do nível mais alto ao mais baixo ou adicionando uma nova dimensão;
- ***Drill up***: diminuir o nível de detalhe, navegando ao longo do nível mais baixo ao mais alto ou eliminando uma dimensão;
- ***Slice***: limitar a visualização da informação com recurso ao corte (*slice*), quando o valor de um atributo duma dimensão é selecionado e fixado, sendo analisado em relação às restantes dimensões;
- ***Dice***: limitar a visualização da informação com recurso à redução (*dice*), que permite definir um sub-cubo selecionando atributos em duas ou mais dimensões;
- ***Pivot*** ou ***Rotate***: permite rodar os eixos de visualização dos dados de forma a obter uma visão diferente do cubo.

2.3 Normalização

A normalização, idealizada por Codd na década de 1970, decorre da necessidade de reduzir a redundância nas bases de dados relacionais, para evitar inconsistências resultantes das transações processadas. Num processamento transacional de dados (OLTP) aplicamos técnicas de normalização seguindo graus de desnormalização a fim de obter o desempenho desejado ao reduzir o número de junções de tabelas (*joins*). Quanto mais "normalizada" uma base de dados se encontra, menor o desempenho na obtenção de dados. Como num processamento analítico de dados OLAP as atualizações são esporádicas e centralizadas no processo ETL, mas as consultas são comuns e envolvem grandes conjuntos de dados, deve ser garantido o desempenho nas interrogações. Por isso, de uma forma geral, as bases de dados relacionais utilizadas no OLTP são normalizados, sendo que as bases de dados utilizadas para OLAP são não normalizadas.

Para sistemas OLTP aplicamos técnicas de normalização seguidas por graus de desnormalização a fim de obter o desempenho desejado ao reduzir o número de junções de tabelas (*joins*). Num sistema OLAP as atualizações são esporádicas, mas as consultas envolvem grandes conjuntos de dados e devem ser muito rápidas.

Alguns autores [33] dizem que "se pretendemos um modelo simples com bom desempenho, não se deve usar o modelo normalizado, sendo preferível uma modelação própria para o sistema OLAP".

2.4 Modelos dimensionais

Um modelo dimensional é uma abordagem metodológica concebida para o desenho de um DW, representando a lógica de modelação da base de dados. Segundo os autores, como Mundy [13, 40], o modelo dimensional apresenta as seguintes vantagens:

1. A informação é descrita de uma forma simples;
2. Os resultados das pesquisas são devolvidos de uma forma mais eficiente;
3. Dá informação relevante para o negócio;
4. O utilizador final compreende o conteúdo estrutural, da informação e a forma como esta está agregada.

Estrutura dos modelos multidimensionais

A modelação multidimensional apresenta como componentes básicas as tabelas de factos e de dimensões [28].

A abordagem mais utilizada e a sugerida por Kimball [33, 34] consiste na **tabela de factos** e as suas dimensões associadas. A tabela de factos é constituída por chaves estrangeiras que permitem a ligação entre as várias dimensões existentes e formam em conjunto (ou subconjunto) a chave primária da tabela e os factos que são as medidas, métricas de desempenho correspondentes a cada processo de negócio ou departamento das organizações. Estas tabelas têm muitos tuplos e relativamente poucas colunas, representando assim quase a totalidade do espaço ocupado pelo DW.

As **tabelas de dimensões** são constituídas por uma chave primária e por um conjunto de atributos descritivos que são principalmente textuais e normalmente relacionados hierarquicamente. Contêm muito menos linhas do que as tabelas de factos, mas o grau do seu esquema é maior, contendo atributos para descrever tanto quanto possível os factos (analisa-los sobre diversas perspetivas). São as dimensões que definem a granularidade do facto [33].

Existem várias abordagens à modelação dimensional. Abaixo serão descritas as mais comuns e a sua constituição:

- Esquema em estrela (*Star Schema*);
- Esquema em constelação (*Constellation Schema*);
- Esquema em floco de neve (*Snowflake Schema*).

O **esquema em estrela**, representado na Figura 2.4 é habitualmente o modelo de dados multidimensional mais utilizado para modelar um DW ou um DMart [19]. O nome foi adotado devido a semelhança do modelo como uma estrela.

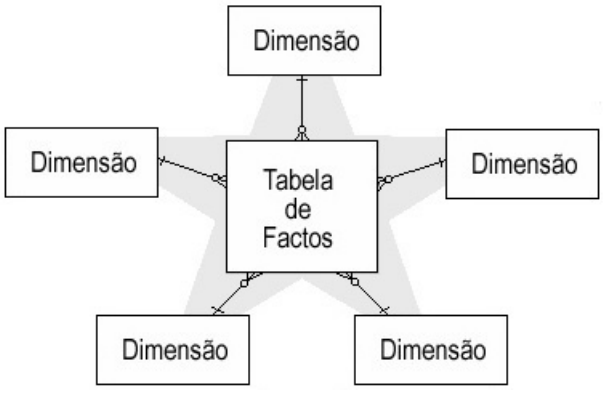


Figura 2.4: Esquema em estrela

No "centro" da estrela, existe uma tabela de factos, rodeada por tabelas de dimensões. A tabela de factos liga-se às demais dimensões por múltiplas junções e, as tabelas de dimensões conectam-se com apenas uma junção do tipo chave primária / chave estrangeira a tabela de factos, como se pode verificar na Figura 2.4.

As tabelas de dimensões não são ligadas entre si e são tabelas não normalizadas obtidas através de contração de hierarquias e agregações [39].

Um **esquema em constelação** é um modelo de dados mais complexo que integra várias tabelas de factos que partilham uma ou mais dimensões. Este esquema pode ser visto como um conjunto de esquemas em estrelas, que se ligam através de dimensões comuns. Neste caso as dimensões têm que ser concordantes (a sua semântica mantêm-se em todas as tabelas de factos que tiverem uma chave estrangeira para ela) e os factos respetivos definidos duma forma coerente. Habitualmente este tipo de esquema pode ser encontrando num DW que contém mais do que um DMart. A representação deste esquema encontra-se na Figura 2.5.

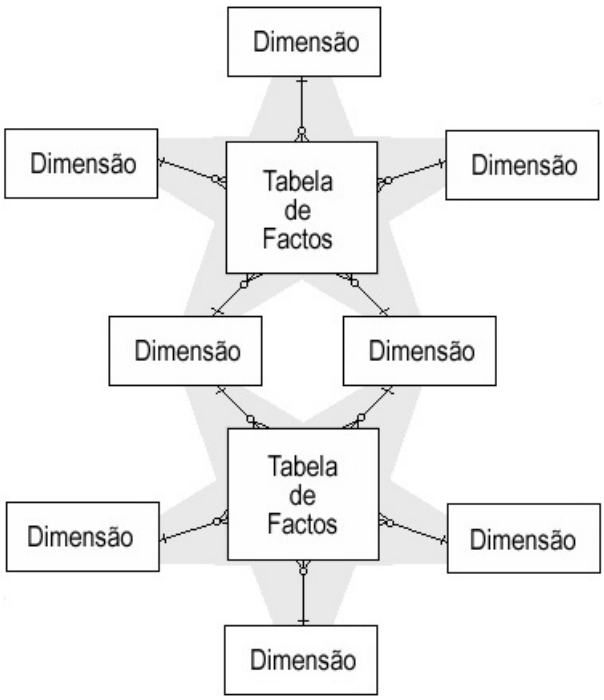


Figura 2.5: Esquema em constelação

Por sua vez, **o esquema em floco de neve** é um esquema em estrela que apresenta algumas (ou todas) as tabelas de dimensão completamente normalizadas.

Tem como vantagem indicar explicitamente a estrutura das suas dimensões e evitar que seja armazenada informação redundante, por causa da normalização. É

por isso possível afirmar que é o mais semelhante ao modelo relacional [32]. Uma representação deste esquema pode ser vista na Figura 2.6.

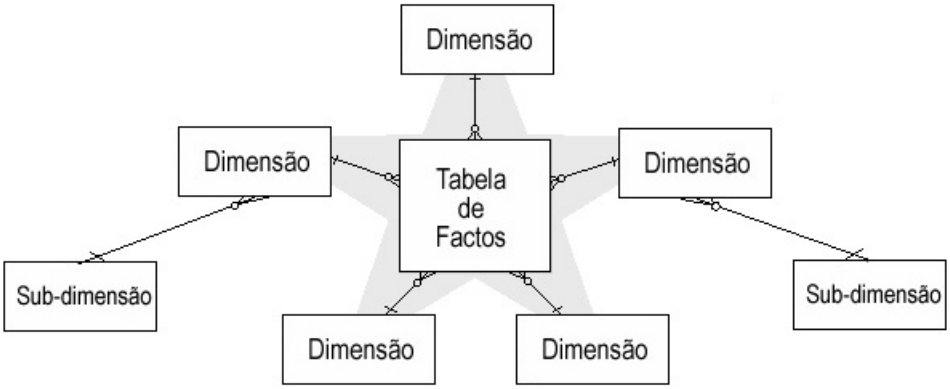


Figura 2.6: Esquema em floco de neve

2.5 Metadados

Metadados é uma expressão utilizada para referir "dados sobre dados". Segundo Inmon [31] metadados serve como *a road map to a miner*, i.e. define-se não para descrever o conteúdo, mas o contexto da informação. Pode-se concluir que um dos pontos mais importantes na documentação das aplicações OLAP e ambiente DW/DMart é a definição dos seus metadados. De forma semelhante, Marco [37] define metadados como "conhecimento" e são "o conjunto de todos os dados físicos e do conhecimento, de dentro e para dentro da organização, incluíndo informação sobre os dados físicos, os processos, regras e restrições dos dados e, estruturas utilizadas por uma organização".

As principais vantagens dos metadados são:

1. Identificação das fontes operacionais;
2. Indicação das nomenclaturas dos objetos;
3. Segurança da informação;
4. Definições físicas e lógicas das colunas, tabelas e atributos da base de dados;
5. Implementação de regras e automatização das hierarquias dimensionais.

2.6 Processo ETL num sistema *Data Warehouse*

A ferramenta ETL tem como função a extração de dados de diversos sistemas, transformação desses dados conforme regras de negócios e por fim a o carregamento dos dados geralmente num DW e DMart. Imhoff [30] descreve o ETL como "o processo de extração de dados de um sistema fonte e depois da transformação os dados são apresentados de uma forma mais aceitável para o DW". Mundy *et al.* [40] concluem que "um sistema ETL é o centro de fiabilidade de um sistema de Business Intelligence".

Consome facilmente 70% dos recursos necessários à implementação e manutenção de um DW típico [33]. É considerada duma importância vital porque permite tratar a complexidade encontrada nos dados, a sua posterior limpeza e transformação de forma a garantir a sua qualidade e permitir depois o respetivo carregamento no DW.

3

Receção dos dados

A recolha dos registos da atividade elétrica depois de contração voluntária é feita em ambulatório, por um dispositivo com capacidades de armazenamento e comunicação reduzidas.

Assim, é necessário que o armazenamento dessa informação seja feita de forma persistente num sistema externo. Na fase de especificação decidiu-se que seria disponibilizado um ponto único para envio de dados, uma vez que a cadência de recolha não obedece a nenhum escalonamento previsível. Ficou definido que o sistema de envio apenas necessita de ter acesso à internet para fazer o envio dos dados. Cada registo contém um ficheiro com meta dados e um outro, com as leituras de EMG. A informação sobre os pacientes e os músculos é disponibilizado em formato xls, pelo facto do cliente final estar mais familiarizado com ferramentas *Office*. Por se tratarem de registos clínicos, pessoais e confidenciais, é imprescindível que o envio dos dados seja feito de forma segura [46].

3.1 Solução proposta

A solução proposta para a recolha dos dados consiste num serviço que permite a receção dos sinais da função muscular, medidas de EMG, e a meta informação necessária para descrever o contexto de recolha.

O serviço permite a receção de qualquer tipo de ficheiro, independentemente do

seu tamanho (limitado à capacidade física da máquina onde fica instalado o serviço). Um dos requisitos funcionais de grande importância foi garantir a segurança e confidencialidade dos dados transferidos. A meta informação inclui, entre outros, informação sobre o músculo utilizado, a data de recolha e o paciente. O serviço servirá principalmente para um envio automático dos dados recolhidos. Foi implementada uma aplicação web que faz uso do serviço e permite um envio manual de dados. Por exemplo, permitirá ao médico que acompanha o doente o envio de informação relacionada com este.

A solução foi construída, utilizando ferramentas *open source* e *standards*, permitindo assim funcionar em sistemas heterogéneos, com maior flexibilidade de implementação. Deu-se particular relevância à simplicidade da solução, com o objetivo de a tornar robusta, eficiente e com um ótimo desempenho.

Segue uma breve descrição das principais técnicas e tecnologias usadas.

Simple Object Access Protocol[1] (SOAP) Protocolo criado para troca de informações estruturadas numa plataforma descentralizada e distribuída. Um *standard* de facto utilizado em tecnologias de serviços *web*, representando um protocolo de troca de mensagens formatados em *eXtensible Markup Language* (XML) com determinadas regras. Um documento SOAP XML é designado por SOAP *message* ou SOAP *envelope* e obedece a um XML *Schema*, como se verifica na Figura 3.1. Os perfis básicos da *Web Services interoperability* (WS-i) definem que em serviços *web* se usa SOAP via *HyperText Transfer Protocol* (HTTP), ou outro como, por exemplo, *Simple Mail Transfer Protocol* (SMTP).

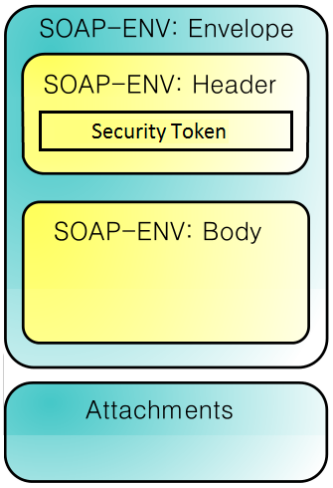


Figura 3.1: Estrutura da mensagem SOAP
(Consultado de: https://en.wikipedia.org/wiki/SOAP, Maio 2015)

[1] http://www.w3.org/TR/soap12/

Foi escolhido o tipo SOAP em vez do tipo *Representational State Transfer* (REST), porque a segurança dos dados transmitidos é um dos requisitos funcionais. Neste caso, a segurança utilizando serviços SOAP está bem padronizada usando *WS-Security* ao contrário dos serviços REST onde não existem normas, sendo necessário implementar a sua própria segurança. Outra razão para a escolha deste protocolo é existir a *Web Service Definition Language* (WSDL) do serviço. Assim podemos descobrir automaticamente o serviço e gerar um *proxy client* de uma forma simples (gerar as chamadas de serviço, os tipos de dados necessários para os métodos e assim por diante) [5]. Por fim, utilizando *Message Transmission Optimization Mechanism* (MTOM) , tem-se um elevado desempenho na transferência de anexos binários.

Apache CXF[2] é uma *framework open source* que dá suporte à criação e consumo de serviços *web* de uma forma intuitiva e simples. Dois projetos foram combinados por pessoas que trabalham em conjunto na *Apache Software Foundation* e o nome, CXF, vem da combinação dos nomes desses mesmos projetos: "Celtix" e "XFire". Suporta uma variedade de modelos de programação tipo *frontend*, como *Java API for XML Web Services* (JAX-WS) e *Java API for RESTful Web Services* (JAX-RS). Os serviços criados com estas APIs utilizam SOAP ou REST e trabalham sobre o protocolo de transporte HTTP [2].

As considerações principais sobre o *design* da CXF incluem:

- separação completa de *frontend*, como JAX-WS, e a implementação do código serviço;
- alto desempenho com sobrecarga computacional mínima;
- simplicidade, por exemplo, na criação de clientes e *endpoints*;
- componentes de serviços *web* embutidos: inclui a *Spring Framework* e *Geronimo*.

Sendo o serviço de tipo SOAP, o *frontend* utilizado foi JAX - WS.

MTOM[3] é uma norma que permite a transferência de ficheiros binários, em tecnologias de serviços *web*, de forma eficiente, e que é apoiada tanto por OASIS WS-i[4] como por múltiplos fabricantes (Apache, Microsoft).

O envio de dados binários de diferentes formatos, utilizando serviços *web* SOAP, pode ser realizado a partir de duas técnicas [6]:

[2] http://cxf.apache.org/
[3] http://www.w3.org/TR/soap12-mtom/
[4] http://www.oasis-ws-i.org/about

1. **por valor** - é conseguida através da incorporação de dados (após a codificação base64) como um elemento duma componente de dados XML. Esta técnica apresenta desvantagens, porque aumenta o tamanho e causa uma sobrecarga de processamento relacionada com os custos de transformação, especialmente quando volta para a descodificação binária.
2. **por referência** - é alcançado adicionando dados binários (não transformados) como entidades não analisadas externas fora do documento XML, e incorporar referências URIs para esta entidade, como elementos ou valores de atributos. Isso impede dados desnecessariamente excessivos e um processamento lento. No entanto, o principal obstáculo para o uso dessas entidades é a sua forte dependência de *Document Type Definition* (DTD), o que impede a modularidade, bem como o uso de *namespaces* XML.

MTOM recorre à especificação XML Binary Optimized Packaging (XOP) para referenciar os anexos binários da mensagem. Com o uso deste elemento, o anexo binário é visto como parte da mensagem SOAP, mesmo que na verdade estejam separados. Isso permite à aplicações processar o XML, mais curto, tornando a dependência de DTDs desnecessária. O MTOM padronizou o mecanismo de referenciar os *SOAP with Attachments* (SwA).

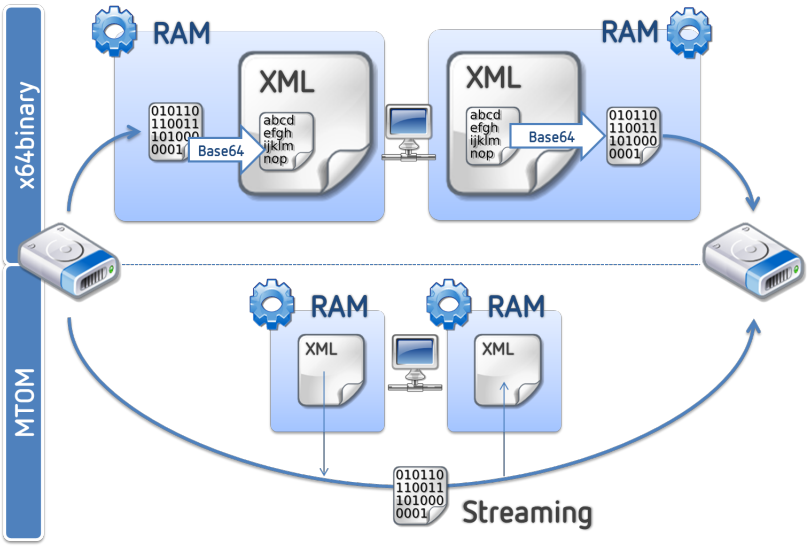


Figura 3.2: Comparação entre envio usando XML base64 e MTOM[5]

Em suma, a utilização deste mecanismo permite enviar o conteúdo dos ficheiros em formato binário e em modo *streaming*. Por estes motivos, tanto o servidor como o cliente do serviço economizam tempo, evitando o processo de transformação e depois a manipulação em memória do XML, como se vê na Figura 3.2, e

[5]Retirado de: https://angelborroy.wordpress.com/2012/08/

remove qualquer limite no tamanho do ficheiro que é transmitido.

Java e Maven[6] Todo o código foi escrito em linguagem Java e os projetos criados foram do tipo Maven. Apache Maven é uma ferramenta para a estruturação de projetos, gestão do ciclo de vida e, em particular, a gestão de dependências. Este último aspeto pode revelar-se complexo quando se integram múltiplos projetos. Para isso, utiliza um arquivo XML *Project Object Model* (POM) para descrever o projeto, as suas dependências sobre módulos e componentes externos, a ordem de compilação, diretórios e *plugins* necessários.

Tomcat[7] Foi utilizado como servidor aplicacional tanto para o serviço como para a aplicação *web*. O Tomcat é desenvolvido pela *Apache Software Foundation* e distribuído como *software* livre. É um servidor de aplicações Java para *web* que implementa entre outras, as tecnologias Java Servlets e JavaServer *Pages* (JSP). Tem a capacidade de atuar também como servidor *web*, ou pode funcionar integrado a um servidor *web* dedicado como o Apache. Tomcat é inteiramente escrito em Java e, portanto, para ser executado necessita de uma Java *Virtual Machine* (JVM) instalada.

3.2 Implementação do serviço

O serviço criado - *UploadFileService* - implementa uma interface que disponibiliza um contrato com as características e operação mostradas na Listagem 3.1.

```
1 @WebService(name = "UploadFile", targetNamespace =
    "http://service.alsmon.isel.pt/")
  @Policies({ @Policy(uri = "SecurityPolicy.xml", placement =
    Policy.Placement.SERVICE) })
3 public interface UploadFile {
  @WebMethod  Boolean uploadFile(@WebParam(name="file")
    FileInformation file);
5 }
```

Listagem 3.1: Contrato do serviço

A operação do contrato - `UploadFile` - descrita na Listagem 3.1, recebe como parâmetro um objeto de tipo `FileInformation` (linha 4) e devolve como resultado do tipo `Boolean`. O parâmetro contém vários atributos, sendo o mais

[6] https://maven.apache.org/
[7] http://tomcat.apache.org/

importante o *Dfile* do tipo `DataHandler`, que tem o conteúdo do ficheiro em formato binário.

Na Listagem 3.2 é ilustrada a implementação da interface, através da classe `UploadFileImpl` (linha 3).

A criação de serviços *web* implica sempre uma série de elementos de configuração. Assim foram usadas anotações para reduzir a quantidade de código e dados de configuração necessários à implementação.

```
1 @WebService(targetNamespace = "http://service.alsmon.isel.pt/",
    endpointInterface = "pt.isel.alsmon.service.UploadFile",
    portName = "UploadFilePort", serviceName =
    "UploadFileService")
  @MTOM(enabled = true, threshold = 10240)
3 public class UploadFileImpl implements UploadFile {
    @WebMethod
5   public Boolean uploadFile(FileInformation file) {
    //...
7   } }
```

Listagem 3.2: A classe que implementa a interface

Na Listagem 3.2, na linha 1, utilizando a anotação @WebService, foi definido o *namespace* xml do wsdl (mapeado do nome do *package* onde está implementado o serviço), a interface que define o contrato do serviço, a porta e o nome do serviço. Utilizando a anotação @MTOM foi configurado o mecanismo MTOM para estar ativado quando o tamanho dos dados for igual ou superior a 10 KB.

O nome dos ficheiros recebidos são gravados em formato "yyyy_MM_dd-HH-mm-ss-SSS", em função da data e da hora da receção. Pretende-se com este esquema de nomes, simplificar o processamento na parte do ETL do DW, permitindo uma rápida deteção da data de receção dos ficheiros. No entanto uma alteração neste esquema de nomes não quebra o funcionamento do ETL. Neste caso, os ficheiros sendo processados por outra ordem e não aquela de receção.

Todas as variáveis suscetíveis a futuras alterações como e.g. a localização onde são guardados os ficheiros recebidos, foram definidas através de ficheiros de configuração.

A estrutura da implementação do serviço contém ainda uma classe que implementa a parte de segurança e que será detalhada na Secção 3.5.

Depois da compilação, o projeto foi exportado em formato "WAR"(*Web Application ARchive*) e instalado no Tomcat.

O endereço com a informação sobre o serviço disponibilizado ficou no URL: `https://alsmon.adeetc.e.ipl.pt:8443/UploadFileService/services/`

O endereço do contrato do serviço ficou no URL:

`https://alsmon.adeetc.e.ipl.pt:8443/UploadFileService/services/UploadFile?wsdl`

3.3 Exemplo de cliente para o serviço

Como vimos na secção anterior, o serviço disponibiliza um endereço com uma descrição WSDL. Utilizando este endereço, a criação do *proxy* para o serviço foi realizada de forma automática, através de uma ferramenta disponibilizada pela *framework* Apache CXF - `wsdl2java` -, como na Listagem 3.3.

```
wsdl2java -client -verbose https://alsmon.adeetc.e.ipl.pt:8443/
    UploadFileService/services/UploadFile?wsdl
```

Listagem 3.3: Criação do *proxy*

O *proxy* foi importado no projeto cliente, permitindo as chamadas à operação disponibilizada pelo serviço, como apresentado na Figura 3.3 e na Listagem 3.4.

```
//...
URL wsdlURL = UploadFileService.WSDL_LOCATION;
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(UploadFile.class);
factory.setAddress(wsdlURL.toString());
UploadFile sproxy = (UploadFile) factory.create();
((SOAPBinding) ((BindingProvider)
   sproxy).getBinding()).setMTOMEnabled(true);
//...
Boolean result = sproxy.uploadFile(_uploadFile);
//...
```

Listagem 3.4: Exemplo de chamada ao método do serviço

Como no serviço, também o cliente foi configurado para obter variáveis sujeitas a alteração num ficheiro de configuração, e.g. a localização dos ficheiros a serem

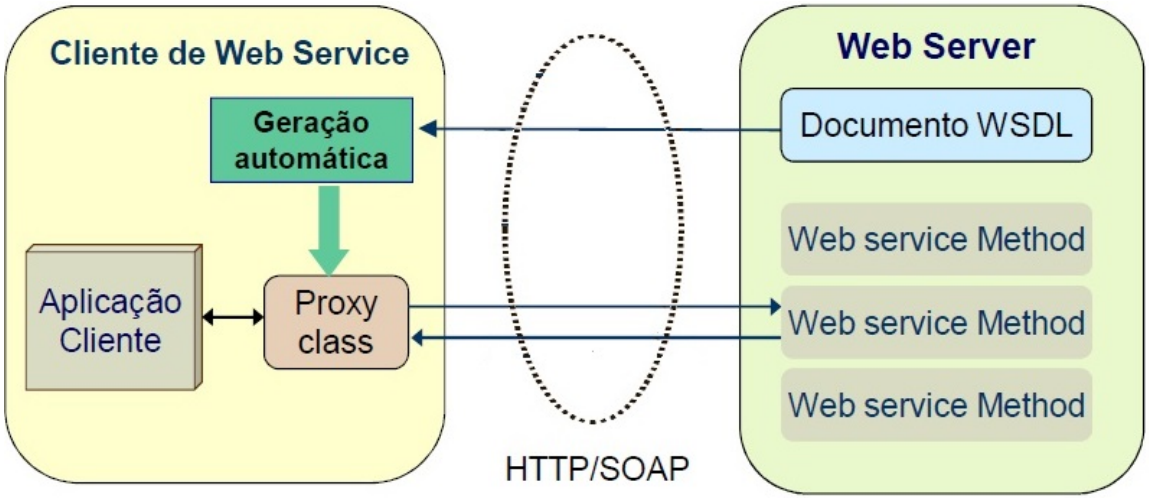


Figura 3.3: A comunicação entre o cliente e o serviço

enviados e a localização onde ficam gravados os ficheiros enviados.

No cliente, antes de serem enviados, os dados são comprimidos. Esta operação permite colocar num *zip* os dados correlatos (eletromiografia e metadados) para depois serem corretamente processados no DW. Esta restrição faz parte de um protocolo definido entre as partes: quem envia os dados EMG e quem os recebe e processa.

A aplicação cliente permite ser tolerante a falhas. O cliente está "à escuta" numa diretoria onde são colocados os dados dos exames, um por diretoria, sendo posteriormente enviados para o serviço, como arquivos. Se o arquivo for transferido com sucesso, ele é movido localmente para outra diretoria. No caso de falha, esta operação não é realizada, o arquivo respetivo fica disponível para ser transferido na próxima vez que o cliente é executado.

A parte de código que trata a segurança dos dados será explicada, juntamente com a do serviço, na Secção 3.5.

O projeto cliente foi exportado em formato "JAR" (*Java ARchive*), podendo ser executado diretamente ou dentro de um *batch*, de forma automática.

3.4 Implementação da aplicação *web*

Além do serviço foi criada uma aplicação *web*, Figura 3.4, que permite que os médicos possam realizar diretamente o *upload* dos ficheiros com a informação sobre os doentes.

Como se pode observar na Listagem 3.5, a aplicação *web* utiliza o serviço para realizar o *upload*.

```
//...
public class UploadFileWebClient extends HttpServlet {
  private static final long serialVersionUID = 1L;
  private static QName SERVICE_NAME;
  protected void doPost(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
   IOException {
    SERVICE_NAME = new
   QName("http://service.alsmon.isel.pt/","UploadFileService");
    URL wsdlURL = UploadFileService.WSDL_LOCATION;
    UploadFileService service = new UploadFileService(wsdlURL,
   SERVICE_NAME);
//...
```

Listagem 3.5: A aplicação *web* utilizando o serviço

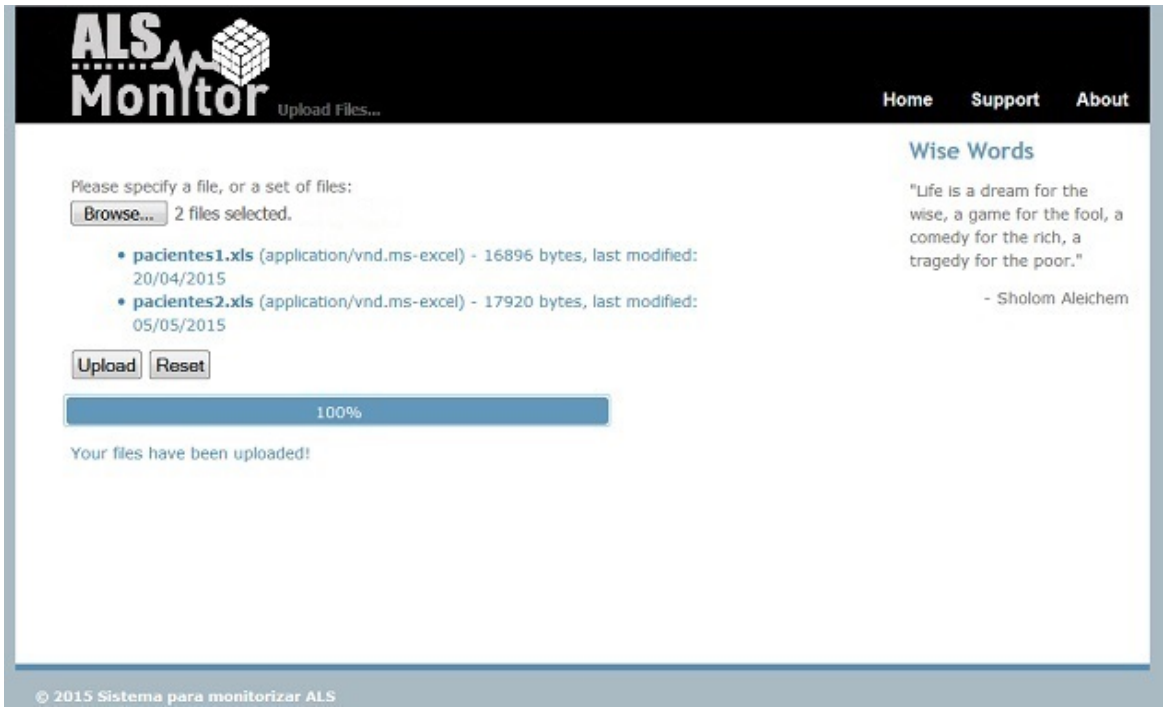


Figura 3.4: A aplicação *web* que permite o *upload* de ficheiros

O projeto foi exportado em formato "war" e instalado no Tomcat.

O endereço da aplicação *web* ficou no URL:

`https://w3id.org/als-telemonitoring/MDUpload`

Esta solução fornece escalabilidade, podendo esta aplicação estar instalada numa máquina diferente daquela em que está instalado o serviço. A aplicação e em especial o serviço podem ser replicados para várias máquinas se se verificar aumento no processamento.

O serviço será utilizado principalmente para o envio automático dos dados, enquanto esta aplicação *web*, fazendo uso do serviço, permite um envio manual de dados, através duma interface *web*.

3.5 Segurança dos dados

Para garantir a segurança dos dados foi utilizado o protocolo *Secure Sockets Layer* (SSL). Este protocolo fornece a confidencialidade e a integridade de dados por intermédio da autenticação do servidor e da encriptação dos dados transmitidos. O protocolo impede ainda que intermediários entre as duas extremidades da comunicação obtenham acesso indevido ou falsifiquem os dados que estão a ser transmitidos.

A autenticação dos clientes do serviço foi implementada utilizando a norma *WS-Security* e dos clientes da aplicação *web* utilizando *basic authentication*, em ambos casos, através do nome do utilizador e palavra-chave.

Configuração do protocolo SSL no Tomcat

A configuração do SSL é realizada usando o ficheiro de configuração do Tomcat, `server.xml`. Inicialmente é criado um repositório de chaves e certificados, contendo a chave privada do servidor e um certificado auto-assinado[8], utilizando a ferramenta java - `keytool` -, como na Listagem 3.6.

```
keytool -genkey -alias tomcat -keyalg RSA -validity 365 -keystore
    /tomcat/conf/keystore/keystore.jks
```

Listagem 3.6: Criação do repositório

Em seguida, o ficheiro de configuração é alterado, Listagem 3.7, para permitir o acesso ao servidor a qualquer aplicação alojada nele utilizando o protocolo SSL.

[8]No futuro, quando o produto passar para a fase de produção, tem de ser utilizado um certificado emitido pela Autoridade de Certificação (CA)

```
<Connector port="8443"
   protocol="org.apache.coyote.http11.Http11NioProtocol"
   sslProtocol="TLS" secure="true" scheme="https"
   maxThreads="200" keystorePass="changeit"
   keystoreFile="conf/keystore/keystore.jks" clientAuth="false"
   SSLEnabled="true"/>
```

Listagem 3.7: Configuração do protocolo SSL no Tomcat

Autenticação do servidor perante os clientes

A autenticação do servidor é realizada através do certificado auto-assinado criado. Para os clientes da aplicação *web*, como o acesso é feito via *browser*, basta confiar no certificado apresentado. Para os clientes do serviço a configuração é mais complexa, dado que não se pode confiar no certificado de uma forma explicita mas só por configuração. Esta configuração pode ser feita de duas maneiras:

1. O certificado é exportado do servidor e depois importado na *cacerts* do java do cliente. Assim o certificado fica "*trusted*" para qualquer aplicação, não sendo necessário mais nenhuma configuração do lado da aplicação cliente.
2. O certificado é exportado do servidor e depois importado num repositório de certificados criado do lado da aplicação cliente. Depois a aplicação cliente é configurada para procurar os certificados "*trusted*", além dos *cacerts* do java, também neste repositório criado. Na Listagem 3.8, o repositório é definido nas linhas 6 e 7, como "*trusted*" na linha 9 e, a configuração do protocolo SSL para usar este repositório é descrita nas linhas 11 e 12.

Foi escolhida a segunda opção, porque mesmo sendo necessário mais desenvolvimento, uma vez feito, tem a vantagem da instalação da aplicação cliente ficar transparente, sem nenhuma configuração extra, i.e necessidade de ter o certificado e depois importá-lo na *cacerts* do java para cada dispositivo onde fica instalado o cliente do serviço.

Autenticação do cliente do serviço perante o servidor

Como todo o tráfego entre o cliente e o servidor é encriptado, para evitar a complexidade da utilização dos certificados do lado dos clientes, a autenticação foi implementada através do nome do utilizador e palavra-chave, utilizando a norma *WS-Security*.

```
private void setupTLS(UploadFile proxy) throws
    FileNotFoundException, IOException, GeneralSecurityException
    {
     HTTPConduit httpConduit = (HTTPConduit)
     ClientProxy.getClient(proxy).getConduit();
     TLSClientParameters tlsCP = new TLSClientParameters();
     String keyPassword = "changeit";
     KeyStore trustStore = KeyStore.getInstance("JKS");
     trustStore.load(this.getClass().getResourceAsStream
     ("clientkeystore.jks"), keyPassword.toCharArray());
     TrustManager[] myTrustKeyStoreManagers =
     getTrustManagers(trustStore);
     tlsCP.setTrustManagers(myTrustKeyStoreManagers);
     httpConduit.setTlsClientParameters(tlsCP);
   }
```

Listagem 3.8: Configuração do repositório de certificados

Neste sentido foi criada uma classe de tipo "`CallbackHandler`", sendo usada no lado do cliente para configurar as credenciais e no lado do servidor para verificar estas credenciais.

Depois foram utilizados os *Interceptors*, que fazem uso das classes acima criadas. No lado do cliente são do tipo `WSS4JOutInterceptor`, criando o Security Token a ser adicionado ao *header* das mensagens SOAP, com as credenciais. No lado do servidor são do tipo `WSS4JInInterceptor`, e servem para para analisar as mensagens SOAP recebidas e verificar o utilizador e a palavra-chave enviados. Na Listagem 3.9 é apresentado um exemplo da utilização do `WSS4JOutInterceptor` na aplicação cliente do serviço. Nesta listagem são utilizadas as constantes provenientes da biblioteca "wss4j", da *framework* CXF, para realizar a configuração necessária.

Por fim, na definição do contrato (declaração da interface), é adicionada uma *WS-SecurityPolicy* de tipo *UsernameToken*, obrigando os clientes a se autenticarem com o nome do utilizador e a palavra-chave antes mesmo de utilizar o serviço.

Autenticação do cliente da aplicação *web* perante o servidor

A autenticação do cliente da aplicação *web* é realizada através do nome do utilizador e palavra-chave, utilizando *basic authentication*. Para aplicações *web* pode ser utilizado o servidor Tomcat para implementar este tipo de autenticação, através

do ficheiro de configuração *tomcat-users.xml*, definindo uma nova "*role*" e, criando utilizadores e palavras-chaves com esta "*role*". Na aplicação *web* é utilizado o ficheiro *web.xml* para realizar a configuração necessária, ver Listagem A.1. Fica depois ao cargo do servidor tomcat realizar a ligação entre a configuração feita nos dois ficheiros, utilizando a "*role*" definida.

```
//...
Map<String, Object> outProps = new HashMap<String, Object>();
outProps.put(WSHandlerConstants.ACTION,
    WSHandlerConstants.USERNAME_TOKEN);
outProps.put(WSHandlerConstants.USER, "alsmon");
outProps.put(WSHandlerConstants.PASSWORD_TYPE,
    WSConstants.PW_TEXT);
outProps.put(WSHandlerConstants.PW_CALLBACK_REF, new
    ClientPasswordCallback());
WSS4JOutInterceptor wsOut = new WSS4JOutInterceptor(outProps);
wsOut.setAllowMTOM(true);
factory.getOutInterceptors().add(wsOut);
//...
```

Listagem 3.9: Configuração da autenticação na aplicação cliente

4

Módulo de *Data Warehouse*

É necessário armazenar de forma persistente os dados recolhidos dos exames efetuados em ambulatório. Como é relevante para este domínio de aplicação a possibilidade de análise histórica dos dados, contextualizados por um conjunto de descritores, decidiu-se que o armazenamento seria feito num DW.

4.1 Solução proposta

O DW foi construído seguindo a metodologia do Kimball [33], de tipo *bottom-up*. Como no âmbito deste projeto foram utilizados dados obtidos da monitorização do EMG, o DW é constituído por um único DMart. A modelação do DMart foi realizada através do modelo habitualmente utilizado, o modelo em estrela. O DW tem uma área temporária, denominada de *data staging*, onde os dados serão transformados, processados e guardados temporariamente antes de passar para o modelo em estrela. Todo este processamento é realizado através do processo de ETL. A arquitetura utilizada está ilustrada na Figura 4.1.

Foi definido um cubo multidimensional, de tipo ROLAP, para permitir consultas OLAP por ferramentas genéricas. Foram criados pré-agregados para otimizar algumas consultas consideradas relevantes.

Na construção do DW teve-se em conta a realização de uma solução robusta, com um bom desempenho e com suporte multilingue. A solução permite adicionar outros DMarts, representando diferentes tipos de exames que futuramente

possam ser necessários. Esta adição não necessita de alterar a estrutura existente, bastando adicionar paralelamente mais tabelas de facto reutilizando as dimensões existentes.

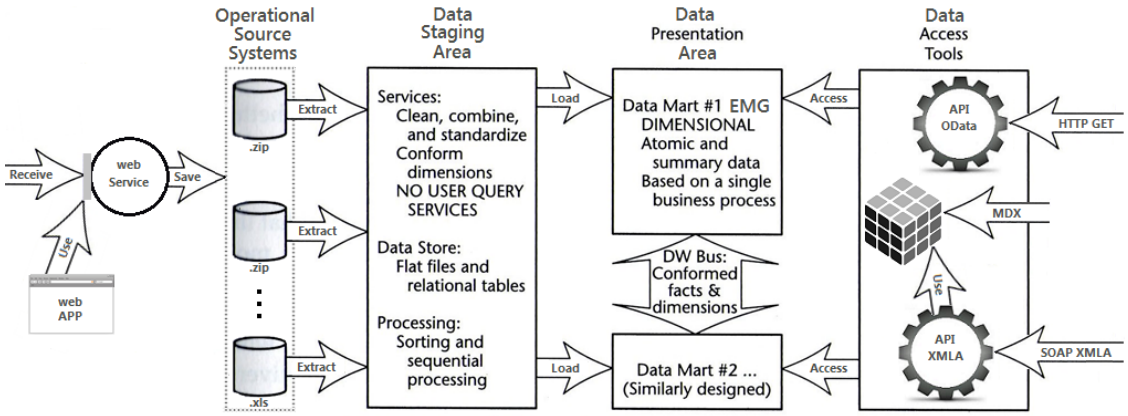


Figura 4.1: Arquitetura da solução (Adaptada de [33])

A solução foi construída utilizando ferramentas *open source*. Todo o processamento ETL foi desenvolvido para ser executado de forma automática, sem intervenção humana. A solução foi configurada para, em caso de erro ou alguma falha, enviar duma forma automática um *email* ao responsável com o relatório detalhado sobre o erro ou falha. Na secção seguinte é feita uma breve descrição destas ferramentas e das tecnologias utilizadas e a razão da sua escolha.

4.2 Ferramentas e tecnologias utilizadas

No mercado atual existe uma grande variedade de ferramentas de BI *open source*, como Pentaho, SpagoBI, OpenI, JasperSoft, Palo, Vanilla [1]. Foi realizado um estudo neste domínio para escolher a plataforma que melhor se adequa aos requisitos.

Como ilustrado na Figura 4.2, as duas plataformas, Pentaho e a SpagoBI, disponibilizam todas as ferramentas necessárias à implementação duma solução completa de BI. A escolha recaiu sobre a *suite* Pentaho, uma vez que é aquela que é mais popular e com melhor potencial [38].

A plataforma Pentaho[1] é uma solução madura, amplamente utilizadas por milhares de empresas. Em Portugal é utilizada pelo Governo de Portugal, a Brisa, a AIRC, entre outras [3]. Mesmo com uma curva de aprendizagem maior, permite

[1] http://www.pentaho.com/

Features	Open Source Business Intelligence					
	JasperSoft	*OpenI*	*Palo*	*Pentaho*	*SpagoBI*	*Vanilla*
Reports	✓	✓	✓	✓	✓	✓
Graphics	✓	✓	✓	✓	✓	✓
Dashboards	✓	✓	✓	✓	✓	✓
OLAP	✓	✓	✓	✓	✓	✓
ETL	✓	✗	✓	✓	✓	✓
Data Mining	✗	✓	✗	✓	✓	✓
KPI	✗	✗	✗	✓	✓	✓
Data export	✓	✗	✓	✓	✓	✓
GEO/GIS	✓	✗	✗	✓	✓	✗
Ad-hoc queries	✓	✗	✓	✓	✓	✓

Figura 4.2: Comparação das características de soluções BI *open source* [15]

um grande leque de opções, tendo todas as ferramentas necessárias à implementação duma solução completa de BI.

A *suite* é disponibilizada em duas versões: comercial e *open source*. Neste projeto foi utilizada a versão *open source*, Pentaho Community Edition[2]. A diferença entre eles é que a versão comercial tem suporte e ainda disponibiliza vários repositórios e tem uma oferta mais alargada em termos de ferramentas OLAP.

Na Figura 4.3 é apresentada a arquitetura da *suite* Pentaho.

Além dos *data sources* utilizados a *suite* é constituída por duas partes: a plataforma de BI propriamente dita (aplicação *web*) e as ferramentas de desenvolvimento, que criam o conteúdo (o DW) necessário à plataforma de BI.

A plataforma de BI, o **Pentaho BI Server**, é utilizada para explorar os dados guardados no DW através de navegador OLAP, relatórios, entre outros. Foi instalada só com propósito de testes, para verificar o funcionamento do DW.

Segue-se uma breve descrição das principais ferramentas de desenvolvimento do Pentaho, que foram utilizadas para a criação dos artefactos da solução do DW. Todas estas ferramentas são programas java e precisam de uma JVM instalada.

[2] http://community.pentaho.com/

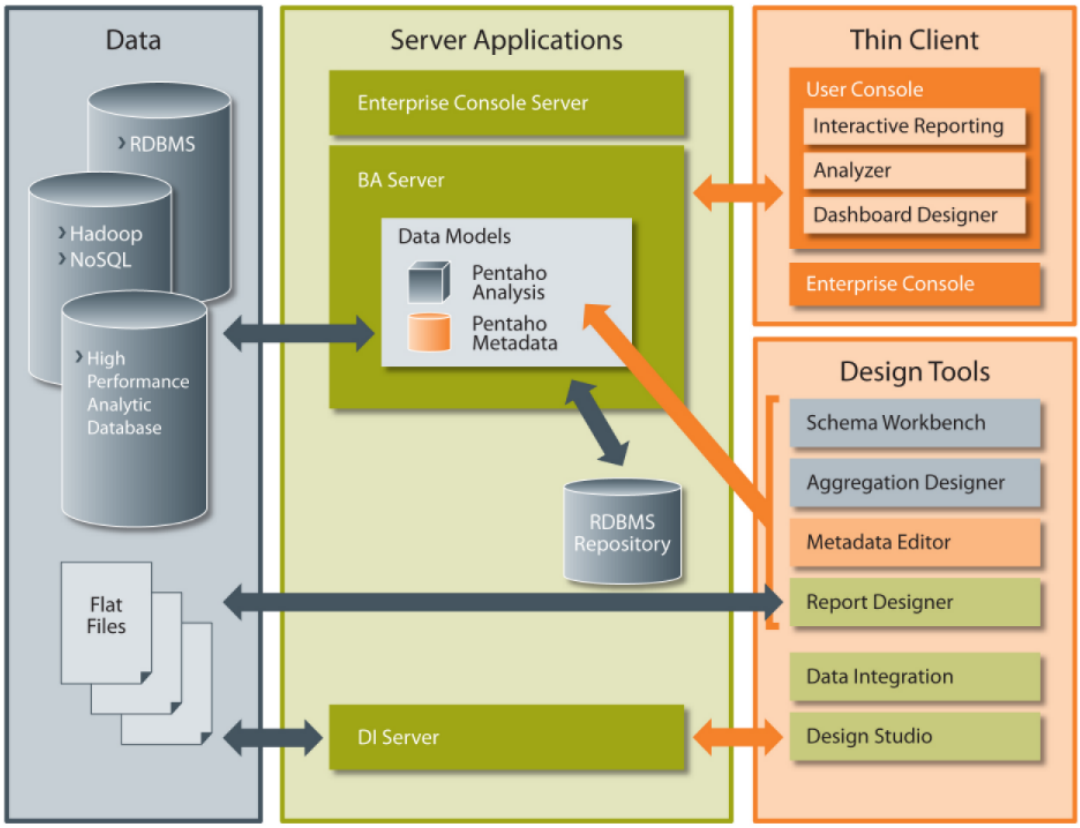


Figura 4.3: A arquitetura da plataforma Pentaho
(Consultado de: `http://infocenter.pentaho.com`, Maio 2015)

Pentaho Data Integration (PDI - Kettle) é a ferramenta utilizada na implementação dos processos de ETL no DW. Utilizando um ambiente gráfico, desenvolvem-se sequências de *transformations* e *jobs*, ligadas por restrições de controlo de fluxo. Estas sequências, semelhantes a *workflows*, são executadas através de utilitários como *kitchen*, *pan* ou *carte*.

Pentaho Schema Workbench (PSW) é utilizado para a criação do cubo multidimensional. A aplicação tem uma interface visual para navegar entre as definições do cubo, permitindo criar métricas, dimensões e hierarquias, entre outras. A informação gerada, metadados do cubo, é guardada em formato xml.

Pentaho Aggregation Designer (PAD) é a ferramenta utilizada para a criação e a instalação de pre-agregações necessárias a otimização do desempenho, na execução das consultas MDX.

A *suite* Pentaho funciona com muitos tipos de SGBD. No âmbito neste projeto de dissertação foi escolhido o MySQL, ferramenta descrita a seguir.

O **MySQL**[3] é um dos SGBDs mais populares sendo utilizado em todo o mundo [4]. Utiliza a linguagem SQL como interface e é disponibilizado em duas versões: comercial e *open source*, denominada MySQL Community Edition, que foi usado no projeto. Entre as características que o tornou um dos mais popular enumeramos: a portabilidade, funcionando praticamente em qualger plataforma; o excelente desempenho e estabilidade; a flexibilidade em termos de recursos *hardware*; a facilidade na utilização; e, a compatibilidade [7].

No desenvolvimento do modelo de dados, foi tido em consideração utilizar tipos de dados mais adequados aos requisitos para otimizar o desempenho e o espaço ocupado. Teve-se a preocupação de não utilizar tipos de dados específicos só a MySQL para não criar dependências a um SGBD específico e poder alterá-lo de forma simples. Nesse sentido, foram feitos testes com MariaDB[4]. A alteração entre os dois SGBDs é feita sem problemas, não sendo necessário alterar nada em termos dos esquemas criados: exportar do MySQL, importar no MariaDB e alterar no Pentaho a configuração da conexão e o *driver* utilizado.

As ferramentas escolhidas foram instaladas e configuradas numa máquina virtual que foi disponibilizada para o efeito. Para cada ferramenta foram criados e configurados *workspaces* onde ficam guardados os artefactos da solução de DW. Na criação destes artefactos foram utilizadas variáveis, para tudo o que possa ser objeto de alteração. Estas variáveis foram guardadas em ficheiros de configuração. No Anexo A.2 segue um exemplo dum ficheiro de configuração, para a ferramenta Pentaho Data Integration.

4.3 Processo de desenvolvimento do *Data Warehouse*

4.3.1 Factos e a sua natureza

Neste trabalho foram utilizados os dados provenientes de um único tipo de exame, a eletromiografia de superfície (sEMG), considerado um método de estudo da função muscular, que consiste em registar a atividade elétrica dos músculos. Assim os factos principais que se pretendem analisar provêm deste exame, representado em forma de série temporal.

Os dois factos principais que foram extraídos desta serie temporal:

[3] http://www.mysql.com
[4] https://mariadb.org

- ***area* (*Area Under Curve* — AUC)**: é a área da superfície sob a curva da série temporal e é utilizada para representar o valor do tónus muscular;
- ***nr_peaks***: representa o número de picos da série temporal e é utilizado para confirmar se o exame foi realizado corretamente (o respetivo registo, para ser considerado válido, tem que ter o valor do *nr_peaks* igual a três).

O facto ***area*** não tem restrição de adição, em qualquer dimensão, portanto é um facto de natureza aditiva. O ***nr_peaks*** é de natureza não-aditiva.

O algoritmo em java para a cálculo destes dois factos, a partir dos dados da série temporal, é descrito na Secção 4.4.

Os outros factos, de natureza não-aditiva, foram extraídos dos metadados correspondentes ao exame:

- ***high_pass***: representa o valor máximo da frequência do sinal;
- ***low_pass***: representa o valor mínimo da frequência do sinal;
- ***electrod***: representa o tipo de elétrodo utilizado na realização do respetivo exame;
- ***protocol***: representa o protocolo que foi utilizado na realização do exame;

Estes factos ficaram guardados na tabela de factos, ***fact_emg***. Cada registo desta tabela representa assim um exame EMG realizado.

4.3.2 Dimensões

Após a identificação dos factos, foram determinadas as dimensões e respetivos atributos necessários à contextualização destes factos. Depois da reunião com a equipa médica, para definir as necessidades de análise, foram identificadas quatro dimensões necessárias para responder a todas as necessidades de análise.

Dimensão *dim_patient* - a tabela que guarda toda a informação sobre os pacientes (Tabela 4.1);

Dimensão *dim_muscle* - a tabela que guarda a informação sobre os músculos (Tabela 4.2);

Dimensão *dim_date* - a tabela que armazena informação relativa a data quando foi realizado o exame EMG (Tabela 4.3);

Dimensão *dim_time* - a tabela que armazena informação relativa a tempo quando foi realizado o exame EMG (Tabela 4.4);

Tabela 4.1: Descrição da tabela de dimensão *dim_patient*

dim_patient		
Atributo	**Tipo**	**Descrição**
keycol	int(identity)	Chave primária
patient_id	varchar(10)	Identificação do paciente
name	varchar(50)	Nome do paciente
gender	char(1)	Sexo
birthdate	date	Data de nascimento
diagnoseon	date	Data quando foi diagnosticado
createon	date	Data quando foi registado
updateon	date	Data quando foi atualizado
rowversion	varchar(18)	Versão da atualização
at_low	decimal(5,3)	Valor min músculo Anterior Tibialis
at_high	decimal(5,3)	Valor max músculo Anterior Tibialis
fcr_low	decimal(5,3)	Valor min músculo Flexor Carpi Radialis
fcr_high	decimal(5,3)	Valor max músculo Flexor Carpi Radialis
scm_low	decimal(5,3)	Valor min músculo Sternocleido Mastoideus
scm_high	decimal(5,3)	Valor max músculo Sternocleido Mastoideus

Tabela 4.2: Descrição da tabela de dimensão *dim_muscle*

dim_muscle		
Atributo	**Tipo**	**Descrição**
keycol	int(identity)	Chave primária
muscle_id	varchar(10)	Identificação do músculo
name	varchar(50)	Nome do músculo
acronym	varchar(10)	Abreviação do nome do músculo
side	varchar(25)	Lado em que se encontra

São as dimensões que definem a granularidade dos factos. A granularidade da tabela de factos ***fact_emg*** indica o valor do tónus muscular para um determinado paciente, por músculo, por dia e por segundo.

4.3.3 Modelo multidimensional

Como o DW foi construído com base só nos dados do exame EMG, ele é constituído por um único DMart. Com os factos e as dimensões definidas anteriormente, foi realizado o esquema multidimensional correspondente ao DMart. Assim, usando o modelo criado, é possível responder, por exemplo, a questões

Tabela 4.3: Descrição da tabela de dimensão *dim_date*

dim_date		
Atributo	**Tipo**	**Descrição**
keycol	int(identity)	Chave primária
date_iso	date	Data em formato internacional — iso
fulldate_en	varchar(50)	Data em formato extenso, em inglês
fulldate_pt	varchar(50)	Data em formato extenso, em português
dayofweek	smallint(6)	Número do dia da semana
dayofmonth	smallint(6)	Número do dia do mês (dd)
dayname_en	varchar(15)	Nome do dia em inglês
dayname_pt	varchar(15)	Nome do dia em português
weekofyear	smallint(6)	Número da semana do ano (ww)
monthnumber	smallint(6)	Número do mês (MM)
monthname_en	varchar(25)	Nome do mês em inglês
monthname_pt	varchar(25)	Nome do mês em português
quarter	smallint(6)	Número do trimestre
year	smallint(6)	Ano (yyyy)
year_weekofyear	char(9)	Ano - Número da semana do ano
year_monthname_en	varchar(32)	Ano - Nome do mês em inglês
year_monthname_pt	varchar(32)	Ano - Nome do mês em português
year_quarter	char(9)	Ano - Número do do trimestre

Tabela 4.4: Descrição da tabela de dimensão *dim_time*

dim_time		
Atributo	**Tipo**	**Descrição**
keycol	int(identity)	Chave primária
fulltime24	time	Tempo em formato HH:mm:ss
fulltime12	char(11)	Tempo em formato HH:mm:ss am/pm
hour24	smallint(6)	Hora (HH)
hour12	char(5)	Hora (HH am/pm)
minute	smallint(6)	Minuto (mm)
second	smallint(6)	Segundo (ss)
period_en	varchar(25)	Período em inglês (ex: *morning*, *afternoon*)
period_pt	varchar(25)	Período em português (ex: manha, tarde)

como: "Qual foi o valor médio do tónus muscular, dum músculo, para um paciente, num intervalo de um ano, durante a parte de manhã?".

O modelo de dados proposto assenta num esquema em estrela, representado na Figura 4.4.

O esquema em estrela integra a tabela de factos (***fact_emg***) e as quatro tabelas de

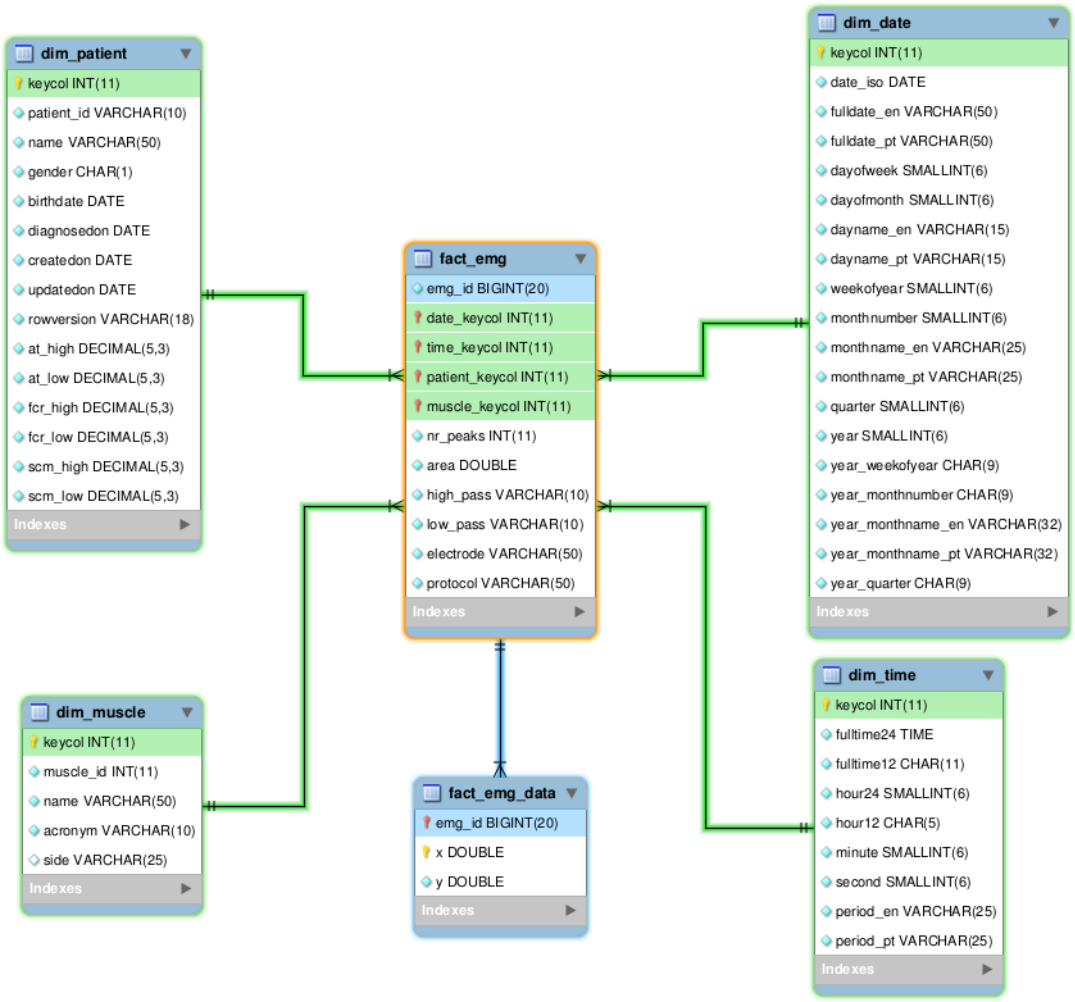


Figura 4.4: Modelo de esquema em estrela

dimensões (***dim_patient***, ***dim_muscle***, ***dim_date***, ***dim_time***).

A tabela de factos ***fact_emg*** está ligada as tabelas de dimensões, com quais está relacionada, por chaves estrangeiras (FK) que, em conjunto, formam a chave primaria (PK) desta tabela.

No esquema foi ainda adicionada a tabela ***fact_emg_data***, onde ficam guardados os dados da série temporal (exame de EMG). Assim, futuramente, se for necessário extrair mais alguma informação a partir desses dados, eles estão disponíveis no DW.

Temos a estrutura necessária, o esquema em estrela, e sabemos qual é a informação que precisamos. Na próxima secção vamos ver como é extraída e formatada esta informação e depois carregada nesta estrutura, de forma automatizada.

4.4 Processo de ETL

O processo do ETL (extração, transformação e carregamentos dos dados) foi dividido em duas etapas. Na primeira etapa foi tratada a complexidade encontrada nos dados. Os dados foram limpos e transformados (extraídos os factos e outra informação relevante) e colocados na área temporária, chamada *data staging*. Na segunda etapa os dados foram transferidos da *data staging* para o esquema em estrela, sendo preenchidas as tabelas de dimensão e as tabelas de factos. Ter uma zona temporária para guardar os dados antes de os passar para o esquema em estrela traz vantagens nomeadamente: flexibilidade na frequência de carregamento de dados no DW, transformações complexas de dados requerem espaço extra para guardar temporariamente os dados, recuperação no caso de falha, *debugging* [10]. No início do processamento ETL, se o processamento interno anterior correr bem, a zona temporária *data staging* é limpa.

A implementação do processo foi realizada através da ferramenta Pentaho Data Integration — Kettle. Ela permite a criação de *workflows*, utilizando os seus dois elementos principais: *transformation* e *job*.

As *transformations* são utilizadas para manipular os fluxos de dados. São constituídas por um ou mais passos que executam diferentes tipos de trabalhos tais como ler dados de ficheiros, filtrar linhas, transformar e carregar dados numa base de dados, entre outros. Para permitir o fluxo de dados entre os passos, eles são ligados por *transformation hops*, representando canais unidirecionais. A unidade de dados é a linha, e um fluxo de dados é o movimento de linhas de um passo para outro. Um exemplo de *transformation* e da utilização destes elementos pode ser visto da Figura 4.7.

Os *jobs* trabalham a nível superior de abstração, permitindo executar uma ou mais tarefas, entre quais também *transformations* ou outros *jobs*, chamadas de *job entries*. Para definir um caminho de execução entre as *job entries* são utilizados os *job hops*. Enquanto os passos duma *transformation* são executados todos em paralelo, as *job entries* são executadas numa ordem certa, determinada pelos *job hops* como também pelo resultado da execução destas mesmo. Um exemplo de *job* e da utilização destes elementos pode ser visto da Figura 4.5.

Nas *transformations* e *jobs* é possível executar código java, através de *items* específicos. No entanto não se pode aproveitar toda a flexibilidade da linguagem java porque só estão disponíveis as bibliotecas do projeto Janino[5]. A ideia principal

[5] http://unkrig.de/w/Janino

aqui é ter código que seja executado mais rápido possível.

Por fim, todo o processamento ETL foi configurado para ser executado duma forma automática, como referido no fim desta secção.

De forma a exemplificar o processo ETL procura-se, através de alguns exemplos de *workflows*, mostrar como todo o processo foi construído.

4.4.1 Implementação do processo ETL para a *data staging*

Nesta etapa, através da extração, limpeza, transformação, os factos e informação relevante são extraídos dos ficheiros zip (dados dos exames EMG e da meta informação associada) e dos ficheiros excel (dos dados sobre os pacientes e músculos). Depois, são colocados na zona temporária, *data staging*, representada por uma base de dados relacional MySQL. Para a implementação desta etapa foram criados oito *workflows*: três *jobs* e cinco *tranformations*.

A seguir é realizada uma descrição sucinta de um destes *workflows* - do *job* principal, ilustrado na Figura 4.5.

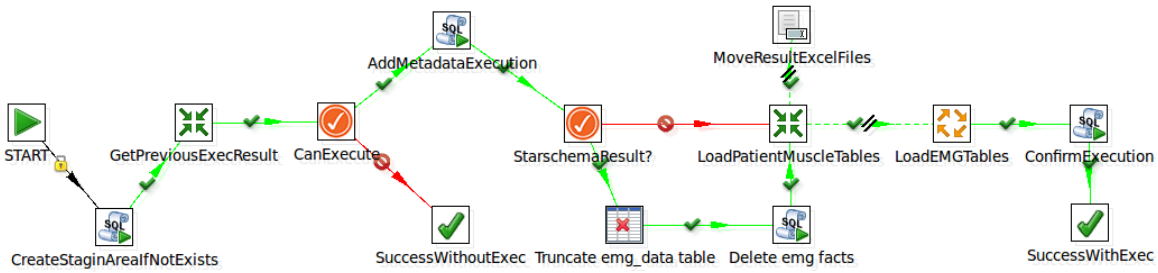


Figura 4.5: O *workflow* principal do processo ETL para a *data staging*

No *workflow* começa-se por verificar a existência da base de dados MySQL, "alsm_stagingarea", na *job entry* "CreateStagingAreaIfNotExists", e se não existir ela é criada e configurada, através do *script* SQL, cuja listagem está no anexo, Listagem A.3. Depois são obtidos os resultados do processamento ETL anterior, das duas etapas, através da *transformation* "GetPreviousExecResult". Estes resultados, com as datas de execução, estão guardados na tabela "metadata_execution", criada para esse objetivo.

Se a avaliação do resultado do processamento ETL anterior, para a *data staging*, for verdadeiro, o *workflow* continua, adicionando uma nova entrada com a data de execução e o resultado deste processamento configurado como falso, na *job entry* "AddMetadataExecution". Se o resultado anterior for falso (o processamento anterior teve erros), a execução não é realizada.

Depois é verificado o resultado do processamento ETL anterior, para o esquema estrela, na job entry "StarSchemaResult?". Se este concluiu com êxito, na zona temporária são limpas as tabelas que contém os factos, porque eles já foram carregados corretamente no DW.

O *workflow* continua com a *transformation* "LoadPatientMuscleTables", que realiza a extração e a transformação dos dados sobre os pacientes e os músculos, dos ficheiros excel, e o respetivo carregamento nas tabelas associadas, na zona temporária. Depois do processamento, estes ficheiros serão movidos para a pasta dos ficheiros processados, onde ficaram guardados.

A seguir, na *job entry* "LoadEMGTables" é realizada a operação mais complexa deste *workflow*, um *job* que é um *workflow* de *workflows* (constituído por outras *transformations*). Aqui é extraída a informação dos ficheiros zip, com os dados do sinal EMG e a meta informação associada. Primeiro é processada e guardada a meta informação. Depois os dados do sinal EMG são extraídos, limpos e transformados para obter os factos necessários. Por fim estes factos, como também os dados do sinal EMG, são guardados nas tabelas correspondentes. Aqui, para relacionar corretamente os factos, a meta informação associada e os dados do sinal EMG, houve a necessidade de processar os ficheiros com os dados do sinal EMG, um de cada vez. Este tipo de processamento foi possível só através duma combinação de *transformations* e ativando a opção de configuração das *transformations*: "*Execute for every input row*", significando que só uma linha de dados é processada cada vez (neste caso uma linha de dados representa um ficheiro). Por fim, neste *job*, os arquivos zip foram movidos para a pasta dos ficheiros processados e apagados os temporários criados durante o processamento.

O *workflow* acaba confirmando o sucesso da execução, na *job entry* "ConfirmExecution", através da atualização da linha inserida na tabela "metadata_execution", no início.

4.4.1.1 Transformação do sinal EMG

Esta transformação permite extrair, a partir do exame EMG, os dois factos principais: a *area* e o *nr_peaks*, descritos na Secção 4.3.1. Para visualizar e entender o sinal de EMG, e ver o resultado de algumas operações de processamento do sinal foi utilizado o programa R[6]. Um exemplo de sinal EMG está ilustrado na Figura 4.6.

[6] https://www.r-project.org/

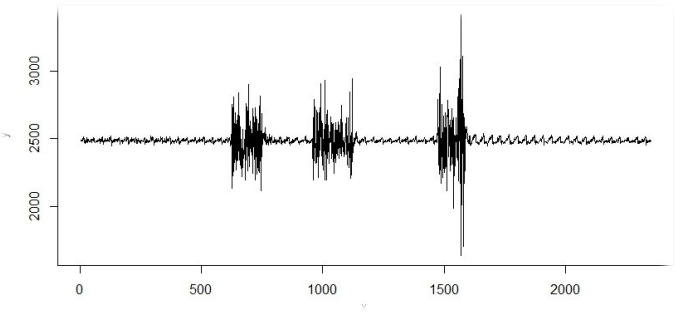


Figura 4.6: O sinal EMG, visualizado utilizando o programa R

No *workflow* da *transformation* onde são processados os ficheiros com os dados do sinal de EMG, ilustrado na Figura 4.7, no passo "*Calculate area and peaks Java class*", foi adicionado código em java que permite obter os dois factos a partir dos dados do sinal.

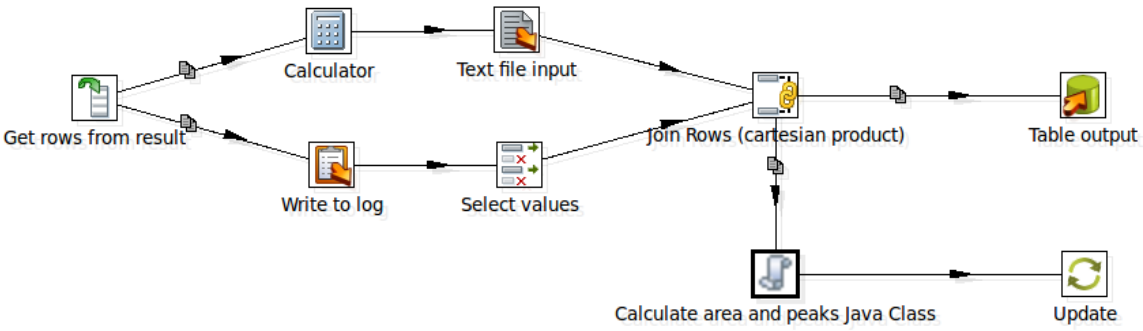


Figura 4.7: O *workflow* do processo ETL onde são extraídos os factos

Antes de extrair os factos foram realizadas algumas operações de pré-processamento de sinal, para permitir uma comparação direta entre estes:

1. *Zero-mean*, colocando a média no valor zero no eixo (y), como se vê no código entre as linhas 2 e 5 da Listagem 4.1;
2. Ordenação e normalização no eixo do tempo (x), para garantir que todos os sinais estão representadas na mesma escala de tempo, como se vê no código entre as linhas 7 e 15 da mesma listagem.

O facto *area* (*Area Under Curve* — AUC), representa a área da superfície sob a curva do sinal e, foi calculado como a soma das áreas de vários trapezóides retângulos, como se pode observar na linha 4 da Listagem 4.2.

O cálculo do facto *nr_peaks*, que representa o número de picos do sinal, é mais complexo. Para implementar o algoritmo foi criado um método separado. O método utiliza dois parâmetros, definidos em ficheiro de configuração: *threshold_noise*

(valores inferiores a este são considerados ruído) e *window*, utilizado no algoritmo para testar todos os valores que se encontram dentro dele. Os valores destes dois parâmetros foram acertados para chegar ao encontro das características do sinal. Este método encontra-se em anexo, Listagem A.5.

```
//envelope
y_envelope = new double[y_array.size()];
average_y = calculateAverage(y_array);
for (i = 0; i < y_array.size(); i++)
  y_envelope[i] = Math.abs((Double)y_array.get(i) - average_y);
//sort into idx and normalization
Collections.sort(x_array);
idx = new int[x_array.size()];
x_normalized = new double[x_array.size()];
min_x = (Double)x_array.get(0);
dif_max_min_x = (Double)x_array.get(x_array.size()-1) - min_x;
for (i = 0; i < x_array.size(); i++){
    idx[i] = tmp.indexOf(x_array.get(i));
  x_normalized[i] = ((Double)x_array.get(i)-
   min_x)/dif_max_min_x;
```

Listagem 4.1: Implementação das operações de processamento de sinal

```
//calculate area AUC
area = 0;
for (i = 0; i < (x_array.size() - 1); i++)
  area += (x_normalized[i+1] -
   x_normalized[i])*(y_envelope[idx[i+1]] +
   y_envelope[idx[i]])/2;
```

Listagem 4.2: Algoritmo para calcular o facto *area*

No método, para contar mais um pico começa-se por verificar se os valores no eixo (y) estão em cima do ruído. Se for o caso, temos o início de um pico. Depois a janela é deslizada ao longo do sinal e, para considerar que o pico acabou e eventualmente vai começar outro, todos os valores dentro da janela, no eixo (y), tem que estar em baixo do valor do parâmetro *threshold_noise*.

Este algoritmo não vai calcular corretamente o número de picos quando o valor do ruído é superior ao valor do parâmetro *threshold_noise* ou quando não é respeitada a rotina de medição do sinal (a medição não é realizada corretamente).

4.4.2 Implementação do processo ETL para o esquema estrela

Nesta etapa os dados existentes na *data staging* são carregados no esquema em estrela. Para a implementação foram criados treze *workflows*: quatro *jobs* e nove *tranformations*.

A seguir é realizada uma descrição sucinta de um dos *workflows* - do *job* principal, ilustrado na Figura 4.8.

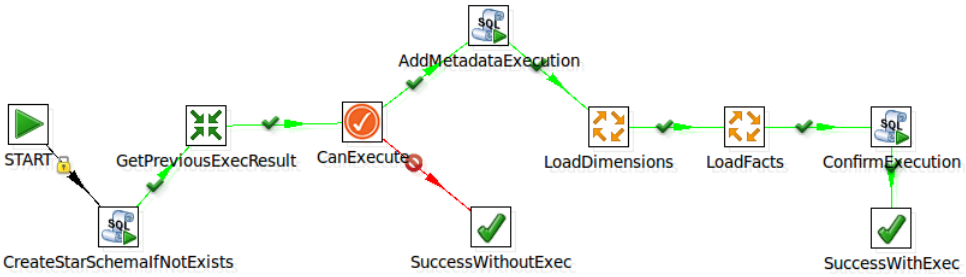


Figura 4.8: O *workflow* principal do processo ETL para o esquema estrela

No *workflow* começa-se por verificar a existência da base de dados MySQL, "alsm_starschem", na *job entry* "CreateStarScchemaIfNotExists", e se não existir ela é criada e configurada, através do *script* SQL, cuja listagem está no anexo, Listagem A.4. Depois são obtidos os resultados do processamento ETL anterior, das duas etapas, através da *transformation* "GetPreviousExecResult". Se existirem dados novos para serem carregados e o processamento ETL anterior correu corretamente, o *workflow* continua, atualizando a linha correspondente da tabela "metadata_execution" e colocando o resultado deste processamento configurado como falso, na *job entry* "AddMetadataExecution". Se não existirem dados novos ou o último processamento não correu bem, a execução não é realizada.

No próximo passo, através do *job* "LoadDimensions" é carregada a informação, em cada uma das tabelas de dimensão, em paralelo. Posteriormente, são carregados os factos nas tabelas de factos, através do *job* "LoadFacts". Como foi referido, cada uma das tabelas de factos está relacionada com um conjunto de tabelas de dimensão através das chaves estrangeiras. Por este motivo, o carregamento é realizado nessa ordem, sendo as tabelas de factos sempre as últimas a serem carregadas.

A dimensão dim_time é uma dimensão estática, toda a informação necessária na tabela desta dimensão, sobre o tempo, sendo gerada e carregada só uma vez.

A dimensão dim_date é uma dimensão semi-estática. No início é gerada e carregada a informação das datas para um período de dez anos. Depois, nos próximos

processamentos, é verificada a diferença entre a data corrente e a última data disponível na dimensão. Se a diferença for inferior a um ano é gerada e carregada automaticamente a informação das datas para mais dez anos.

Os valores dos atributos destas duas dimensões são criados de forma autónoma, através de geração de fluxo de dados, produto cartesiano, código java, e utilizando *worflows* criados para o efeito.

Nas tabelas de dimensão dim_patient e dim_muscle, a estratégia da alteração dos valores dos atributos é de tipo *slowly changing dimensions type 1*, sendo realizada a atualização destes valores, quando for necessário.

Nas tabelas de factos para fazer a ligação com as tabelas das dimensões, têm de ser obtidas as chaves estrangeiras. É usado o item "*Database value lookup*" para obter as chaves, como ilustrado na Figura 4.9.

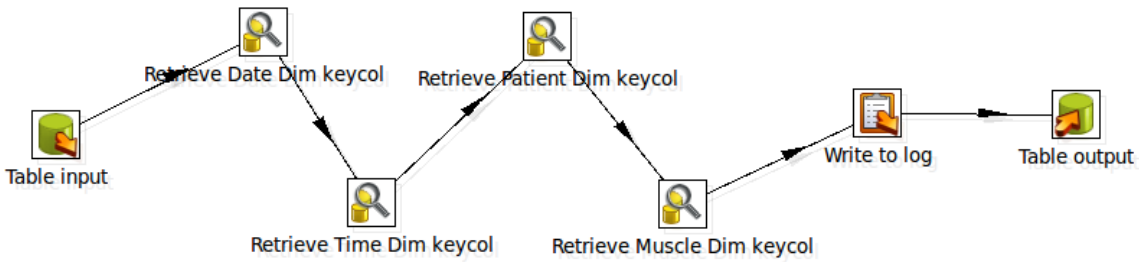


Figura 4.9: O *workflow* do processo ETL para o carregamento dos factos no DW

O *workflow* acaba confirmando o sucesso da execução, através da atualização da linha utilizada no início, na tabela "metadata_execution".

4.4.2.1 Internacionalização

Neste trabalho teve-se a preocupação de encontrar uma melhor maneira de ter a informação no DW traduzida em varias línguas. A solução encontrada foi utilizando ficheiros de configuração, contendo a informação traduzida, e os *resource bundles* java que vão buscar e disponibilizar esta informação. É uma solução transparente, fácil de implementar e com um bom desempenho, aproveitando-se as características dos *resource bundles* java. Para efeitos de demonstração apenas uma parte da informação foi traduzida. No entanto, seguindo o mesmo procedimento pode ser efetuada a tradução completa.

Para mostrar como foi implementado vamos ver um exemplo concreto, as traduções dos períodos do dia, em português e inglês.

No primeiro passo foram criados dois ficheiros de configuração correspondentes às duas línguas. No *workflow* da *transformation* onde são gerados e carregados os valores dos atributos da dimensão dim_time, ilustrado na Figura 4.10, no passo "Create time dim fields", foi adicionado o código da Listagem 4.3.

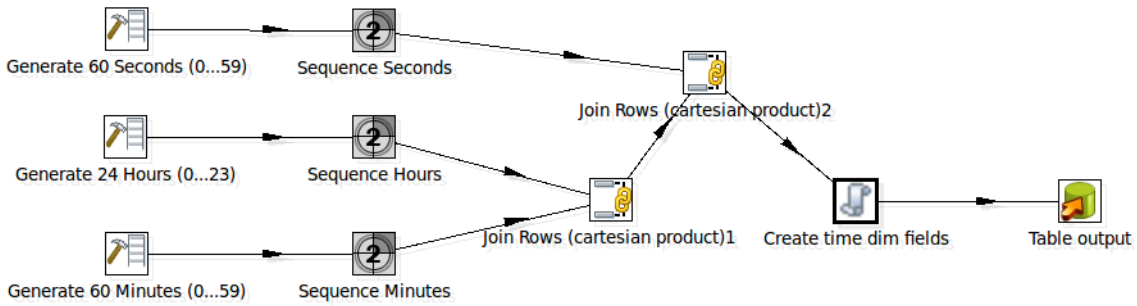


Figura 4.10: O *workflow* do processo ETL para a dimensão dim_time

```
//..
Locale locale_en = new Locale(
  getVariable("language_code_en"),
  getVariable("country_code_en"));
Locale locale_pt = new Locale(
  getVariable("language_code_pt"),
  getVariable("country_code_pt") );
//..
  URL[] urls = {file.toURI().toURL()};
  ClassLoader loader = new URLClassLoader(urls);
  rb_en = ResourceBundle.getBundle("translations", locale_en,
   loader);
  rb_pt = ResourceBundle.getBundle("translations", locale_pt,
   loader);}
//...
if ((hour24 >=0  & hour24 < 7) || (hour24 >= 21 & hour24 <=
   23)) {
    period_en = rb_en.getString("night");
    period_pt = rb_pt.getString("night");}
//..
```

Listagem 4.3: Utilização de *bundles* java para a tradução do conteúdo

Na Listagem 4.3, nas linhas 11 e 12 foram criados e configurados os *resource bundles* para as duas línguas, utilizando os objetos *Locale* correspondentes, criados

nas linhas 2 e 5. Depois pode-se obter a tradução dos *strings* existentes nos ficheiros de configuração, a partir do *resource bundle* da língua correspondente, utilizando o método *getString* e a chave do respetivo *string*, como se pode ver nas linhas 15 e 16.

A implementação do processo ETL terminou com a criação de dois *jobs* para executar os *jobs* principais das duas etapas e que permitem, em caso de erro ou falha, em qualquer ponto do processamento ETL, enviar um email ao responsável com o log detalhado sobre o problema em causa.

Para a automatização do processamento ETL foram criados dois *scripts*, que utilizam a ferramenta Pentaho Kitchen para executar, através de comandos, os *jobs* acima criados, como se pode ler na linha 8 da Listagem 4.4.

A execução destes *scripts* foi configurada para ser realizada de forma automática, através da criação de *schedules*, no servidor onde se encontra o DW. A periodicidade escolhida foi diária, durante a noite. Esta periodicidade pode ser sempre alterada, consoante as necessidades do processamento.

```
1 #!/bin/sh
BASE_DIR=/alsmon/dw/kettle/
3 STAGINGAREA_JOB=/alsmon/dw/workspace_kettle/stagingarea/jobs/
execstagingarea.kjb
5 STAGINGAREA_LOGFILE=/alsmon/dw/workspace_kettle/logs/
stagingarea.log
7 cd $BASE_DIR
./kitchen.sh -file=$STAGINGAREA_JOB -Level=Basic >>
   $STAGINGAREA_LOGFILE
```

Listagem 4.4: *Script* que executa o processamento ETL para a *data staging*

4.5 Servidor OLAP

Foi desenhado e implementado um servidor OLAP, do tipo ROLAP, para permitir um processamento analítico consistente e rápido. Nele foi disponibilizado um cubo, que assenta sobre o motor relacional MySQL. A "ligação" entre o cubo e o DW é feita através da meta-informação guardada no servidor OLAP.

4.5.1 Implementação

Para criar o esquema do cubo multidimensional, foi utilizada a ferramenta Pentaho Schema Workbench (PSW), como ilustrado na Figura 4.11.

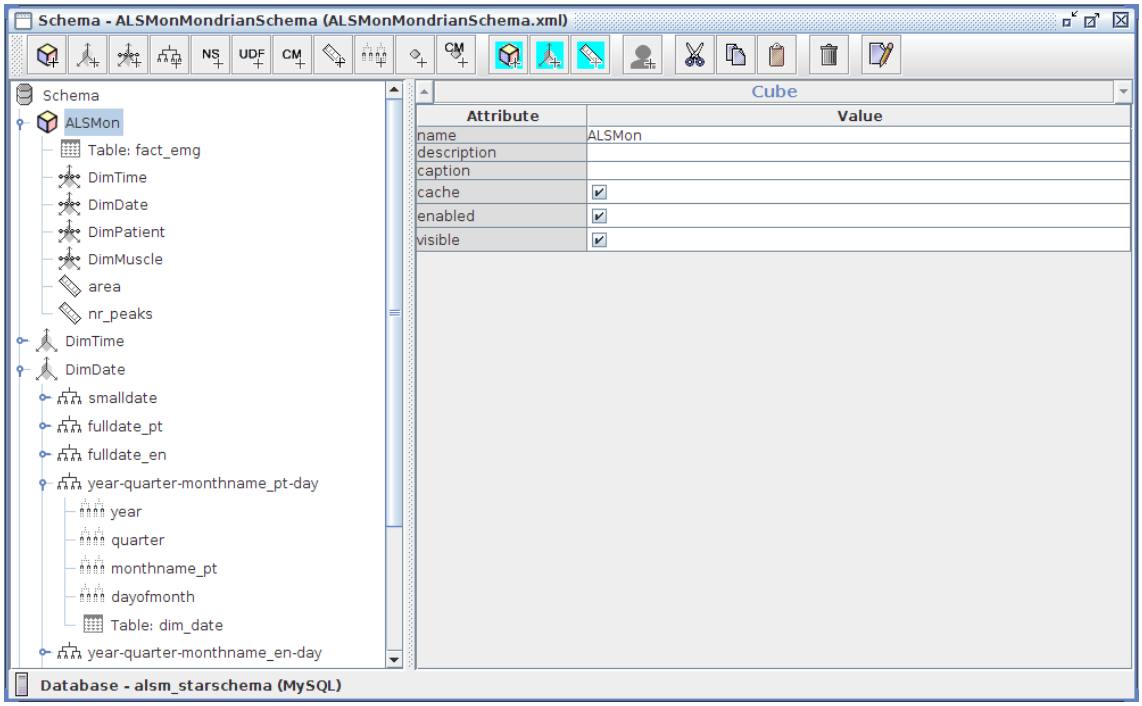


Figura 4.11: A criação do cubo multidimensional utilizando PSW

Assim foi definida a base de dados relacional, criado o cubo e os elementos necessários à definição do cubo: métricas, dimensões e hierarquias.

Como **métricas** foram usados os factos da tabela de factos, fact_emg. Para cada uma delas foi definida a função de agregação utilizada. Por exemplo, no casa da métrica "*area*", foi escolhida a função de agregação avg(). Além destas métricas podem ser definidas métricas calculadas. De notar que cada cubo só permite definir métricas a partir duma única tabela de factos.

A definição das **dimensões** pode ser realizada de duas maneiras: dentro da definição do cubo ou duma forma geral (fora do esquema do cubo), podendo depois ser utilizada por qualquer outro cubo criado. Neste caso foram criadas quatro dimensões consoante as quatro tabelas de dimensão existentes no modelo multidimensional: DimPatient, DimMuscle, DimDate e DimTime. Estas dimensões foram criadas com possibilidade de ser partilhadas por outros cubos que futuramente poderão vir a ser necessários.

Em cada uma das dimensões foram definidos os atributos que vão ser visualizados (excluindo por exemplo as chaves) e foram criadas todas as hierarquias possíveis.

As **hierarquias** criadas permitem o acesso à informação com maior ou menor detalhe. Por exemplo a hierarquia {*year-quarter-monthname_pt-dayofmonth*}, criada na dimensão DimDate, permite analisar as métricas por dia, e através da operação *drill-up*, obter a análise a um nível superior e saber as métricas por mês, trimestre e por ano. Quando são realizadas consultas em níveis hierárquicos menos detalhados, a informação apresentada é agregada.

No fim a estrutura do cubo foi guardada em formato xml.

Neste trabalho como temos só um DMart, representando o exame EMG, foi criado um único cubo. Posteriormente, quando houver mais exames (mais DMarts), vai haver necessidade de adicionar outros cubos.

4.5.2 *Queries* MDX

O DW pode ser interrogado, através do cubo multidimensional, utilizando a linguagem MDX.

A seguir são indicados exemplos de consultas MDX relevantes para obter informação do DW sobre um paciente específico (nestes exemplos o paciente com o id '2015–443').

1. A consulta MDX para obter a média do valor do tónus muscular para o músculo Anterior Tibialis, por mês do ano 2015, para um paciente como na Listagem 4.5.

2. A consulta MDX para obter a média do valor do tónus muscular, por músculo, por ano, e para um paciente ilustrado na Listagem 4.6.

Nas duas consultas a função de agregação para a métrica "*area*" é aquela que foi definida no cubo: avg() — média.

```
select
    {[DimDate.year-monthname_pt-day].[2015].Children} on columns,
    {[DimPatient.patient_id].[2015-443]} on rows
from ALSMon
where [DimMuscle.name].[Anterior Tibialis]
```

Listagem 4.5: Primeiro exemplo de consulta MDX

```
select
    {CrossJoin([DimMuscle.name].Children,
    [DimDate.year-quarter-monthname_pt-day].Children)} on
    columns,
    {[DimPatient.patient_id].[2015-443]} on rows
from ALSMon
where [Measures].[area]
```

Listagem 4.6: Segundo exemplo de consulta MDX

4.5.3 Otimização utilizando agregações

A otimização das consultas foi realizada através de criação de pré-agregações. Utilizando a ferramenta Pentaho Aggregation Designer (PAD), como ilustrado na Figura 4.12, foram definidas as ligações à bases de dados relacional e ao cubo multidimensional e criados dois tipos de agregações, para otimizar as duas consultas MDX apresentadas no parágrafo anterior.

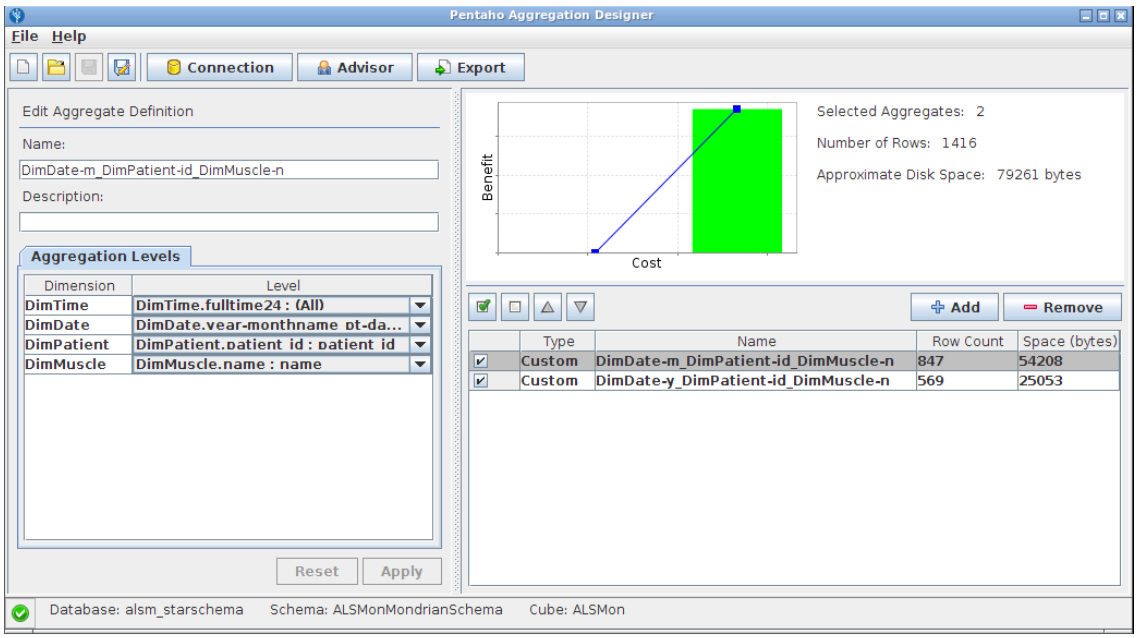


Figura 4.12: A criação de agregações utilizando PAD

Na Figura 4.12 pode ser observado que o benefício aumentou mas também o custo, sendo necessário mais espaço (79261 bytes) para guardar os novos 1416 registos que vão ser gerados com os valores agregados.

Por exemplo, na agregação para otimizar a primeira consulta MDX, foram configurados os seguintes níveis de agregação, para cada dimensão:

- na dimensão DimTime não foi configurado nenhum nível: *DimTime.fulltime24:All*;
- na dimensão DimDate o nível de agregação por mês: *DimDate.year-quarter-monthname_pt-dayofmonth:monthname_pt*;
- na dimensão DimPatient o nível de agregação por patient_id: *DimPatient.patient_id:patient_id*;
- na dimensão DimMuscle o nível de agregação por nome: *DimMuscle.name:name*.

A ferramenta PAD permite depois guardar a configuração das agregações na definição do cubo, para utiliza-las futuramente. Além disso, permite executar ou exportar os *scripts* sql necessários a criação e preenchimento das tabelas com os dados agregados, na base de dados relacional.

Mais tarde podem ser adicionadas outras agregações para otimizar outras consultas consideradas relevantes. No entanto, a criação de pré-agregações tem que ser feita utilizando heurísticas, considerando o custo vs. o benefício. Por exemplo, se foi criada uma agregação por mês talvez não vale a pena criar uma por trimestre porque neste caso vão ser agregados só 3 elementos, aproveitando-se as agregações disponíveis por mês, o custo neste caso sendo maior do que o benefício.

5

Módulo de análise

A informação disponível no DW só é relevante se estiver acessível aos utilizadores. Nesse sentido, decidiu-se desenvolver API de acesso a dados que pode servir como base ao desenvolvimento de aplicações de análise.

5.1 Solução proposta

A solução consiste em duas APIs, OData e XMLA que permitem o acesso à informação do DW e que serão usadas por aplicações de análise. São serviços *web* que funcionam como camadas de acesso a dados (DAL) e que recebem pedidos das aplicações de análise, executam interrogações sobre o DW e depois devolvem o resultado a estas aplicações de análise.

Esta solução foi implementada com o objetivo de ter acesso aos dados do DW duma forma padronizada, utilizando ferramentas *open source* e *standards*.

As duas APIs foram desenvolvidas em java, no IDE Eclipse, em projetos de tipo Apache Maven, como *web application* e utilizando, além das tecnologias descritas nos próximos parágrafos, a *framework* Apache CXF, que dá suporte a criação e consumo de serviços *web*, através dos modelos de programação *frontend* JAX-RS e JAX-WS. O *deployment* das APIs foi realizado no servidor *web* aplicacional, Apache Tomcat.

Todas estas ferramentas (Apache CXF, Apache Maven e Apache Tomcat) foram

descritas no Capítulo 3, onde foram também utilizadas para a criação do serviço e da aplicação *web*, necessárias a receção dos dados.

Como os dados disponibilizados para consulta são confidenciais, um dos objetivos principais foi garantir a segurança destes. A segurança dos dados, quer no acesso, quer na transmissão, foi garantida através de protocolos e cifras modernas como descrito na Secção 5.4, onde é descrita a implementação da segurança.

Segue-se a descrição da solução para cada uma das APIs implementadas.

5.2 API OData

A API OData é um serviço REST que utiliza o protocolo OData[1]. Este serviço permite interrogar e obter os dados diretamente da base de dados MySQL.

Como é um serviço REST, as interrogações para a consulta dos dados são realizadas através de pedidos HTTP *GET* que respeitam a convenção OData URI[2], ficando ao cargo do serviço transformar depois estes pedidos em interrogações SQL sobre a respetiva base de dados. Os dados são devolvidos num formato padrão, como xml ou json.

A acesso aos dados, através do serviço, foi configurado apenas para leitura, não sendo possível a inserção ou a alteração dos dados.

Segue uma breve descrição das ferramentas e tecnologias utilizadas.

A ***Java Persistence API***[3] (JPA) é uma API que padroniza a persistência dos dados, especificando uma interface comum para as *frameworks* de persistência dos dados. Ela define a forma de mapeamento objeto-relacional para objetos java simples e comuns (POJOs), denominados *beans* de entidade.

Diversas *frameworks* de mapeamento objeto-relacional implementam a JPA, tendo sido utilizada a EclipseLink[4]. É uma API *open source*, desenvolvida pela Eclipse Foundation e por isso disponível no IDE Eclipse.

Representational State Transfer (REST) é um estilo de arquitetura para desenvolver aplicações em redes. A ideia é que, em vez de utilizar mecanismos complexos, tais como CORBA, RPC ou SOAP para se conectar entre as máquinas, é utilizado

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

[2] http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html

[3] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[4] http://www.eclipse.org/eclipselink/

HTTP simples para fazer pedidos entre máquinas. Assim, representa uma alternativa mais leve a estes mecanismos.

Entre as regras de *design* aplicadas ao estilo de arquitetura REST enumeramos: modelo cliente-servidor, *stateless*, *cacheable*, interface uniforme, *layered system*, *code-on-demand* [27].

Em muitos aspetos, a *World Wide Web* em si, com base em HTTP, pode ser vista como uma arquitetura baseada em REST. As aplicações RESTful utilizam chamadas HTTP para enviar dados (criar e/ou atualizar), ler dados (por exemplo, fazer consultas) e apagar dados. Assim, REST usa HTTP para todas as quatro operações CRUD (*Create / Read / Update / Delete*).

REST não é um "padrão". Embora haja estruturas de programação em REST, trabalhando com REST é tão simples que muitas vezes podem ser construídas estruturas próprias com os recursos às bibliotecas padrão em linguagens como Perl, Java ou C#.

Um serviço REST é independente de plataforma, da linguagem de programação. Como serviços *web*, REST não oferece recursos internos de segurança, criptografia, gestão de sessão, garantias de QoS, etc. Estes só podem ser adicionados através da construção em cima de HTTP [5].

Open Data Protocol (OData) é uma iniciativa *open source* da Microsoft e outras empresas (OASIS) para criar um padrão que simplifica a manipulação e partilha de dados entre aplicações. Sendo um protocolo baseado em REST, OData utiliza HTTP, AtomPub e JSON, através de URIs, para aceder aos respetivos recursos e realizar operações de consulta, inserção, atualização ou apagar os dados. Ele permite aceder informação a partir de uma variedade de fontes, como bases de dados relacionais, sistemas de ficheiros, sistemas de gestão de conteúdo. Além disso, é possível efetuar uma filtragem sobre um recurso, diretamente na fonte.

Os serviços OData são definidos usando um modelo de dados comum. O serviço expõe o seu modelo de dados concretos de uma forma legível, permitindo que clientes genéricos interagem com o serviço duma forma bem definida, como se vê na Figura 5.1:

Um serviço OData expõe dois recursos que descrevem o seu modelo de dados:

- **documento de serviço**: lista os conjuntos de entidades, funções e *singletons* que podem ser utilizados. Os clientes podem utilizar o documento de serviço para navegar no modelo duma forma *hypermedia-driven*;

- **documento de metadados**: descreve os tipos de metadados, conjuntos, funções e ações disponibilizados pelo serviço OData. Os clientes podem utilizar o documento de metadados para entender como consultar e interagir com as entidades do serviço. [8]

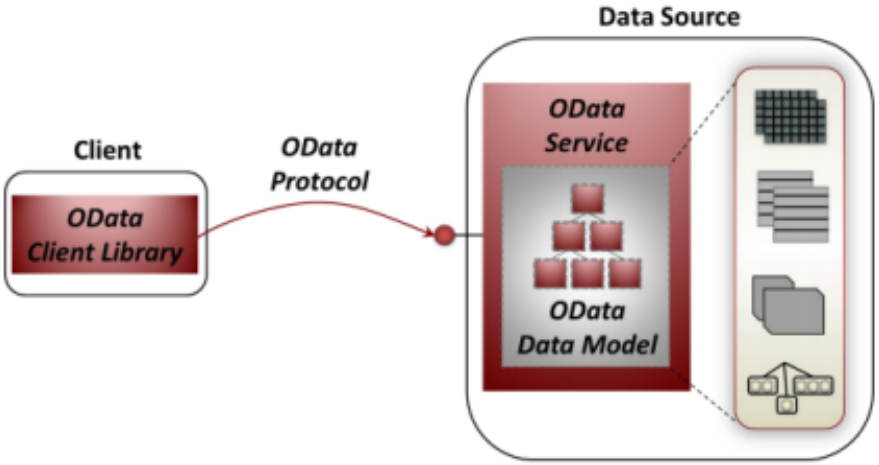


Figura 5.1: Tecnologia OData [18]

Como o OData é um protocolo web baseado em padrões amplamente difundidos, fica fácil fazer a integração de aplicações em várias plataformas e sistemas operativos.

Apache Olingo[5] formou-se a partir do Apache Incubator para se tornar um projeto *top-level* da Apache. É um projeto *open source* que disponibiliza as bibliotecas java, cliente e servidor, utilizadas na implementação do protocolo OData.

Apache Olingo suporta múltiplas idiomas, incluindo Java e JavaScript, para clientes e servidores OData. As extensões do Olingo possuem características adicionais, tais como o suporte a Java Persistence API (JPA) e classes anotadas. O projeto vem com uma documentação completa, que inclui exemplos de implementação de um serviço OData personalizado, construído com Apache Maven e que pode ser instalado em qualquer servidor de aplicações *web*, como o Apache Tomcat.

Para a criação da API OData, foram utilizadas as bibliotecas java do servidor Apache Olingo.

[5] https://olingo.apache.org/

5.2.1 Implementação do DAL para a API OData

Num novo projeto Maven foi configurada uma *database connection*, para criar o acesso a base de dados MySQL, utilizando o driver MySQL JDBC Driver.

Foi adicionado ao projeto Maven a JPA Facet (JPA Persistence), foi escolhida a *framework* EclipseLink e a ligação à base de dados (definida anteriormente). Esta operação criou automaticamente o ficheiro META-INF/*persistence.xml* onde ficam guardadas todas estas configurações sobre a persistência nomeadamente:

- a framework utilizada,
- a configuração da ligação à base de dados,
- as classes que realizam o mapeamento para o modelo relacional (descritas mais a frente).

A listagem do conteúdo deste ficheiro está em anexo, Listagem A.6.

Para realizar o mapeamento para o modelo relacional foram criadas as entidades JPA necessárias. Estas não são nada mais do que classes POJO marcadas como entidades, utilizando a anotação *@Entity*, como mostrado na Listagem 5.1.

```
//...
@Entity
public class DimPatient implements Serializable {
  private int keycol;
  private String name;
  private Date birthdate;
//...
```

Listagem 5.1: Exemplo de uma entidade JPA

Neste exemplo, a entidade ***DimPatient*** precisa saber como mapear os atributos para a tabela ***dim_patient***. Isso é realizado por meio de anotações JPA, como apresentado na Listagem 5.2. Para mapear as relações entre as tabelas, foram adicionados os atributos necessários e utilizadas anotações específicas. Esses atributos permitem posteriormente obter numa interrogação OData várias tabelas.

Para definir, por exemplo, o mapeamento bidirecional JPA "*one to many*" e "*many to one*", entre as tabelas ***dim_patient*** e ***fact_emg***, foram utilizadas as anotações *@OneToMany*, *@ManyToOne*, *@JoinColumn* e criados os atributos mostrados na Listagem 5.3, factEmgs e dimPatient.

Em todas as classes implementadas, os "*setters*" não foram configurados, permitindo assim só o acesso à leitura, através dos "*getters*".

```
//...
@Entity
@Table(name="dim_patient")
@NamedQuery(name="DimPatient.findAll", query="SELECT d FROM
    DimPatient d")
public class DimPatient implements Serializable {
  @Id
  @Column(name="keycol", unique=true, nullable=false)
  private int keycol;

  @Column(name="name", nullable=false, length=50)
  private String name;

  @Temporal(TemporalType.DATE)
  @Column(name="birthdate", nullable=false)
  private Date birthdate;
//...
```

Listagem 5.2: Exemplo de entidade JPA com as anotações necessárias

```
//...
//bi-directional one-to-many association to FactEmg
  @OneToMany(mappedBy="dimPatient")
  private List<FactEmg> factEmgs;
//...
//bi-directional many-to-one association to DimPatient
  @ManyToOne
  @JoinColumn(name="patient_keycol",
   referencedColumnName="keycol", nullable=false,
   insertable=false, updatable=false)
  private DimPatient dimPatient;
//...
```

Listagem 5.3: Exemplo do mapeamento JPA para as relações entre tabelas

5.2.2 Disponibilização em formato de serviço OData

O servidor Apache Olingo foi utilizado para implementar uma *OData Service Factory*. Assim, foi adicionada ao projeto uma nova classe java, estendendo a classe ODataJPAServiceFactory.

Foi configurada a ligação ao mapeamento JPA através da variável PUNIT_NAME, que se refere à configuração existente no ficheiro *persistence.xml* criado anteriormente. Para obter o ODataJPAContext foi feita a redefinição do método initializeODataJPAContext, e utilizada a variável PUNIT_NAME, como está apresentado na Listagem 5.4.

```
//...
public class ODataServiceFactory extends ODataJPAServiceFactory
    {
  private static final String PUNIT_NAME = "ODataDAL";
  @Override
  public ODataJPAContext initializeODataJPAContext()
  throws ODataJPARuntimeException {
    ODataJPAContext oDataJPAContext = this.getODataJPAContext();
    try {
      EntityManagerFactory emf = Persistence
          .createEntityManagerFactory(PUNIT_NAME);
      oDataJPAContext.setEntityManagerFactory(emf);
      oDataJPAContext.setPersistenceUnitName(PUNIT_NAME);
      return oDataJPAContext;
//...
```

Listagem 5.4: A classe java que estende ODataJPAServiceFactory

Os pedidos são recebidos pela biblioteca REST do Apache Olingo[6] utilizando a *framework* Apache CXF, através da *frontend* JAX-RS. Depois são passados à classe ODataServiceFactory, criada anteriormente, que faz a ligação ao mapeamento JPA. Toda esta configuração é realizada através do ficheiro *web.xml*, como ilustrado na Listagem 5.5.

Por fim o projeto Maven criado, foi exportado em formato "war" e instalado no Tomcat.

O endereço geral (página principal) da API OData ficou no URL:

https://alsmon.adeetc.e.ipl.pt:8443/ODataDAL/

O URL base para a criação das *queries* é:

https://alsmon.adeetc.e.ipl.pt:8443/ODataDAL/DW.svc/

O endereço dos metadados da API ficou no URL:

https://alsmon.adeetc.e.ipl.pt:8443/ODataDAL/DW.svc/$metadata

[6]org.apache.olingo.odata2.core.rest.app.ODataApplication

```
<servlet>
  <servlet-name>ODataDALDWServlet</servlet-name>
  <servlet-class>org.apache.cxf.jaxrs.servlet.
  CXFNonSpringJaxrsServlet</servlet-class>
  <init-param>
    <param-name>javax.ws.rs.Application</param-name>
    <param-value>org.apache.olingo.odata2.core.rest.app.
    ODataApplication</param-value>
  </init-param>
  <init-param>
    <param-name>org.apache.olingo.odata2.service.factory
    </param-name>
    <param-value>pt.isel.alsmon.odata.ODataServiceFactory
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Listagem 5.5: Configuração da API OData

5.2.3 Exemplo de cliente OData

Como a API OData é uma API REST disponível só para leitura, pode ser utilizado qualquer *browser* para testá-la, usando pedidos HTTP GET. O resultado é devolvido em formatos padrão como xml ou json. Como é necessário utilizar HTTPs e autenticação de tipo *basic* do lado do cliente, para comunicar com a API é preciso aceitar e instalar o certificado no *browser* e indicar o nome de utilizador e a palavra-chave.

Para testes mais exaustivos, existem vários clientes REST, *open source*, que possibilitam vários tipos de pedidos HTTP. Esses clientes permitem também configurar vários tipos de autenticação entre outras funcionalidades como, por exemplo, configurar os parâmetros ou ver o tempo de resposta do serviço OData. Nos testes feitos foi utilizada a ferramenta Postman do Google Chrome.

Para saber como utilizar e interagir com o serviço OData pode ser consultado o endereço que disponibiliza a informação sobre os metadados. Nele são descritos os tipos de metadados, conjuntos, funções e ações disponibilizados pelo serviço, e a convenção OData URI, para saber como criar as interrogações necessárias.

Na Figura 5.2 é mostrado como é pedida a informação sobre os pacientes, existente na tabela ***dim_patient***, utilizando a API OData, através da interrogação HTTP GET.

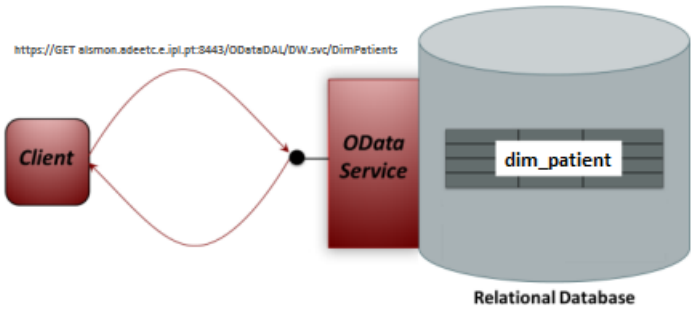


Figura 5.2: Cliente do serviço OData [18]

Outro exemplo, ilustrado na Listagem 5.6, mostra uma interrogação para obter todos os factos das EMGs do paciente com o "PatientKeycol=1", com informação completa (data, tempo, músculo). O retorno vem em formato json.

```
1 https://alsmon.adeetc.e.ipl.pt:8443/ODataDAL/DW.svc/FactEmgs?
  $format=json&
3 $filter=PatientKeycol eq 1&
  $expand=DimDateDetails,DimTimeDetails,DimMuscleDetails
```

Listagem 5.6: Exemplo de interrogação para o serviço OData

Para utilizar o serviço OData a partir duma aplicação cliente, basta fazer uso da biblioteca OData cliente, correspondente a linguagem de programação utilizada.

5.3 API XMLA

A API XMLA é uma API que possibilita a consulta dos dados do DW, através do protocolo XMLA e suportado pelo projeto Mondrian. A API permite interrogar e obter os dados do DW, diretamente da base de dados relacional ou através do cubo OLAP. A vantagem desta quando comparada com a API OData, é a possibilidade de utilizar todas as funcionalidades do cubo (dimensões, métricas, hierarquias), tirando partido das pré-agregações criadas. No entanto ao contrário do serviço OData, onde os dados são obtidos através de pedidos HTTP simples, aqui a consulta dos dados é realizada através de interrogações MDX, transformadas em mensagens SOAP, utilizando um cliente OLAP XMLA. Os dados são devolvidos depois em formato padrão, SOAP formatado em xml-analysis.

O acesso aos dados, através deste serviço, como no caso da API OData, é só de leitura.

Segue uma breve descrição das ferramentas e tecnologias utilizadas.

Mondrian[7] é uma projeto *open source, business analytics*, da Pentaho, escrito em java, que possibilita o acesso aos dados do DW, através de um cubo OLAP.

A arquitetura do sistema Mondrian consiste em quatro camadas: a camada de apresentação, a camada dimensional, a camada de estrela e a camada de armazenamento, como mostrado na Figura 5.3.

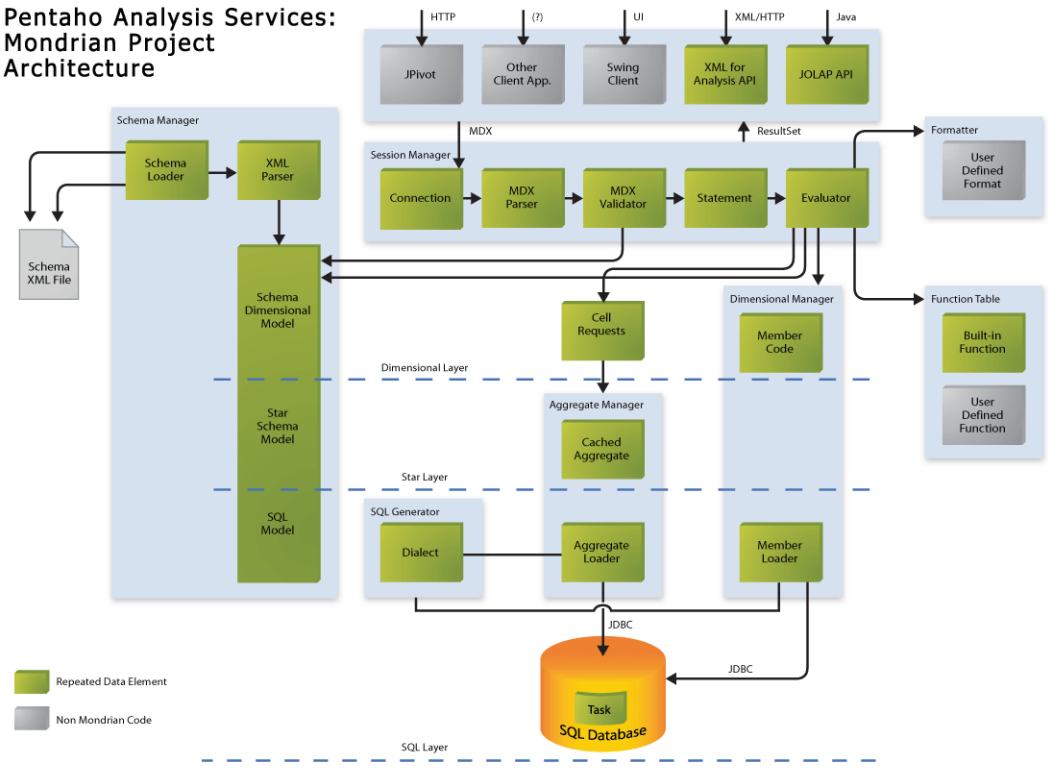


Figura 5.3: A arquitetura do sistema Mondrian [9]

Utilizando o Mondrian como motor OLAP, podem ser construidas soluções de BI que permitem consultas multi-dimensionais, utilizando a linguagem MDX.

O projeto Mondrian contém as bibliotecas necessárias para disponibilizar a informação do DW através do protocolo XMLA. As vantagens do XMLA só descritas a seguir.

XML for Analysis (XMLA) é um protocolo *standard* SOAP baseado em XML, projetado especificamente para acesso a qualquer fonte de dados multidimensional,

[7] http://community.pentaho.com/projects/mondrian/

através duma ligação HTTP. Este protocolo foi criado para uniformizar a comunicação entre aplicações cliente e fornecedores de dados multi-dimensionais ou servidores OLAP.

Assim, o XMLA permite às aplicações analíticas clientes deslocarem-se de uma arquitetura cliente-servidor tradicional para um ambiente mais flexível, evitando acoplamento e resolvendo assim os problemas de dependência cliente-servidor. XMLA está otimizado para a Internet, minimizando o número de pedidos do cliente ao servidor [11].

API Olap4j[8] é uma API OLAP, *open source*, escrita em java. Tem as bibliotecas necessárias para aceder a vários motores Olap (por exemplo, Mondrian, Analysis Services, Palo).

Foi utilizada na implementação da aplicação cliente para aceder ao serviço XMLA, através do módulo olap4j-xmla.

5.3.1 Implementação da API XMLA

Para implementar a API XMLA foi copiado o ficheiro xml que contém o esquema do cubo multidimensional (chamado de catálogo nas configurações da API) num novo projeto Maven, na pasta /WEB-INF/schema/, e criado o ficheiro de configuração WEB-INF/*datasources.xml* que contém, entre outras configurações, a informação sobre a conexão ao esquema em estrela e a localização do catálogo. O conteúdo deste ficheiro está em anexo, Listagem A.7.

Para utilizar a configuração realizada foi criada uma nova classe java, estendendo a classe MondrianXmlaServlet, que implementa o serviço XMLA e é disponibilizada pelo projeto Mondrian. Foi redefinido o método makeDataSourcesUrl para ler a configuração do ficheiro *datasources.xml*, como é ilustrado na Listagem 5.7, na linha 6.

Na configuração do *servlet* XMLA, no ficheiro *web.xml*, foi usada a classe criada, como está na Listagem 5.8, na linha 3.

Foram aproveitadas as classes existentes no projeto Mondrian, que possibilitam testar online o serviço XMLA e ver as mensagens SOAP trocadas. Além disso, também permite executar interrogações MDX. Aqui foram feitas as alterações necessárias no código para utilizar a conexão ao esquema em estrela e a localização do catálogo, configurados anteriormente. Estas classes, que disponibilizam as

[8] http://http://www.olap4j.org//

funcionalidades extra enumeradas em cima, foram colocadas num *package* separado e podem ser retiradas mais tarde, sem afetar o serviço XMLA disponibilizado.

```
public class XmlaServlet extends MondrianXmlaServlet {
  private static final long serialVersionUID = 1L;
  public static final String DEFAULT_DATASOURCE_FILE =
   "datasources.xml";
    protected MondrianServer server;
@Override
    protected String makeDataSourcesUrl(ServletConfig
   servletConfig) {
        String paramValue =
   servletConfig.getInitParameter(PARAM_DATASOURCES_CONFIG);
//...
        URL dataSourcesConfigUrl = null;
        try {
            if (paramValue == null) {
                // fallback to default
                String defaultDS = "/WEB-INF/" +
   DEFAULT_DATASOURCE_FILE;
//...
```

Listagem 5.7: A classe java que estende MondrianXmlaServlet

```
<servlet>
   <servlet-name>MondrianXmlaServlet</servlet-name>
   <servlet-class>pt.isel.alsmon.olap.XmlaServlet
   </servlet-class>
   <init-param>
     <param-name>CharacterEncoding</param-name>
     <param-value>UTF-8</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
   <servlet-name>MondrianXmlaServlet</servlet-name>
   <url-pattern>/xmla</url-pattern>
 </servlet-mapping>
```

Listagem 5.8: Configuração do *servlet* XMLA

Por fim o projeto Maven criado, foi exportado em formato "war" e instalado no Tomcat.

O endereço geral (página principal) da API XMLA ficou no URL:

`https://alsmon.adeetc.e.ipl.pt:8443/OlapXMLA/`

O endereço do serviço XMLA ficou no URL:

`https://alsmon.adeetc.e.ipl.pt:8443/OlapXMLA/xmla`

5.3.2 Cliente XMLA

Os testes à API XMLA podem ser realizados *online*, através do endereço disponibilizado. Funcionam como um cliente XMLA, permitindo realizar uma série de testes como, por exemplo, executar interrogações MDX, ver mensagens SOAP que utilizam o xml-*analysis*. Como no caso da API OData, sendo necessário utilizar o HTTPS e autenticação de tipo *basic* do lado do cliente, para comunicar com a API, sempre tem que se aceitar e instalar o certificado no *browser* e colocar o nome do utilizador e a palavra-chave.

No entanto para utilizar o serviço XMLA a partir duma aplicação cliente, basta fazer uso das bibliotecas Olap XMLA cliente, correspondente à linguagem de programação utilizada: Perl, Java ou C#.

Assim foi criada uma pequena aplicação cliente, em java, um projeto Maven que utiliza as bibliotecas da API Olap4j. Para ligar-se ao serviço XMLA, a aplicação cliente utiliza o driver XMLA Olap4j e o endereço do serviço XMLA, especificando o catálogo utilizado, como se vê na Listagem 5.9.

Depois, através das bibliotecas olap4j, foi criado un novo OlapStatement, utilizando a olapConnection criada anteriormente, e executada uma interrogação MDX.

O utilização do certificado e a autenticação segue o mesmo esquema descrito anteriormente. Por fim a aplicação cliente foi exportada em formato "jar".

5.4 Segurança dos dados

Para as duas APIs, OData e XMLA, sendo aplicações *web*, a segurança dos dados foi implementada através de protocolo HTTPS e da autenticação de tipo "basic" do lado dos clientes (nome de utilizador, palavra-chave).

```
//...
public static void main( String[] args )
throws ClassNotFoundException, SQLException
  {
    // Register driver.
    Class.forName("org.olap4j.driver.xmla.XmlaOlap4jDriver");
    //Create XMLA connection
    Connection connection = DriverManager.getConnection(
          "jdbc:xmla:Server=
          https://alsmon.adeetc.e.ipl.pt:8443/OlapXMLA/xmla;"
          + "Catalog=ALSMon");
    OlapWrapper wrapper = (OlapWrapper) connection;
    OlapConnection olapConnection =
   wrapper.unwrap(OlapConnection.class);
//...
```

Listagem 5.9: Exemplo de utilização do serviço XMLA numa aplicação cliente

```
//...
String MDX = "SELECT CrossJoin( \n"
+ "[DimDate.year-monthname_pt-day].[2015].Children, \n"
+ "[Measures].[area]) ON COLUMNS, \n"
+ "{ [DimPatient.name].Children } ON ROWS \n"
+ "FROM ALSMon";
OlapStatement statement = olapConnection.createStatement();
CellSet result = statement.executeOlapQuery(MDX);

//...
```

Listagem 5.10: Executar uma interrogação MDX numa aplicação XMLA cliente

A configuração do protocolo HTTPS foi realizada no servidor *web* aplicacional, Apache Tomcat. Esta configuração está detalhada na Secção 3.5, sendo utilizada a mesma que foi realizada para o serviço e a aplicação *web*, criadas para a receção dos dados.

A configuração da autenticação é também igual à da aplicação *web* para a receção dos dados, descrita no mesmo capítulo. Neste caso, através do ficheiro de configuração *tomcat-users.xml*, foram definidas mais duas novas "*role*" e criados utilizadores e palavras-chaves com estas "*role*", para as duas APIs. Depois, nas

duas aplicações *web*, foram utilizados os ficheiros *web.xml* para realizar a configuração necessária, ver anexo, Listagem A.1 (na listagem, neste caso, as "*role*" são "odata" e "xmla").

Assim para os clientes conseguirem comunicar e utilizar as APIs tem que utilizar o protocolo SSL e o certificado auto-assinado e também configurar a autenticação, de tipo *basic*, utilizando os nomes de utilizador e as palavras-chaves disponibilizadas.

6

Conclusões

Neste capítulo será feita uma síntese do trabalho realizado e realçadas as principais contribuições do projeto. O capítulo termina apontado alguns caminhos para o trabalho futuro.

6.1 Síntese e discussão de resultados

Neste trabalho implementou-se uma solução de *data warehouse* que permite analisar os sinais fisiológicos da doença ELA, auxiliando a tomada de decisão dos profissionais de saúde. Entre os requisitos principais foi garantir a segurança dos dados e implementá-la utilizando ferramentas *open source*.

Assim foi necessária a definição e a implementação de um modelo dimensional, esquema em estrela, representada por uma base de dados relacional MySQL, que servisse de suporte ao armazenamento dos dados. Para processar os dados recebidos foi definido e implementado um processo ETL automatizado que permite a extração e a transformação destes dados e o carregamento deles no DW. No DW foi criada uma zona temporária, representada por uma base de dados relacional MySQL, utilizada pelo processo ETL para extrair e transformar os dados antes de os carregar no DW. Foi definido e implementado também um servidor OLAP para permitir o acesso e a exploração do DW através da linguagem MDX. No servidor OLAP foram criadas pré-agregações para otimizar algumas consultas utilizadas com frequência.

Na implementação do DW foi utilizada a *suite* Pentaho. A escolha desta ferramenta superou as expetativas, provando-se ser muito flexível. Embora apresentando uma curva de aprendizagem maior, por exemplo comparando com a solução comercial da Microsoft, no leque de opções e ferramentas ultrapassa a solução a Microsoft.

Para receber os dados necessários procurou-se implementar uma solução que não seja especifica e que permite enviar qualquer tipo de dados de qualquer tamanho. Encontrou-se uma solução que oferece um bom desempenho e que utiliza padrões, sendo assim fácil de utilizar e possível de integrar com outras aplicações. A solução foi um serviço SOAP que utiliza o mecanismo MTOM para a transferência dos dados. Foi ainda implementada uma aplicação *web*, que utiliza de uma forma fácil o serviço para a receção dos dados, dadas as características do serviço acima mencionadas.

Depois da implementação foram configurados envios automatizados de dados que simulam os dados reais do exame EMG e a meta informação para testar a solução.

No última parte, a ideia principal foi disponibilizar em formato padrão o acesso aos dados do DW. Foram encontradas duas soluções, e implementadas utilizando ferramentas *open-source*. São dois serviços, um de tipo REST que utiliza o padrão OData e outro de tipo SOAP que utiliza o formato xml-analysis. Enquanto na API OData a consulta dos dados é realizadas através de pedidos HTTP *GET*, na API XMLA a consulta é realizada através da linguagem MDX. A outra diferença é na forma como é obtido internamente o resultado da consulta. Enquanto na API OData a interrogação é feita diretamente sobre o modelo em estrela na, API XMLA a interrogação é realizada através do cubo multidimensional, podendo-se aproveitar as funcionalidades deste, e.g. a utilização de pré-agregações, para diminuir o tempo de resposta para tipos de consultas realizadas com frequência.

Assim foram disponibilizadas duas possibilidades diferentes, de interrogar e obter os dados necessários ao DW. Cabe depois as aplicações cliente utilizar uma ou outra, conforme as necessidades.

6.2 Principais contribuições

O meu contributo neste projeto, que faz parte de um projeto multidisciplinar que envolve outros campos da engenharia, foi encontrar um serviço e uma aplicação *web* que fazem a receção dos dados de forma segura; um *data warehouse* onde são

verificados, processados e gravados de forma autónoma através de um processo ETL. No final, podem ser analisados os sinais fisiológicos, resultantes de uma eletromiografia de superfície, da doença ELA. A solução encontrada foi concretizada através da implantação dos artefactos, divididos em dois módulos, descritos nesta dissertação.

Ela faz a ponte entre os outros dois projetos envolvidos neste projeto multidisciplinar. Primeiro, disponibiliza mecanismos de receção dos dados para o projeto que realiza a recolha dos sinais. Depois processa e guarda os dados recebidos, duma forma automática, numa solução de armazenamento adequada, um *data warehouse*. Por fim, disponibiliza várias formas de acesso e consulta da informação deste DW, ao projeto que implementa uma aplicação *web* e a outras aplicações cliente que possam vir a ser construídas.

Tratando-se de um projeto inovador na investigação desta doença, o sistema concebido e implementado nesta dissertação, juntamente com os outros dois, poderá proporcionar informação útil, contribuir para a formulação de novo conhecimento na área e permitir aos profissionais de saúde de agir em tempo real, de uma forma mais adequada e personalizada.

6.3 Trabalho futuro

O trabalho desenvolvido inclui o processamento do exame EMG mas futuramente outros exames poderão vir a ser necessários para obter o máximo conhecimento sobre a doença.

O modelo de DW escolhido permite a evolução do esquema em estrela para o esquema em constelação, adicionando paralelamente mais tabelas de facto, reutilizando as dimensões existentes ou adicionando outras novas. Assim outros exames, representados através de novos DMarts, poderão ser incluídos e processados. O processo de ETL foi pensado também a permitir esta evolução, sendo necessário praticamente adicionar elementos novos à estrutura existente.

Em termos de otimização da performance do trabalho desenvolvido, existe a possibilidade de testes para melhoria, no segundo módulo, que disponibiliza a informação do DW.

Na API OData poderiam ser exploradas e utilizadas as melhores formas de obter os dados através de diferentes tipos de interrogações OData e diferentes formatos de resposta. Por exemplo, para um volume de dados pequeno, poder-se-iam

pedir os dados não agregados duma vez e realizar diferentes tipos de agregação do lado da aplicação cliente ou ao contrário, para um volume de dados grande, pedir os dados já agregados.

Para otimizar o tempo de resposta da API XMLA, as interrogações necessárias podem ser testadas e consequentemente criar vários tipos de pré-agregações no servidor OLAP.

Além destas duas APIs poderão ser exploradas outras possibilidades de disponibilizar a informação do DW às aplicações cliente, por exemplo, ligando-as diretamente como cliente OLAP para interrogar o DW, e não por intermédio de um serviço XMLA.

Por fim, pode ser investigada a possibilidade desta solução vier a ser aproveitada e adaptada a outros tipos de doenças.

Referências

[1] 33 Open Source and Free Business Intelligence Solutions - Predictive Analytics Today. URL http://www.predictiveanalyticstoday.com/open-source-free-business-intelligence-solutions/. [Online; consultado em Julho 2015].

[2] Apache CXF - Wikipedia, the free encyclopedia. URL https://en.wikipedia.org/wiki/Apache_CXF. [Online; consultado em Junho 2015].

[3] Casos de Sucesso. URL http://www.xpand-it.com/pt/casos-de-sucesso. [Online; consultado em Julho 2015].

[4] DB-Engines Ranking - popularity ranking of database management systems. URL http://db-engines.com/en/ranking. [Online; consultado em Setembro 2015].

[5] Difference between SOAP and RESTful Webservices - Interview Questions. URL https://sites.google.com/site/interviewsharing/asp-net/difference-between-soap-and-restful-webservices. [Online; consultado em Junho 2015].

[6] Apache Axis2 - Handling Binary data with Axis2 (MTOM/SwA). URL http://axis.apache.org/axis2/java/core/docs/mtom-guide.html. [Online; consultado em Junho 2015].

[7] MySQL :: Top 10 Reasons to Choose MySQL for Next Generation Web Applications. URL https://www.mysql.com/why-mysql/white-papers/top-10-reasons-to-choose-mysql-for-next-generation-web-applications/. [Online; consultado em Julho 2015].

[8] OData Version 4.0 Part 1: Protocol Plus Errata 02. URL http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398201. [Online; consultado em Setembro 2015].

[9] Pentaho Mondrian Documentation. URL http://mondrian.pentaho.com/documentation/. [Online; consultado em Setembro 2015].

[10] Why do we need Staging Area during ETL Load. URL http://dwbi.org/etl/etl/52-why-do-we-need-staging-area-during-etl-load.

[11] XML for Analysis Specification. URL https://msdn.microsoft.com/en-us/library/ms977626.aspx. [Online; consultado em Setembro 2015].

[12] Data warehouse - Wikipedia, the free encyclopedia. URL https://en.wikipedia.org/wiki/Data_warehouse. [Online; consultado em Junho 2015].

[13] Fundamentos e Modelagem de Bancos de Dados Multidimensionais. URL https://msdn.microsoft.com/pt-br/library/cc518031.aspx#XSLTsection126121120120. [Online; consultado em Junho 2015].

[14] Carlos Barbieri. BI–Business Intelligence–Modelagem & Tecnologia. *Rio de*, 2001.

[15] Jorge Bernardino. Open source business intelligence platforms for engineering education. *WEE2011, September*, pages 27–30, 2011.

[16] Mary Breslin. Data warehousing battle of the giants. *Business Intelligence Journal*, page 7, 2004.

[17] Susan Byrne, Cathal Walsh, Catherine Lynch, Peter Bede, Marwa Elamin, Kevin Kenna, Russell McLaughlin, and Orla Hardiman. Rate of familial amyotrophic lateral sclerosis: a systematic review and meta-analysis. *Journal of Neurology, Neurosurgery & Psychiatry*, 82(6):623–627, 2011.

[18] DA Chappell. Introducing OData: Data Access for the Web, the cloud, mobile devices, and more. *Microsoft Whitepaper, May*, 2011.

[19] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[20] Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. An overview of business intelligence technology. *Communications of the ACM*, 54(8):88–98, 2011.

[21] A Chiò, G Logroscino, BJ Traynor, J Collins, JC Simeone, LA Goldstein, and LA White. Global epidemiology of amyotrophic lateral sclerosis: a systematic review of the published literature. *Neuroepidemiology*, 41(2):118, 2013.

[22] Gari D Clifford and David Clifton. Wireless technology in disease management and medicine. *Annual Review of Medicine*, 63:479–492, 2012.

[23] Sérgio António Ramos da Costa. Sistema de business intelligence como suporte à gestão estratégica. 2012.

[24] Susana Cristina da Costa Pinto. *Estudos Clínicos e Neurofisiológicos para a Compreensão e a Reabilitação da Fraqueza e da Fadiga Respiratórias na Esclerose Lateral Amiotrófica*. PhD thesis, Faculdade de Medicina de Lisboa da Universidade de Lisboa, Outubro 2012.

[25] JP de Almeida, Pinto AC, Pereira J, Pinto S, and de Carvalho M. Implementation of a wireless device for real-time telemedical assistance of home-ventilated amyotrophic lateral sclerosis patients: a feasibility study. *Telemedicine J E Health*, 16(8):883–888, Oct 2010. doi: 10.1089/tmj.2010.0042.

[26] Bruno Nascimento de Ávila, Rodrigo Vitorino Moravia, Maria Renata Furtado, and Viviane Rodrigues Silva. Implantação do DW na ANVISA.

[27] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[28] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The dimensional fact model: a conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7(02n03):215–247, 1998.

[29] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.

[30] Claudia Imhoff, Nicholas Galemmo, and Jonathan G. Geiger. *Mastering Data Warehouse Design, Relational and Dimensional Techniques*. Wiley Publishing, Inc., 2003. ISBN 0-471-32421-3.

[31] William H Inmon. *Building the data warehouse*. John Wiley & Sons, 2005.

[32] Han Jiawei and Micheline Kamber. Data mining: concepts and techniques. *San Francisco, CA, itd: Morgan Kaufmann*, 5, 2001.

[33] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit*. John Wiley & Sons, Inc., 2nd edition edition, 2002.

[34] Ralph Kimball and Margy Ross. *The Kimball Group Reader; Relentlessly Practical Tools for Data Warehousing and Business Intelligence*. Wiley Publishing, Inc., 2010. ISBN 978-0-470-56310-6.

[35] Ralph Kimball, Laura Reeves, Warren Thornthwaite, Margy Ross, and Warren Thornwaite. The data warehouse lifecycle toolkit: Expert methods for designing, developing and deploying data warehouses with cd rom. 1998.

[36] Giancarlo Logroscino, Bryan J Traynor, Orla Hardiman, Adriano Chiò, Douglas Mitchell, Robert J Swingler, Andrea Millul, Emma Benn, and Ettore Beghi. Incidence of amyotrophic lateral sclerosis in europe. *Journal of Neurology, Neurosurgery, and Psychiatry*, 81(4):385–390, 04 2010.

[37] David Marco. Building and managing the meta data repository. *A full lifecycle guide*, 2000.

[38] Antonio Marinheiro and Jorge Bernardino. Analysis of open source business intelligence suites. In *Information Systems and Technologies (CISTI), 2013 8th Iberian Conference on*, pages 1–7. IEEE, 2013.

[39] Daniel L Moody and Mark AR Kortink. From enterprise models to dimensional models: a methodology for data warehouse and data mart design. In *DMDW*, page 5, 2000.

[40] Joy Mundy and Warren Thornthwaite. *The Microsoft data warehouse toolkit: With SQL server 2005 and the microsoft business intelligence toolset*. John Wiley & Sons, 2007.

[41] Risto Roine, Arto Ohinmaa, and David Hailey. Assessing telemedicine: a systematic review of the literature. *Canadian Medical Association Journal*, 165 (6):765–771, 2001.

[42] S Scalvini, M Vitacca, L Paletta, A Giordano, and B Balbi. Telemedicine: a new frontier for effective healthcare services. *Monaldi Arch Chest Dis*, 61(4): 226–233, 2004.

[43] Sanjay Sood, Victor Mbarika, Shakhina Jugoo, Reena Dookhy, Charles R Doarn, Nupur Prakash, and Ronald C Merrell. What is telemedicine? a collection of 104 peer-reviewed perspectives and theoretical underpinnings. *Telemedicine and e-Health*, 13(5):573–590, 2007.

[44] Michele Vitacca, Laura Comini, Giuliano Assoni, Domenico Fiorenza, Sonia Gilè, Palmira Bernocchi, and Simonetta Scalvini. Tele-assistance in patients with amyotrophic lateral sclerosis: long term activity and costs. *Disability and Rehabilitation: Assistive Technology*, 7(6):494–500, 2012.

[45] Heather M Young. Challenges and solutions for care of frail older adults. *Online journal of issues in nursing*, 8(2), 2003.

[46] Rui Zhang and Ling Liu. Security Models and Requirements for Healthcare Application Clouds. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 268–275, July 2010. doi: 10.1109/CLOUD.2010.62.

Anexos

```
//...
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>
               SecuredUploadSite
            </web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
          <description>
              Let only upload users use this app
            </description>
            <role-name>upload</role-name>
        </auth-constraint>
        <user-data-constraint>
          <transport-guarantee>
            CONFIDENTIAL
          </transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <security-role>
      <role-name>upload</role-name>
```

```
    </security-role>
24  <login-config>
      <auth-method>BASIC</auth-method>
26  </login-config>
//...
```

Listagem A.1: A configuração da autenticação no ficheiro *web.xml*

```
# This file was generated by Pentaho Data Integration version
    5.3.0.0-213.
# Note: lines like these with a # in front of it are comments

#general
#databases scripts location
scripts_dir=/alsmon/dw/workspace_kettle/databases_scripts

#locale
language_code_en=en
country_code_en=US
language_code_pt=pt
country_code_pt=PT

#stagingarea
#jobs location
sa_jobs_dir=/alsmon/dw/workspace_kettle/stagingarea/jobs
#transformations location
sa_transf_dir=/alsmon/dw/workspace_kettle/stagingarea/transformations
#upload dir location
sa_upload_dir=/alsmon/service/uploads
#processed dir location
sa_processed_dir=/alsmon/dw/workspace_kettle/files_processed
#tmp for unzip  dir location
sa_tmp_dir=/alsmon/dw/workspace_kettle/tmp_for_unzip

#signal
#threshold noise
threshold_noise=100
#window
window=50

#starschema
#jobs location
ss_jobs_dir=/alsmon/dw/workspace_kettle/starschema/jobs
#transformations location
ss_transf_dir=/alsmon/dw/workspace_kettle/starschema/transformations
```

Listagem A.2: Ficheiro de configuração da ferramenta PDI

```
#
# Create Data Staging
#

CREATE DATABASE IF NOT EXISTS `alsm_stagingarea` DEFAULT
   CHARACTER SET utf8 COLLATE utf8_unicode_ci;
GRANT ALL ON alsm_stagingarea.* TO '*****'@'localhost'
   IDENTIFIED BY '*****';
USE `alsm_stagingarea`;
START TRANSACTION;
CREATE TABLE IF NOT EXISTS patient
  (
    patient_id VARCHAR(10) NOT NULL,
    name VARCHAR(50) NOT NULL,
    gender CHAR(1) NOT NULL,
    birthdate DATE NOT NULL,
    diagnosedon DATE NOT NULL,
    createdon DATE NOT NULL,
    updatedon DATE NOT NULL,
    rowversion VARCHAR(18) NOT NULL,
    at_high DECIMAL(5,3) NOT NULL,
    at_low DECIMAL(5,3) NOT NULL,
    fcr_high DECIMAL(5,3) NOT NULL,
    fcr_low DECIMAL(5,3) NOT NULL,
    scm_high DECIMAL(5,3) NOT NULL,
    scm_low DECIMAL(5,3) NOT NULL,
    PRIMARY KEY (patient_id)
);
CREATE TABLE IF NOT EXISTS muscle
  (
    muscle_id INT NOT NULL,
    name VARCHAR(50) NOT NULL,
    acronym VARCHAR(10) NOT NULL,
    side VARCHAR(25),
    PRIMARY KEY (muscle_id)
);
CREATE TABLE IF NOT EXISTS emg
  (
    emg_id BIGINT NOT NULL AUTO_INCREMENT,
```

```
38      patient_id VARCHAR(10) NOT NULL,
        muscle_id INT NOT NULL,
40      fulldate DATE NOT NULL,
        fulltime TIME NOT NULL,
42      nr_peaks INT,
        area DOUBLE,
44      high_pass VARCHAR(10) NOT NULL,
        low_pass VARCHAR(10) NOT NULL,
46      electrode VARCHAR(50) NOT NULL,
        protocol VARCHAR(50) NOT NULL,
48      PRIMARY KEY (emg_id),
        FOREIGN KEY (patient_id) REFERENCES patient(patient_id),
50      FOREIGN KEY (muscle_id) REFERENCES muscle(muscle_id)
  );
52 CREATE TABLE IF NOT EXISTS emg_data
    (
54      emg_id BIGINT NOT NULL,
        x DOUBLE NOT NULL,
56      y DOUBLE NOT NULL,
        PRIMARY KEY (emg_id, x),
58      FOREIGN KEY (emg_id) REFERENCES emg(emg_id)
  );
60 SET @exists = (SELECT count(1) FROM information_schema.tables
     WHERE table_schema = 'alsm_stagingarea' AND table_name =
     'metadata_execution');
  CREATE TABLE IF NOT EXISTS metadata_execution
62    (
        execution_number INT NOT NULL AUTO_INCREMENT,
64      stagingarea_execution_date DATE NOT NULL,
        starschema_execution_date DATE,
66      stagingarea_result BOOLEAN NOT NULL,
        starschema_result BOOLEAN NULL,
68      PRIMARY KEY (execution_number)
  );
70 SET @s = IF(!@exists, "INSERT INTO
     alsm_stagingarea.metadata_execution
     (`stagingarea_execution_date`, `starschema_execution_date`,
     `stagingarea_result`, `starschema_result`) VALUES
     ('2015-04-01', '2015-04-01', true, true)", 'DO SLEEP(0)');
  PREPARE stmt FROM @s;
```

```
72 EXECUTE stmt;
   COMMIT;
```

Listagem A.3: O Script SQL para a criação e a configuração da *data staging*

```
#
# Create Star Schema
#

CREATE DATABASE IF NOT EXISTS `alsm_starschema` DEFAULT
    CHARACTER SET utf8 COLLATE utf8_unicode_ci;
GRANT ALL ON alsm_starschema.* TO '*****'@'localhost'
    IDENTIFIED BY '*****';
USE `alsm_starschema`;
START TRANSACTION;
CREATE TABLE IF NOT EXISTS dim_date
   (
     keycol INT NOT NULL AUTO_INCREMENT,
     date_iso DATE NOT NULL,
     fulldate_en VARCHAR(50) NOT NULL,
     fulldate_pt VARCHAR(50) NOT NULL,
     dayofweek SMALLINT NOT NULL,
     dayofmonth SMALLINT NOT NULL,
     dayname_en VARCHAR(15) NOT NULL,
     dayname_pt VARCHAR(15) NOT NULL,
     weekofyear SMALLINT NOT NULL,
     monthnumber SMALLINT NOT NULL,
     monthname_en VARCHAR(25) NOT NULL,
     monthname_pt VARCHAR(25) NOT NULL,
     quarter SMALLINT NOT NULL,
     year SMALLINT NOT NULL,
     year_weekofyear CHAR(9) NOT NULL,
     year_monthnumber CHAR(9) NOT NULL,
     year_monthname_en VARCHAR(32) NOT NULL,
     year_monthname_pt VARCHAR(32) NOT NULL,
     year_quarter CHAR(9) NOT NULL,
     UNIQUE (date_iso),
     PRIMARY KEY (keycol)
);
CREATE TABLE IF NOT EXISTS dim_time
   (
     keycol INT NOT NULL AUTO_INCREMENT,
     fulltime24 TIME NOT NULL,
     fulltime12 CHAR(11) NOT NULL,
```

```
    hour24 SMALLINT NOT NULL,
39  hour12 CHAR(5) NOT NULL,
    minute SMALLINT NOT NULL,
41  second SMALLINT NOT NULL,
    period_en VARCHAR(25) NOT NULL,
43  period_pt VARCHAR(25) NOT NULL,
    UNIQUE (fulltime24),
45  PRIMARY KEY (keycol)
);
47 CREATE TABLE IF NOT EXISTS dim_patient
  (
49  keycol INT NOT NULL AUTO_INCREMENT,
    patient_id VARCHAR(10) NOT NULL,
51  name VARCHAR(50) NOT NULL,
    gender CHAR(1) NOT NULL,
53  birthdate DATE NOT NULL,
    diagnosedon DATE NOT NULL,
55  createdon DATE NOT NULL,
    updatedon DATE NOT NULL,
57  rowversion VARCHAR(18) NOT NULL,
    at_high DECIMAL(5,3) NOT NULL,
59  at_low DECIMAL(5,3) NOT NULL,
    fcr_high DECIMAL(5,3) NOT NULL,
61  fcr_low DECIMAL(5,3) NOT NULL,
    scm_high DECIMAL(5,3) NOT NULL,
63  scm_low DECIMAL(5,3) NOT NULL,
    UNIQUE (patient_id),
65  PRIMARY KEY (keycol)
);
67 CREATE TABLE IF NOT EXISTS dim_muscle
  (
69  keycol INT NOT NULL AUTO_INCREMENT,
    muscle_id INT NOT NULL,
71  name VARCHAR(50) NOT NULL,
    acronym VARCHAR(10) NOT NULL,
73  side VARCHAR(25),
    UNIQUE (muscle_id),
75  PRIMARY KEY (keycol)
);
77 CREATE TABLE IF NOT EXISTS fact_emg
```

```
  (
79    emg_id BIGINT NOT NULL,
      date_keycol INT NOT NULL,
81    time_keycol INT NOT NULL,
      patient_keycol INT NOT NULL,
83    muscle_keycol INT NOT NULL,
      nr_peaks INT NOT NULL,
85    area DOUBLE NOT NULL,
      high_pass VARCHAR(10) NOT NULL,
87    low_pass VARCHAR(10) NOT NULL,
      electrode VARCHAR(50) NOT NULL,
89    protocol VARCHAR(50) NOT NULL,
      UNIQUE (emg_id),
91    PRIMARY KEY (date_keycol, time_keycol, patient_keycol,
   muscle_keycol),
      FOREIGN KEY (date_keycol) REFERENCES dim_date(keycol),
93    FOREIGN KEY (time_keycol) REFERENCES dim_time(keycol),
      FOREIGN KEY (patient_keycol) REFERENCES dim_patient(keycol),
95    FOREIGN KEY (muscle_keycol) REFERENCES dim_muscle(keycol)
);
97 CREATE TABLE IF NOT EXISTS fact_emg_data
  (
99    emg_id BIGINT NOT NULL,
      x DOUBLE NOT NULL,
101   y DOUBLE NOT NULL,
      PRIMARY KEY (emg_id, x),
103   FOREIGN KEY (emg_id) REFERENCES fact_emg(emg_id)
);
105 COMMIT;
```

Listagem A.4: O Script SQL para a criação e a configuração do esquema estrela

```
public long calculateNrPeaks(double[] y_envelope, double
    pat_muscle, double threshold_noise, int window) {
  int nr_peaks = 0;
  boolean peak_found = false;
  i=0;
  while (i < (y_envelope.length - window)) {
    if ((y_envelope[i] - pat_muscle) > threshold_noise) {
      if(!peak_found) {
        nr_peaks += 1;
        peak_found = true;
      }
    }
    else {
      if(peak_found) {
        for (j = (i + 1); j <=( i + window) ; j++) {
          if ((y_envelope[j] - pat_muscle) > threshold_noise) {
            break;
          }
        }
        if (j > (i + window))
          peak_found = false;

        i = j + 1;
      }
    }
    i++;
  }
  //last window - maybe one more peak
  for (j = i; j < y_envelope.length ; j++) {
    if ((y_envelope[i] - pat_muscle) > threshold_noise) {
      if(!peak_found) {
        nr_peaks += 1;
        break;
      }
    }
  }
  return (long)nr_peaks;
}
```

Listagem A.5: Algoritmo para calcular o facto *nr_peaks*

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
   xmlns="http://xmlns.jcp.org/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="ODataDAL"
   transaction-type="RESOURCE_LOCAL">
    <class>pt.isel.alsmon.model.DimDate</class>
    <class>pt.isel.alsmon.model.DimMuscle</class>
    <class>pt.isel.alsmon.model.DimPatient</class>
    <class>pt.isel.alsmon.model.DimTime</class>
    <class>pt.isel.alsmon.model.FactEmg</class>
    <class>pt.isel.alsmon.model.FactEmgPK</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
   value="jdbc:mysql://localhost:3306/alsm_starschema"/>
      <property name="javax.persistence.jdbc.user"
   value="*****"/>
      <property name="javax.persistence.jdbc.password"
   value="*****"/>
      <property name="javax.persistence.jdbc.driver"
   value="com.mysql.jdbc.Driver"/>
    </properties>
  </persistence-unit>
</persistence>
```

Listagem A.6: Ficheiro de configuração JPA, *persistence.xml*

```
<?xml version="1.0"?>
<DataSources>
  <DataSource>
    <DataSourceName>Provider=Mondrian;DataSource=ALSMonSource;
    </DataSourceName>
    <DataSourceDescription>MondrianALSMon Data Warehouse
    </DataSourceDescription>
    <URL>https://alsmon.adeetc.e.ipl.pt:8443/OlapXMLA/xmla</URL>
    <DataSourceInfo>Provider=mondrian;
    Jdbc=jdbc:mysql://localhost:3306/alsm_starschema;
    JdbcUser=*****;JdbcPassword=*****;
    JdbcDrivers=com.mysql.jdbc.Driver;
    Catalog=/WEB-INF/schema/ALSMonMondrianSchema.xml;
    </DataSourceInfo>
    <ProviderName>Mondrian</ProviderName>
    <ProviderType>MDP</ProviderType>
    <AuthenticationMode>Unauthenticated</AuthenticationMode>
    <Catalogs>
      <Catalog name="ALSMon">
        <Definition>/WEB-INF/schema/ALSMonMondrianSchema.xml
        </Definition>
      </Catalog>
    </Catalogs>
  </DataSource>
</DataSources>
```

Listagem A.7: Ficheiro de configuração da API XMLA, *datasources.xml*