
Universidade de Aveiro
2012
Departamento de Electrónica, Telecomunicações e Informática

Mário Seoane Veiga

Representação dinâmica de dados de fontes heterogéneas

Universidade de Aveiro
2012
Departamento de Electrónica, Telecomunicações e Informática

Mário Seoane Veiga

Representação dinâmica de dados de fontes heterogéneas

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Cláudio Jorge Vieira Teixeira, Equiparado a Investigador Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Professor Doutor Diogo Nuno Pereira Gomes, Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro. Esta dissertação contou ainda com a colaboração do Engenheiro Filipe António Rodrigues Barreto Trancho, Coordenador da Área de Sistemas e Gestão de Informação da Universidade de Aveiro.

Dedico este trabalho a quem não poderia deixar de ser.

o júri

presidente

Professor Doutor Osvaldo Manuel da Rocha Pacheco
Professor Auxiliar da Universidade de Aveiro

vogais

Mestre Benjamim Pereira Mendes Júnior
Director na PT Comunicações

Doutor Cláudio Jorge Vieira Teixeira
Equiparado a Investigador Auxiliar da Universidade de Aveiro

Professor Doutor Diogo Nuno Pereira Gomes
Professor Auxiliar Convidado da Universidade de Aveiro

agradecimentos

Agradeço aos meu orientadores, Dr. Cláudio Teixeira e Prof. Dr. Diogo Gomes, por todo o apoio, acompanhamento e disponibilidade no decorrer deste trabalho.
Agradeço também ao Eng. Filipe Trancho toda a sua colaboração.

palavras-chave

Open Data, *reports*, representação de dados, dados heterogéneos, visualização de dados

resumo

Esta dissertação surge no contexto de *Open Data*. O objectivo é propor uma solução que permita às instituições que aderem a este princípio representar de forma dinâmica os dados disponibilizados, bem como compreender quais as diferenças na implementação dessa solução na representação de dados heterogéneos provenientes de uma só fonte e na representação de dados provenientes de fontes heterogéneas.

Após uma apresentação do conceito de *Open Data*, dando-se a conhecer alguns portais *web* que o aplicam, bem como do conceito de *Linked Open Data* derivado do primeiro, são abordadas algumas técnicas e heurísticas que podem ser utilizadas na criação de visualizações gráficas de dados, apresentando-se também algumas ferramentas que podem ser utilizadas para esse fim.

Nesta dissertação procura-se ainda dar a conhecer algumas ferramentas que podem ser utilizadas na criação de *reports*.

O cenário utilizado para o desenvolvimento desta solução foi a Universidade de Aveiro, sendo que o mesmo permitiu a criação de um protótipo para uma aplicação que possibilita a representação e visualização de forma dinâmica dos dados facultados por esta instituição por meio de tabelas, gráficos e *reports*, independentemente da proveniência dos mesmos.

keywords

Open Data, reports, data representation, heterogeneous data, data visualization

abstract

This dissertation is inserted in the Open Data context. Its objective is to propose a solution that allows institutions adherent to this principle the ability to represent data in a dynamic fashion, as well as to understand the main differences in the implementation of this solution for the representation of heterogeneous data from only one source and the representation of data from heterogeneous sources.

After a presentation of the Open Data and Linked Open Data concepts, enumerating some web portals that apply these principles, some techniques and heuristics that can be used in the creation of graphic data visualizations, as well as some tools that can be used to this end are described. Also, some tools that permit the creation of reports are described in this dissertation.

The scenario used to develop this solution was Aveiro University. This allowed the creation of a prototype application that permits the dynamic representation and visualization of data provided by this institution by means of tables, charts and reports, independently of its source.

Índice

Índice de figuras

Índice de tabelas

Lista de acrónimos

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CLI	Call Level Interface
CORBA	Common Object Request Broker Architecture
CSV	Comma-separated values
DLL	Dynamic-Link Library
DOC	Microsoft Word file format
DOM	Document Object Model
GCT	Google Chart Tools
HTML	HyperText Markup Language
HTML5	HyperText Markup Language 5
HTTP	Hipertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IBM	International Business Machines Corporation
IPS	Internet Protocol Suite
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
LD	Linked Data
LOD	Linked Open Data
N3	Notation 3
OD	Open Data
OD@UA	Open Data @ UA
ODBC	Open Database Connectivity
PDF	Portable Document Format
PNG	Portable Network Graphics
RDF	Resource Description Framework
RDF/XML	Resource Description Framework/Extensible Markup Language
REST	Representational State Transfer

RSS	Rich Site Summary
SGBD	Sistema de Gestão de Bases de Dados
SMTP	Simple Mail Transfer Protocol
SO	Sistema Operativo
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TIFF	Tagged Image File Format
UA	Universidade de Aveiro
UA-API	Camada de Serviços Academic Playground & Innovation
UA-BD	Base de dados dos serviços académicos da UA
UDDI	Universal Description, Discovery and Integration
URI	Uniform Ressource Identifier
URL	Uniform Resource Locator
VML	Vector Markup Language
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XHTML/CSS	Extensible HyperText Markup Language/Cascading Style Sheets
XLS	Microsoft Excel file format
XML	Extensible Markup Language

1 Introdução

1.1 Contextualização

Na sociedade actual, qualquer pessoa dispõe de instrumentos que lhe permitem efectuar uma recolha de dados sobre o seu dia-a-dia. Smartphones, câmaras fotográficas, tablets, ou mesmo o veterano computador pessoal, são dispositivos que constantemente recolhem dados sobre as vidas dos seus utilizadores. Surge então o desafio sobre o que fazer com tais colecções de dados e como agregar as mesmas, por forma a permitir criar representações a partir das quais possa ser retirada informação útil. No entanto, uma vez que segundo Teraoka [1] estes dados são heterogéneos, ou seja, contêm uma grande diversidade de formatos e provêm de fontes diversas, é necessário encontrar estratégias para os organizar e representar, para que deles se possa retirar informação de forma simples, clara e eficaz.

No que respeita às instituições, sejam elas públicas ou privadas, estas geram diariamente vastas colecções de dados no decorrer da sua actividade. Visto que numa só instituição existem habitualmente diversos sistemas de informação, é possível que cada sistema utilize um critério próprio na criação e armazenamento dos dados, o que os torna heterogéneos, sendo por isso muitas vezes incompreensíveis fora do contexto em que foram gerados.

Para as instituições, é do maior interesse compreender todos os dados de que dispõem da forma mais clara possível, pois estes podem ser uma poderosa ferramenta de auxílio ao seu bom funcionamento. Numa empresa, relacionar dados obtidos a partir de diferentes fontes, e que aparentemente não têm qualquer relação entre si, pode ser revelador de importante informação que de outra forma ficaria oculta. O mesmo se passa com os Estados, ao nível das instituições públicas. Por esta razão, recentemente tem-se assistido a uma tendência crescente por parte destes em disponibilizar as suas vastas colecções de dados, permitindo assim aos cidadãos estarem mais próximos da informação, tornando o processo democrático mais transparente.

É neste contexto que surge o conceito de *Open Data* (OD), o qual defende que as instituições devem disponibilizar as suas colecções de dados, permitindo que qualquer indivíduo tenha acesso aos mesmos e os possa estudar, cruzar e republicar.

Seguindo esta filosofia de OD, a Comissão Europeia, órgão executivo da União Europeia, anunciou uma nova estratégia de disponibilização de dados para a Europa [2], o que irá forçar os Estados membros a tornar todos os dados do sector público acessíveis num formato *machine-readable*, permitindo assim a sua posterior reutilização. No entanto, para que os dados possam ser utilizados pelos cidadãos na monitorização dos seus representantes políticos ou agências governamentais, é necessário fazer muito mais do que disponibilizá-los num formato *machine-*

readable, uma vez que a extracção de informação útil a partir dos mesmos é, tal como afirmam Böhm et al. [3], bastante complexa: os dados podem estar em formatos diferentes, podem conter erros e é difícil obter uma eficaz execução de *queries ad hoc* em colecções de dados de grandes dimensões.

Apesar de todas as dificuldades, são já muitos os países que disponibilizam portais *web* onde se encontram vastas colecções de dados, e que permitem criar visualizações sobre estes. Alguns dos exemplos internacionais mais conhecidos são os portais *data.gov* [4] dos Estados Unidos da América, *data.gov.uk* [5] do Reino Unido e *data.gov.au* [6] da Comunidade da Austrália. Em Portugal existem igualmente portais equivalentes: *pordata.pt* [7] e *dados.gov.pt* [8].

No âmbito científico, com o aparecimento de instrumentos de medição cada vez mais complexos e sofisticados, transformando a ciência em algo crescentemente digital, tal como afirma Uselton [9], surgem novas oportunidades de analisar dados experimentais, bem como de comparação de dados simulados com estes. Talvez mais do que em qualquer outro âmbito, em ciência, dados compilados num determinado contexto podem ser relevantes num contexto completamente diverso, pois a explicação dos fenómenos nem sempre se limita a contextos previamente definidos. Segundo Andorf et al. [10], também em ciência a heterogeneidade dos dados é um problema, já que a comunidade científica é composta por diversos indivíduos ou grupos autónomos que se servem de distintas semânticas, o que poderá, eventualmente, dificultar a transposição de determinados dados para um contexto diferente daquele no qual foram adquiridos, e impedir um cruzamento interdisciplinar dos mesmos.

Com todos estes dados, sejam eles de acesso livre ou disponíveis apenas a um grupo restrito de pessoas, surge a necessidade de criar ferramentas capazes de os representar de uma forma compreensível para o utilizador, permitindo o cruzamento de dados de diversas fontes e dando-lhes um contexto comum, gerando informação e potenciando o conhecimento. No entanto, esta não é uma tarefa trivial, pois para além de ser necessário criar uma semântica comum para os dados, é também importante pensar a melhor forma de os representar, uma vez que caso a representação não seja a mais favorável, a compreensão dos dados não será conseguida. Por esta razão, na criação de uma ferramenta deste tipo devem utilizar-se técnicas e heurísticas que permitam melhorar a representação da informação disponibilizada ao utilizador, bem como a forma como este interage com a ferramenta para melhor adquirir conhecimento.

1.2 Motivação

Com o crescente interesse por parte das instituições em disponibilizarem informação, surge a necessidade de se criarem ferramentas capazes de a representar de forma a produzir

conhecimento. Uma vez que essa informação é por vezes proveniente de diversas fontes, torna-se imperativo que as ferramentas tenham a capacidade de integrar os dados de uma forma lógica.

No contexto da Universidade de Aveiro (UA), cenário de estudo para esta dissertação, existem diversas colecções de dados provenientes de fontes distintas. Estes dados são relativos a diferentes temas, como por exemplo informação respeitante às diversas unidades curriculares leccionadas na UA ou às obras bibliográficas disponíveis na biblioteca, temas estes que por vezes não têm uma relação aparente entre eles, mas que se analisados de uma forma mais aprofundada podem gerar conhecimento até aí oculto.

Tendo a UA decidido adoptar uma filosofia de OD, torna-se agora mais fácil criar conhecimento a partir dos dados da instituição, pois se até ao momento estes se encontravam em bases de dados de difícil acesso à comunidade académica, podem agora ser vistos e interpretados por esta comunidade, desde que existam ferramentas que facilitem esse processo.

Qualquer ferramenta desenvolvida no âmbito académico deve ter em conta que nem todos os membros da academia possuem conhecimentos informáticos. Por esta razão, uma ferramenta de representação de dados deve abstrair-se da complexidade de acesso aos dados subjacentes, e permitir ao utilizador menos experiente conseguir obter o conhecimento que pretende sobre os mesmos de uma forma autónoma e eficaz. Assim sendo, a forma como o utilizador navega até aos dados pretendidos deve ser livre de especificações técnicas e as representações devem ser efectuadas de uma forma familiar ao utilizador recorrendo, nomeadamente, a tabelas ou gráficos.

Este trabalho visa a criação de um protótipo de uma aplicação que permita cruzar e representar dados de distintas fontes no universo da UA de uma forma dinâmica, ou seja, será a própria aplicação a ter a capacidade de identificar quais os dados que podem ser cruzados entre si, bem como de criar as representações e as visualizações gráficas dos mesmos, sem que para isso seja necessária a intervenção do utilizador para além da escolha dos dados que pretende ver representados. Com esta aplicação a comunidade académica passará a dispor de uma fonte de conhecimento mais aprofundada sobre a instituição que integra, pois as diversas fontes de dados disponibilizadas por esta instituição poderão ser utilizadas pela aplicação de forma dinâmica, permitindo assim a esta comunidade servir-se das funcionalidades disponibilizadas pela aplicação para melhor explorar e compreender as colecções de dados provenientes das diversas fontes.

1.3 Objectivos

Com a realização deste trabalho pretende-se criar um protótipo de uma aplicação que permita a representação e visualização dinâmicas de dados. Esta deve ter a capacidade de permitir que os dados utilizados sejam de fontes heterogéneas, sendo irrelevante para o funcionamento da

mesma qual a sua proveniência. Por forma a que a aplicação seja independente das fontes de dados, para cada fonte deve ser implementado um interface que ofereça funções padrão que permitam à aplicação aceder aos dados de um modo uniforme, independentemente da sua origem. Para o utilizador da aplicação não devem existir distinções dos dados pela sua proveniência. Assim sendo, deve ser possível, caso seja essa a vontade do utilizador, cruzar dados procedentes da mesma fonte ou de fontes distintas. Além disso, depois de seleccionados, a aplicação deve permitir que os dados sejam visualizados pelo utilizador de uma forma dinâmica e compreensível para si, recorrendo a visualizações gráficas, tabelas e *reports* para atingir este fim. Deverá ainda ser possível ao utilizador gravar a pesquisa efectuada, bem como as visualizações gráficas criadas, para consulta futura e publicação aos demais utilizadores.

As fontes de dados a utilizar neste trabalho serão a base de dados dos serviços académicos da UA (UA-BD) e a camada de serviços disponibilizada pelo projecto *Academic Playground & Innovation* (UA-API) [11]. A primeira diz respeito a dados heterogéneos provenientes de uma única fonte. Já a segunda diz respeito a dados provenientes de fontes heterogéneas. Assim sendo, importa compreender de que forma o acesso aos dados por parte da aplicação é diferente em cada uma delas.

Esta dissertação tem como universo de estudo a UA podendo-se, no entanto, facilmente extrapolar o conhecimento aqui adquirido para qualquer tipo de instituição, seja ela pública ou privada, de forma a satisfazer as necessidades de representação de dados dessa entidade.

1.4 Organização da dissertação

Neste capítulo, sendo o primeiro, procura-se enquadrar e contextualizar o trabalho que irá ser desenvolvido, bem como definir as motivações para a sua realização e objectivos a alcançar com a sua conclusão.

No segundo capítulo é abordado o conceito OD, sendo focadas algumas das tecnologias mais utilizadas na sua implementação e três exemplos distintos de portais que se servem deste.

O terceiro capítulo trata de interacções de dados, sendo descritas tecnologias utilizadas na transferência dos mesmos, bem como modelos de comunicação destes.

No quarto capítulo, dedicado à visualização de dados, são focadas algumas técnicas que permitem a redução da desordem em visualizações gráficas de dados e também heurísticas que melhorem a capacidade de o utilizador adquirir informação a partir das mesmas. Por fim, descrevem-se algumas ferramentas que permitem a criação de visualizações gráficas de dados.

No quinto capítulo procura-se compreender o que são *reports*, sendo descritas ferramentas que possibilitam a sua criação.

No sexto capítulo descrevem-se os pormenores do protótipo criado ao longo deste trabalho para uma aplicação que permite a representação e visualização dinâmicas de dados.

No sétimo capítulo apresenta-se a conclusão desta dissertação e discute-se o trabalho a desenvolver no futuro.

2 *Open Data*

2.1 Contextualização

O conceito de OD, que significa literalmente dados abertos, tem vindo a ganhar reconhecimento, existindo actualmente cada vez mais portais *web* a apostar nos ideais que defende. Este conceito advoga que os dados, informação electronicamente armazenada [12], independentemente do tipo ou fonte, devem ser disponibilizados ao público sem quaisquer tipos de entraves, nomeadamente no que diz respeito a direitos de autor ou de propriedade, para que deles se possa servir com o intuito de aumentar o seu conhecimento sobre os mesmos. Machado e Oliveira [12] definem OD como sendo a publicação de dados em formatos não proprietários e em bruto, por meios que os tornem facilmente acessíveis e disponíveis ao público, permitindo o seu processamento automático bem como a sua reutilização.

A razão que possivelmente torna o conceito OD tão aliciante, é a de que se um indivíduo tem acesso a determinados dados pode deles retirar conhecimento. Desta forma, quanto mais dados estiverem disponíveis, maior será a compreensão do indivíduo sobre o universo em que se insere.

Para o comum dos cidadãos, a ideia do OD será provavelmente mais visível no contexto dos Estados e das instituições públicas. Neste contexto, denominado de *Open Government Data*, que limita o conceito de OD apenas a dados provenientes de instituições públicas, o cidadão tem a possibilidade de aumentar o seu conhecimento sobre as entidades que influenciam a sua vida e, assim, tomar decisões informadas no acto eleitoral, bem como exercer uma cidadania activa, pressionando os decisores políticos mais conscientemente. Neste sentido, o *Open Government Working Group* definiu os seguintes princípios que determinam quando é que os dados facultados pelos Estados devem ser considerados abertos, tendo esses princípios sido referidos em [12]:

- Quando são **completos**, ou seja, todos os dados devem ser disponibilizados, desde que não estejam sujeitos a limitações de segurança, privacidade ou de privilégios;
- Quando estão no seu **estado primário**, tendo sido recolhidos directamente da fonte, encontrando-se no estado mais atómico possível, não tendo sido previamente agregados ou modificados de nenhuma forma;
- Quando são **oportunos**, visto que os dados devem ser disponibilizados logo que deles exista necessidade para manter o seu valor;
- Quando são **acessíveis**, pois devem estar disponíveis para qualquer tipo de utilizador e para qualquer propósito;

- Quando são facilmente **processáveis**, uma vez que devem ter uma estrutura que permita um processamento automático;
- Quando não são **discriminatórios**, pois os dados devem estar acessíveis a qualquer pessoa sem necessidade de qualquer tipo de registo;
- Quando são disponibilizados em formatos **não proprietários**, uma vez que devem ser disponibilizados num formato não controlado por nenhuma entidade especifica;
- Quando são **livres de licenças**, já que estes não devem estar sujeitos a nenhum tipo de patentes ou direitos reservados.

Caso estes princípios estejam simultaneamente presentes numa determinada colecção de dados, esta pode considerar-se aberta. No entanto, apesar de estes princípios se referirem ao conceito de *Open Government Data*, podem ser extrapolados para o conceito mais abrangente de OD, já que nenhum deles é susceptível de ser exclusivo às instituições públicas, podendo mesmo ser aplicados no contexto de instituições privadas, nomeadamente de empresas.

No âmbito científico, o conceito de OD faz também todo o sentido, uma vez que a ciência partilha do mesmo objectivo que este: a criação de conhecimento. Assim sendo, é importante clarificar de que forma beneficia a ciência ao adoptar o paradigma OD. Com este intuito, Pardo e Sayogo [13] identificaram os principais benefícios relativos à partilha de dados científicos, sendo eles:

- O enriquecimento do conhecimento científico e aceleramento do seu progresso, pois desta forma é possível aos investigadores criar novo conhecimento a partir de colecções de dados antigas, mas inseridas em novo contexto, bem como a combinação de diferentes colecções de dados que de outra forma não estariam disponíveis;
- A promoção de trabalhos colaborativos entre investigadores, uma vez que ao serem disponibilizados dados científicos de uma forma aberta, um determinado investigador pode tomar conhecimento do trabalho realizado por outro, criando-se assim a possibilidade de um futuro trabalho colaborativo;
- Melhorar a responsabilização, já que assim os dados gerados durante determinada investigação estão mais disponíveis para que os pares possam avaliar o trabalho do investigador; aumentar a eficiência do esforço de investigação, sendo que assim será evitada a recolha redundante de dados, bem como serão reduzidos os custos temporais e monetários dispendidos na recolha dos dados;

- Expandir a reputação e o mérito científico, pois ao disponibilizar de forma pública os dados das suas investigações, o investigador aumenta a sua reputação junto do público, de publicações científicas e, por consequência, dos seus pares.

Apesar de todas as motivações existentes à adopção do paradigma OD, subsistem alguns desafios que tornam a sua implementação mais difícil. Do ponto de vista tecnológico, vários autores ([13], [12] e [14]) concordam que a heterogeneidade dos dados e das fontes dos mesmos é o factor que mais dificuldades coloca quando se pretende adoptar este paradigma, uma vez que habitualmente os dados se encontram em formatos distintos, alguns deles com erros ou omissões, não existe uma coerência semântica entre as várias colecções.

Além dos problemas tecnológicos inerentes à adopção do OD, existem também, segundo Pardo e Sayogo [13], algumas barreiras que influem negativamente na partilha de dados e, por consequência, dificultam a implementação do paradigma OD. Aspectos sociais, organizacionais ou económicos [13] podem inibir a partilha dos dados, pois muitas vezes é necessário despender algum dinheiro para atingir este fim e as verbas podem não estar disponíveis. Por vezes existem também barreiras legais ou entraves políticos [13], já que nem sempre existe interesse político em divulgar determinados dados ou estes podem estar de alguma forma legalmente impedidos de ser publicados. Também os contextos locais e especificidades [13] dificultam este processo, uma vez que dependendo do contexto, os dados podem adoptar significados diversos.

2.2 *Linked Open Data*

No contexto OD existe um conceito que alia os princípios deste com os princípios de *Linked Data* (LD), sendo denominado de *Linked Open Data* (LOD). Segundo Bizer et al. [15] LD significa a disponibilização de colecções de dados na *internet* para que estas sejam *machine-readable*, o seu significado seja explicitamente definido e que seja possível criar ligações entre elas e colecções externas ao âmbito no qual estão inseridas. Desta forma, LOD pode ser considerado como uma evolução natural do OD. Depois de disponibilizadas as colecções de dados, quanto mais ligações existirem entre elas, mais conhecimento se pode adquirir. Para Hausenblas [16], a nuvem composta por todas as ligações permitidas entre colecções de dados distintas pode mesmo ser interpretada como uma base de dados à escala da *internet*. Conforme se pode verificar pela análise da Figura 1, são já imensas as ligações existentes entre as distintas colecções de dados, estando assim criada e em constante transformação, uma extensa nuvem de conhecimento.

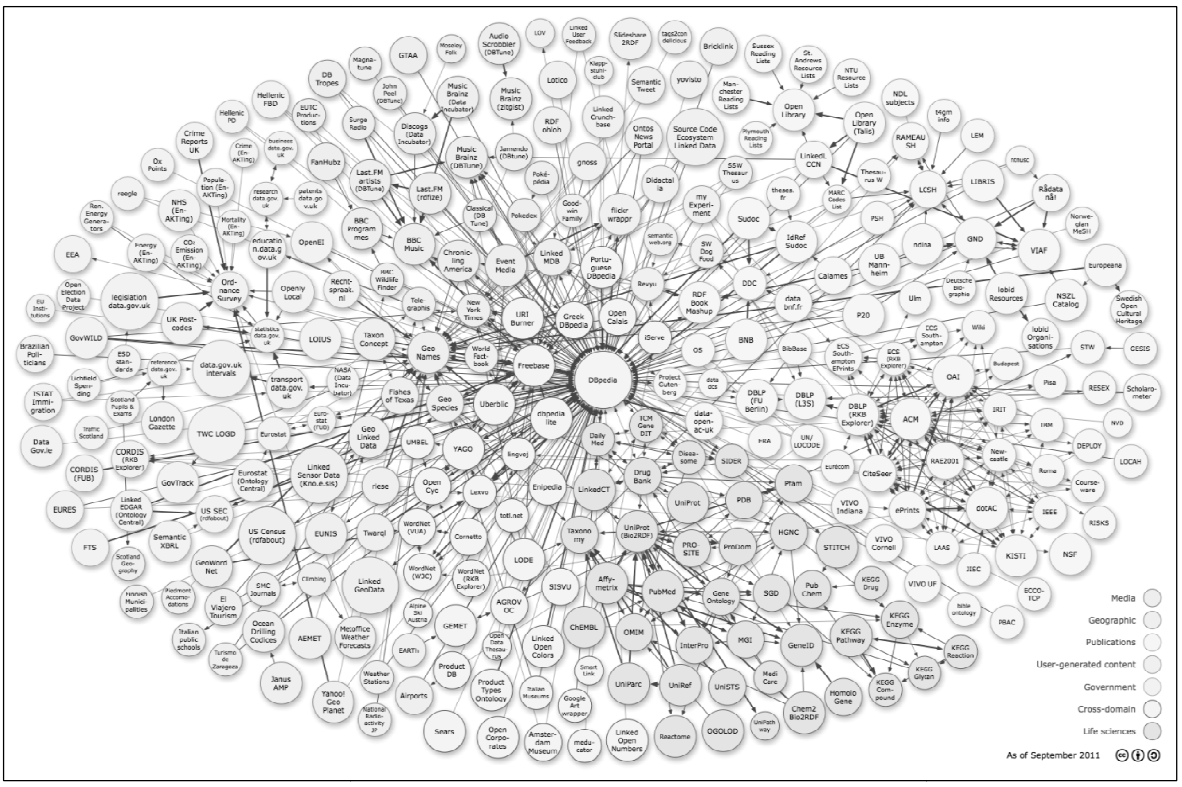


Figura 1: Diagrama da nuvem LOD em Setembro de 2011 [17]

Para se conseguir uma nuvem como a da Figura 1, foi necessário que os responsáveis pelas diversas colecções de dados as publicassem respeitando os princípios de LD apontados por Berners-Lee [18], sendo eles:

- Todos os dados de uma determinada colecção devem ser identificados por meio de *Uniform Ressource Identifiers* (URI);
- Devem ser utilizados URIs de *Hipertext Transfer Protocol* (HTTP) para que os dados possam ser facilmente alcançáveis;
- Deve ser prestada informação útil sempre que cada URI é visitado, por meio dos standards *Resource Description Framework* (RDF) e *SPARQL Protocol and RDF Query Language* (SPARQL);
- Devem ser incluídas ligações para outros URIs, permitindo assim que outros dados sejam alcançados.

Assim sendo, cada colecção de dados deve ser disponibilizada por meio de um URI, formatada em RDF, sendo permitida a execução de *queries* sobre a mesma utilizando SPARQL *Query Language*. Cada um destes standards será discutido nos próximos pontos desta dissertação.

2.2.1 URI

Um URI [19] é um conjunto de caracteres dispostos de acordo com uma sintaxe própria e que permite um meio simples de identificar um determinado recurso num contexto informático. É utilizando um URI que no âmbito de LOD é possível identificar dados específicos pertencentes a determinada colecção.

A sintaxe de um URI integra os elementos "scheme", "authority", "path", "query" e "fragment", conforme se pode verificar nos exemplos da Figura 2.

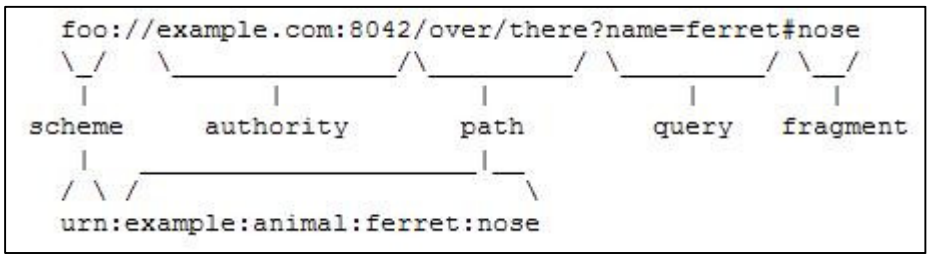


Figura 2: Exemplos de URIs e seus componentes sintáticos [19]

De acordo com a com a sua especificação sintáctica, um URI deve sempre conter, pelo menos, os elementos "scheme" e "path", sendo que o elemento "path" pode estar vazio.

Cada URI distinto inicia-se com o elemento "scheme". Este elemento representa o nome de uma especificação responsável por atribuir identificadores dentro desse "scheme". Por esta razão, este elemento acaba por influenciar a restante sintaxe e semântica de um determinado URI. Seguidamente, caso esteja presente, o elemento "authority" serve para definir qual a autoridade responsável pela gestão do "namespace" para os caracteres restantes do URI. Já os elementos "path" e "query", no seu conjunto, servem para identificar determinado recurso no âmbito do "scheme" do URI. No entanto, enquanto que normalmente os dados do elemento "path" estão organizados de uma forma hierárquica, o mesmo não acontece com os dados do elemento "query". Por fim, o elemento "fragment", caso tenha sido definido, permite a identificação indirecta de um recurso secundário por referência a um recurso primário devolvido pela "query" e por meio de informação identificativa adicional. O recurso secundário pode dizer respeito a uma porção do subconjunto do recurso primário, uma visão específica das representações do recurso primário ou um recurso distinto que seja definido ou descrito por essas especificações.

2.2.2 RDF

RDF é uma linguagem desenvolvida pelo *World Wide Web Consortium* (W3C), descrita em [20] e [21], sendo utilizada para representar informação na *internet*. Inicialmente foi criada com o intuito de representar *metadata* relativa a documentos *web*. No entanto, cedo se percebeu que poderia ter aplicações muito mais interessantes, passando a ser utilizada na descrição e modelação de informação que pudesse ser identificada por meio de um URI.

O RDF foi desenvolvido com o intuito de tornar possível o processamento automático da informação por meio de aplicações, e não tanto de tornar a exibição da mesma mais compreensível a humanos. Assim sendo, o RDF oferece a possibilidade de troca de informação entre aplicações distintas sem que o sentido da mesma seja perdido, uma vez que é fornecido um modelo de dados comum.

O modelo de dados do RDF é denominado de *graph data model*, já que este toma a forma de um grafo. Este grafo é um conjunto de *triples*, sendo que cada *triple* é composto por três elementos: o sujeito, o predicado e o objecto, conforme se pode ver na Figura 3.

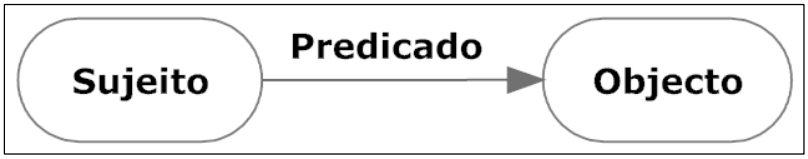


Figura 3: Representação genérica de um *triple* RDF, baseado em [20]

Os nós do grafo RDF são o sujeito e o objecto, enquanto que o arco que liga ambos os nós é o predicado. É importante salientar que o arco correspondente ao predicado aponta sempre para o objecto. Existem duas formas de serializar grafos RDF sendo elas *Resource Description Framework/Extensible Markup Language* (RDF/XML) [22] e *Notation 3* (N3) [23]. Contudo, uma vez que a definição das mesmas está fora do âmbito desta dissertação, optou-se apenas por se fazer referência a ambas.

Atentando na Figura 3, é possível compreender que cada triple RDF representa uma relação entre o sujeito e o objecto, indicada pelo predicado. Por conseguinte, para se conhecer o significado de um grafo RDF na sua totalidade é necessário determinar o significado de cada um dos triples que o compõem e, posteriormente, determinar a sua conjunção lógica.

De acordo com a especificação do RDF descrita em [20], os componentes dos *triples* devem ser compostos da seguinte forma: o sujeito pode ser um URI ou um nó em branco; o predicado pode ser apenas um URI; o objecto pode ser um URI, um nó em branco ou um literal. Uma vez que o URI já foi definido no ponto anterior desta dissertação, resta apenas definir nó em

branco e literal. Assim sendo, no contexto RDF um literal é uma *string Unicode*, enquanto que um nó em branco não tem uma estrutura interna definida, mas deve ser distinto de um literal ou de um URI.

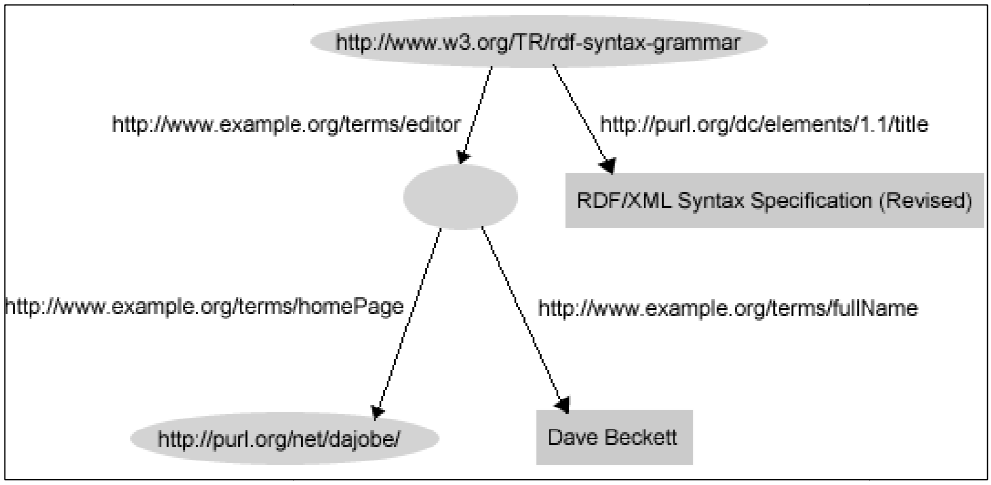


Figura 4: Representação de um grafo RDF com os distintos tipos de nós [21]

Na Figura 4 é possível ver, a título de exemplo, a representação de um grafo RDF em que cada um dos seus componentes assume todas as formas possíveis: URIs, nós em branco e literais. Começa-se por definir o sujeito principal como sendo o documento "http://www.w3.org/TR/rdf-syntax-grammar", que é representado por um URI. A este sujeito correspondem dois predicados distintos, ambos representados também por URIs. O predicado mais à direita representa uma relação de "o título do sujeito é" entre o sujeito e o objecto, em que o objecto é representado pelo literal "RDF/XML Syntax Specification (Revised)" que é o título do documento. Já o predicado mais à esquerda representa uma relação de "o editor do sujeito é" entre o sujeito e o objecto, em que o objecto é representado por um nó em branco. Neste caso, o nó em branco diz respeito a uma pessoa, como é possível verificar através dos predicados "a página pessoal do sujeito é" e "o nome completo do sujeito é", ambos representados por URIs. Os objectos destas relações são, na primeira, mais à esquerda, a página pessoal da pessoa "http://purl.org/net/dajobe/" sendo representada por um URI , enquanto que na segunda, mais a direita, é o nome completo da pessoa "Dave Beckett" representado por um literal. Desta forma, fica descrito o documento "http://www.w3.org/TR/rdf-syntax-grammar" através de um grafo RDF.

No contexto LOD, a interligação de distintos *triples* RDF provenientes de fontes diversas vai permitir a criação de uma nuvem de conhecimento como a da Figura 1.

2.2.3 SPARQL *Query Language*

Para permitir a execução de *queries* sobre grafos RDF foi criada a *SPARQL Protocol and RDF Query Language* (SPARQL). Esta *query language* é uma recomendação oficial do W3C [24], estando directamente relacionada com o SPARQL *Protocol* [25], não devendo no entanto, ser confundida com o mesmo. O SPARQL *Protocol* tem por objectivo a transmissão de *queries* desde o cliente onde é inserida a *query* até ao processador das *queries*. Já a SPARQL *Query Language* é a linguagem utilizada para fazer essas mesmas *queries*, sendo esta que aqui será abordada.

A *SPARQL Query Language* permite efectuar *queries* sobre distintas fontes de dados RDF. Estas *queries* podem ser efectuadas sobre dados nativamente formatados em RDF ou que sejam visualizados como RDF por meio de *middleware*. No âmbito de LOD é a partir destas *queries* que é possível obter conhecimento contido nas colecções de dados, que podem ou não estar formatadas em RDF.

Segundo Arenas e Pérez [26], uma *query* nesta linguagem é composta por um corpo e por um cabeçalho. O corpo da *query* é uma expressão que verifica a correspondência de padrões com um grafo RDF, tendo este corpo o formato de um ou vários *triples* RDF com as respectivas qualidades de sujeito, predicado e objecto. Quanto ao cabeçalho, é uma expressão que indica como construir a resposta à *query*.

O cabeçalho de uma *query* pode ser construído de quatro formas distintas, sendo elas [24]:

- **"Select"**, que devolve todos os resultados da *query* numa forma tabelar;
- **"Construct"**, que devolve os resultados da *query* formatados como um grafo RDF;
- **"Ask"**, que devolve um valor booleano que indica se a *query* tem ou não resultados;
- **"Describe"**, que devolve um grafo RDF que descreve o recurso ao qual se efectuou a *query*.

O corpo da *query* é opcional apenas nas *queries* em que o cabeçalho seja construído recorrendo à forma "Describe".

```
SELECT ?Author
WHERE {
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:isPartOf   ?InPods .
  ?InPods     sw:series      conf:pods .
  ?DbpPerson  owl:sameAs     ?Author .
  ?DbpPerson  dbo:birthPlace dbpedia:Oklahoma .  }
```

Figura 5: Exemplo de uma *query* SPARQL [26]

Para avaliar uma *query* são necessários dois passos distintos [26]. Em primeiro lugar são atribuídos valores às variáveis que se encontram no corpo da *query*, que no exemplo da Figura 5 são as palavras precedidas pelo carácter "?", onde o corpo da *query* corresponde ao que está entre chavetas. Os valores são atribuídos às variáveis por forma a que os *triples* no corpo da *query* correspondam a *triples* no grafo RDF sobre o qual se está a executar a *query*. Por fim, os valores atribuídos às variáveis são processados construindo-se assim o resultado que, neste caso, será devolvido em forma tabelar, uma vez que no cabeçalho da *query* foi utilizado o "Select". Assim sendo, para esta *query* seria devolvida uma tabela contendo todos os autores incluídos no grafo RDF sobre o qual a mesma é executada e que respeitassem as condições impostas pelos *triples* contidos no corpo desta. Mais concretamente, uma tabela contendo todos os autores criadores ("dc:creator") de um *paper* ("?Paper") que faça parte do ("dct:isPartOf") *Symposium on Principles of Database Systems* ("?inPods") de um determinado ano, sendo que esse simpósio deve pertencer às séries "conf:pods". Os autores resultantes da query devem também existir ("owl:sameAs") nos *triples* disponibilizados pela DBpedia [27], sendo que o seu local de nascimento ("dbo:birthPlace") deve ser *Oklahoma* ("dbpedia:Oklahoma").

É utilizando *queries* SPARQL que se pode obter conhecimento de forma automática a partir de grafos RDF como o da Figura 4.

2.3 Exemplos de portais *Open Data*

Neste ponto descrevem-se três portais OD distintos. A razão pela qual se escolheram estes portais específicos deve-se ao facto de cada um deles adoptar um método diferente para disponibilizar os dados aos seus utilizadores. O primeiro portal converte as suas colecções de dados num formato único, eliminando assim a heterogeneidade das mesmas, e permitindo a execução de *queries* sobre estas. Por sua vez, o segundo disponibiliza as suas colecções de dados em diferentes formatos *machine-readable*, enquanto que o último utiliza um vector comum a todas as colecções de dados para conseguir interligar as mesmas de forma lógica.

2.3.1 *open.data.al*

O projecto *Open Data Albania* [28] é um projecto que pretende implementar os princípios de *Open Government Data* na Albânia. Segundo Brahaj et al. [29] o objectivo deste projecto é aumentar a abertura dos dados disponibilizados pelas diversas instituições públicas deste país, e encontrar um meio de representar a informação contida nesses dados de uma forma compreensível,

motivando o debate público sobre os mesmos. Para isso foi implementado o portal *open.data.al* que permite o acesso a todos esses dados.

Uma vez que os dados disponibilizados pelas instituições públicas são bastante heterogéneos quanto ao seu formato e fontes, este projecto teve de encontrar uma estratégia para superar este problema. De acordo com Brahaj et al. [29] foi necessário criar um sistema que fosse capaz de converter os dados em bruto que são disponibilizados pelas diversas instituições em *triples* RDF, permitindo assim a execução de *queries* simples ou complexas sobre estes dados e a sua posterior visualização.

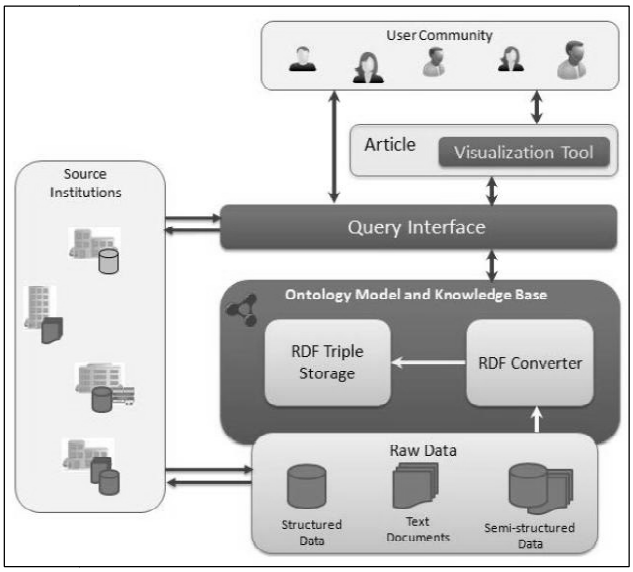


Figura 6: Arquitectura do sistema implementado no âmbito do projecto open.data.al [28]

Na Figura 6 está representada a arquitectura utilizada na implementação deste sistema. Segundo Brahaj et al. [29] a recolha inicial dos dados é efectuada manualmente, devido à grande heterogeneidade dos seus formatos e fontes. Após a recolha dos dados em bruto, procede-se à sua estruturação no formato *Comma-separated values* (CSV), o que resulta na construção de colecções dos mesmos. Posteriormente, estas colecções de dados são convertidas para o formato RDF, sendo que estas são também disponibilizadas aos utilizadores.

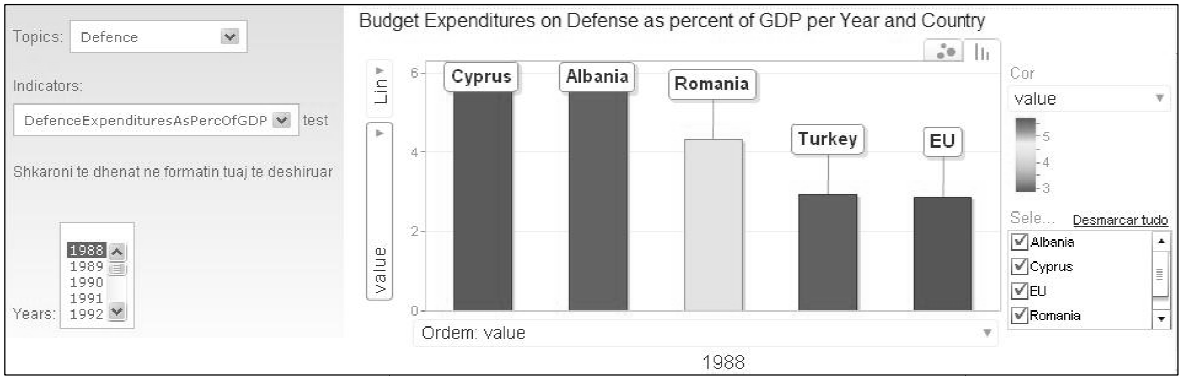


Figura 7: Exemplo de utilização do portal *open.data.al*

Depois de convertidos os dados, é possível executar *queries* simples ou complexas sobre estes, utilizando SPARQL *Query Language*. Conforme se pode ver no lado esquerdo da Figura 7, as *queries* são efectuadas de uma forma simplificada para o utilizador, ou seja, existe uma camada de *software* que abstrai a SPARQL *Query Language* permitindo que as *queries* sejam construídas por meio de ferramentas de utilização simples, como é o caso de *drop down lists* ou similares. Desta forma não é necessário o utilizador ter conhecimentos da linguagem de *query*. Depois de executadas as *queries*, os resultados destas podem ser visualizados pelos utilizadores de uma forma gráfica, como mostra a Figura 7 do lado direito, uma vez que é utilizada a *Google Visualization API* que permite uma visualização dinâmica, ou seja, o resultado das *queries* é imediatamente mostrado ao utilizador por meio de gráficos ou tabelas.

2.3.2 *data.gov*

O portal *data.gov* [4] foi criado com o intuito de aumentar o conhecimento dos cidadãos acerca do governo dos Estados Unidos da América e suas instituições. Para que isto seja possível, este portal disponibiliza colecções de dados em formatos *machine-readable*, nomeadamente CSV, *Extensible Markup Language* (XML), *Keyhole Markup Language* (KML), *JavaScript Object Notation* (JSON), entre outros, sendo estes dados gerados pelo ramo executivo do governo federal deste país. Simplificando, este site pode ser interpretado como um repositório de colecções de dados, das quais se podem ver alguns exemplos na Figura 8.

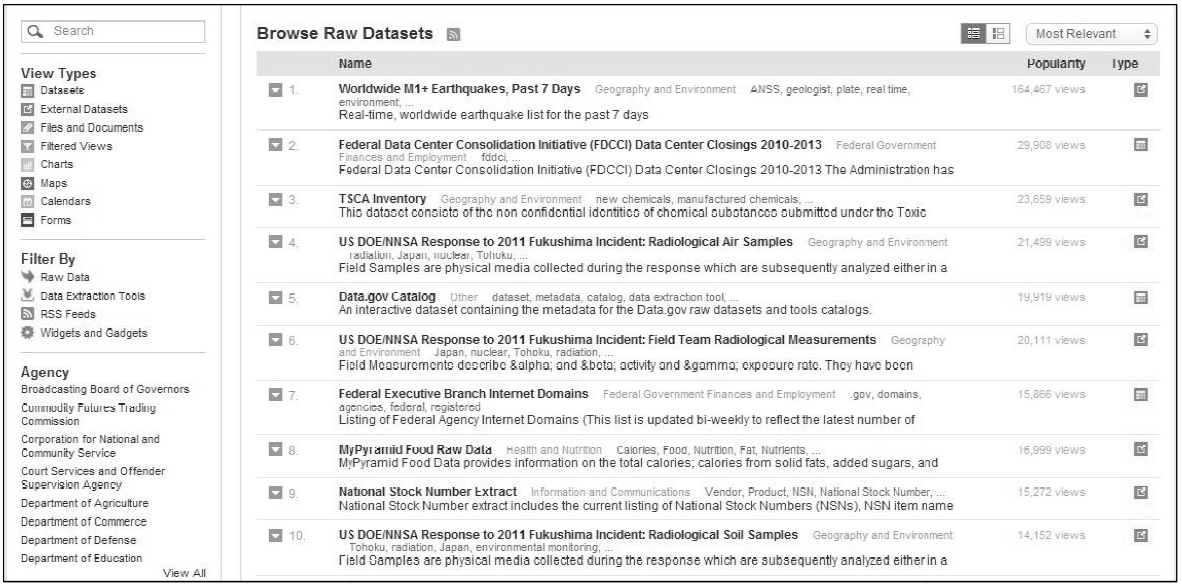


Figura 8: Exemplo de utilização do portal *data.gov*

De acordo com Brahaj et al. [29], está a decorrer um esforço para traduzir todos estes dados para RDF, permitindo assim a este portal abraçar os princípios de LOD.

Este portal disponibiliza aos seus utilizadores diversas aplicações, nomeadamente *widgets* ou *feeds Rich Site Summary* (RSS). É também incentivada a redistribuição dos dados disponibilizados por meio de aplicações ou portais desenvolvidos pelos utilizadores, por forma a que um maior número de pessoas tenha acesso à informação neles contida.

O portal *data.gov* permite também a representação dos dados de uma forma interactiva, nomeadamente por meio de mapas geográficos ou tabelas. No entanto, são ainda poucas as colecções para as quais estas representações estão disponíveis.

Existem alguns portais semelhantes a este em diversos países, nomeadamente, *data.gov.uk* [5] do Reino Unido, *data.gov.au* [6] da Comunidade Australiana e *dados.gov.pt* [8] da República Portuguesa.

2.3.3 *pordata.pt*

O portal *pordata.pt* [7] é um projecto de iniciativa privada que tem por objectivo dar a conhecer dados estatísticos sobre Portugal e a União Europeia aos seus cidadãos. Estes dados dizem respeito às mais diversas áreas da sociedade, nomeadamente à saúde, economia, justiça, entre outros. O que diferencia este portal dos outros já referidos, é que todos estes dados têm um vector comum, o tempo, sendo que assim é possível efectuar o cruzamento de dados respeitantes a áreas completamente distintas, como por exemplo, comparar o número de doutorados nas

universidades com o consumo de água *per capita* num determinado ano, como mostra a Figura 9. Desta forma, o utilizador pode descobrir nos dados informação que habitualmente lhe estaria oculta.

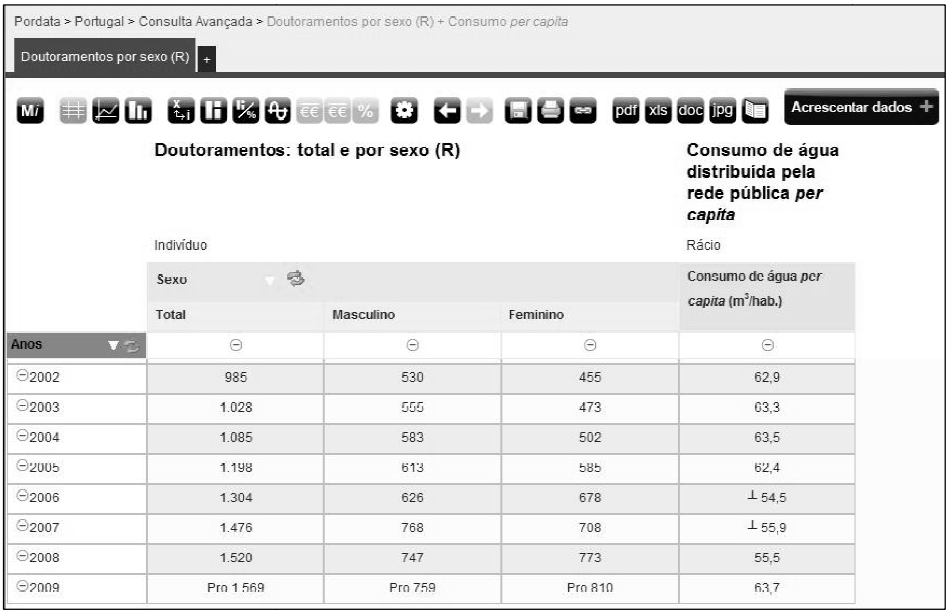


Figura 9: Exemplo de utilização do portal *pordata.pt*

Neste portal é também possível criar visualizações dinâmicas, tanto por meio de gráficos como de tabelas, dos dados seleccionados. Além disso, o utilizador pode exportar os dados, bem como as respectivas visualizações para os formatos *Portable Document Format* (PDF), *Microsoft Word file format* (DOC), *Microsoft Excel file format* (XLS) e *Joint Photographic Experts Group* (JPEG). No entanto, este portal não disponibiliza os dados em formato RDF.

Segundo informações disponibilizadas na ficha técnica do portal, são utilizadas as seguintes tecnologias: *Outsystems*, *Google Motions Charts*, *Fusion Charts*, *GeoPortal* e *Real Time Statistics.org*.

3 Interacções de dados

Qualquer aplicação informática, seja qual for o seu âmbito, tem por objectivo o processamento de dados. Por vezes, os dados que se pretendem processar, encontram-se dispersos pelas distintas aplicações informáticas de uma determinada instituição ou até mesmo de instituições diversas. Por esta razão, a possibilidade de permitir que aplicações diferentes interajam entre si de uma forma eficaz, abstraindo-se da linguagem de programação ou do hardware de cada uma delas, é muito apetecível.

Foi com o intuito de permitir estas interacções entre distintas aplicações que surgiram os *Web Services*. O W3C definiu um *Web Service* [30] como sendo:

"Aplicação de software identificada por um URI, cujos interfaces possam ser definidos, descritos e descobertos como artefactos XML. Um Web Service suporta interacções directas com outros agentes de software, utilizando mensagens baseadas em XML, sendo estas trocadas por meio de protocolos baseados na internet."

Tendo em conta esta definição, facilmente se deduzem algumas vantagens da utilização de *Web Services* [31], nomeadamente na integração dos diversos componentes de uma aplicação, pois, ao serem desenvolvidos como *Web Services*, permitem que de cada vez que ocorra uma alteração no funcionamento interno de cada componente, esta não interfira ou quebre as interacções dos mesmos. Para além disso, os *Web Services* permitem também a atribuição de um interface a aplicações antigas que não foram desenvolvidas a pensar na interoperabilidade. Desta forma, consegue-se que estas passem a integrar este ambiente de interacção entre aplicações, sem que seja necessário alterar o seu funcionamento original.

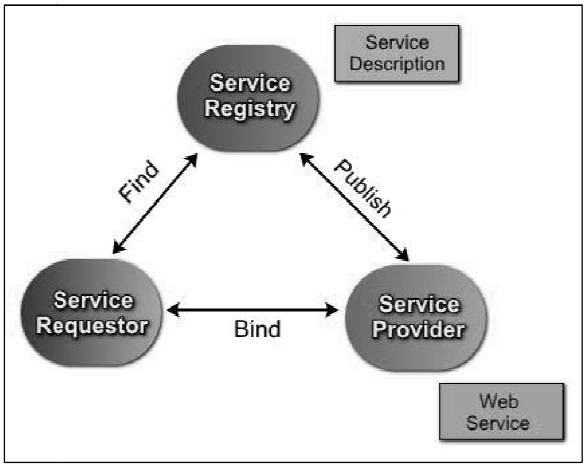


Figura 10: Arquitectura dos *Web Services*[32]

A arquitectura mais usual dos *Web Services* está representada na Figura 10. Nesta figura é possível verificar a existência de três entidades distintas. Em primeiro lugar, o "Service Provider", que é a entidade onde o *Web Service* está alojado, deve publicar, junto de um "Service Registry", esse mesmo serviço. Este "Service Registry" ou registo *Universal Description, Discovery and Integration* (UDDI) pode ser privado (para uso apenas numa *intranet*), pode ser exposto numa *extranet* (acessível apenas a entidades seleccionadas), ou então pode ser público (disponibilizado na *internet*) [33]. Uma forma de fazer esta publicação é disponibilizar ao "Service Registry" informações que descrevam o interface do *Web Service* num formato *machine-readable*. Um modo de o conseguir é utilizando a *Web Service Description Language* (WSDL), a qual permite a definição de estruturas gramaticais em XML capazes de descrever o *Web Service* [34]. Assim sendo, o "Service Registry" pode ser visto como um *broker* de serviços, uma vez que quando um "Service Requestor" pretenda descobrir serviços para os poder utilizar, contacta o "Service Registry" que lhe disponibiliza a descrição dos serviços que conhece em WSDL. Posteriormente, o "Service Requestor", depois de conhecer os serviços disponíveis, pode comunicar directamente com o "Service Provider" que pretenda, para assim consumir os dados disponibilizados pelo serviço. A comunicação entre estas duas entidades pode ser efectuada com recurso ao protocolo de transferência de dados *Simple Object Access Protocol* (SOAP), que será descrito mais adiante neste capítulo, bem como recorrendo ao estilo arquitectural para sistemas distribuídos *Representational State Transfer* (REST).

3.1 Transferência de dados - SOAP

3.1.1 SOAP

O protocolo SOAP que, a partir da sua ultima revisão, deixou de ser um acrónimo para *Simple Object Access Protocol*, é um dos protocolos de transferência de dados mais utilizados actualmente, sendo extensivamente usado na implementação de *Web Services*.

O W3C definiu SOAP como sendo [35]:

"um protocolo que tem por objectivo a transferência de informação estruturada num ambiente distribuído e descentralizado. Utiliza tecnologias XML para definir um framework que permita a construção de mensagens que podem ser trocadas por meio de diversos protocolos subjacentes. Este framework foi construído por forma a ser independente de qualquer modelo de programação ou especificidades semânticas da implementação."

Tal como a sua definição indica, este protocolo serve-se de XML na formatação das suas mensagens, o que as torna bastante compreensíveis quer para as aplicações que delas se servem, quer para humanos [36]. Desta forma, o protocolo SOAP pode ser utilizado independentemente da linguagem de programação em uso nos pontos opostos da comunicação, uma vez que estes apenas necessitam compreender o XML das mensagens, o que é uma tarefa simples, pois XML é um padrão aberto existindo *parsers* para a maioria das linguagens de programação utilizadas actualmente. Assim sendo, SOAP é um protocolo indicado para ambientes heterogéneos quando se pretendem interligar aplicações com diferentes características [37].

Atentando ainda na sua definição, pode também concluir-se que, o protocolo SOAP não está restringido na forma como é feita a transmissão das suas mensagens, ou seja, pode servir-se de diversos protocolos da camada de aplicação do *Internet Protocol Suite* (IPS), nomeadamente o HTTP, o *Hypertext Transfer Protocol Secure* (HTTPS) e o *Simple Mail Transfer Protocol* (SMTP) [38].

Os protocolos mais utilizados pelo SOAP na sua transmissão de dados, permitem também que este consiga transpor *firewalls* sem qualquer concessão específica para este efeito, tal como nos diz Albrecht [36], uma vez que estas interpretam o tráfego SOAP como simples pedidos de páginas Web. No entanto, apesar deste aspecto ser interessante do ponto de vista de quem pretende efectuar a transmissão SOAP, pois não tem de se preocupar com eventuais bloqueios, para os responsáveis pela segurança das redes isto pode ser um problema, já que existe a possibilidade de este ser um ponto de entrada para eventuais ataques [36].

As mesmas características que tornam SOAP um protocolo interessante para a transferência de dados, são aquelas que também lhe colocam os principais problemas. Em primeiro lugar, os métodos utilizados para a transmissão das mensagens deste protocolo criam um elevado tráfego na rede pela qual são transmitidas. Em segundo, a latência envolvida neste processo é elevada. No que diz respeito às mensagens, a sua formatação pode ter um elevado custo computacional quando estas são geradas ou é efectuado o *parsing* das mesmas [34].

Existem já alguns estudos, tais como [39] e [40], que concluíram que o protocolo SOAP tem um tempo de resposta significativamente mais elevado quando comparado com tecnologias idênticas, como é o caso, por exemplo, do *Common Object Request Broker Architecture* (CORBA). Para Elfwing et al. [39] esta disparidade temporal é tão relevante, que chegam a afirmar que mesmo uma versão optimizada do protocolo SOAP será sempre mais lenta que o CORBA.

Depois de conhecer as vantagens e desvantagens deste protocolo é importante ficar a conhece-lo de uma forma mais aprofundada, pelo que se optou por abordar, nesta dissertação, os seus detalhes mais relevantes.

Em primeiro lugar, convém clarificar alguns dos conceitos respeitantes a este protocolo, nomeadamente o conceito de nó SOAP [41], que é a entidade responsável por garantir que a troca de mensagens SOAP obedece às regras do protocolo, bem como por aceder aos serviços disponibilizados pelos protocolos subjacentes sobre os quais é feita a transmissão destas mesmas mensagens (HTTP ou SMTP, por exemplo). Assim sendo, é nesta entidade que está contida toda a lógica necessária à transmissão, recepção, processamento e retransmissão das mensagens SOAP. Por sua vez, uma mensagem SOAP, traduz-se numa transmissão unidireccional entre nós SOAP, isto é, desde um nó que transmite a mensagem SOAP (remetente) até ao nó que aceita mensagens SOAP (receptor).

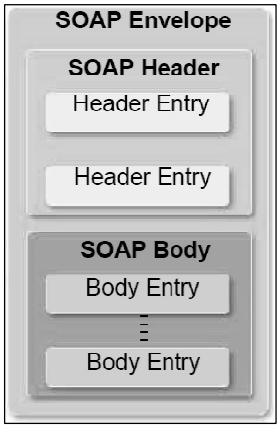


Figura 11: Estrutura de uma mensagem SOAP [38]

Atentando na Figura 11 é possível perceber que as mensagens SOAP têm uma estrutura bem definida, podendo identificar-se três componentes distintos: o elemento raiz "SOAP Envelope", contendo os elementos "SOAP Header" e "SOAP Body". O "SOAP Envelope", sendo o elemento raiz de toda a mensagem [35], é responsável por fornecer informação sobre o *namespace*, bem como contexto de serialização aos elementos e parâmetros da mensagem [34]. O conceito serialização diz respeito ao processo de converter um determinado objecto que se encontra em memória num objecto XML, para que este possa ser transmitido numa mensagem SOAP [34]. Quanto ao "SOAP Header" [41], é um elemento opcional numa mensagem SOAP que, no interior da própria mensagem, permite que seja enviada informação de controlo da mesma. A título de exemplo, essa informação pode relacionar-se com a forma como a mensagem deve ser processada ou pode conter dados sobre o seu emissor e receptor [34]. Mais concretamente, um "SOAP Header" [35] é uma colecção de zero ou mais elementos denominados SOAP *Header blocks* (representados por "Header Entry" na Figura 11), sendo que cada um desses elementos pode ser direccionado a qualquer receptor SOAP que se encontre no caminho da mensagem, ou seja, numa única mensagem diferentes blocos podem dizer respeito a diferentes receptores. Por fim, o

elemento “SOAP Body” [41] é o elemento obrigatório dentro do “SOAP Envelope”, uma vez que nele estão contidos os dados que se pretendem transferir por meio de uma mensagem SOAP. Este elemento compreende uma colecção de zero ou mais sub-elementos [35] (representados na Figura 11 por “Body Entry”) contendo informação direccionada a um receptor final SOAP no caminho percorrido pela mensagem.

```
<?xml version="1.0" ?>
<env:Envelope
xmlns:env="http://www.w3.org/2001/12/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

Figura 12: Exemplo de uma mensagem SOAP [39]

Na Figura 12 está representado um exemplo de uma mensagem SOAP onde é possível visualizar os três elementos descritos anteriormente: “Envelope”, “Header” e “Body”, em que no primeiro é visível informação relativa ao controlo da mensagem e no segundo estão representados os dados que se pretendem transmitir.

É importante salientar que as mensagens SOAP suportam diferentes padrões de funcionamento [34], nomeadamente:

- Mensagens unidireccionais;
- Mensagens *request-response*, em que um nó SOAP faz um *request* de informação a outro determinado nó que, por sua vez, envia essa informação num *response*;
- Mensagens *remote procedure call* (RPC) que, tal como o nome indica, permitem que um determinado nó SOAP evoque procedimentos de forma remota num outro nó, sendo para isso determinante que a mensagem contenha todas as informações necessárias à evocação do procedimento [41];
- Mensagens *peer-to-peer*.

Depois de conhecer alguns conceitos relacionados com o protocolo SOAP, é interessante compreender de que forma se processa a troca de mensagens em maior detalhe.

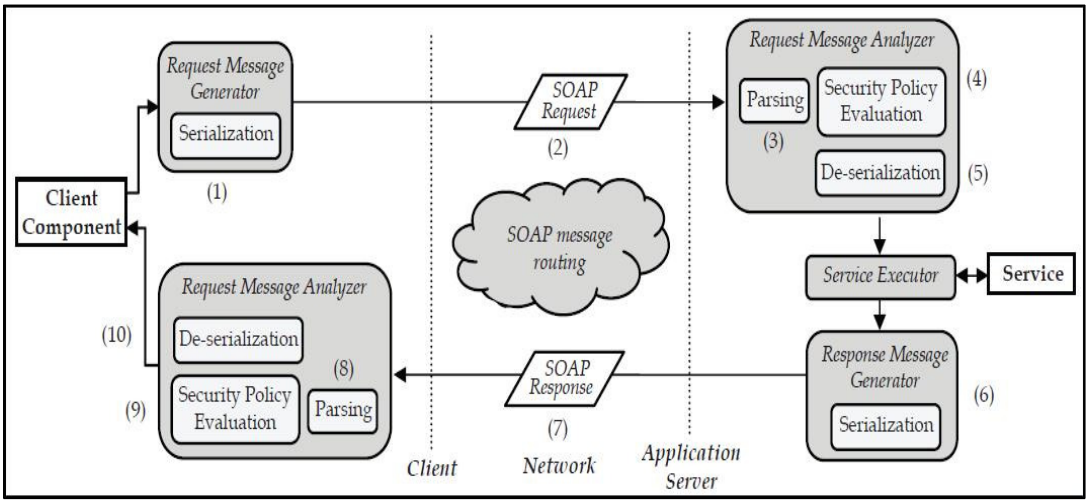


Figura 13: Exemplo de um cenário de troca de mensagens SOAP [34]

Na Figura 13 está representada uma troca de mensagens SOAP. Os passos constituintes desta troca de mensagens foram descritos em [34], resumindo-se aqui essa descrição.

Em primeiro lugar é criada uma mensagem SOAP *request* no nó iniciador da transmissão, que na Figura 13 é representado pelo "Client Component". Ao ser criada esta mensagem é necessário serializar os dados que se pretendem transmitir. Seguidamente, a mensagem é enviada para o nó SOAP de destino, utilizando um protocolo de transferência de dados da camada de aplicação do IPS. Ao chegar ao nó de destino é feito *parsing* à mensagem recebida com o intuito de a validar e efectuar uma análise léxica. Caso este passo tenha sucesso, a mensagem é posteriormente submetida a uma avaliação por forma a identificar as partes da mensagem que têm restrições de segurança, seguindo-se a desserialização da mesma para que esta possa ser processada pelo nó de destino do SOAP *request* representado na Figura 13 pelo elemento "Service". Logo que o nó destino disponha de uma resposta para o *request* que lhe foi dirigido, é criada uma mensagem SOAP *response* que sofrerá as mesmas transformações que a SOAP *request*, mas que seguirá o percurso em sentido inverso, ficando assim concluída a troca de mensagens SOAP.

3.2 Modelos de comunicação de dados

Para além dos *Web Services*, existem outros meios para que uma determinada aplicação possa interagir com dados disponibilizados por outras aplicações distintas. Para que isso seja possível, é necessário encontrar modelos de comunicação de dados, por forma a que estes sejam perceptíveis às diversas aplicações. Neste capítulo serão abordados dois modelos de comunicação de dados diferentes: JSON e *Direct database access*.

3.2.1 JSON

O JSON é um formato de texto para a serialização e comunicação de dados estruturados [42]. Este formato é baseado nos objectos da linguagem de programação *JavaScript*, sendo, no entanto, completamente independente desta ou de qualquer outra linguagem de programação [43]. Apesar desta independência, as estruturas base do JSON podem ser consideradas universais à grande maioria das linguagens de programação modernas, tornando-o num formato ideal para a comunicação de dados [43]. Estas estruturas são as seguintes [43]:

- Uma colecção de pares nome/valor, que nas distintas linguagens de programação pode tomar a forma de objecto, estrutura, *hash table*, etc.;
- Uma lista ordenada de valores, que é habitualmente representada por *array*, vector, lista, etc., nas linguagens de programação modernas.

Ainda que baseado em aspectos da linguagem de programação *JavaScript* [44], JSON não é uma linguagem de programação em si mesmo, pois não é passível de ser compilado e executado. É apenas uma forma de conter dados, sendo completamente neutro à semântica destes.

Os dados formatados em JSON podem ser facilmente analisados e processados utilizando *JavaScript*, uma vez que se baseiam nesta linguagem. Por esta razão, é o formato indicado para a comunicação de dados em aplicações *Web* que se sirvam de *Asynchronous JavaScript and XML* (AJAX) [45], pois, ao contrário de XML [46], cujos objectos são analisados como *Document Object Model* (DOM), sendo por isso necessária a inclusão de bibliotecas extra para esse efeito [47], JSON é visto pela aplicação como um simples *array* de *strings*. Uma vez que se trata de um formato de dados independente de qualquer linguagem de programação, JSON pode ser utilizado em todos os cenários em que seja necessário trocar ou armazenar informação em formato de texto [45], como por exemplo em *Web Services* [48] que, actualmente, muitas empresas começam a disponibilizar neste formato [49]. Também no contexto dos dispositivos móveis, por ser um formato bastante leve, o JSON está a ser amplamente utilizado [49].

Uma característica importante do JSON [47], para além das outras já apontadas, é que este foi construído para ser de fácil utilização por computadores, mas também para permitir a sua compreensão por humanos. A título de exemplo, na Figura 14 estão representadas algumas informações sobre a cidade de Aveiro formatadas em JSON. Se esta informação fosse facultada por um indivíduo a outro, facilmente o receptor compreenderia que o emissor já havia visitado (“Visitada”) uma cidade denominada Aveiro (“NomeCidade”), com 78463 habitantes (“NumeroHabitantes”), composta pelas freguesias (“Freguesias”) denominadas Glória, Santa Joana

e Vera Cruz ("NomeFreguesia"), tendo cada uma delas a população de 9099, 8094 e 9657 habitantes ("População"), respectivamente.

```
{
  "NomeCidade": "Aveiro",
  "NumeroHabitantes": 78463,
  "Visitada": true,
  "Freguesias": [
                  { "NomeFreguesia": "Glória", "População": 9099},
                  { "NomeFreguesia": "Santa Joana", "População": 8094},
                  { "NomeFreguesia": "Vera Cruz", "População": 9657}
                ]
}
```

Figura 14: Exemplo de dados formatados em JSON

Não obstante as vantagens que o formato JSON encerra em si, são-lhe apontadas algumas falhas que é importante referir. Em primeiro lugar, os críticos deste formato afirmam que este não tem suporte para *namespaces* [47], enquanto os seus defensores respondem que cada objecto é um *namespace* [50] servindo-se do contexto para evitar ambiguidades. Uma outra crítica também apontada a este formato é a inexistência de validação de input [47], cabendo, por isso, essa tarefa a cada aplicação que dele se sirva. Por fim, os seus críticos afirmam ainda que JSON não é um formato extensível [47], ou seja, caso seja necessário adicionar novos elementos a uma representação de dados formatada em JSON já existente, podem surgir problemas.

Para compreender qual o formato que melhor se adequa à comunicação de dados, foi realizado um estudo que comparava as performances de JSON e XML neste âmbito [47], tendo o estudo concluído que JSON é mais rápido e usa menos recursos do que XML. Talvez por esta razão, e por todas as suas outras vantagens, JSON é, actualmente, o formato dominante na comunicação de dados entre aplicações [48].

Depois de perceber as razões que tornam o formato JSON interessante, importa conhecer em pormenor quais os tipos que os dados podem tomar neste formato. Podem ser representados [42] em JSON os tipos primitivos *string*, número, booleano e *null*. Quanto a tipos estruturados, podem representar-se objectos e *arrays*, já mencionados anteriormente nesta dissertação.

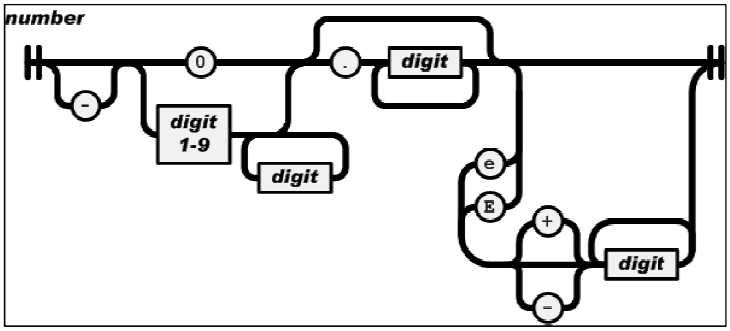


Figura 15: Diagrama representando a sintaxe do tipo número de JSON [43]

O tipo primitivo "número" de JSON permite a representação de qualquer número, seja ele inteiro ou de virgula flutuante, tal como são representados nas linguagens de programação C ou JAVA, com a excepção de que JSON não suporta números no formato octal ou hexadecimal [43]. Na Figura 15 está representada, por meio de um diagrama, a sintaxe permitida pelo JSON para a representação de números.

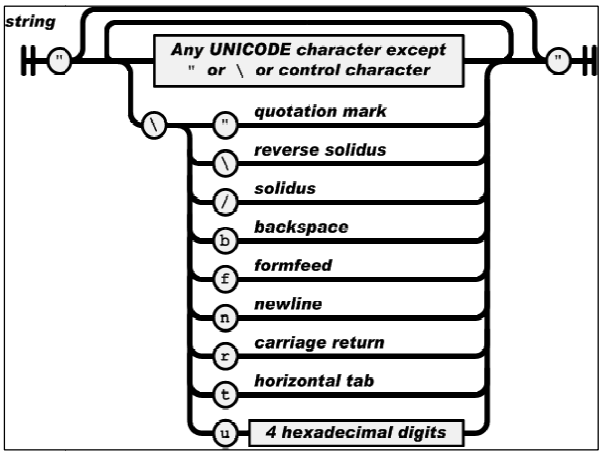


Figura 16: Diagrama representando a sintaxe do tipo "string" de JSON [43]

Um outro tipo primitivo deste formato é o "string". Uma *string* [43], em JSON, pode ser representada por uma sequência de zero ou mais caracteres *Unicode* delimitada por aspas. Caso se pretendam representar caracteres especiais dentro da *string* é possível fazê-lo, utilizando para isso o carácter *backslash* antes do carácter especial. Podem também representar-se caracteres únicos, equivalentes ao *char* da linguagem C, por meio de *strings* com um só carácter. Na Figura 16 está representada a sintaxe permitida pelo JSON para a representação de *strings*.

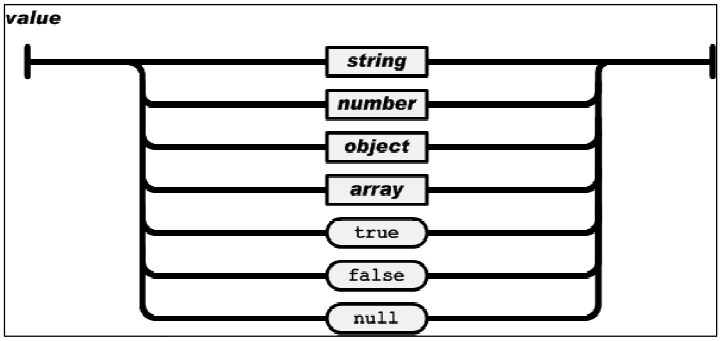


Figura 17: Diagrama representando os diversos tipos JSON [43]

A Figura 17 representa todos os tipos possíveis em JSON sendo-lhes, atribuído o nome de valor. Com esta atribuição pretende-se criar uma entidade abstracta que represente qualquer tipo em JSON, permitindo assim efectuar o encadeamento desta nos tipos estruturados objecto e *array*.

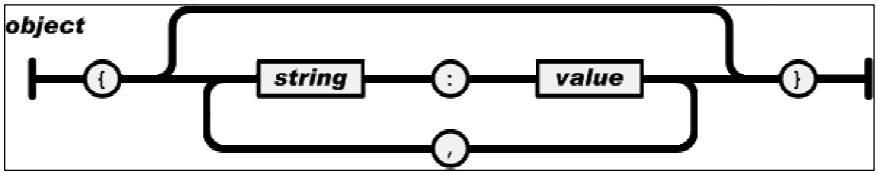


Figura 18: Diagrama representando o tipo estruturado objecto [43]

Tal como já foi referido anteriormente, um objecto em JSON é uma colecção desordenada de pares nome/valor [43]. Conforme se pode observar pela Figura 18, estes objectos são delimitados por chavetas, sendo cada nome (uma "string") separado do seu respectivo valor pelo carácter dois pontos. A separação de cada par nome/valor é feita por meio de uma vírgula. Tendo em conta que o valor é uma abstracção que representa qualquer tipo em JSON, facilmente se compreende pela análise da Figura 18 a possibilidade de encadear os diversos tipos, estruturados ou não, dentro de um objecto, ou seja, objectos dentro de objectos e *arrays* dentro de objectos.

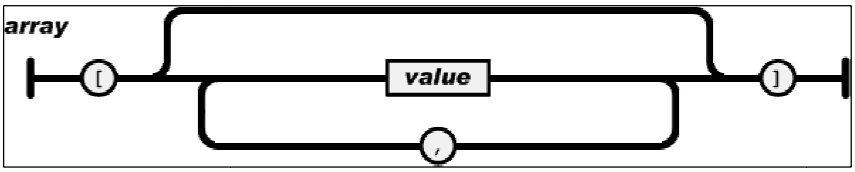


Figura 19: Diagrama representando o tipo estructurado *array* [43]

Finalmente, o tipo estruturado *array* [43], cuja sintaxe está representada na Figura 19, é uma colecção de valores delimitados por parêntesis rectos e separados por vírgulas, sendo que esses valores podem tomar a forma de todos os tipos existentes em JSON, tornando assim possível a criação de *arrays* de *arrays* ou *arrays* de objectos, por exemplo.

3.2.2 Direct database access

Uma forma das aplicações comunicarem dados entre si é por meio de bases de dados. Diferentes aplicações podem servir-se da mesma base de dados, sem que para isso tenham de conhecer os pormenores umas das outras. Necessitam apenas conhecer o esquema da base de dados para com ela poderem interagir. Por outras palavras, dados criados e armazenados por uma aplicação em determinada base de dados podem ser consumidos por outra aplicação distinta. Além disso, uma aplicação pode necessitar de dados armazenados em bases de dados distintas, cujo sistema de gestão de bases de dados (SGBD) é também diferente em cada uma delas, bem como o sistema operativo (SO) sobre o qual corre o SGBD. Segundo Sarinder et al. [51], o processo de efectuar uma *query* sobre distintas bases de dados e devolver os resultados a uma determinada aplicação cria algumas dificuldades, sendo que estas se tornam ainda mais complexas caso as bases de dados tenham sido desenvolvidas em distintos formatos ou plataformas.

Ao processo de uma aplicação comunicar directamente com uma base de dados pode dar-se o nome de *direct database access*, que em Português significa, literalmente, acesso directo à base de dados.

Uma maneira de efectuar a comunicação entre uma aplicação e uma base de dados é por meio de uma *Call Level Interface* (CLI). Uma CLI [52] é uma *Application Programming Interface* (API) que tem por objectivo facilitar a integração entre *software* que pretende aceder a uma base de dados e a própria base de dados. Apesar de aceitar instruções *Structured Query Language* (SQL) geradas dinamicamente [53], uma CLI não faz qualquer tipo de verificação de sintaxe ou de tipos, pelo que alguma incorrecção existente nas instruções só será detectada aquando da execução das mesmas na base de dados.

Ainda que a maioria dos SGBD providenciem uma CLI para que a eles se possa aceder, por vezes é preferível isolar a aplicação do SGBD, permitindo assim uma maior independência sem que, caso se queira mudar de SGBD, seja necessário efectuar alterações [54]. Uma forma de conseguir esta independência é recorrendo a *Open Database Connectivity* (ODBC) [54].

Na década de noventa, começou a aparecer um elevado número de SGBD, o que despoletou a necessidade de criar algo que permitisse às aplicações comunicar com distintos SGBD através de uma interface comum [55]. Desta forma, foi desenvolvida a interface ODBC que define um conjunto de funções baseadas nas especificações da CLI do *SQL Access Group* e do consórcio X/*Open* [55]. Esta interface, sendo independente do SGBD, oferece consistência e portabilidade no acesso das aplicações à base de dados, pois a comunicação é feita utilizando a linguagem de programação C, não sendo necessário à aplicação conhecer os pormenores do SGBD [56].

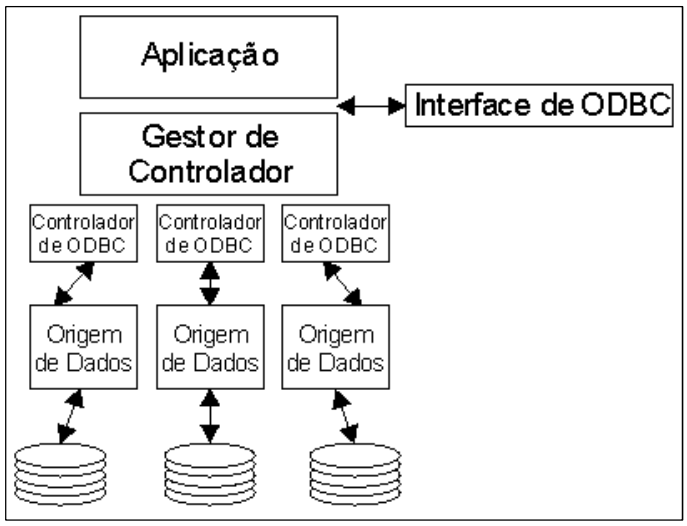


Figura 20: Diagrama representando os distintos componentes de ODBC [57]

Conforme se pode ver na Figura 20, são quatro os componentes principais que interagem no contexto ODBC [57] [55]. O componente que inicia todo o processo, a aplicação, chama as funções ODBC de que necessita para executar instruções SQL, com vista a conseguir obter dados a partir de um ou mais *data sources*. Já o gestor de controlador é responsável, tal como o seu nome indica, por gerir os controladores ODBC que podem ser úteis a uma dada aplicação, ou seja, este processa as chamadas de funções ODBC efectuadas pela aplicação e reencaminha as instruções para o controlador correcto. Por sua vez, cabe aos controladores ODBC abrir e fechar a ligação para os *data sources* respectivos ou, como estão representados na Figura 20, "origem de dados", bem como enviar as instruções SQL para estes por forma a serem executadas. São muitos os controladores ODBC disponíveis para os mais diversos SGBD, mesmo operando em sistemas operativos distintos, nomeadamente *Microsoft SQL Server*, *DB2*, *PostgreSQL*, *MySQL*, *Sybase*, *Oracle* e *Pervasive SQL* ou até para *Microsoft Access* e *FileMaker* [54]. Por fim, o componente "origem dos dados" (Figura 20), é onde o SGBD é contactado pelo controlador ODBC e lhe são facultadas as instruções SQL que este irá executar por forma a extrair dados da base de dados da qual é responsável, enviando os mesmos de volta ao controlador ODBC, que os faz seguir de volta para a aplicação que os solicitou.

Para além de ODBC, existem outras interfaces que têm um objectivo semelhante, nomeadamente JDBC e *Perl Database Interface* que em lugar de disponibilizarem funcionalidades compatíveis com a linguagem de programação C, facultam essas funcionalidades nas linguagens de programação JAVA e *Perl*, respectivamente.

4 Visualização de dados

O valor dos dados não está em si mesmo, pois são apenas letras ou números, mas na informação que deles se pode retirar logo que o seu significado se clarifique. Uma forma de compreender o significado dos dados é por meio de visualizações. Para Yi et al. [58], as visualizações têm por objectivo servir-se das capacidades perceptivas visuais dos humanos, conseguindo assim identificar tendências, padrões e ocorrências estranhas num determinado conjunto de dados. Por esta razão, importa conhecer de que forma o conhecimento é adquirido por meio destas visualizações.

Foi com o intuito de compreender de que maneira o conhecimento é extraído das visualizações de dados, que Yi et al. [58] procuraram identificar os processos responsáveis por este fenómeno, tendo concluído que existem quatro procedimentos distintos e independentes que, quando combinados entre si ou utilizados em todo o seu conjunto, permitem a compreensão dos dados representados nas visualizações. Em primeiro lugar, concluíram que o fornecimento de um panorama geral dos dados na visualização permite que o utilizador fique com uma ideia abrangente dos mesmos, o que, apesar de muitas vezes não lhe permitir extrair imediatamente conhecimento, permite que este identifique quais as áreas na visualização que pretende explorar melhor. Em segundo lugar, apontam a possibilidade de permitir o ajustamento da visualização como sendo um outro processo de adquirir conhecimento a partir da mesma, já que, ao ajustar o nível de abstracção e extensão dos dados, o utilizador consegue compreender melhor os mesmos, uma vez que lhe é permitido explorar um subconjunto de dados para ele mais interessante do que outros, bem como agrupá-los por forma a abstrair-se do conjunto geral destes. O terceiro processo que identificam como forma de compreender os dados através de visualizações dos mesmos é a detecção de padrões. Este processo diz respeito à análise de tendências, frequências ou estruturas nos dados que permitam detectar padrões que mostrem ao utilizador a informação que ele procurava, bem como informação inesperada que motivará nova busca de conhecimento. Por fim, a correspondência com o modelo mental é outro processo que permite a extracção de informação a partir de visualizações de dados, uma vez que estas facilitam a aproximação entre os dados e a ideia que o utilizador tem sobre os mesmos, proporcionando, desta forma, uma maneira mais simples de se estabelecer uma relação entre os dados e os fenómenos para os quais se procuram informações.

Para além dos processos descritos anteriormente, Yi et al. [58] acreditam que um dos factores mais importantes para a extracção de conhecimento a partir de visualizações de dados é a afinidade do utilizador com os mesmos, sendo que, uma forma de conseguir essa afinidade, é permitir a interactividade do utilizador com as visualizações.

Uma vez que hoje em dia se verifica que são muitas as pessoas que se servem das redes sociais como fontes de informação, bem como forma de trocar ideias, conseguir que estas pessoas interajam na utilização de ferramentas de visualização de dados preparadas para estas redes, pode trazer muitas vantagens na descoberta de conhecimento. Esta ideia é defendida por Macdonald et al. [59], que afirmam que a visualização de dados pode ser vista como um meio de comunicação social de conhecimento, ou seja, a análise dos dados representados nas visualizações pode passar a ser uma tarefa colectiva em lugar de ser uma tarefa individual, em que todos os participantes vão colaborando na busca do conhecimento. A forma como esta colaboração é conseguida é transformando a visualização dos dados num artefacto social [59], artefacto esse a que cada participante pode adicionar a sua opinião sobre aquilo que está representado. Desta forma, todas as opiniões dos diversos participantes vão passar a fazer parte do próprio artefacto, o que poderá revelar conhecimento até então escondido nos dados, uma vez que determinado utilizador pode-se aperceber de algum pormenor que não tenha sido detectado por outro. Por esta razão, para Macdonald et al. [59], no que diz respeito às visualizações de dados, a ênfase não deve ser colocada na parte estética das mesmas, mas sim em permitir aos utilizadores interagirem por meio do artefacto social que são as visualizações, bem como em permitir a manipulação e controlo das mesmas por parte destes.

Um outro aspecto a ter em conta no âmbito das visualizações de dados é apontado por Hauser [60] que afirma que:

"Devido ao aumento contínuo dos dados disponíveis para visualização, a questão de o que mostrar (ou o que não mostrar) torna-se crescentemente importante quando comparada com a questão de como visualizar determinados dados."

Tendo em conta esta afirmação, convém conhecer algumas técnicas que visam reduzir a desordem visual das visualizações de dados, uma vez que como as colecções de dados são muito extensas e habitualmente os *displays* onde as visualizações são mostradas têm um tamanho reduzido, o processo de recolha de conhecimento poderá ser dificultado [61]. Assim sendo, Dix e Ellis [61] propõem um conjunto de técnicas com essa finalidade, estando elas divididas em três grupos distintos: aparência, distorção espacial e técnicas temporais. As técnicas contidas no primeiro grupo, tal como o seu nome indica, são técnicas que afectam a aparência das visualizações. Neste grupo, em primeiro lugar, temos a técnica "alterar tamanho dos pontos" que, tal como a técnica "alterar opacidade", tem no seu nome a sua descrição. Também neste grupo estão contidas as técnicas "amostragem" e "filtragem", sendo elas responsáveis por fazer desaparecer itens desnecessários nas visualizações. A diferença entre ambas é que a primeira diz

respeito à selecção de um subconjunto aleatório de dados a mostrar na visualização, enquanto que na segunda essa selecção deve satisfazer um determinado critério. Por fim, ainda neste grupo, está contida a técnica "agrupamento" que consiste em reduzir o número de itens, passando a representar vários pontos ou linhas de uma determinada visualização em apenas um ponto ou uma linha que represente todo o conjunto. Para além destas cinco técnicas, existem alguns aspectos ligados à aparência das visualizações que também permitem reduzir um pouco a desordem nas mesmas, nomeadamente as cores utilizadas ou as texturas.

O grupo das técnicas de distorção espacial [61] contém cinco técnicas que de alguma forma deslocam os pontos ou linhas da sua localização original. Primeiramente surge a técnica "deslocamento de ponto/linha", responsável por ajustar a posição de cada item de dados, ou seja, encontrar a melhor posição para ele na visualização sem perder o seu valor no contexto geral da mesma. Uma outra técnica neste grupo é a "distorção topológica" que permite estender o pano de fundo sobre o qual estão dispostos os dados nas visualizações, de uma forma uniforme ou não uniforme, sendo que os dados adequar-se-ão a este novo pano de fundo. Um exemplo de uma extensão uniforme pode ser o "zoom", que obriga a que toda a visualização sofra um deslocamento; quanto à extensão não uniforme pode ser o "fish-eye" que, utilizando, por exemplo, uma metáfora de uma lupa na visualização, permite que apenas os dados seleccionados sofram um deslocamento. O "preenchimento de espaços" é também uma técnica pertencente a este grupo, consistindo em rearranjar, de modo a que não se sobreponham, os pontos correspondentes aos dados, transformando-os em rectângulos que respeitem uma determinada estrutura, como por exemplo *treemaps*. Ainda neste grupo se inclui também a técnica "pixel-plotting" que transforma cada elemento da colecção de dados a visualizar em píxeis, para desta forma conseguir incluir a maior quantidade de dados possíveis no espaço disponível para a visualização. Por fim, a técnica "reordenamento dimensional" permite que os eixos da visualização sejam reordenados para que esta se torne mais compreensível.

O grupo das técnicas temporais [61] contém apenas a técnica "animação" que, tal como o seu nome indica, permite criar animações com os dados tendo em vista a redução da desordem dos mesmos na visualização.

Com o objectivo de maximizar o conhecimento que pode ser retirado de visualizações de dados, importa conhecer, para além das técnicas já abordadas anteriormente, algumas heurísticas que podem ser utilizadas para melhorar a qualidade das ferramentas que disponibilizam estas visualizações. Assim sendo, Forsell e Johansson [62] propõem um conjunto de dez heurísticas que auxiliem os criadores destas ferramentas no seu trabalho, sendo elas:

- **Codificação da informação** – uma vez que a percepção da informação por parte dos utilizadores destas ferramentas é dependente da correspondência dos dados à forma como são mostrados nas visualizações, esta percepção deve ser maximizada recorrendo a símbolos adicionais, bem como a técnicas ou características realistas que a aumentem;
- **Acção mínima** – Diz respeito ao número de acções que são necessárias executar na ferramenta para atingir determinado fim, sendo que esse número deve ser o mínimo possível;
- **Flexibilidade** – Diz respeito ao número de maneiras distintas disponíveis na ferramenta para atingir um determinado objectivo, permitindo a costumização da mesma, por forma a que esta melhor se adeqúe aos hábitos e estratégias de trabalho dos utilizadores;
- **Orientação e ajuda** – Diz respeito à disponibilização ao utilizador de funções de suporte que permitam controlar os níveis de detalhe na ferramenta, bem como a capacidade de refazer e desfazer acções previamente executadas. Para além disso, devem ser também disponibilizadas ao utilizador informações adicionais sobre utilização da ferramenta;
- **Organização espacial** – É respeitante à forma como os elementos estão distribuídos pelo *layout*, tendo em consideração a precisão e legibilidade dos mesmos, bem como a eficiência na utilização do espaço, sendo estes factores que irão influenciar a forma como o utilizador se orienta na aplicação;
- **Consistência** – Dentro de um mesmo contexto a aplicação deve manter a consistência na forma como se apresenta ao utilizador;
- **Reconhecimento em vez de lembrança** – Para que o utilizador possa executar as tarefas que pretende, não deve necessitar de memorizar todos os passos, devendo estes estar acessíveis de uma forma intuitiva;
- ***Prompting*** – Quando num determinado contexto são possíveis executar diferentes acções, devem ser facultados todos os meios para que o utilizador possa expressar da forma mais clara qual a que pretende;
- **Remover o que está a mais** – Qualquer informação que não seja realmente necessária pode distrair o utilizador dos dados que está a analisar, devendo por isso minimizar-se a informação desnecessária;
- **Redução do conjunto de dados** – Diz respeito às técnicas que podem ser empregues na redução do conjunto de dados, por forma a aumentar a sua eficiência e facilidade de utilização;

Deve ter-se em atenção que estas heurísticas [62], bem como as técnicas de redução de desordem abordadas anteriormente [61], não devem ser interpretadas como definitivas e únicas,

mas sim como mais um instrumento disponível aos criadores de ferramentas de visualização que pode ser melhorado.

Depois de conhecer a forma como os utilizadores adquirem conhecimento por meio de visualizações de dados, bem como de algumas técnicas e heurísticas que melhoram essas visualizações, importa conhecer agora ferramentas que permitam visualizar dados numa forma gráfica.

4.1 *Google Chart Tools*

A ferramenta *Google Chart Tools* (GCT) [63], desenvolvida pela empresa *Google Inc.*, permite a criação de gráficos interactivos que podem ser incorporados em *Web Sites* [64]. Cada visualização de um gráfico nesta ferramenta depende de três blocos distintos: a biblioteca de gráficos ("Chart Library"), uma tabela de dados ("DataTable") e uma fonte de dados ("Datasource") [65].

A "Chart Library" expõe cada gráfico como uma classe *JavaScript*, o que permite utilizar qualquer linguagem de programação do lado do servidor. São disponibilizados distintos tipos de gráficos, sendo que novos tipos podem ser desenvolvidos pela comunidade. No entanto, a galeria principal é composta por apenas catorze tipos, sendo eles [66]: "area chart", "bar chart", "bubble chart", "candlestick chart", "column chart", "combo chart", "gauge chart", "geo chart", "line chart", "pie chart", "scatter chart", "stepped area chart", "table chart" e "tree map chart". O *render* de cada gráfico [65] é efectuado recorrendo às tecnologias *HyperText Markup Language 5* (HTML5) e *Scalable Vector Graphics* (SVG), o que permite a compatibilidade desta ferramenta com os browsers modernos bem como com diversas plataformas, sendo que, caso seja utilizada uma versão antiga do *Internet Explorer* é utilizado *Vector Markup Language* (VML) no *rendering*. Assim, esta ferramenta dispensa qualquer tipo de *plug-ins* para o seu funcionamento.

Cada um dos tipos de gráfico pode ser utilizado sem qualquer costumização [67], no entanto, a ferramenta permite a costumização de diversos aspectos dos gráficos, nomeadamente a largura ou a altura, de uma forma simples, uma vez que cada tipo de gráfico expõe opções no formato "nome:valor", opções essas que podem ser personalizadas.

Para além das costumizações, esta ferramenta permite que os gráficos disparem eventos [68] que podem ser manipulados no lado do cliente utilizando *JavaScript*. Estes eventos podem ser desencadeados pelo utilizador ao interagir com um gráfico, quando clica nele, por exemplo, ou podem ser programados pelo criador da visualização para disparar quando determinada condição se verifica. Importa referir que os eventos associados aos gráficos são distintos dos eventos DOM.

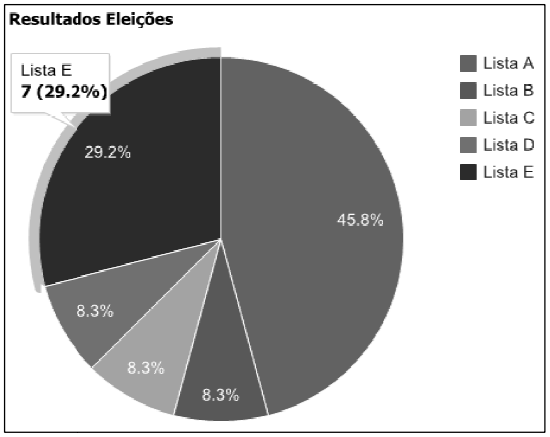


Figura 21: Exemplo de um "pie chart" criado com recurso à ferramenta GCT

Na Figura 21 está representado um "pie chart" no qual é possível ver o exemplo de um evento a ser disparado. Quando o utilizador clicou sobre a superfície do gráfico respeitante à "Lista E", o evento disparado fez aparecer sobre ela os detalhes da mesma, conforme se pode verificar na figura.

```
<html>
   <head>
     <script type="text/javascript" src="https://www.google.com/jsapi"></script>
     <script type="text/javascript">
       google.load("visualization", "1", {packages:["corechart"]});
       google.setOnLoadCallback(drawChart);
       function drawChart() {
         var data = google.visualization.arrayToDataTable([
           ['Lista', 'Votos'],
           ['Lista A',     11],
           ['Lista B',      2],
           ['Lista C',  2],
           ['Lista D', 2],
           ['Lista E',    7]
         ]);
         var options = {
           title: 'Resultados Eleições'
         };
         var chart = new google.visualization.PieChart(document.getElementById('chart_div'));
         chart.draw(data, options);
       }
     </script>
 </head>
 <body>
     <div id="chart_div" style="width: 900px; height: 500px;"></div>
 </body>
</html>
```

Figura 22: *Markup* e código utilizados para criar o gráfico da Figura 21

Na Figura 22 estão representados o *markup* e código necessários à criação do gráfico da Figura 21. No código representado nesta figura, é possível ver a forma como os dados podem ser passados à visualização por meio de um *array* de *arrays,* que posteriormente é transformado numa "DataTable" pela função "arrayToDataTable", ficando assim declarada a variável "data". É também visível neste código uma pequena costumização da visualização, nomeadamente o título da mesma que é definido como sendo "Resultados Eleições", conforme se pode verificar na

declaração da variável "options", a qual é um objecto que contém um par nome/valor, em que o nome é "title" e o valor é o título do gráfico.

Por forma a permitir uma melhor organização e controlo dos gráficos quando estes são mostrados numa página *Web,* a ferramenta GCT disponibiliza ainda um painel de instrumentos ("Dashboard") [69] que pode ser utilizado para controlar simultaneamente diferentes gráficos que partilhem os mesmos dados, bastando apenas escolher determinada opção no "Dashboard" para que todos os gráficos a ele associados sofram as alterações pretendidas. Desta forma, o programador da visualização não tem de se preocupar com a programação das ligações entre todos os gráficos relacionados.

Uma outra característica também importante relativa aos gráficos é que estes não têm limite de pontos de dados [70], o que pode ser bastante vantajoso quando a colecção de dados a mostrar for extensa.

Um outro bloco vital na criação de visualizações utilizando esta ferramenta é o "DataTable". Este bloco [65] serve para povoar todos os tipos distintos de gráficos com os dados a serem representados na visualização. Uma vez que "DataTable" é uma classe *JavaScript*, o mesmo objecto implementado a partir desta classe pode ser utilizado em qualquer tipo de gráfico, desde que os dados respeitem o formato relativo a cada tipo, ou seja, um objecto "DataTable" com dados utilizados para povoar um "bar chart", pode ser utilizado para povoar um "column chart", sem que para isso seja necessário fazer qualquer alteração ao objecto inicial. Mais concretamente, um "DataTable" [71] é uma tabela de duas dimensões em que todos os elementos de cada coluna devem ter o mesmo tipo de dados. Associado a cada coluna [71], de forma a diferenciar as mesmas umas das outras, existe um *id* cujo objectivo é permitir a referência a cada uma das colunas individualmente. Além disso, cada coluna tem a ela associado um *label*, que habitualmente é utilizado nas visualizações para designar a coluna, por forma a que os utilizadores compreendam a que se referem os dados constantes da mesma.

Um "DataTable" [71] é composto por células, em que cada uma delas contém um valor, podendo esse valor ser nulo ou então ser do tipo especificado para a coluna em que se encontra. Para permitir um maior controlo sobre os dados constantes da tabela, a classe "DataTable" define alguns métodos [65] que oferecem a possibilidade de ordenar, modificar ou filtrar os mesmos, possibilitando também [71] adicionar novas colunas ou linhas a uma tabela já criada anteriormente.

Cada "DataTable" pode ser povoado [65] programaticamente recorrendo a qualquer tipo de fonte de dados. No entanto, caso essa fonte de dados suporte o protocolo *Chart Tools DataSource* este processo é muito mais simples, uma vez que é apenas necessário executar *queries* ao "DataSource" sendo imediatamente retornados "DataTables" com todos os dados.

Importa agora abordar o último bloco necessário à construção de uma visualização no GCT, o "DataSource". Tal como já foi dito anteriormente, este pode ser de qualquer tipo, seja uma base dados ou mesmo um *web service*. Contudo, por simplificar bastante o processo de criação de "DataTables", interessa conhecer melhor a forma como esta ferramenta permite executar *queries* [65] [72] em fontes de dados que implementem o protocolo *Chart Tools DataSource,* como por exemplo o *Google Spreadsheets* ou *Google Fusion Tables*, podendo este protocolo ser também implementado por qualquer fonte de dados que se pretenda transformar num provedor de dados para esta ferramenta.

De acordo com este protocolo, um "DataSource" [73] é um *web service* para o qual se podem enviar *queries*, recebendo-se em resposta um "DataTable" povoado com os dados relativos à *query* em questão. Desta forma, para se efectuar um pedido [73] a um "DataSource" deve ser instanciado um objecto do tipo "Query" [74], objecto esse que representa uma *query* a ser enviada a um "DataSource", sendo que nessa instanciação devem indicar-se eventuais opções, como por exemplo o método que deve ser utilizado no envio da informação, bem como o *Uniform Resource Locator* (URL) do "DataSource". Neste URL deve estar indicado de uma forma perceptível ao "DataSource" quais os dados que se pretendem. Posteriormente, caso se pretendam filtrar ou ordenar os dados de alguma forma, pode incluir-se também neste pedido uma *string* formatada numa linguagem derivada do SQL, especificando as opções pretendidas. Caso o "DataSource" suporte esta linguagem, ele vai devolver um "DataTable" com todos os dados dispostos da maneira pretendida, enquanto que se a linguagem não for suportada, esta *string* será ignorada sendo devolvido o "DataTable" sem qualquer filtragem ou ordem. Depois de o objecto do tipo "Query" ser instanciado, e se ter ou não adicionado a referida *string* para eventuais filtragens ou ordenações, pode enviar-se finalmente o pedido para o "DataSource", especificando um método de *callback* que será chamado quando a resposta for recebida.

Logo que a resposta seja recebida [73] o método de *callback* é executado, sendo este método responsável por verificar se a resposta foi ou não recebida com sucesso: se ocorre algum problema, a resposta contém informação sobre o erro; se o pedido ao "DataSource" teve sucesso, então a resposta contém o "DataTable" com os dados pretendidos, podendo esse "DataTable" ser directamente passado ao gráfico para assim ser criada uma visualização dos dados.

Esta ferramenta simplifica bastante o processo de criação de visualizações de dados por meio de gráficos, no entanto, são-lhe apontadas algumas desvantagens. Em primeiro lugar é de notar que apesar de o *rendering* das visualizações nesta ferramenta ser feito com recurso a SVG [75], não é possível exportar as mesmas neste formato, o que seria útil, por exemplo, caso se pretendesse incluir as visualizações gráficas criadas em relatórios. Além disso, [76] o criador da visualização não tem qualquer controlo sobre o processo de *rendering* da mesma, o que pode levar

a alguns problemas, nomeadamente quando as colecções de dados são muito extensas podendo ocorrer desordem na representação dos dados, bem como no texto presente nos eixos de cada gráfico que, devido à sobreposição de valores, se pode tornar ilegível, dificultando o processo de compreensão da informação representada. Também a visualização de dados em tempo real perde qualidade com esta ferramenta, uma vez que cada vez que se pretende juntar novos dados à mesma a ferramenta força um novo *render* da visualização [76].

Apesar das desvantagens, esta ferramenta é de simples de utilização, por conseguinte, caso o criador das visualizações esteja disposto a abdicar do controlo da parte do *rendering* para ganhar em simplicidade e uniformidade na parte de passagem dos dados à visualização, esta é uma ferramenta que lhe irá ser muito útil.

4.2 *Highcharts*

Highcharts [77] é uma biblioteca que permite a criação de gráficos interactivos, possibilitando a sua integração em *web sites* ou aplicações *web*. Esta biblioteca [78] foi inteiramente desenvolvida em *JavaScript*, servindo-se apenas de tecnologias presentes nos *browsers*, não sendo por isso necessários quaisquer tipos de *plug-ins* para o seu funcionamento. Esta opção de desenvolvimento permite também que esta biblioteca seja completamente independente da tecnologia utilizada no servidor, bem como compatível com praticamente todos os *browsers* disponíveis actualmente. Esta compatibilidade é também conseguida [79] devido às diferentes tecnologias utilizadas no processo de *rendering* das visualizações, uma vez que nem todos os browsers suportam as mesmas tecnologias. Por conseguinte, a tecnologia mais amplamente utilizada é o SVG, sendo que quando esta não é suportada pelos *browsers*, como é o caso das versões mais antigas do *Internet Explorer*, é utilizado VML.

Para além de ser amplamente compatível com os diversos sistemas, esta biblioteca tem o seu código fonte disponível e aberto [78], permitindo assim aos seus utilizadores editar e alterar a ferramenta por forma a melhor satisfazer as suas necessidades de visualizações gráficas. Aliada a esta liberdade, está também a liberdade de utilização, já que esta é gratuita para usos não comerciais.

São vários os tipos de gráficos que são possíveis criar com esta ferramenta [78] [80]: “line chart”, “spline chart”, “area chart”, “areaspline chart”, “column chart”, “bar chart”, “pie chart” e “scatter chart”, sendo que todos eles podem ser combinados em apenas uma visualização, permitindo assim uma grande flexibilidade. Além disso, é também possível criar gráficos com diversos eixos [78], para além dos habituais dois, ou seja, quando se pretendem comparar valores

que não estão na mesma escala, pode ser adicionado um eixo para cada uma dessas escalas, sendo permitida inclusão de um eixo y por série de dados e um eixo x por categoria de dados.

Por forma a melhorar a visibilidade dos dados representados nas visualizações [78], a ferramenta *Highcharts* oferece a funcionalidade de fazer *zoom* nas mesmas, bem como a possibilidade de inverter a posição dos gráficos, inverter a ordem dos valores representados nos eixos e também rodar os *labels* textuais associados à visualização, digam eles respeito a pontos de dados ou aos valores representados nos eixos. Em relação aos eixos, é também facilitada a criação de visualizações com eixos na escala data/tempo, pois a ferramenta permite uma precisão até aos milissegundos nestes casos.

```
$(function () {
    var chart;
    $(document).ready(function() {
        chart = new Highcharts.Chart({
            chart: {
                renderTo: 'container',
                plotBackgroundColor: null,
                plotBorderWidth: null,
                plotShadow: false
            },
            title: {
                text: 'Resultados Eleições',
                x:-20
            },
            tooltip: {
                formatter: function() {
                    return '<b>'+ this.point.name +'</b>:'+
                        this.y +' Votos';
                }
            },
            plotOptions: {
                pie: {
                    allowPointSelect: true,
                    cursor: 'pointer',
                    dataLabels: {
                        enabled: true,
                        color: '#000000',
                        connectorColor: '#000000',
                        formatter: function() {
                            return '<b>'+ this.point.name+
                                '</b>: '+ this.y +' Votos';
                        }
                    ]
                }
            },
            series: [{
                type: 'pie',
                name: 'Listas',
                data: [
                    ['Lista A',11],
                    ['Lista B',2],
                    ['Lista C',2],
                    ['Lista D',2],
                    ['Lista E',7]
                ]
            }]
        });
    });
});
```

```
$(function () {
    var chart;
    $(document).ready(function() {
        chart = new Highcharts.Chart({
            chart: {
                renderTo: 'container',
                type: 'column',
                marginRight: 130,
                marginBottom: 25
            },
            title: {
                text: 'Resultados Eleições',
                x: -20
            },
            xAxis: {
                categories: ['Listas']
            },
            yAxis: {
                title: {
                    text: 'Votos'
                }
            },
            tooltip: {
                formatter: function() {
                        return '<b>'+ this.series.name+
                            '</b><br/>'+this.y +' Votos';
                }
            },
            legend: {
                layout: 'vertical',
                align: 'right',
                verticalAlign: 'top',
                x:  10,
                y: 100,
                borderWidth: 0
            },
            series: [{name: 'Lista A',data: [11]},
                     {name: 'Lista B',data: [2]},
                     {name: 'Lista C',data: [2]},
                     {name: 'Lista D',data: [2]},
                     {name: 'Lista E',data: [7]}]
        });
    });

});
```

Figura 23: Exemplos do código necessário à criação de dois tipos de gráficos distintos com os mesmos dados em *Highcharts*. "Pie chart" à esquerda e "column chart" à direita

Duas grandes vantagens desta ferramenta em relação ao GCT são que, em primeiro lugar, a ferramenta *Highcharts* permite que as visualizações sejam exportadas para os formatos *Portable Network Graphics* (PNG), JPEG, PDF ou SVG, podendo assim ser utilizadas em *reports* ou outras finalidades; em segundo lugar, após a criação da visualização é possível modificar, adicionar ou eliminar séries de dados ou eixos, sendo assim permitido criar visualizações em tempo real, uma

vez que a visualização pode estar a receber dados de um *data provider* mostrando-os imediatamente após os receber.

Para que se consigam executar as funções disponibilizadas por esta biblioteca numa página ou aplicação *web* é apenas necessário incluir um de entre os *frameworks JQquery*, *MooTools* ou *Prototype*, bem como a própria biblioteca através do ficheiro "*highcharts.js*" [78].

Na Figura 23 está representado o código necessário à criação de dois gráficos de tipos distintos com recurso ao *framework JQuery*. Conforme se pode verificar nesta figura, a sintaxe utilizada [78] para a configuração das visualizações é bastante simples, uma vez que respeita a notação dos objectos *JavaScript*, podendo mesmo estas ser configuradas por indivíduos sem conhecimentos de programação. Para configurar os gráficos é necessário apenas saber para que serve cada opção [81] de configuração.

Uma vez que no código da Figura 23 está a ser utilizado o *framework JQuery*, a função "ready" deste *framework* vai executar a função que lhe é passada como parâmetro logo que o DOM da página esteja pronto. Nessa altura, é feito o *render* da visualização para o elemento da página que se pretende, que neste caso, conforme se pode verificar na opção "renderTo" da figura, é o elemento com o identificador "container". A visualização é então criada respeitando todas a opções impostas pelo criador da mesma.

Um aspecto interessante desta ferramenta é que ela permite que os dados provenham de qualquer tipo de fonte externa [78], sendo apenas necessário fazer o *parsing* dos mesmos para o formato *array* de *JavaScript*. No entanto, não existe uniformidade na forma como esses dados são depois passados aos distintos tipos de gráficos, conforme se pode observar no elemento "series" de ambos os tipos de gráfico da Figura 23, onde no código relativo ao "pie chart" (esquerda) existe apenas uma série de dados denominada Listas, a qual contém um *array* "data" com cinco elementos, sendo cada um desses elementos uma *slice* do gráfico. Já no código relativo ao "column chart" (direita) é possível verificar que existem cinco séries, representando cada uma dessas séries uma coluna no gráfico. Esta falta de uniformidade pode tornar complexa a utilização desta ferramenta caso se pretenda automatizar o processo de escolha da visualização de acordo com os dados que se pretendem visualizar.

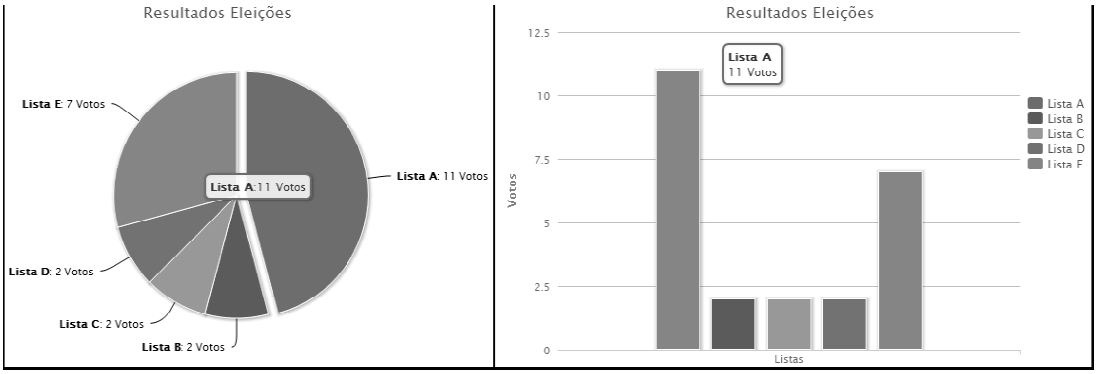


Figura 24: Visualizações gráficas criadas pelo código da Figura 23. "Pie chart" à esquerda e "column chart" à direita

Conforme se pode ver na Figura 24, esta ferramenta permite adicionar informação complementar e passível de costumização a cada ponto ou série de dados por meio de *tooltips* [78]. Estes *tooltips* ocorrem quando o utilizador faz *hover* sobre os dados em questão, sendo a informação contida neles programável, conforme se pode ver no código da Figura 23 no elemento com o mesmo nome. Neste caso optou-se por mostrar o nome da lista à qual correspondem os dados, bem como o número de votos de cada lista. De notar que para os dois tipos de gráficos representados, o código utilizado para o mesmo efeito difere novamente (Figura 23), já que para o "pie chart" é utilizado "this.point.name" para determinar o nome da lista a mostrar, enquanto que para o "column chart" é utilizado "this.series.name" com o mesmo objectivo. Importa ainda salientar que as visualizações permitem que a eventos provocados pelos utilizadores, ao clicar num ponto desta, por exemplo, sejam atribuídos métodos de *callback*, para tornar as mesmas mais dinâmicas e interactivas [78].

Um outro aspecto interessante desta ferramenta [82] é que ela oferece um módulo que permite o desenho de formas ou texto nos gráficos, bem como desenhar directamente numa página HTML.

Tendo em conta todos os aspectos do seu funcionamento, a ferramenta *Highcharts* é ideal caso se pretenda abdicar da uniformidade na passagem dos dados aos diferentes tipos de gráficos tendo em vista o ganho da possibilidade de criação de gráficos dinâmicos em *real-time*.

4.3 *Many Eyes*

Uma outra ferramenta que permite a criação de visualizações gráficas a partir de colecções de dados é a ferramenta *Many Eyes* [83] desenvolvida pela *International Business Machines Corporation* (IBM). A diferença principal em relação ás outras abordadas anteriormente, é que esta ferramenta não se trata de uma API ou de uma biblioteca que o criador das visualizações integra no

seu projecto, mas sim de um *web site* que permite aos seus utilizadores a criação de visualizações interactivas de uma forma intuitiva, sem que sejam necessários quaisquer conhecimentos de programação. Além disso, esta ferramenta permite a interacção social entre os vários utilizadores, tendo essa interacção por objecto as visualizações e as colecções de dados. Pretendeu-se [84] pois, com este *web site*, oferecer tecnologias de visualização de dados a uma audiência que até então a elas não tinha acesso por falta de conhecimentos técnicos. Para além disso procurou-se fomentar também a colaboração na análise dessas mesmas visualizações e dos dados subjacentes. Assim sendo, as actividades habituais no web site [84] são a transferência de colecções de dados para o mesmo, a construção de visualizações tendo por base essas colecções e o comentário e discussão sobre ambas. De notar que, por forma a facilitar a interacção entre os utilizadores, todos estes elementos são públicos [85], ou seja, quando uma visualização de dados ou um comentário é adicionado ao *web site*, fica imediatamente disponível a qualquer pessoa com acesso à *Internet* por meio de um URL.

Uma vez que as colecções de dados disponíveis no *web site* são, na sua grande maioria, facultadas pelos utilizadores, dos quais muitos não têm conhecimentos técnicos de programação, o modelo de dados [84] deve ser facilmente compreensível e simples permitindo, no entanto, que a partir dele sejam criadas visualizações algo complexas. Desta forma a ferramenta permite a introdução de dados em dois formatos diferentes [84]: na forma tabelar, ou seja, um conjunto de colunas com o mesmo comprimento, sendo que a cada uma delas corresponde um tipo que pode ser textual ou numérico, e na forma de texto não estruturado, ou seja, o equivalente a uma *string*. Para transferir os dados para o *Many Eyes* o utilizador deve introduzi-los num formulário disponibilizado para esse efeito. Caso os dados sejam introduzidos na forma de uma grelha [84], em que a primeira linha representa o nome de cada coluna e as restantes os elementos das mesmas, sendo os elementos de cada coluna separados por *tabs*, a ferramenta vai interpretar estes dados como uma tabela, tentando atribuir automaticamente um tipo a cada uma das colunas. Se os dados forem introduzidos em texto não estruturado [84], a ferramenta interpreta-os como sendo dados não estruturados. De notar que todas as colecções de dados são armazenadas nos servidores da ferramenta num formato de texto [84] por forma a aumentar a eficiência dos mesmos. Além disso, não é permitido editar uma colecção de dados após a sua criação, pois isso poderia tornar obsoletas algumas visualizações e comentários a ela associados. No entanto, caso não tenha sido criada nenhuma visualização da mesma, o seu criador pode eliminá-la por forma a corrigir eventuais erros.

Importa ainda referir que a cada colecção de dados vai ficar associada uma colecção de metadados [84], nomeadamente o título, a fonte e a descrição da mesma, bem como *tags* que permitem facilmente a um utilizador encontrar colecções de dados relativas a um determinado tema

do seu interesse. Toda a informação respeitante a uma colecção de dados vai ficar disponível numa página [84] que é criada automaticamente para esse efeito, na qual se encontram todas as visualizações criadas até ao momento para essa colecção, bem como a discussão gerada entre os utilizadores sobre a mesma por meio de comentários.

No que diz respeito às visualizações, o *web site Many Eyes* disponibiliza distintos tipos [86] para objectivos específicos. Por forma a conseguir compreender as relações entre os diversos pontos de dados existem os tipos "scatterplot", "matrix chart" e "network diagram". Já para comparar conjuntos de valores são disponibilizados os tipos "bar chart", "block histogram" e "bubble chart". Para conseguir encontrar variações ao longo do tempo existe o "line graph" e o "stack graph". Por forma a conseguir ver melhor as partes do universo do conjunto de dados são disponibilizados os tipos "pie chart" e "treemap". Caso se pretenda analisar texto os tipos "word tree", "tag cloud", "phrase net" e "world cloud generator" são facultados. Por fim, para integrar os dados em visualizações geográficas existem o "world map", o "US county map" e outros mapas para distintas regiões nos Estados Unidos da América. A grande maioria destas visualizações estão implementadas como sendo *applets Java* [84], no entanto, algumas delas foram também implementadas usando *Adobe Flash* [87]. Esta opção de implementação implica que os utilizadores tenham instalados nos seus *browsers* os *plug-ins* necessários ao bom funcionamento destas tecnologias.

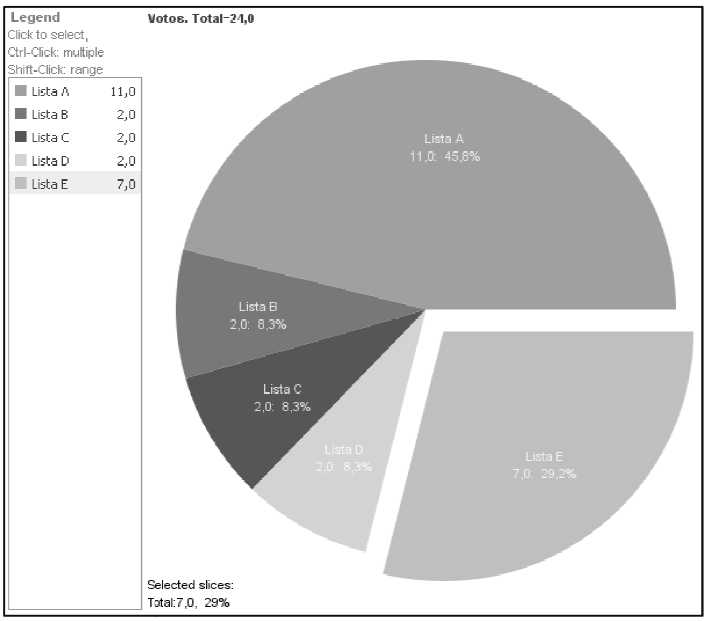


Figura 25: Exemplo de uma visualização do tipo *pie chart* na ferramenta *Many Eyes*

Para criar uma visualização [84] na ferramenta *Many Eyes*, como por exemplo a da Figura 25, basta apenas associar um dos tipos de visualizações mencionadas anteriormente com os dados

que se pretendem visualizar. Logo que esta esteja criada é-lhe atribuída uma página [84] onde os utilizadores podem discutir a mesma, contendo essa página todos os metadados, bem como uma ligação para a colecção de dados que lhe deu origem.

Para que uma determinada visualização mostre apenas colecções de dados que fazem sentido no seu contexto, foi atribuído a cada uma delas um esquema que permite excluir as colecções de dados inválidas para visualização. Cada um desses esquemas é composto por um conjunto de distintos elementos [84], nomeadamente:

- **Elemento não estruturado**, composto por texto sem uma estrutura definida;
- **Elemento numérico**, composto por uma coluna única de dados numéricos;
- **Elemento textual**, composto por uma coluna única de dados textuais;
- **Elemento multi-numérico**, composto por uma ou mais colunas de dados numéricos;
- **Elemento multi-textual**, composto por uma ou mais colunas de dados textuais;

Combinando os diversos elementos descritos anteriormente, cada visualização descreve eficazmente quais os dados que permite visualizar.

Importa agora abordar o aspecto que mais diferencia a ferramenta *Many Eyes* das demais: o aspecto social. Conforme já foi referido anteriormente, este *web site* permite aos utilizadores [84] adicionar comentários a todas as colecções e visualizações de dados, sendo esta funcionalidade a responsável por permitir uma experiência social na análise dos mesmos. No entanto, para melhorar um pouco mais essa experiência, é também oferecida a possibilidade de cada utilizador salvar o estado [84] como está a experienciar a visualização, ou seja, as filtragens que tenha efectuado na mesma, os parâmetros ou cores utilizadas, sendo assim guardada esta informação que será tornada pública por meio de um URL disponibilizado na página principal da visualização. Este URL pode depois ser incluído em outros *web sites*, permitindo assim uma ligação directa a cada um dos estados das visualizações. Importa também referir que cada visualização pode ser votada [85] pelos utilizadores, criando-se assim uma noção de qualidade das mesmas.

Um outro aspecto interessante, ainda de âmbito social, é que é permitido aos utilizadores criarem um *hub* de tópicos [85], ou seja, uma página que serve de ponto de entrada para colecções de dados e visualizações relacionadas com um determinado tema, conseguindo-se assim um ponto centralizado de discussão.

Apesar de toda a funcionalidade oferecida, o *web site Many Eyes* ainda não conseguiu criar uma comunidade de utilizadores muito vasta. Segundo *Karmakar* e *Zhu* [88] os utilizadores desta ferramenta são casuais, sendo o número de utilizadores com uma participação activa muito reduzido.

Esta é uma ferramenta interessante para utilizadores sem experiência de programação. A forma de introdução dos dados é simples e intuitiva, sendo oferecida uma vasta colecção de tipos de visualizações interactivas. Também a necessidade de apenas um *browser* para criar e visualizar as colecções de dados é uma vantagem, estando esta informação imediatamente acessível a todo o mundo. O facto de ser uma ferramenta social pode ser uma vantagem pela busca partilhada de conhecimento que permite, mas também uma desvantagem por questões [85] de privacidade e gestão de audiência dos dados. Talvez a questão da privacidade, a impossibilidade de inclusão das visualizações interactivas em outros *web sites* e a inexistência de uma API para utilizadores mais experientes sejam as principais desvantagens desta ferramenta.

5 *Reports*

Ao longo de um dia são vários os *reports* que passam pelas mãos de qualquer indivíduo sem que este, muitas vezes, faça a associação do nome ao objecto. Simples facturas ou extractos de conta bancária são os exemplos mais simples que diariamente fazem parte da vida das pessoas. No entanto, *reports* podem também ser complexos documentos, contendo informação sobre as operações de uma determinada empresa ou um catálogo descrevendo todos os seus produtos. Mais concretamente, Turley e Bruckner [89] definem um *report* como sendo:

"Qualquer output formatado de dados provenientes de uma base de dados ou de qualquer outra fonte."

Assim, caso se disponha de uma fonte de dados e de uma ferramenta capaz de os formatar é possível criar-se um *report* dos mesmos. No entanto, nem sempre é simples escolher a forma de formatação a utilizar [90], uma vez que para além de os modelos utilizados nas fontes de dados poderem sofrer alterações ao longo do tempo, também os requisitos de informação que devem estar contidos nos *reports* podem alterar-se. Desta forma, é possível apontar dois paradigmas distintos de *reports* [91]: os estáticos e os dinâmicos. Tal como o nome indica, os *reports* estáticos tem a sua formatação estabelecida de uma forma estática [91], isto é, não sofrem qualquer alteração tendo em conta os dados neles contidos. Por esta razão, sempre que houver alguma alteração ao modelo dos dados utilizados para povoar o *report* [90] ou sempre que surjam novos requisitos de informação, é também necessário modificar todo *report*, o que pode ser uma tarefa árdua, principalmente no caso de serem vários a necessitar de alterações. Já no que diz respeito aos *reports* dinâmicos [90], estes são gerados dinamicamente de acordo com os dados que neles se pretendem mostrar, sendo para isso preferível utilizar modelos de dados genéricos que conferem uma maior flexibilidade aos mesmos. Assim, já não é necessário efectuar alterações ao *report* sempre que existam mudanças de requisitos ou de modelo de dados utilizado.

De acordo com o seu conteúdo e formatação, os *reports* podem também ser classificados em várias categorias que por vezes podem incluir elementos umas das outras, sendo elas [89]: "reports de suporte operacional", "reports de análise e informação de negócio", "reports para formulários, etiquetas e cartas" e "reports integrados na aplicação". Na categoria de "reports de suporte operacional" estão contidos os tipos de relatórios que dizem respeito [89], tal como o seu nome indica, a dados do foro operacional das instituições, ou seja, mais relacionados com o funcionamento diário ou de curto prazo das mesmas. Um dos tipos de *reports* incluídos nesta categoria é o das ordens de venda, facturas e formulários de inventário. Habitualmente estes *reports*

contêm um cabeçalho com as informações relativas à entidade a que se referem seguidos por uma secção tabelar onde se incluem os dados pretendidos, seguindo-se a esta, por vezes, um rodapé contendo informações adicionais relativas à entidade [89]. Um outro tipo pertencente a esta categoria são os *reports* tabelares e de lista [89], os quais, como seria de esperar, permitem formatar os dados que se pretendem mostrar em tabelas ou listas. Também os catálogos [89] pertencem a esta categoria de *reports*, sendo estes utilizados habitualmente para agrupar produtos ou serviços disponibilizados por uma determinada entidade, fornecendo detalhes sobre cada um deles. Um requisito dos catálogos é que estes devem ser fáceis de ler.

Ainda na categoria de "reports de suporte operacional" encontram-se os *reports* de sumário de actividade [89]. Este tipo de *reports* pode, no entanto, ser incluído na categoria "reports de análise e informação de negócio", estando esta inclusão dependente apenas do âmbito dos sumários [89], ou seja, caso estes digam respeito a um intervalo temporal reduzido ou apenas a uma pequena parte da entidade a que se referem, como por exemplo um escritório de uma empresa multinacional, então enquadram-se na primeira categoria. Caso os sumários sejam utilizados para auxiliar a tomada de decisões ao nível da entidade no seu todo ou digam respeito a intervalos temporais alargados, então devem ser incluídos no segundo tipo.

Finalmente, os *reports* de estado pertencem também à categoria de "reports de suporte operacional". Estes têm o objectivo [89] de apresentar dados concisos, passíveis de ser comparados ao longo do tempo, como por exemplo o valor das vendas conseguidas por vendedor numa determinada empresa.

A categoria "reports de análise e informação de negócio" contém os tipos de *reports* que auxiliam no processo [89] de tomada de decisão e criação de estratégias das entidades às quais se referem, servindo-se de dados actuais e históricos para neles encontrar tendências e relações. Um dos tipos de *reports* pertencentes a esta categoria é o "dashboard". Este tipo de *report* [89] é ele próprio uma colecção de *reports* que informam sobre o estado de métricas relacionadas entre si no âmbito de uma determinada instituição ou empresa. Os "dashboards" [89] devem ser simples e a informação neles contida deve ter o nível de detalhe adequado às necessidades dos seus utilizadores, por forma a maximizar a informação que é fornecida. Um outro tipo de *report* contido nesta categoria são as "pivot tables", as quais [89] permitem agrupar os dados correspondentes quer às colunas quer às linhas de uma tabela, sendo o valor desse agrupamento mostrado nas intersecções das mesmas.

A categoria "reports de análise e informação de negócio" contém ainda os *reports* com gráficos e mapas [89]. Os primeiros dizem respeito aos *reports* que contém gráficos, servindo estes para visualizar os dados subjacentes. Os segundos são respeitantes a *reports* onde os dados são

integrados num mapa geográfico, por forma a associar os dados representados a uma determinada localização geográfica.

Na categoria de "reports para formulários, etiquetas e cartas" estão contidos os tipos de *report* que fornecem os *layouts* necessários à criação de formulários, etiquetas ou cartas, podendo estes ser ou não preenchidos com dados. De notar que os formulários e as etiquetas podem também ser incluídas na categoria "reports de suporte operacional" [89].

Finalmente, no que diz respeito à categoria de "reports integrados na aplicação" [89], estão contidos todos os tipos de *reports*, inclusive das outras categorias, que de alguma forma integram uma aplicação, ou seja, caso estes contenham alguns menus e botões de funcionalidade ou caso integrem uma aplicação que lhes ofereça esta funcionalidade.

Esta secção apresenta algumas ferramentas que possibilitam a criação de *reports*.

5.1 *Microsoft SQL Server Reporting Services*

A ferramenta *Microsoft SQL Server Reporting Services* , que é uma plataforma que integra o *Microsoft SQL Server*, permite a criação e administração de *reports* em diversos formatos, a partir de variadas fontes de dados [92]. Para além de permitir aceder por meio de uma aplicação ou *web site* a *reports* já criados, permite também integrar a criação dos mesmos a partir destes meios [92].

Com esta ferramenta é possível criar *reports* de uma forma *standard* ou de uma forma *ad hoc* [93]. Os *reports standard* são construídos por alguém que conheça a estrutura das fontes de dados a serem utilizadas para popular o *report*. Desta forma, esta acção fica restringida aos programadores do meio no qual o *report* está inserido. Assim sendo, o utilizador apenas pode visualizar o mesmo não tendo oportunidade de seleccionar o que integrará o *report*. Já os *reports ad hoc* dão a possibilidade ao utilizador de definir qual o conteúdo que constará dos mesmos.

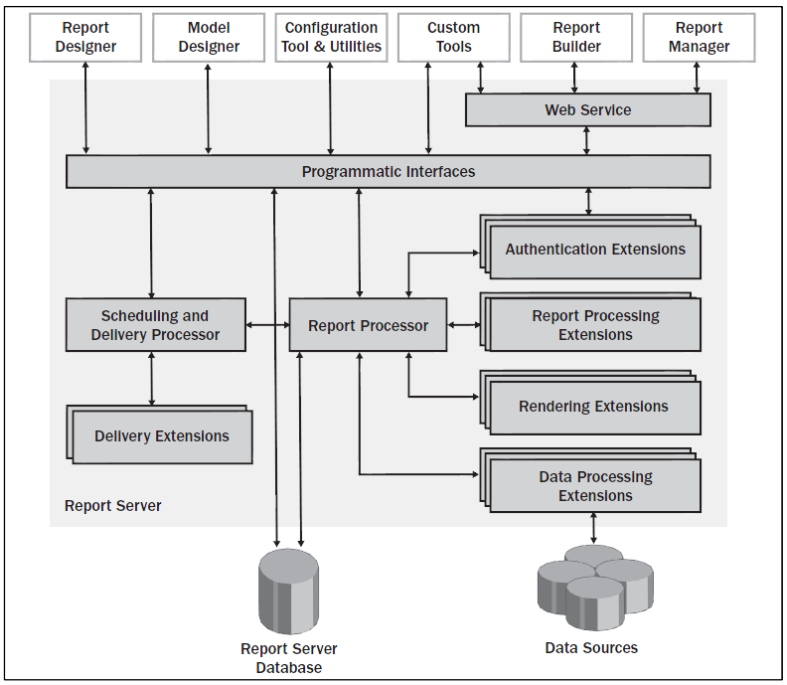


Figura 26: Arquitectura da ferramenta *Microsoft SQL Server Reporting Services* [94]

Do ponto de vista da arquitectura deste sistema, o núcleo central é o "Report Server", que é uma camada intermédia entre as fontes de dados e os componentes desta arquitectura que funcionam do lado do cliente. Este "Report Server" [93] é responsável por gerar, bem como efectuar o *render* de *reports*, sendo também responsável por enviá-los para o componente da arquitectura que os requisitou. Este componente funciona como um *web service*, permitindo assim um acesso simples [94]. O "Report Server" é composto por diversos componentes, como se pode observar na Figura 26, como por exemplo as várias extensões, nomeadamente de processamento de dados, *rendering*, processamento de *report*, autenticação e entrega, sendo cada uma destas extensões responsável por executar tarefas específicas relacionadas com o nome das mesmas [94]. Um outro componente integrante do "Report Server" é o "Report Processor", que é responsável por fundir os dados que se pretendem mostrar num determinado *report* com o *layout* do mesmo, enviando posteriormente o *report* para a extensão de *rendering* adequada à forma como o *report* vai ser visualizado. De notar que esta ferramenta permite *rendering* em diversos formatos de imagem [89], nomeadamente *Tagged Image File Format* (TIFF), bem como nos formatos [89] HTML, CSV, XML, PDF, XLS e DOC. "Quanto ao Scheduling and Delivery Processor" [94], é responsável por executar *reports* que estejam agendados, bem como entregá-los segundo o agendamento definido.

Fazem também parte da arquitectura desta ferramenta, conforme se pode observar na Figura 26, alguns componentes instalados do lado do cliente. O "Report Designer" [94] é um desses componentes, permitindo aos utilizadores criar *reports* mesmo sem conhecimentos de

programação, uma vez que oferece diversos *templates* para este efeito. Este componente permite aos utilizadores incluir os dados nas estruturas que pretendem para a sua visualização, como tabelas por exemplo, permitindo também efectuar cálculos com os dados e aceder a distintas opções de formatação do *report*. Um outro componente do lado do cliente é o "Report Builder" [94], sendo este mais utilizado na criação de *reports ad hoc*. Este componente permite a criação de *reports* com dados a partir de uma única fonte. A principal diferença [94] entre o "Report Builder" e o "Report Designer" é que o primeiro não necessita que seja construída uma *query* para determinar os dados, em vez disso serve-se de vários modelos de *report* que já contêm todas as informações necessárias à criação e execução das *queries*. Para criar os modelos de *report* dos quais o "Report Builder" se vai servir, é utilizado o componente "Model Designer" [94]. Neste componente é especificada qual a fonte de dados para cada *report*, bem como quais as tabelas e relações a serem utilizadas. Posteriormente o modelo do *report* é gerado e está pronto a ser utilizado pelo "Report Builder". Por fim, o elemento "Report Manager" tem duas valências, serve de consola de administração do "Report Server", bem como sistema de menu para os *reports* [89].

Devido a todas as características aqui descritas, esta é uma ferramenta bastante completa e interessante, apesar de não ser de utilização livre.

5.2 *SAP Crystal Reports*

A ferramenta *SAP Crystal Reports* [95] permite a criação de *reports* interactivos cujos dados podem ser provenientes de diversos tipos de fonte de dados, como por exemplo bases de dados com distintos SGBD ou ficheiros XLS.

Esta ferramenta permite reduzir o recurso a pessoal especializado na criação dos *reports* [96], uma vez que permite executar funções de filtragem, ordenação e reformatação directamente no *report*, bem como pesquisa interactiva dos dados sem que para isso seja necessário executar novas *queries* na fonte dos mesmos. É também possível, por meio desta ferramenta, construir visualizações gráficas dinâmicas a partir de dados e integrá-las nos *reports*, recorrendo à tecnologia *Adobe Flash*. Uma outra característica interessante desta ferramenta é que oferece suporte para mensagens SOAP, permitindo assim ter *web services* como fontes de dados e podendo passar directamente dados provenientes de *mashups* para os *reports*. Recorrendo a esta ferramenta podem ainda integrar-se nos *reports* códigos de barras já que esta contém funcionalidade que suporta este tipo de códigos.

A ferramenta *SAP Crystal Reports* permite ainda que o *render* dos *reports* nela criados possa ser feito a partir de diversos formatos, nomeadamente XML, o que possibilita a sua fácil integração com outras aplicações.

De notar que esta é uma ferramenta comercial, tendo estado integrada no *Microsoft Visual Studio* até à versão 2008 do mesmo, sendo que agora a sua disponibilidade neste ambiente de desenvolvimento está apenas restringida a um *add-on*. No entanto a mesma encontra-se hoje mais disseminada com o seu próprio ambiente de desenvolvimento.

5.3 *iText*

A biblioteca *iText* permite a criação e manipulação de ficheiros PDF. Esta não é propriamente uma ferramenta de criação de *reports*, mas devido às suas características e tendo em conta a definição de *report* dada por *Turley* e *Bruckner* [89] mencionada anteriormente, pode ser usada para tal. Esta biblioteca, que está disponível [97] numa versão para *Java* e noutra para C#, é de utilização gratuita para projectos não comerciais, estando o seu código fonte aberto.

Pode utilizar-se esta ferramenta para uma grande variedade de tarefas, como por exemplo a criação de facturas ou de bilhetes para espectáculos, sendo a mesma indicada [97] para quando os conteúdos a incluir no ficheiro PDF não estão disponíveis à partida, mas necessitam ser extraídos de uma base de dados em tempo real ou calculados tendo em conta o *input* de um utilizador. Esta ferramenta é também indicada quando o número de documentos ou o número de páginas num documento que se pretende criar é elevado [97], uma vez que permite automatizar todo o processo, bem como a costumização e personalização dos mesmos, possibilitando a sua criação em *batch*.

As funcionalidades permitidas pela biblioteca *iText* são muitas, no entanto, importa referir aquelas por que é mais conhecida. Para além de permitir a criação de documentos e *reports* baseados em dados provenientes de ficheiros XML, bases de dados ou de qualquer outra fonte [98], possibilita a criação de mapas e livros que se servem das características interactivas dos ficheiros PDF [97]. Caso se pretendam alterar ficheiros PDF já existentes [97], esta biblioteca oferece funcionalidades que permitem adicionar numeração, marcas de água e outras características às páginas dos documentos, bem como a concatenação de vários documentos ou a divisão dos mesmos.

O *layout* de cada documento criado por meio desta ferramenta é completamente configurável, podendo inserir-se texto livre, parágrafos, listas, imagens, tabelas, ou ainda o desenho de arcos, círculos e rectângulos [98], o que permite uma grande flexibilidade na geração de *reports*.

De referir ainda que esta biblioteca oferece classes que permitem funcionalidades de cifra, bem como de suporte de integração de aplicações desenvolvidas em *Adobe Flash* nos ficheiros PDF. Também ficheiros nos formatos XML e *eXtensible HyperText Markup Language/Cascading Style Sheets* (XHTML/CSS) podem ser convertidos para PDF utilizando esta mesma biblioteca [98].

Concluindo, segundo Yanmei e Xueya [99], sendo esta uma biblioteca de código aberto, escalável e robusta, torna-se mais fácil a sua manutenção e costumização, apresentando como principal desvantagem o facto de o resultado não poder ser visto em *run time*, pois não é possível visualizar o *report* durante o processo de criação do mesmo. Além disso, a documentação disponível para esta biblioteca é muito reduzida, principalmente para a versão C#, o que dificulta a compreensão e utilização da mesma.

6 Aplicação para representação dinâmica de dados de fontes heterogéneas

Actualmente, a UA dispõe de uma ferramenta [100] que permite a representação de dados e criação de *reports* sobre os mesmos. No entanto, para além de não permitir a criação de visualizações gráficas baseadas nos dados, esta ferramenta não foi desenhada para ter um desempenho dinâmico, sendo por isso necessário reprogramá-la sempre que se necessita de adicionar novas colecções de dados, o que obriga a uma elevada manutenção e gasto temporal. Por esta razão, existe a necessidade de tornar todo este processo dinâmico, permitindo também a criação de visualizações gráficas baseadas nos dados seleccionados. Foi com o intuito de suprir estas necessidades que se procurou, no decorrer deste trabalho, desenvolver um protótipo para uma aplicação *web* capaz de representar dados de forma dinâmica, facultando também a sua visualização gráfica e criação de *reports* sobre os mesmos, sendo esta denominada de *Open Data @ UA* (OD@UA). Ao longo deste capítulo proceder-se-á à descrição deste protótipo.

6.1 Requisitos da aplicação

Uma aplicação informática tem, habitualmente, na sua génese, os requisitos da mesma. Antes da sua criação importa compreender de que forma uma aplicação deve satisfazer as necessidades dos seus utilizadores, bem como os aspectos a ter em conta para que estas sejam satisfeitas da melhor forma possível. Segundo Sommerville [101] os requisitos de uma aplicação informática tomam a seguinte definição:

"As descrições daquilo que o sistema deve fazer – os serviços que oferece e as restrições ao seu funcionamento."

Desta forma, neste ponto serão focados os requisitos da aplicação OD@UA sendo feita uma diferenciação entre requisitos funcionais e não funcionais.

6.1.1 Requisitos funcionais

Os requisitos funcionais de uma aplicação [101] especificam quais os serviços prestados por esta, bem como a forma como a mesma deve reagir a *inputs* ou a situações particulares. Por vezes, este tipo de requisitos [101] pode também ser utilizado para explicitar quais as acções que

determinada aplicação não deve permitir. Desta forma, importa definir os requisitos funcionais da aplicação OD@UA:

- Para a pesquisa de dados, o utilizador deve ter a possibilidade de seleccionar *queries* simples ou de criar *queries* compostas a partir das *queries* simples, tendo como ponto de partida um determinado tema e podendo, numa *query* composta, cruzar dados relativos a diversos temas distintos;
- Para conseguir criar *queries* o utilizador não deve necessitar de ter conhecimentos de informática nem conhecimentos sobre os *data providers*. Deve ser a aplicação a traduzir a linguagem entendida pelo utilizador para a linguagem entendida pelo *data provider*;
- Cada *query* deve pertencer a um determinado *namespace*. É esse *namespace* que define sobre que *data provider* será efectuada a *query*;
- Quando o utilizador selecciona *queries* simples os dados resultantes da mesma devem ser-lhe apresentados na forma de tabelas simples;
- Quando o utilizador cria *queries* compostas os dados resultantes da mesma devem ser-lhe apresentados na forma de *nested tables*, oferecendo-lhe a possibilidade de expandir e colapsar as linhas conforme a sua vontade;
- A cada *query* deve ser associada pelo menos uma *tag*, por forma a que os utilizadores identifiquem rapidamente de que trata a mesma, bem como para permitir alguma consistência semântica quando se criam *queries* compostas;
- Deve ser permitido ao utilizador a criação automática (sem a sua intervenção) de *reports* das representações tabelares dos dados resultantes das *queries* que seleccionou ou criou;
- Os tipos de visualizações gráficas estudados nesta dissertação serão os especificados na Tabela 2, sendo que a aplicação deve possibilitar a criação de cada um destes tipos;
- As visualizações gráficas devem ser criadas tendo como estrutura de suporte uma tabela simples, sendo que serão estudados dezassete tipos distintos de tabelas de suporte que permitem visualizações (especificados na Tabela 1);
- As combinações possíveis entre o tipo de tabela e tipo de visualização gráfica estudadas nesta dissertação serão as definidas na Tabela 3, sendo que a aplicação deve ter a capacidade de automaticamente seleccionar quais os tipos de visualizações gráficas a utilizar, tendo em conta o tipo da tabela de dados seleccionada pelo utilizador;
- Deve ser permitido ao utilizador seleccionar quais as colunas, bem como a sua ordem, da tabela que serve de suporte à criação das visualizações gráficas;
- A aplicação deve ter a capacidade de escolher qual a melhor escala para visualizar os dados graficamente (escala linear ou escala logarítmica);

Uma vez que os requisitos funcionais de uma aplicação especificam, além de outras coisas, quais os serviços prestados pela mesma [101], importa conhecer alguns casos de utilização permitidos a todos os tipos de utilizador da aplicação OD@UA, sendo que os mesmos estão representados na Figura 27.

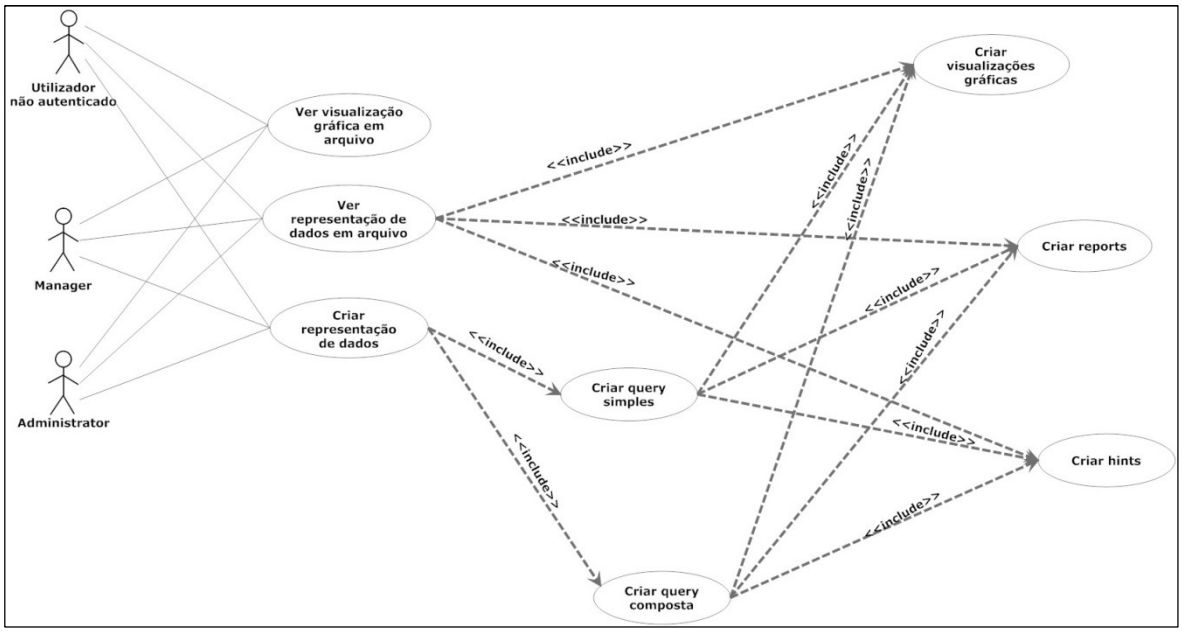


Figura 27: Diagrama de casos de utilização disponíveis a todos os tipos de utilizador da aplicação OD@UA

Além das funcionalidades disponibilizadas a todos os utilizadores, a aplicação OD@UA dispões de outras que apenas estão disponíveis a utilizadores que tenham um determinado *role*, nomeadamente "Administrator" e "Manager", sendo que as mesmas são acessíveis apenas no *back-office* da aplicação, após o utilizador se autenticar. Estas funcionalidades estão representadas no diagrama da Figura 28.

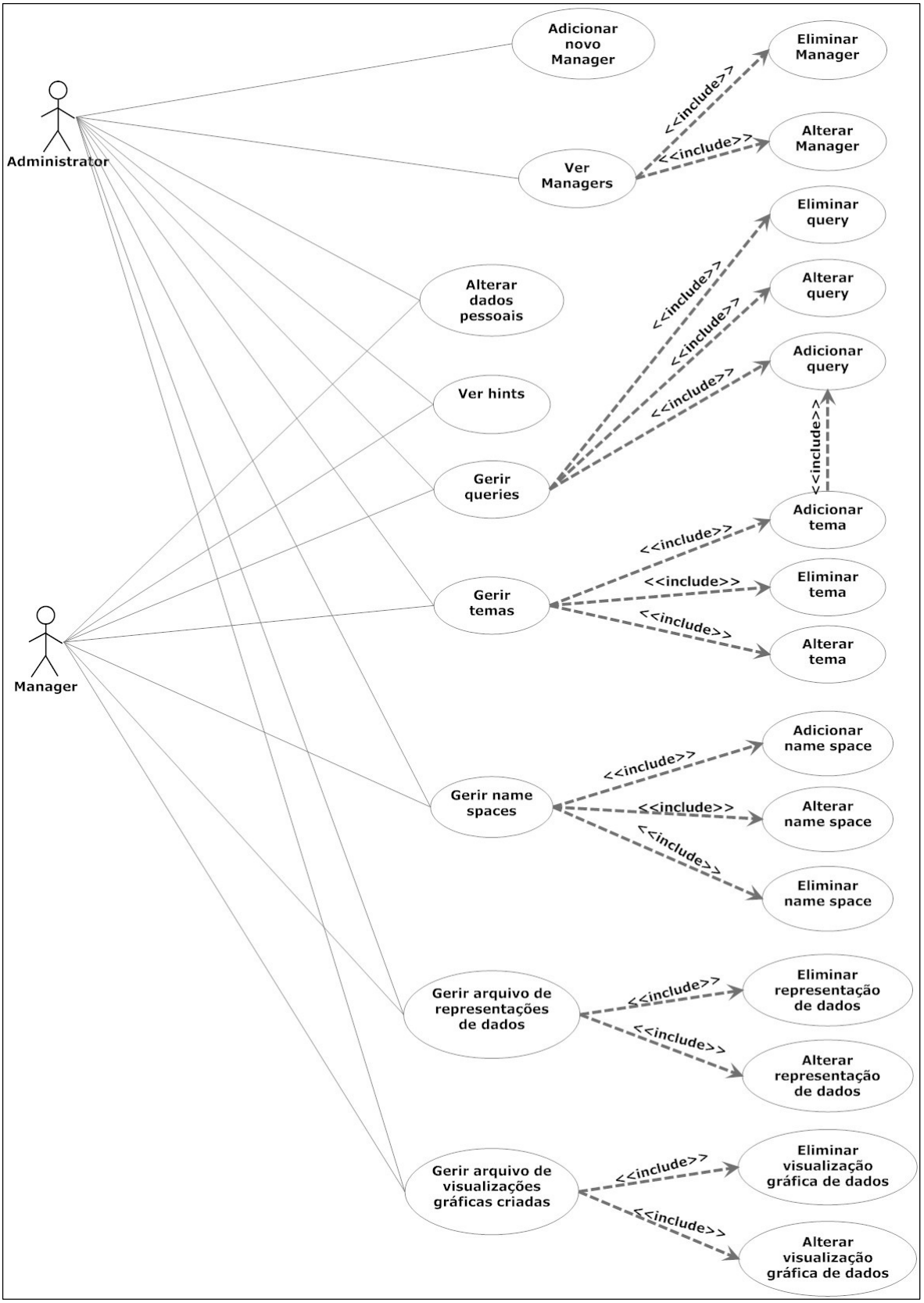


Figura 28: Diagrama de casos de utilização disponíveis no *back-office* da aplicação OD@UA

Tipo de tabela	Descrição
1cls1ro	Tabela composta por uma coluna e uma linha, em que a coluna é de tipo *string*
1clsYro	Tabela composta por uma coluna e várias linhas, em que a coluna é de tipo *string*
1cln1ro	Tabela composta por uma coluna e uma linha, em que a coluna é de tipo numérico
1clnYro	Tabela composta por uma coluna e várias linhas, em que a coluna é de tipo numérico
2clsn1ro	Tabela composta por duas colunas e uma linha, em que a primeira coluna é de tipo *string* e a segunda de tipo numérico
2clsnYro	Tabela composta por duas colunas e várias linhas, em que a primeira coluna é de tipo *string* e a segunda de tipo numérico
2clcon1ro	Tabela composta por duas colunas e uma linha, em que a primeira coluna é de tipo *string* sendo essa *string* o nome de um país ou o seu respectivo código de acordo com a norma ISO-3166-1. A segunda coluna é de tipo numérico
2clconYro	Tabela composta por duas colunas e várias linhas, em que a primeira coluna é de tipo *string* sendo essa *string* o nome de um país ou o seu respectivo código de acordo com a norma ISO-3166-1. A segunda coluna é de tipo numérico
2clnn1ro	Tabela composta por duas colunas e uma linha, em que ambas as colunas são de tipo numérico
2clnnYro	Tabela composta por duas colunas e várias linhas, em que ambas as colunas são de tipo numérico
Xclsn1ro	Tabela composta por várias colunas e uma linha, em que a primeira coluna é de tipo *string* e as restantes de tipo numérico
XclsnYro	Tabela composta por várias colunas e várias linhas, em que a primeira coluna é de tipo *string* e as restantes de tipo numérico
Xclss1ro	Tabela composta por várias colunas e uma linha, em que a as duas primeiras colunas são de tipo *string* e as restantes de tipo numérico
XclssYro	Tabela composta por várias colunas e várias linhas, em que a as duas primeiras colunas são de tipo *string* e as restantes de tipo numérico
Xclnn1ro	Tabela composta por várias colunas e uma linha, em que todas as colunas são de tipo numérico
XclnnYro	Tabela composta por várias colunas e várias linhas, em que todas as colunas são de tipo numérico
Xdefault	Tabela que não se enquadra em qualquer outro tipo

Tabela 1: Tipos de tabelas permitidas pela aplicação OD@UA como estrutura de suporte à criação de visualizações gráficas

Tipo de visualização gráfica	Descrição
area	Area Chart do GCT
bar	Bar Chart do GCT
column	Column Chart do GCT
gauge	Gauge Chart do GCT
geo	Geo Chart do GCT
line	Line Chart do GCT
pie	Pie Chart do GCT
scatter	Scatter Chart do GCT
steppedarea	Stepped Area Chart do GCT
tablechart	Tabela do GCT

Tabela 2: Definição dos tipos de visualizações gráficas permitidas pela aplicação OD@UA

	1cls1ro	1clsYro	1cln1ro	1clnYro	2clsn1ro	2clsnYro	2clcon1ro	2clconYro	2clnn1ro	2clnnYro	Xclsn1ro	XclsnYro	Xclss1ro	XclssYro	Xclnn1ro	XclnnYro	Xdefault
area						X		X				X					
bar					X	X	X	X			X	X					
column					X	X	X	X			X	X					
gauge			X		X	X	X	X	X						X		
geo							X	X									
line					X	X	X	X			X	X					
pie					X	X	X	X									
scatter									X	X					X	X	
steppedarea					X	X	X	X			X	X					
tablechart	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabela 3: Combinações possíveis (marcadas com X) na aplicação OD@UA entre o tipo de tabela e o tipo de visualização gráfica.

6.1.2 Requisitos não funcionais

Os requisitos não funcionais de uma aplicação [101] especificam as restrições nos serviços e funcionalidade oferecidos pela mesma, incluindo restrições temporais, do processo de desenvolvimento e impostas pelos *standards*. Assim sendo, os requisitos não funcionais da aplicação OD@UA são:

- Por forma a integrar o universo das aplicações da UA que correm no servidor aplicacional *Internet Information Services* da *Microsoft*, a aplicação OD@UA deve ser uma aplicação *web* que se sirva da *framework* ASP.NET, sendo desenvolvida na linguagem de programação C#;
- Deve ser utilizada a biblioteca *iText* (versão para C#) para a criação de *reports*;

- Os *data providers* aos quais a aplicação pode recorrer para obter os dados a mostrar aos utilizadores podem ser de qualquer tipo (por exemplo bases de dados, *web services* e ficheiros CSV);
- A aplicação deve ser independente dos *data providers*;
- Deve ser utilizada a ferramenta GCT para a criação das visualizações gráficas;
- Toda a aplicação, à excepção do *back-office*, deve ser de acesso livre, sem a necessidade de registo do utilizador.

6.2 Arquitectura da aplicação

O modelo arquitectural de uma aplicação [101] é habitualmente utilizado para mostrar os distintos componentes que integram a mesma, evidenciando as ligações entre eles. Desta forma, por meio de um diagrama em que figure a arquitectura de determinada aplicação, é possível compreender como esta está organizada e estruturada. Assim sendo, na Figura 29 encontra-se representada a arquitectura da aplicação OD@UA.

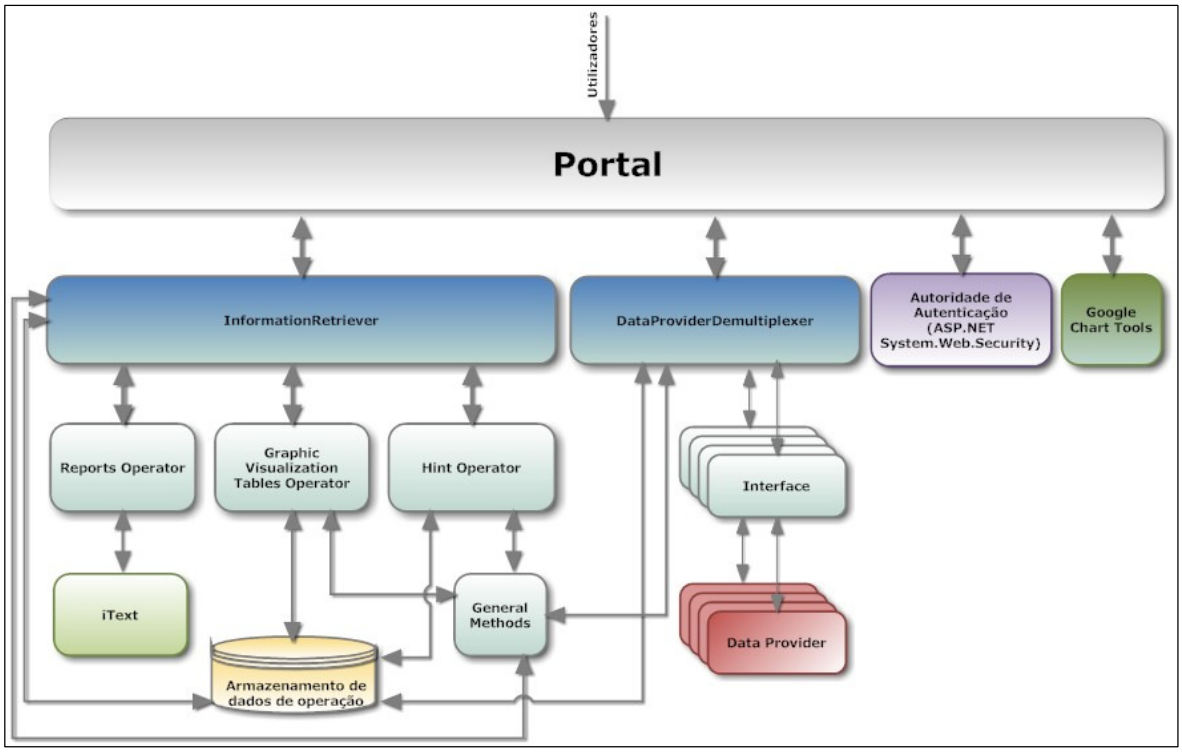


Figura 29: Arquitectura da aplicação OD@UA

Uma vez que se trata de uma aplicação *web*, uma das componentes mais importantes da aplicação OD@UA é, conforme se pode verificar na Figura 29, o "Portal".

O "Portal" é a componente a partir da qual os utilizadores interagem com a aplicação, sendo também este o responsável por controlar o fluxo de execução da mesma. O "Portal" está dividido em *back-office* e zona de acesso livre, sendo que o acesso ao *back-office* é condicionado pela autenticação do utilizador por meio da componente "Autoridade de Autenticação" que se serve de classes do *namespace System.Web.Security* da *framework* ASP.NET para proceder a essa mesma autenticação, bem como a criação de utilizadores ou *roles*.

Para conseguir criar as visualizações gráficas a partir dos dados pretendidos, a aplicação OD@UA serve-se da ferramenta GCT, sendo que a interacção entre ambas é, conforme se pode observar na Figura 29, efectuada por meio do "Portal".

Além das duas componentes apresentadas anteriormente, o "Portal" interage ainda directamente com outras duas pertencentes à arquitectura, sendo elas os *handlers* HTTP "InformationRetriever" e "DataProviderDemultiplexer". A interacção é conseguida através de *strings* formatadas em JSON trocadas entre o "Portal" e ambos os *handlers*. A razão pela qual se optou pela criação de dois *handlers* distintos foi, simplesmente, porque estes têm propósitos diferentes que se clarificarão de seguida.

O objectivo do *handler* "InformationRetriever" é efectuar a retransmissão de informações de controlo das diversas funcionalidades da aplicação, entre o "Portal" e os outros componentes da arquitectura responsáveis por essas mesmas funcionalidades, nomeadamente o "Reports Operator", o "Graphic Visualization Tables Operator", o "Hint Operator" e a base de dados "Armazenamento de dados de operação". Cada um destes componentes tem, por sua vez, o seu próprio desígnio.

Começando pelo componente "Reports Operator", importa referir que o mesmo engloba um conjunto de classes responsáveis pela criação dinâmica de *reports* baseados nos dados retornados pelas *queries* seleccionadas pelos utilizadores no "Portal". De notar ainda que este componente se serve de funcionalidade disponibilizada pela biblioteca *iText* para a criação de *reports* no formato PDF.

No que diz respeito ao componente "Graphic Visualization Tables Operator" a sua função consiste em, dada uma tabela simples contendo os dados seleccionados pelo utilizador a serem utilizados na criação de uma visualização gráfica, determinar quais os tipos de visualizações gráficas podem ser utilizados para representar esses dados, bem como determinar qual a melhor escala para o fazer (linear ou logarítmica).

Quanto ao componente "Hint Operator", o mesmo oferece funcionalidade para que os utilizadores possam deixar comentários para serem lidos pelos "managers" e "administrators". Um utilizador envia uma *hint* através do "Portal" que por sua vez irá comunicá-la, por meio do *handler*

"InformationRetriever", ao "Hint Operator" que irá então inserir a nova *hint* na base de dados "Armazenamento de dados de operação".

Finalmente, a base de dados "Armazenamento de dados de operação" tem por finalidade, tal como o seu nome indica, armazenar todos os dados necessários ao funcionamento da aplicação OD@UA, sendo as suas características explicitadas mais adiante nesta dissertação.

Em relação ao *handler* "DataProviderDemultiplexer", o seu objectivo é efectuar o agulhamento no processo de troca de informações entre a aplicação e o "Interface" com o "Data Provider" pretendido, ou seja, encaminha os dados provenientes da aplicação para o "Interface" correcto e vice-versa. Já cada um dos componentes "Interface" permite a comunicação entre o "Data Provider"ao qual está associado e a aplicação, sem que ambos tenham conhecimentos dos pormenores um do outro, ficando esse conhecimento reservado a cada "Interface". Quanto ao "Data Provider" é uma fonte de dados a partir da qual a aplicação OD@UA irá retirar os dados que pretende por meio de *queries*. De notar que a arquitectura do conjunto de componentes "DataProviderDemultiplexer", "Interface" e "Data Provider" foi inspirada na arquitectura de ODBC.

Existe ainda um outro componente na arquitectura da aplicação OD@UA, o "General Methods", que tem por objectivo oferecer funcionalidade a ser utilizada por vários dos outros componentes já descritos. Este componente não é mais que uma classe estática que disponibiliza vários métodos úteis aos outros componentes da arquitectura.

Concluindo, tendo em conta a arquitectura descrita anteriormente, facilmente se percebe que o "Portal" é o suporte lógico responsável por integrar as funcionalidades oferecidas pelos restantes componentes.

6.3 Modelo de dados da aplicação

A aplicação OD@UA não armazena quaisquer colecções de dados para a sua posterior representação aos utilizadores. Uma vez que estes se encontram disponíveis nos *data providers*, optou-se por não se replicar os mesmos na aplicação em si. Assim sendo, a aplicação apenas armazena informação que permita ao utilizador aceder aos dados que pretenda de uma forma automática e sem que este tenha conhecimento das particularidades do *data provider* de onde estes provêm. Desta forma, é possível dinamizar o processo de inclusão de um novo *data provider*, não sendo necessário efectuar alterações ao nível da programação da aplicação sempre que um *data provider* é adicionado.

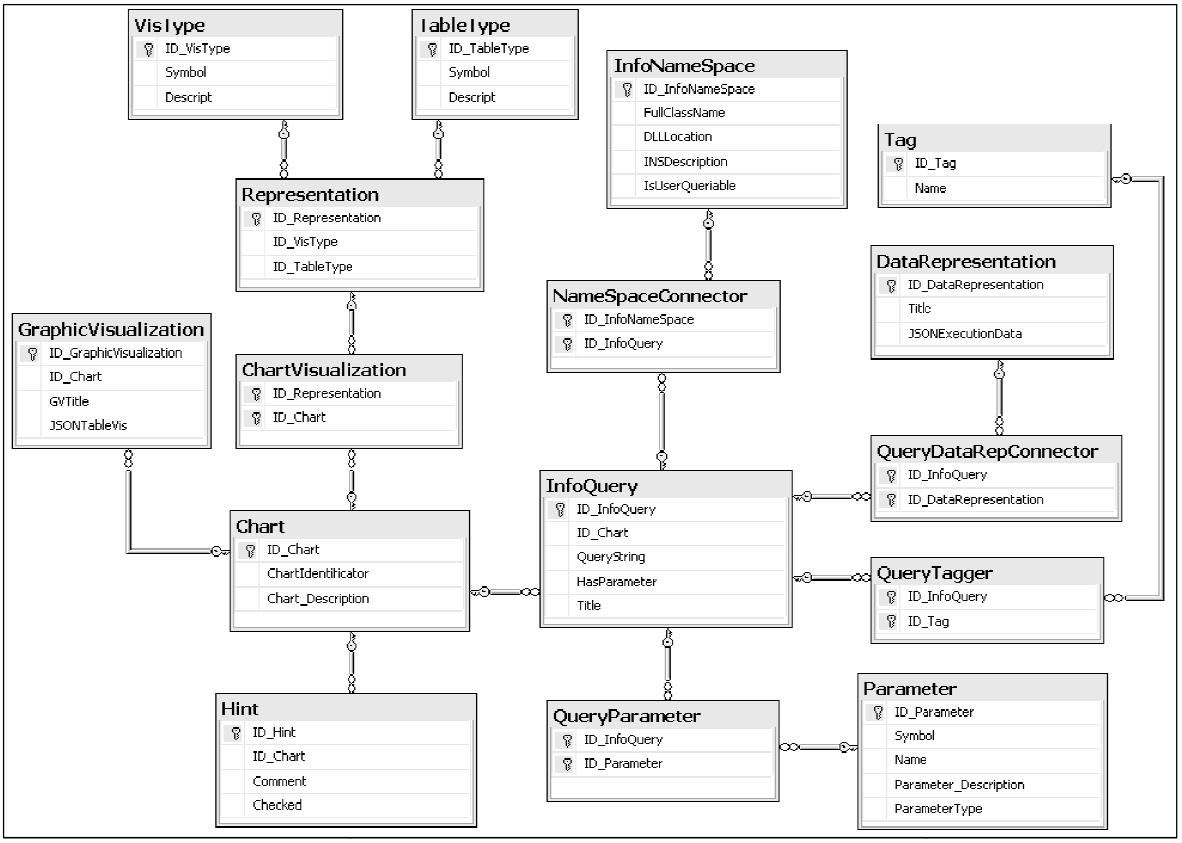


Figura 30: Diagrama da base de dados "Armazenamento de dados de operação"

Na Figura 30 está representado o diagrama da base de dados "Armazenamento de dados de operação". Os dados armazenados nas diversas tabelas desta base de dados serão utilizados pela aplicação OD@UA na sua execução para, entre outras utilizações, determinar quais as *queries* simples que podem ser encadeadas em *queries* compostas, na decisão de como serão visualizados determinados dados ou para determinar qual o *data provider* que serve de fonte a esses mesmos dados.

Importa compreender qual utilidade para a aplicação de cada uma das tabelas representadas no diagrama da Figura 30. Por conseguinte, essa utilidade será aqui discutida. Os comandos para a criação e inicialização da base de dados representada na Figura 30 encontram-se no Anexo A1.

Quando um utilizador pretende criar uma nova representação de dados, deve primeiro escolher qual o tema que pretende explorar, como por exemplo "Registos na Biblioteca" ou "Informações de Cursos na Universidade", sendo que fica a cargo do "manager" a criação de novos temas. A tabela do diagrama da Figura 30 onde são armazenados os temas é a tabela "Chart". Nesta tabela estão contidas informações relativas ao nome do tema no campo "Chart_Description" e ao identificador apresentado ao "manager" no campo "ChartIdentificator". Esta tabela tem um nome que não corresponde à sua função devido ao facto de a aplicação ter sido repensada várias vezes no decorrer deste trabalho. Tendo a aplicação adquirido um grau de complexidade elevado, quando

finalmente ficou estável já era tarde para se efectuar um *refactoring* à mesma. Talvez os nomes "Theme" ou "Subject" fossem os mais indicados para esta tabela.

A cada tema (tabela "Chart") podem estar associadas várias *queries* (tabela "InfoQuery"), sendo estas as *queries* a partir das quais os utilizadores podem conseguir os dados para criar as representações e visualizações que pretendem. Conforme já foi referido anteriormente o utilizador não necessita de ter qualquer conhecimento de informática para utilizar estas *queries*. O que acontece é que ao utilizador apenas é apresentado o título da *query*, como por exemplo "Notas de acesso à Universidade por ano" ou "Fases de acesso por curso", sendo-lhe assim escondida a complexidade da mesma. É no campo "QueryString" da tabela "InfoQuery" que é armazenada a *query* a ser executada sobre um determinado *data provider*. Depois de executada, o retorno de cada *query* é uma tabela de dados. Importa referir que, dependendo dos *data providers* sobre os quais vão ser executadas estas *queries*, as mesmas podem tomar o formato de métodos caso os *data providers* não permitam a execução directa de *queries*, sendo que nesse caso o campo "QueryString" irá conter o nome do método em questão. Por uma questão de simplicidade de linguagem, ao longo desta dissertação, o termo *query* é utilizado indiferenciadamente para ambas as hipóteses.

Outro aspecto importante é o de que as *queries* podem ter parâmetros (armazenados na tabela "Parameter"). Estes parâmetros são o que permite encadear as *queries*, construindo uma *query* composta, ou seja, o *output* da *query* anterior vai ser, na sua totalidade ou apenas as colunas seleccionadas pelo utilizador, o *input* da próxima *query* na cadeia. Para que o encadeamento seja possível, apenas *queries* com parâmetros podem ser usadas após a selecção da *query* raiz. Além disso, os tipos dos parâmetros devem coincidir, ou seja, quando o *output* de uma *query* é uma tabela em que todas as colunas são do tipo *string*, esta só pode ser encadeada com outras *queries* que tenham pelo menos um parâmetro deste tipo.

Outro dos aspectos importantes no encadeamento das *queries* são as *tags* a elas associadas. Conforme se pode observar na Figura 30, cada *query* tem a si associada pelos menos uma *tag* (tabela "Tag"). Estas *tags*, para além de servirem para dar a conhecer ao utilizador dicas semânticas sobre os dados que estão relacionados com cada *query*, são também utilizadas na política de encadeamento das *queries* da seguinte forma: a próxima *query* a ser incluída no encadeamento deve ter, pelo menos, uma *tag* que resulte da conjunção entre as *tags* da *query* anterior e da *query* actual. A título de exemplo, imaginando que à *query* anterior correspondem as *tags* "A", "B", "C" e "D" e à *query* actual as *tags* "A" e "C", a próxima *query* a ser incluída no encadeamento deverá ter a si associada pelo menos uma de entre as *tags* "A" e "C". Desta forma, e tendo em conta as condições expostas anteriormente relativas ao tipo dos parâmetros das *queries*, é possível manter alguma validade semântica no encadeamento das mesmas. Importa referir que o utilizador não precisa de se

preocupar com estas condições, pois apenas as *queries* que as respeitam lhe são mostradas em cada instante.

A associação entre uma *query* e um determinado *data provider* é feita por intermédio das tabelas "NameSpaceConnector" e "InfoNameSpace" do diagrama da Figura 30. Um *namespace* armazenado na tabela "InfoNameSpace" contém informações sobre a classe que implementa o interface que permite criar a ligação entre um *data provider* e a aplicação OD@UA para uma determinada *query*. As informações mais importantes contidas nesta tabela dizem respeito à localização da *Dynamic-Link Library* (DLL) relativa ao *namespace* e o nome da classe que implementa o interface com o *data provider*.

Como seria de esperar, as *queries* não necessitam de pertencer ao mesmo *namespace* para poderem ser encadeadas. Basta apenas que respeitem as condições de *tags* e tipos dos parâmetros descritas anteriormente.

Sempre que um utilizador cria uma *query* composta, encadeando *queries* simples, servindo-se para isso das funcionalidades oferecidas pelo "Portal", vai sendo criada uma estrutura de dados utilizando a notação JSON que contém os dados que a aplicação necessita para executar todas as *queries* na cadeia. Logo que o utilizador decida que a sua *query* composta está completa, pode informar a aplicação deste facto. Assim que isso suceda a estrutura JSON vai ser armazenada na base de dados na tabela "DataRepresentation", ficando associada às *querys* que integram o encadeamento. Desta forma, outro utilizador não necessita de repetir todo o processo para ver a mesma representação de dados, uma vez que basta apenas seleccionar, por meio do seu título, a representação que pretende, já que ao título da representação está associada na tabela "DataRepresentation" a estrutura JSON no campo "JSONExecutionData", onde se encontram todos os dados necessários para o processo ser automaticamente repetido pela aplicação.

Depois de lhe ser apresentada a representação dos dados escolhidos, na forma de uma tabela simples ou *nested table*, o utilizador pode optar por criar um *report* sobre a mesma ou criar visualizações gráficas servindo-se dos dados. Para escolher quais os dados a serem visualizados graficamente estes serão expostos ao utilizador por intermédio de uma tabela simples, onde ele deve escolher quais as linhas e colunas que pretende. Essas linhas e colunas vão, por sua vez, formar uma outra tabela cujo tipo será determinado pela aplicação a partir do universo da Tabela 1, correspondendo o "Tipo de Tabela" da Tabela 1 ao campo "Symbol" da tabela "TableType" do diagrama da Figura 30. Depois de determinado qual o tipo da tabela, a aplicação vai servir-se do conteúdo da tabela "Representation" do diagrama da Figura 30 para determinar quais as visualizações gráficas permitidas para um determinado tema (tabela "Chart"), tendo em conta o tipo da tabela de dados. De notar que ao campo "Symbol" da tabela "VisType" do diagrama da

Figura 30 corresponde cada um dos "Tipo de visualização gráfica" da Tabela 2, enquanto que as combinações da Tabela 3 estão armazenadas na tabela "Representation".

As visualizações gráficas dos dados, tal como as representações dos mesmos descritas anteriormente, também são armazenadas para utilização posterior, sendo este armazenamento efectuado na tabela "GraphicVisualization" e, num processo idêntico ao já referido para as representações gráficas, a estrutura JSON necessária à criação da visualização fica armazenada no campo "JSONTableVis" desta mesma tabela. Importa referir que as visualizações armazenadas ficam associadas ao tema (tabela "Chart") da *query* raiz que lhes deu origem, e mesmo que as *queries* deixem de existir na aplicação a visualização dos dados pode manter-se no sistema enquanto existir este tema, uma vez que os dados necessários a sua criação estão todos guardados no campo "JSONTableVis".

O utilizador pode ainda criar *hints* associadas aos vários temas, que ficarão disponíveis aos "manager" da aplicação, sendo que estas serão armazenadas na tabela "Hint" da base de dados representada no diagrama da Figura 30.

6.4 Criação de *reports*

Uma das funcionalidades mais importantes da aplicação OD@UA é a criação de *reports* a partir dos dados obtidos pelas *queries* escolhidas pelo utilizador. Por esta razão, importa conhecer mais detalhadamente o componente "Reports Operator" do diagrama da Figura 29, uma vez que esta responsabilidade lhe está atribuída.

Numa fase inicial deste trabalho procurou-se, para além de representar os dados nos *reports* na forma tabelar, permitir também a inclusão de visualizações gráficas dos mesmos. Para se conseguir isso utilizou-se a ferramenta *Google Image Charts* [102], mas devido às grandes limitações da mesma, nomeadamente na quantidade de dados que era possível conter em cada visualização, optou-se por abandonar esta funcionalidade.

O componente "Reports Operator" engloba diversas classes, estando estas relacionadas entre si recorrendo a "composições" e "agregações". Segundo Fowler [103] uma "agregação" é uma relação do tipo "parte de", ou seja, por exemplo, uma pessoa faz "parte de" um clube. Também a "composição" diz respeito a uma relação "parte de". No entanto, ao contrário da "agregação", numa relação de "composição" [103], apesar de uma classe poder ser um componente de várias classes distintas, qualquer instância desta deve pertencer a apenas um dono.

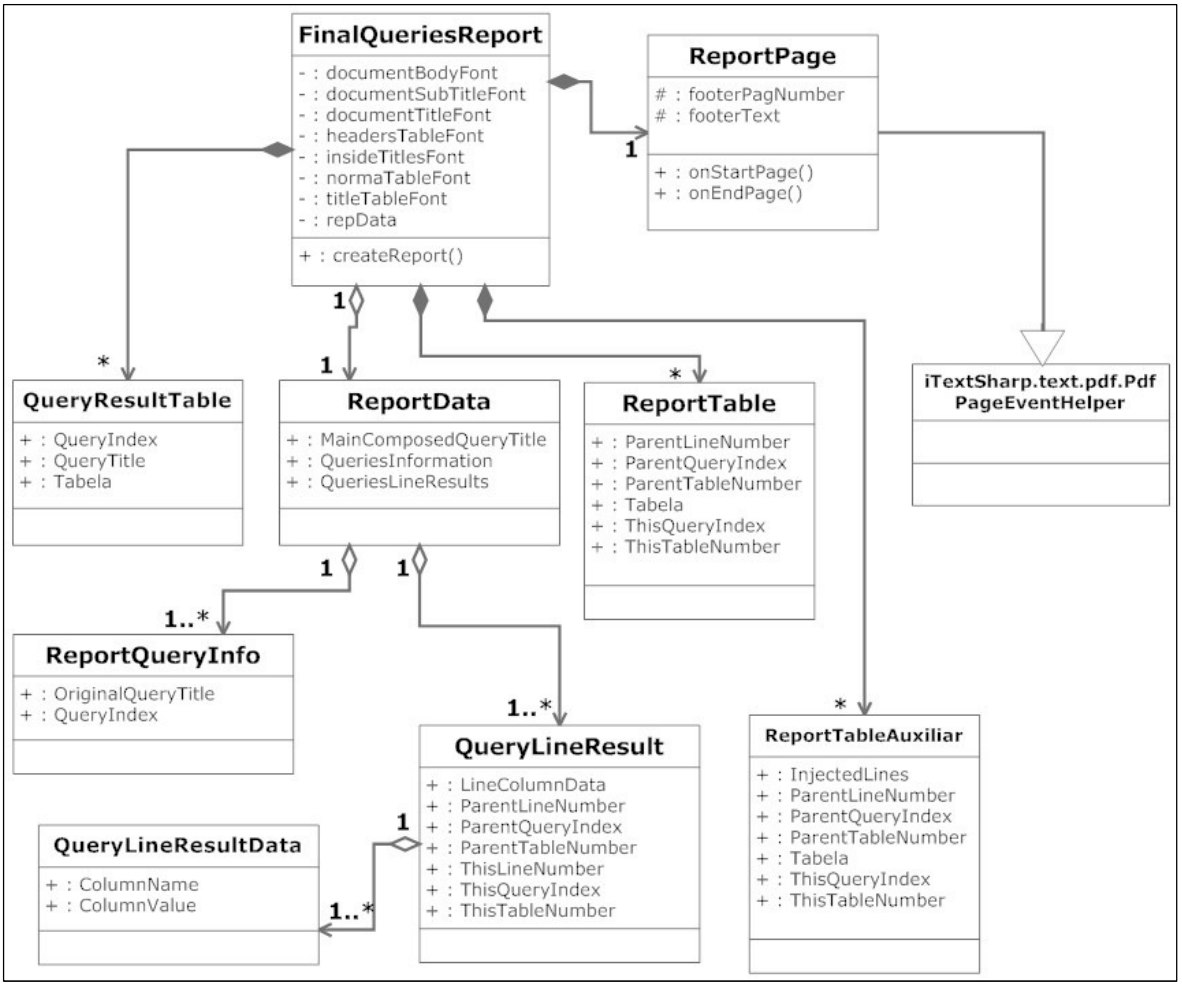


Figura 31: Diagrama de classes do componente "Reports Operator" da aplicação OD@UA

Para que melhor se perceba qual a forma utilizada pela aplicação OD@UA na criação de *reports*, será aqui descrita de forma sucinta esse processo.

Após o utilizador ter criado uma representação dos dados retornados pelas *queries* simples ou compostas que seleccionou, é-lhe dada a possibilidade de criar um *report* sobre a mesma, por meio de uma operação específica para esse efeito. Ao fazê-lo, o portal cria uma estrutura de dados servindo-se da notação JSON, estrutura essa que irá armazenar toda a informação sobre a representação de dados em questão. Essa estrutura será depois encaminhada para o *handler* HTTP "InformationRetriever" que, servindo-se de funcionalidades da componente "General Methods" da aplicação, irá converter a estrutura JSON num objecto C#, utilizado pelo componente "Reports Operator" para criar o *report*.

Inicialmente, servindo-se do objecto contendo as informações relativas à representação dos dados, o *handler* "InformationRetriever" irá criar um *array* de objectos instâncias da classe "ReportQueryInfo" representada na Figura 31, contendo informações sobre cada uma das *queries*, caso os dados sejam relativos a uma *query* composta, ou sobre a *query* simples à qual

correspondem os dados. Seguidamente, é também criado um *array* de objectos instâncias da classe "QueryLineResult" da Figura 31, que por sua vez vão ter agregados, a cada um deles, um *array* de objectos instâncias da classe "QueryLineResultData".

As informações contidas em cada objecto do tipo "ReportQueryInfo" são respeitantes ao título e índice, caso a *query* seja composta, de cada uma das *queries* simples no universo da *query* composta. Caso os dados provenham de apenas uma *query* simples o índice da mesma será 0. Já cada objecto do tipo "QueryLineResult" contém informações sobre cada uma das linhas da representação de dados resultante da *query* simples ou composta, sendo que é nos objectos "QueryLineResultData" associados a cada objecto "QueryLineResult" que estão os valores dos dados, ou seja, o nome da coluna e o seu valor. Importa compreender que a representação de uma *query* composta se faz utilizando uma *nested table*. No entanto, os dados são passados do "Portal" ao *handler "*InformationRetriever" por linha, em que cada uma das linhas pertence a uma tabela, sendo que essa tabela não é a *nested table* da representação final dos dados, mas sim uma de entre as tabelas resultantes do *output* de cada uma das *queries* tendo por *input* cada uma das linhas do *output* da *query* anterior. Desta forma, cada objecto "QueryLineResult" vai conter informação sobre a linha pai que deu origem à linha à qual o objecto diz respeito, conforme se pode ver na Figura 31.

Depois de criados os *arrays* de objectos dos tipos "QueryLineResult" e "ReportQueryInfo" será criado um objecto que irá agregar estes dois *arrays*, contendo assim todos os dados necessários à criação de um *report*. Esse objecto será do tipo "ReportData", conforme a Figura 31.

O objecto do tipo "ReportData" criado será por sua vez utilizado pelo construtor da classe "FinalQueriesReport", dando assim origem ao objecto a partir do qual se irá obter o *report*. A classe "FinalQueriesReport" disponibiliza um método denominado "createReport", o qual é responsável por criar o *report* e devolver a localização do mesmo no servidor onde a aplicação está a ser executada.

Para além de agregar a classe "ReportData", a classe "FinalQueriesReport" é composta por outras quatro classes distintas, como se pode observar na Figura 31. Três destas classes irão compor a "FinalQueriesReport" apenas se a *query* à qual o *report* diz respeito for composta. Caso contrário, somente a classe "ReportPage" irá compor a classe "FinalQueriesReport".

A classe "ReportPage" vai herdar a partir da classe "iTextSharp.text.pdf.PdfPageEventHelper", sendo que vai fazer *override* a dois métodos desta: "onStartPage" e "onEndPage". Estes dois métodos podem ser vistos como *event handlers*, sendo que o método "onStartPage" vai ser executado logo que uma página do *report* começa a ser criada, enquanto que o método "onEndPage" irá ocorrer logo que a mesma página esteja completa. Desta forma, é possível utilizar estes métodos para incluir um cabeçalho e um rodapé em cada página do

report, sendo essa a razão pela qual a classe "FinalQueriesReport" é composta pela classe "ReportPage".

A classe "iTextSharp.text.pdf.PdfPageEventHelper", tal como o seu nome completo indica, pertence à biblioteca *iText*. Também a classe "FinalQueriesReport" vai ser composta por outras classes desta biblioteca, como por exemplo "iTextSharp.text.pdf.PdfCell" e "iTextSharp.text.pdf.PdfTable" que têm como função a criação de uma célula e de uma tabela, respectivamente, tendo-se optado por não incluir as mesmas no diagrama da Figura 31 por uma questão de simplificação.

A razão pela qual se optou por utilizar a biblioteca *iText* para auxiliar na criação dos *reports* é devido ao facto de esta ser simples e intuitiva, oferecendo bastante controlo sobre o processo de criação dos mesmos. Apesar de existir pouca documentação disponível para esta biblioteca, é possível, após um processo de "tentativa e erro", compreender o funcionamento da mesma por forma a criar os *reports* que se pretendem.

Importa também conhecer as restantes três classes que compõem a classe "FinalQueriesReport". Uma dessas classes é a "ReportTable", utilizada para representar, numa *query* composta, a tabela resultante da execução de uma *query* simples integrante do encadeamento, servindo-se do *output* de uma das linhas da *query* executada anteriormente como *input* dos seus parâmetros. Um objecto instanciado a partir desta classe vai armazenar essa mesma tabela e algumas informações a ela relativas, nomeadamente o índice da *query* que lhe deu origem, bem como informações que possibilitam a identificação de quais os dados que foram utilizados como *input*. Assim sendo, uma lista contendo objectos deste tipo pode ser vista como uma árvore que contém todas as tabelas resultantes da *query* composta, sabendo-se qual a linha resultante de cada *query* simples que serviu de *input* à *query* simples seguinte. Uma vez que esta lista armazena todas as informações sobre os resultados da *query* composta e as relações entre as diversas tabelas, é possível utilizar a mesma para incluir no *report*, em primeiro lugar, uma *nested table* com os resutados desta *query* composta, em segundo lugar, as várias tabelas resultantes de cada uma das *queries* simples que compõem a *query* composta.

Para construir a *nested table* será criada uma lista de objectos do tipo "ReportTableAuxiliar" que vai ser um clone da lista de objectos do tipo "ReportTable" referida anteriormente. Esta lista clonada será então processada até se transformar numa *nested table* sendo então incluída no *report*. Já para criação das tabelas resultantes de cada uma as *queries* simples, a lista de objectos "ReportTable" vai ser processada por forma a ser transformada numa lista de objectos "QueryResultTable", sendo que cada objecto desta lista corresponderá a uma tabela resultado de cada *query* simples. Farão parte de cada tabela todos os resultados relativos à mesma

query, inclusivamente quando os parâmetros que lhe deram origem sejam distintos. Logo que a lista esteja completa estas tabelas serão incluídas no *report*.

Finalmente, depois de tudo o que foi descrito anteriormente, o método "createReport" retorna a localização do *report* no servidor, sendo esta devolvida ao *browser* do utilizador por meio do *handler* "InformationRetriever" que permitirá então ao utilizador visualizar e, se necessário, salvar o *report* no seu computador.

6.5 Criação de visualizações gráficas

A criação de visualizações gráficas a partir dos dados provenientes das *queries* seleccionadas pelos utilizadores é uma das funcionalidades de maior interesse da aplicação OD@UA. Desta forma, compreender de que maneira a aplicação produz estas visualizações é muito importante para uma melhor compreensão da mesma como um todo.

Depois de ser criada a representação dos dados provenientes da *query* simples ou composta seleccionada pelo utilizador, é permitida a criação de visualizações gráficas utilizando esses mesmos dados, bastando para isso que o utilizador indique no "Portal" essa intenção por meio de um botão específico para esse efeito. Logo que essa intenção seja comunicada ao "Portal", tal como acontece no caso dos *reports*, é criada uma estrutura de dados na notação JSON contendo todas as informações relativas à representação dos dados.

Ao contrário do que acontece no caso dos *reports*, essa estrutura não é imediatamente enviada para um *handler* HTTP. Em vez disso, ela é processada pelo "Portal" e a partir dela é criada uma tabela simples, independentemente da *query* que lhe deu origem ser simples ou composta. No caso da *query* ser composta, cada linha dessa tabela corresponde a um caminho de execução possível das várias *queries* simples que a compõem, uma vez que cada linha da tabela resultante da *query* anterior vai servir de *input* à *query* seguinte. No caso da *query* ser simples, cada linha da tabela corresponde apenas aos resultados da execução da *query*.

A tabela criada é então apresentada ao utilizador, dando-lhe a possibilidade de escolher quais as linhas e colunas que quer incluir numa visualização gráfica, bem como a ordem das colunas. Neste passo, o utilizador pode seleccionar várias visualizações distintas a partir dos mesmos dados, sendo que a cada uma delas deverá atribuir um título que a distinga das outras. Logo que esteja satisfeito com as selecções que efectuou pode indicar ao "Portal" que pretende avançar para a criação automática das visualizações, mediante opção para esse efeito. Neste caso, o "Portal" vai criar uma nova estrutura de dados formatada em JSON, contendo toda a informação necessária à criação de visualizações gráficas, sendo esta enviada para o *handler* "InformationRetriever". Assim que a estrutura de dados é recebida pelo *handler*, esta vai ser

transformada em três listas distintas. A primeira irá ser uma lista de objectos do tipo "DataTable", sendo que cada objecto representa uma tabela para a qual será criada uma visualização gráfica. A segunda será uma lista de *strings*, em que cada um dos elementos da lista é o título de cada uma das tabelas de dados. Já a última é também uma lista de *strings*, mas que servirá para armazenar o *id* do tema (tabela "Chart" da Figura 30) ao qual pertence a *query* raiz (no caso de uma *query* composta), ou a *query* simples (em caso contrário). Depois de criadas as listas, estas vão ser encaminhadas para o componente "Graphic Visualization Tables Operator" representado na Figura 29. Este encaminhamento é efectuado através da instanciação no *handler* "InformationRetriever" de um objecto do tipo "TableProcessor", sendo que o mesmo irá receber como parâmetros no seu construtor as várias listas mencionadas anteriormente.

A classe "TableProcessor", representada no diagrama da Figura 32, é a mais importante do componente "Graphic Visualization Tables Operator", pois é a partir dela que é desencadeado todo o processamento das tabelas para determinar dinamicamente as formas possíveis de visualização gráfica. Este processamento é efectuado pelo método "process" desta classe, sendo o mesmo invocado pelo *handler* "InformationRetriver", dando início a todo o processo.

Conforme se pode observar na Figura 32, a classe "TableProcessor" é composta por várias outras classes, sendo que estas relações de composição irão auxiliar no processamento das tabelas da forma que se passa então a descrever.

Em primeiro lugar, o método "process" irá instanciar um objecto do tipo "TableTypeDeterminer", objecto este que irá servir-se do método "determine" para determinar a quais tipos de entre os da Tabela 1 pertencem as tabelas para as quais se pretende criar as visualizações gráficas. De notar que estes tipos estão declarados como constantes na classe estática "TableType", classe essa da qual se servirá o método "determine".

Depois de determinados quais os tipos das tabelas, o método "process" do objecto do tipo "TableProcessor" irá determinar qual a escala (logarítmica ou linear) que melhor se adequa à visualização, tendo em conta os dados a incluir nela. Para atingir este fim vai servir-se da classe "PreferredScaleTypeDeterminer" e do seu método "determine", criando um objecto deste tipo, que por sua vez utilizará as constantes declaradas na classe estática "PreferredScaleType" neste processo.

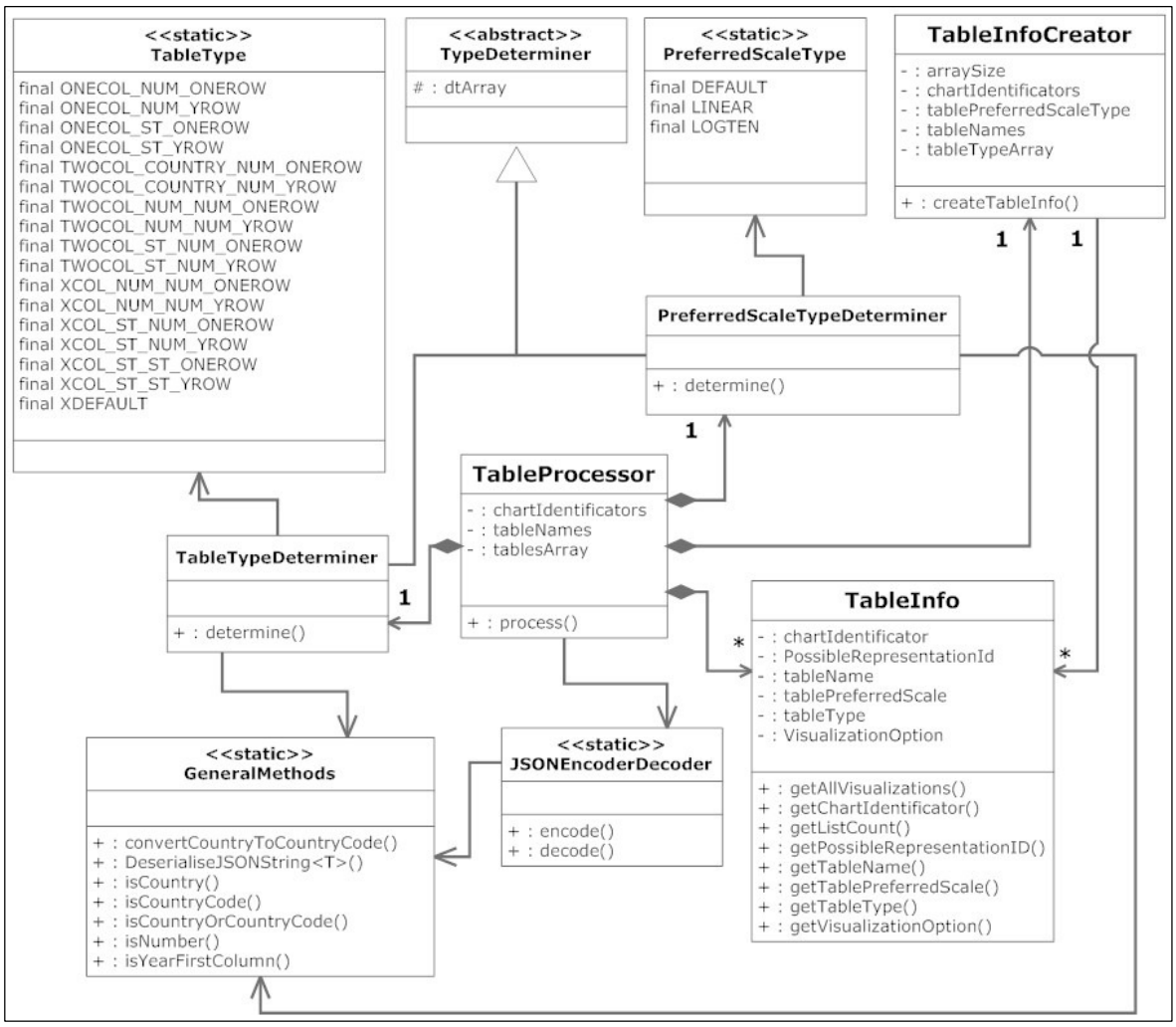


Figura 32: Diagrama de classes do componente "Graphic Visualization Tables Operator" da aplicação OD@UA

Como se pode observar na Figura 32, importa referir que tanto a classe "TableTypeDeterminer" como a "PreferredScaleTypeDeterminer" vão herdar da classe "TypeDeterminer", sendo esta uma generalização de ambas.

Os resultados obtidos a partir dos processos de determinação de tipo de tabela, bem como de tipo de escala, vão ser armazenados em dois *arrays* de *strings* distintos, sendo estes utilizados em conjunto com a lista dos nomes das tabelas, bem como a dos *id* dos temas (tabela "Chart" da Figura 30), na criação de um objecto do tipo "TableInfoCreator", uma vez que são passados como parâmetro ao seu construtor.

A classe "TableInfoCreator" dispõe de um método "createTableInfo" que, partindo dos dados que são passados ao construtor desta classe, permite a criação de um *array* de objectos do tipo "TableInfo", contendo cada um deles todas as informações necessárias à criação das visualizações gráficas para cada tabela de dados. Depois de executado este método, ele retorna o *array* de objectos "TableInfo" por valor ao método "process" do objecto do tipo "TableProcessor",

como representa a relação de composição entre estas duas classes do diagrama da Figura 32. Ao receber este *array*, o objecto do tipo "TableProcessor" vai servir-se da classe estática "JSONEncoderDecoder", mais concretamente do método "encode" da mesma, para codificar o *array* de objectos "TableInfo" numa *string* formatada em JSON que represente esse mesmo *array*. Logo que este processo esteja concluído, o método "process" do objecto do tipo "TableProcessor" termina e retorna ao *handler* "InformationRetriever" a *string* criada.

Depois de concluído o que foi referido anteriormente, o *handler* "InformationRetriever" vai repetir o mesmo processo, mas agora para cada uma das tabelas individualmente, em vez de em simultâneo como anteriormente, para criar uma *string* JSON com todas as informações necessárias à criação de uma visualização gráfica para cada uma das tabelas de dados. As *strings* resultantes de cada processamento serão então armazenadas na base de dados "Armazenamento de dados de Operação" (Figura 29) na tabela "GraphicVisualization" (Figura 30), permitindo assim que as visualizações gráficas possam ser utilizadas no futuro sem que seja necessário repetir todo o processo. Finalmente, o *handler* devolve para o "Portal" a *string* JSON que contém as informações relativas à criação das visualizações de todas as tabelas.

Quando o "Portal" recebe a *string* converte-a imediatamente num objecto *JavaScript*. A partir deste objecto vai determinar quantas visualizações gráficas são necessárias apresentar ao utilizador. Logo que esta tarefa esteja concluída será gerado dinamicamente *markup HyperText Markup Language* (HTML) para os *containers* de cada visualização. A informação sobre os *containers*, bem como sobre a visualização que este irá incluir, será então processada por funções disponibilizadas pela biblioteca *ChartOperations.js*, criada no decorrer deste trabalho para esse efeito, que por sua vez se irá servir da ferramenta GCT para a criação e *render* das visualizações gráficas que serão então apresentadas ao utilizador.

A opção pela ferramenta GCT para a criação e *render* das visualizações gráficas prende-se principalmente com a simplicidade na criação das mesmas, bem como a existência de uma documentação vasta que permite compreender a ferramenta com algum detalhe. Além disso, ao contrário de outras ferramentas, a forma como os dados lhe são passados é uniforme, por meio de uma estrutura "DataTable", a qual já foi referida no capítulo 4.1 desta dissertação, o que facilita em muito o processo. Também a disponibilização, por parte desta ferramenta, de uma biblioteca com variados tipos de visualizações gráficas foi um ponto favorável à sua escolha. Por fim, o controlo que a mesma permite sobre as visualizações criadas, nomeadamente nos tipos e tamanhos de letra utilizados, nas cores, nos tamanhos das visualizações, pesaram também para que esta fosse a ferramenta eleita para incluir na aplicação OD@UA.

Importa referir que apesar de ter sido implementada a técnica "alterar tamanho dos pontos", mencionada no capítulo 4 desta dissertação, tendo sido, neste caso, aplicada ao texto dos

eixos de cada visualização reduzindo-se dinamicamente o tamanho de letra utilizado nestes quanto maior fosse a colecção de dados para a qual se pretendesse criar a visualização gráfica, a mesma não foi suficiente para eliminar a desordem visual que pode ocorrer caso uma visualização seja composta por uma colecção de dados muito extensa. Para conseguir eliminar estas situações, o ideal teria sido a inclusão de funcionalidades de "zoom" ou "fish-eye", conforme as mesmas são descritas no capítulo 4 desta dissertação, em cada visualização, mas que no entanto não foi possível implementar devido a restrições temporais.

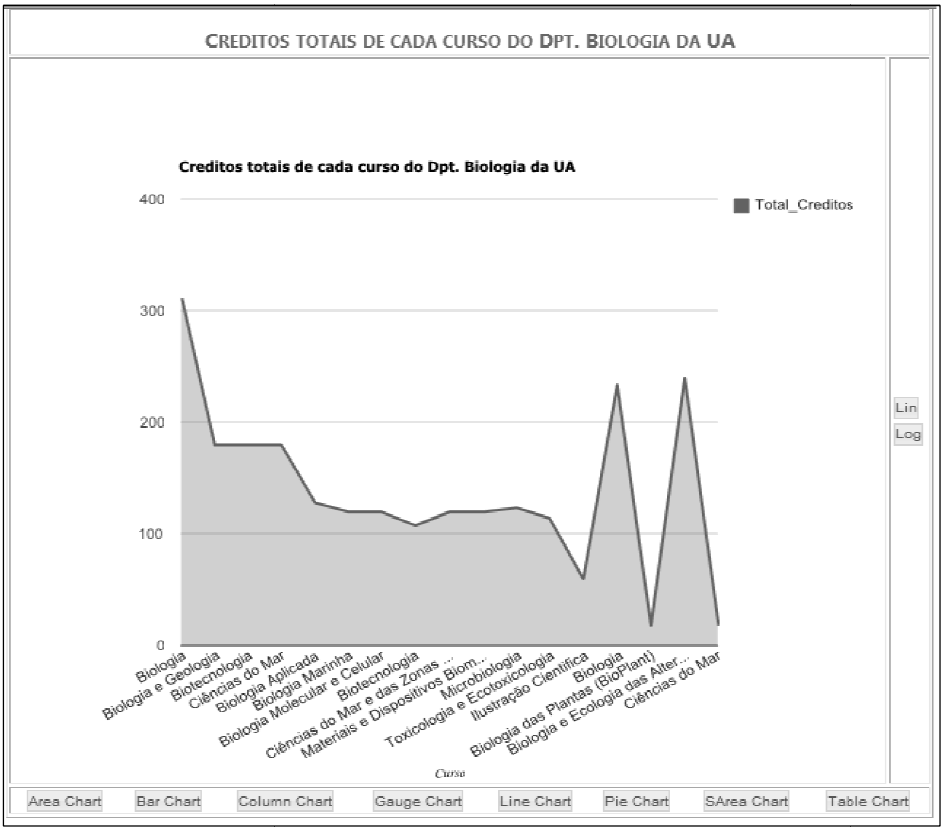


Figura 33: Exemplo de uma visualização gráfica criada pela aplicação OD@UA

Por forma a tentar diminuir a desordem visual de cada visualização gráfica, o texto dos eixos de cada uma delas é também rodado dinamicamente de acordo com o número de pontos de dados, ou seja, o grau de rotação para poucos pontos é de 0 graus, sendo que vai aumentando até 90 graus consoante as visualizações tenham mais pontos de dados.

No que diz respeito às visualizações, bem como à aplicação OD@UA no seu todo, foram também tidas em conta algumas das heurísticas descritas no capítulo 4 desta dissertação, por forma a melhorar a experiência do utilizador, nomeadamente "acção mínima", "organização espacial", "consistência" e "reconhecimento em vez de lembrança". Alguns exemplos destas heurísticas podem ser vistas na Figura 33. O utilizador não necessita de muito trabalho para alterar o tipo ou

escala de uma visualização, basta apenas clicar no botão correspondente à mudança que pretende efectuar e a mesma terá automaticamente lugar, ou seja "acção mínima". Também no que respeita à "organização espacial" houve o cuidado de se criar uma estrutura para cada visualização que definisse claramente os espaços, conforme é possível observar na Figura 33 onde existe um espaço para o título da visualização, para a visualização em si e para os botões de mudança de escala ou tipo de visualização. No que se relaciona com a heurística de "reconhecimento em vez de lembrança", em cada visualização a função dos botões a ela associados está descrita no próprio botão, não sendo assim necessário o utilizador memorizar cada um deles. Finalmente, uma vez que para cada visualização os critérios são os mesmos, ocorre "consistência".

6.6 Interacção com os *data providers*

A aplicação OD@UA está preparada para aceitar dados de qualquer tipo de *data provider*, desde bases de dados que utilizem distintos SGBD até *web services* ou ficheiros CSV, entre outros. Esta interacção é efectuada de uma forma dinâmica, ou seja, a aplicação não tem de ser recompilada sempre que se necessita adicionar um novo *data provider*, basta indicar à mesma qual o *data provider* que se pretende. Para se conseguir este dinamismo foi adoptada na aplicação, no que diz respeito aos *data providers*, uma arquitectura inspirada em ODBC.

Tendo em conta o diagrama da Figura 29, facilmente se compreendem as semelhanças entre as arquitecturas de ODBC e da aplicação OD@UA. No caso da aplicação OD@UA, o componente que inicia todo o processo de acesso a um *data provider* é o "Portal", pois determina quando este acesso é necessário. Logo de seguida, este componente da arquitectura vai interagir com o "DataProviderDemultiplexer", um *handler* HTTP que, à semelhança do "Gestor de Controlador" de ODBC (diagrama da Figura 20), tem por função processar as instruções recebidas a partir do componente que o precede, sendo no caso da aplicação OD@UA o "Portal", e reencaminhá-las para o "Interface" do "Data Provider*"* correcto, ou seja, tem uma função de agulhamento. Já cada componente "Interface" representado no diagrama da Figura 29 tem o seu paralelo ODBC no "Controlador ODBC" (Figura 20), pois é este componente que conhece os detalhes de cada "Data Provider" permitindo desta forma a interacção entre cada um destes com a aplicação, sem que ambas as partes tenham conhecimentos das especificidades uma da outra. Desta forma, sempre que se pretende incluir um novo "Data Provider", em vez de se reprogramar e recompilar toda a aplicação, é apenas necessário criar um novo "Interface" que conheça os detalhes do mesmo, sendo a informação sobre esse mesmo "Interface" dada a conhecer à aplicação, conseguindo-se assim dinamismo.

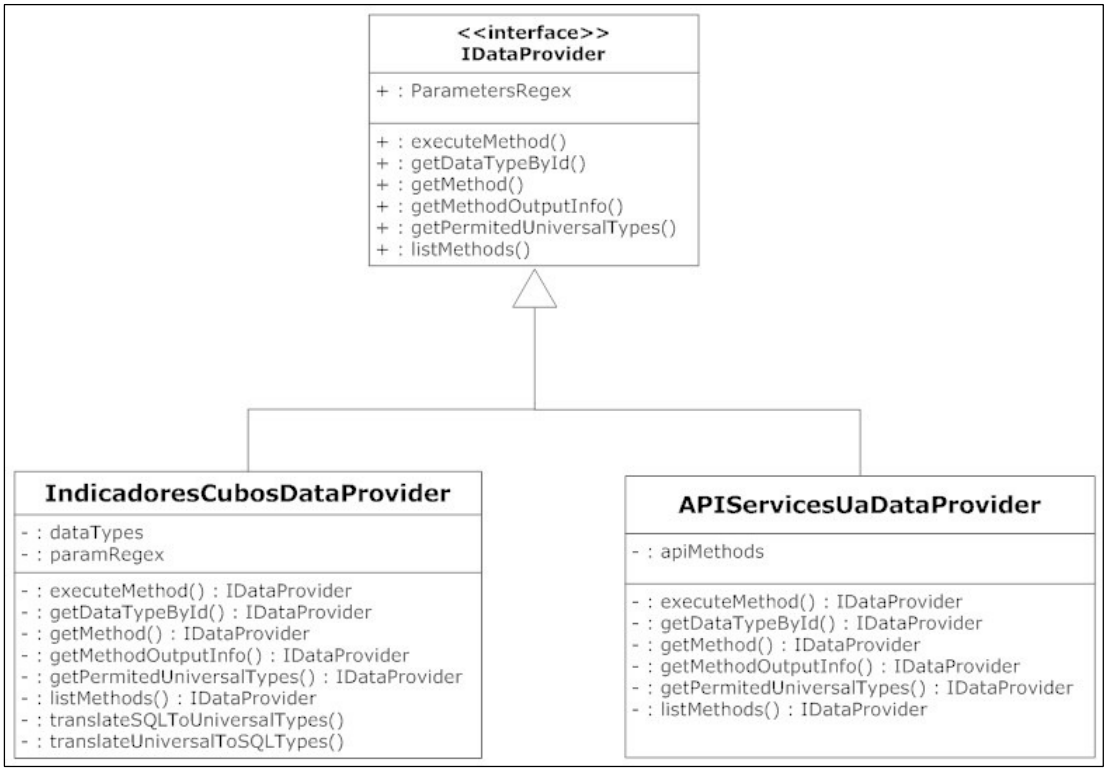


Figura 34: Diagrama de classes dos "Interfaces" de teste da aplicação OD@UA

Um "Interface" a ser incluído na aplicação deve ser compilado, ficando-se assim com uma DLL para o mesmo. Por forma a que a aplicação se possa servir do "Interface" em questão, necessita apenas de saber qual a localização deste no servidor, bem como do *namespace* ao qual este pertence. Cada "Interface" é uma implementação do interface "IDataProvider" representado no diagrama da Figura 34. Por conseguinte, a aplicação deve ser capaz de decidir em cada momento qual o "Interface" a ser utilizado, sendo que o mesmo deve ser implementado de forma dinâmica. Este dinamismo é conseguido com o recurso ao conceito de *reflection*.

Segundo alguns autores ([104] e [105]), *reflection* é a capacidade que um programa tem de apresentar o seu próprio comportamento, ou seja, reflectir o próprio código que o compõe, bem como os seus tipos e membros. Desta forma, é possível obter informações sobre uma determinada classe ou tipo e criar, em *run-time*, objectos que sejam instâncias dos mesmos [105].

A *framework* .NET disponibiliza funcionalidades [106] que permitem *reflection*, sendo as mesmas utilizadas no componente "DataProviderDemultiplexer" (Figura 29) da aplicação OD@UA.

Conforme já foi referido anteriormente, um dos componentes que permite a interacção da aplicação com os seus *data providers* é o *handler* "DataProviderDemultiplexer". Tal como o seu nome indica, o objectivo deste componente é agulhar as instruções recebidas a partir do "Portal" para o "Interface" do "DataProvider" correcto e, além disso, devolver os dados recolhidos no

"DataProvider" ao "Portal". Para que isto seja possível, deve ser passada pelo "Portal" ao "DataProviderDemultiplexer" informação sobre qual é o *id* na base de dados "Armazenamento de dados de operação" do *namespace* ao qual pertence o "Interface" que se pretende utilizar. Logo que disponha desta informação o *handler* irá determinar, acedendo à base de dados "Armazenamento de dados de operação", a localização da DLL correspondente ao "Interface" pretendido no servidor onde está a ser executada a aplicação, bem como o nome completo da classe que lhe corresponde. Adquirido este conhecimento, o *handler* irá servir-se das funcionalidades de *reflection* [106] da *framework* .NET para determinar se a classe correspondente ao "Interface" existe na DLL. Caso exista, será declarado um interface do tipo "IDataProvider" (diagrama da Figura 34) que instanciará um objecto da classe determinada a partir da DLL por *reflection*. Desta forma, criar-se-á então um "Interface" com o "Data Provider" pretendido, sendo que este disponibiliza diversos métodos para possibilitar a interacção da aplicação com o "Data Provider". Estes métodos vão ser utilizados pelo *handler* "DataProviderDemultiplexer" consoante as instruções que este recebe do "Portal", sendo que toda a interacção entre ambos é feita recorrendo à notação JSON.

Conforme se pode observar no diagrama da Figura 34, os métodos disponibilizados pelo interface são os seguintes:

- **"executeMethod"**, responsável por executar uma dada *query* no *data provider*, caso este o permita, ou executar um método que retira dados de um ou vários data providers, em caso contrário. Em ambos os casos é devolvido um objecto do tipo "DataTable" com os resultados da execução;
- **"getDataTypeById"**, responsável por devolver um tipo de dados específico do *data provider* dado o *id* respectivo, também ele específico ao *data provider*. É implementado apenas quando o *data provider* não reconhece os tipos nativos da linguagem C#;
- **"getMethod"**, responsável por devolver toda a informação relacionada com um determinado método dado o seu nome. É implementado apenas quando o *data provider* não permite a execução de *queries* directamente;
- **"getMethodOutputInfo"**, responsável por devolver informação sobre a tabela de retorno de cada *query* ou método, nomeadamente os tipos de cada coluna e os respectivos *headers*. Os tipos devolvidos já estão traduzidos para tipos nativos da linguagem C#, mesmo quando o data *provider* não os reconhece;
- **"getPermitedUniversalTypes"**, responsável por devolver os tipos de dados nativos da linguagem C# que podem ser convertidos para os tipos utilizados pelo

data provider em questão. Este método só é implementado quando os tipos utilizados pelo *data provider* são diferentes dos tipos nativos da linguagem C#;

- **"listMethods",** responsável por devolver um *array* contendo todos os métodos que são possíveis de executar no *data provider*. Este método só é implementado quando o *data provider* não permite a execução de *queries* directamente.

De notar que apesar de cada um dos *data providers* poder funcionar com os seus tipos próprios, os mesmos devem ser mascarados para a aplicação sob os tipos nativos da linguagem C#, sendo que a tradução nos dois sentidos, caso seja necessária, é da responsabilidade de cada "Interface".

Por forma a testar o dinamismo da aplicação no acesso a distintos *data providers*, criaram-se duas classes que implementam o interface "IDataProvider*"* conforme se pode ver no diagrama da Figura 34. Cada uma destas classes tem as suas próprias características, sendo que as mesmas implementam duas formas distintas de adquirir os dados.

A classe "IndicadoresCubosDataProvider" utiliza uma pequena porção da UA-BD como fonte de dados, sendo que esta forma de adquirir os dados pode ser denominada de recolha de dados heterogéneos em cenários conhecidos. Já a classe "APIServicesUaDataProvider" utiliza os serviços disponibilizados pela UA-API como fonte dos seus dados, podendo esta forma de adquirir os dados ser denominada de recolha de dados de fontes heterogéneas, uma vez que cada serviço representa uma fonte de dados distinta.

Como as formas de adquirir dados diferem nas duas classes, também o seu funcionamento tem algumas diferenças. No entanto, uma vez que os tipos de dados devolvidos por ambas são traduzidos para os tipos nativos da linguagem C#, caso já não o sejam inicialmente, dados provenientes de ambas as abordagens podem ser cruzados pela aplicação, desde que cumpram as politicas de *tags* e tipos de dados já descritas anteriormente nesta dissertação.

Importa então perceber mais pormenorizadamente o funcionamento de ambas as classes, para melhor compreender as suas diferenças.

A classe "IndicadoresCubosDataProvider" implementa todos os métodos do interface "IDataProvider" com a excepção do método "getMethod*"* pois, uma vez que o "Data Provider" para o qual é criado este "Interface" aceita queries directas, o interface não necessita de métodos específicos para aceder aos dados. Importa referir que como o "Data Provider" neste caso não reconhece os tipos nativos da linguagem C#, o "Interface" criado pela classe "IndicadoresCubosDataProvider" necessita de ter meios de traduzir entre estes tipos e os tipos nativos do "Data Provider". Por esta razão, e conforme é possível observar no diagrama da Figura 34, esta classe disponibiliza dois métodos privados que têm esta finalidade:

- **"translateSQLToUniversalType"** que, tal como o seu nome indica traduz os tipos conhecidos pelo *Microsoft SQL Server,* SGBD utilizado pelo "Data Provider", para tipos nativos da linguagem C#;
- **"translateUniversalTypeToSQL"** que, tal como o seu nome indica faz o inverso do método anterior.

É também importante compreender de que forma os outros quatro métodos são implementados. Começando pelo método "executeMethod" o funcionamento do mesmo é muito simples, pois executa apenas a *query* SQL sobre o "Data Provider" e retorna uma *DataTable* com os resultados. Quanto ao método "getDataTypeById", dado o *id* associado no construtor desta classe a um determinado tipo de dados nativo da linguagem C# vai devolver um objecto que representa esse mesmo tipo onde está incluído o nome e *id* deste. De notar que é a própria classe "IndicadoresCubosDataProvider" que no seu construtor define quais os tipos nativos da linguagem C# que permitem uma tradução directa, de acordo com o "Data Provider". Já o método "getMethodOutputInfo" vai executar a *query* SQL sobre o "Data Provider", mas em vez de devolver o resultado da mesma, devolve apenas o nome de cada coluna da tabela resultado e os tipos das mesmas, já traduzidos para os tipos nativos da linguagem C#. Quanto ao método "getPermitedUniversalTypes" devolve todos os tipos nativos da linguagem C# que têm tradução directa com os tipos do "Data Provider". Como já foi referido anteriormente estes tipos que são permitidos traduzir são definidos no construtor da classe "IndicadoresCubosDataProvider". Finalmente, o método "listMethods" devolve apenas um *array* vazio, uma vez que o "Interface" implementado com a classe "IndicadoresCubosDataProvider" não oferece nenhum método específico para aceder a dados do "Data Provider", uma vez que isso é feito com recurso a *queries* directas sobre o mesmo.

A classe "IndicadoresCubosDataProvider", que permite a recolha de dados heterogéneos em cenários conhecidos, neste caso a BD-UA, não efectua qualquer processamento nos dados recolhidos, com excepção da sua transformação em "DataTable" e tradução dos seus tipos originais para tipos nativos da linguagem C#. Resumindo, a classe serve apenas para passar uma *query* proveniente da aplicação ao "Data Provider" onde será executada e devolver o resultado da mesma à aplicação, efectuando uma tradução dos tipos utilizados por ambas as partes.

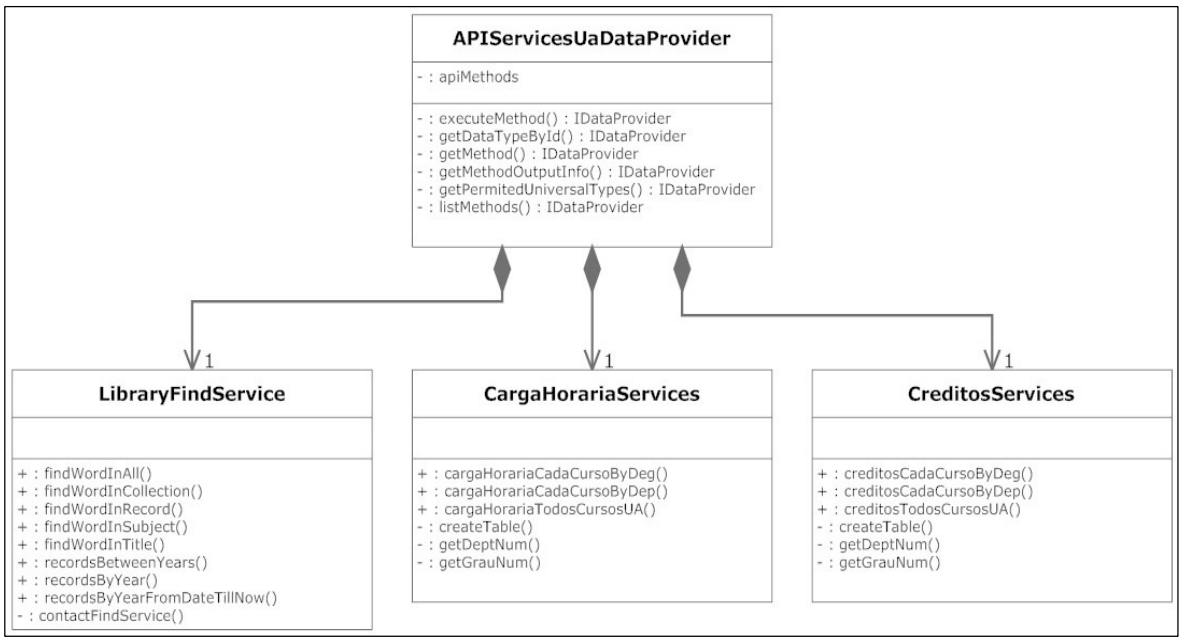


Figura 35: Diagrama de classes utilizadas no teste do paradigma de recolha de dados de fontes heterogéneas

No que diz respeito à classe "APIServicesUADataProvider", esta implementa apenas quatro dos métodos do interface "IDataProvider", não implementando os métodos "getDataTypeById" e "getPermitedUniversalTypes". A razão pela qual estes métodos não são implementados é que a própria "Interface" implementada através desta classe determina os tipos dos dados retornados por cada método executado nos "Data Providers" a ela associados, não sendo por isto necessária uma tradução dos mesmos. De notar que ao contrário da classe "IndicadoresCubosDataProvider", esta não executa *queries* directamente sobre um "Data Provider". Em vez disso, serve-se de diversos métodos disponibilizados por classes que representam distintos serviços, sendo cada um desses serviços um "Data Provider", conforme se pode observar no diagrama da Figura 35. De notar que os serviços utilizados nas classes "LibraryFindService", "CargaHorariaServices" e "CreditosServices" são *RESTful*. No entanto estes poderiam ser de qualquer outro tipo, nomeadamente SOAP. Por conseguinte, esta é uma classe que permite a recolha de dados de fontes heterogéneas.

Optou-se por não descrever os métodos disponibilizados pelas classes "LibraryFindService", "CargaHorariaServices" e "CreditosServices" nesta dissertação uma vez que no lugar destes poderiam estar quaisquer outros, dependendo das necessidades de dados que se pretendessem colmatar com cada "Data Provider".

Quanto aos métodos implementados na classe "APIServicesUADataProvider" importa descrever o seu funcionamento sucintamente. Começando pelo método "executeMethod", o funcionamento do mesmo consiste na execução do método de uma das classes "LibraryFindService", "CargaHorariaServices" ou "CreditosServices" cujo nome lhe foi passado

por parâmetro e devolver a "DataTable" resultante dessa execução. Já o método "getMethod" tem por objectivo devolver todas as informações relativas ao método cujo nome lhe é passado por parâmetro. Quanto ao método "getMethodOutputInfo" o seu funcionamento é idêntico ao método homónimo implementado na classe "IndicadoresCubosDataProvider". Finalmente, o método "listMethods" devolve a lista de todos os métodos de acesso a dados disponibilizados pelos vários "Data Providers" associados ao interface do tipo "APIServicesUADataProvider". De notar que esta lista é definida no construtor desta classe. Importa ainda referir que, se os serviços utilizados fossem baseados em XML, como por exemplo *web services* SOAP, poderia ser utilizada funcionalidade facultada pela classe *ServiceDescription* [107] da *framework* .NET para automatizar o processo de definição dos métodos de acesso a dados disponibilizados pelos *data providers*. Devido a restrições temporais não foi possível investigar de uma forma mais aprofundada esta possibilidade.

Em suma, pode-se afirmar que as duas classes aqui descritas representam dois paradigmas distintos na recolha de dados, sendo que no caso da recolha de dados heterogéneos em cenários conhecidos (representado pela classe "IndicadoresCubosDataProvider") é possível executar *queries* mais ou menos complexas directamente sobre o "Data Provider", sendo o seu resultado devolvido à aplicação no formato de um "DataTable". Já no caso da recolha de dados de fontes heterogéneas (representado pela classe "APIServicesUADataProvider"), é necessário primeiro recolher a informação de cada um dos "Data Providers", processar de seguida essa informação tendo em conta os resultados que se pretendam obter e, finalmente, disponibilizá-la no formato de um "DataTable".

6.7 Principais funcionalidades da aplicação

As funcionalidades oferecidas por uma aplicação informática são, habitualmente, a sua razão de existir. Assim sendo, no que diz respeito à aplicação OD@UA, importa conhecer um pouco melhor quais as principais funcionalidades oferecidas por esta.

A principal finalidade da aplicação OD@UA é a criação de representações dinâmicas de dados a partir de fontes heterogéneas. Desta forma, a mesma deve oferecer funcionalidades que permitam a criação dessas representações.

Na Figura 36 está exposta a forma como esta aplicação permite a criação de representações de dados. De notar que a representação resultante diz respeito ao encadeamento de três *queries* distintas, formando assim uma *query* composta. O utilizador não necessita ter quaisquer conhecimentos sobre as especificidades das fontes de dados pois estas estão mascaradas pela aplicação. Na Figura 36 as escolhas das três *queries* já foram tomadas, estando a representação

pronta a ser criada, bastando para isso o utilizador clicar no botão "Visualizar". O processo de selecção das *queries* simples a incluir na *query* composta que dará lugar à representação de dados, explicar-se-á de seguida. Importa referir que o utilizador pode também criar uma representação a partir de uma *query* simples, bastando para isso clicar no botão "Visualizar" após a selecção da primeira *query*.

Figura 36: Funcionalidade de criação de uma nova representação de dados na aplicação OD@UA

Conforme se pode obeservar na Figura 36, o utilizador é inicialmente convidado a seleccionar qual o tema raiz sobre o qual quer criar a sua representação de dados, sendo esta selecção efectuada na primeira *drop-down list* a partir do topo desta Figura. Logo que o tema esteja seleccionado, são apresentadas ao utilizador quais as *queries* simples relacionadas com esse tema, sendo que deve ser seleccionada uma para se poder prosseguir. Na Figura 36, o utilizador seleccionou a *query* "Cursos na Universidade de Aveiro", conforme se pode observar na segunda *drop-down list*. Depois de o utilizador seleccionar a *query* é-lhe dada a opção de visualizar a representação da mesma imediatamente ou então continuar a compor uma *query* composta que tenha por raiz esta *query* simples. Observando a Figura 36, compreende-se que o utilizador optou por continuar a compor. Assim sendo, foi-lhe pedido que atribuísse um título à nova *query* composta que foi criada a partir dali e qual a próxima *query* simples a incluir na *query* composta. A escolha do utilizador, conforme se pode ver na terceira *drop-down list* da Figura 36 foi pela *query* "Fases de acesso anuais por nome de curso". Uma vez que as *queries* compostas resultam do encadeamento de várias *queries* simples, o *output* da *query* anterior vai ser o *input* da *query*

seguinte. Por esta razão a *query* seguinte tem de ter pelo menos um parâmetro de entrada que seja um valor de *output* da *query* anterior. Conforme se pode ver na Figura 36, para o parâmetro "Nome_Curso" o utilizador seleccionou a opção "CursoUA". Esta opção corresponde ao nome de uma coluna na tabela resultante da *query* anterior que irá ser usada como *input* nesta *query*, ou seja, para cada linha da coluna "CursoUA" da tabela resultado da *query* anterior, vai ser executada a *query* seguinte tendo como parâmetro o valor dessa mesma linha, criando-se assim uma *nested table* com os resultados da *query* composta. De notar que as *queries* encadeadas não necessitam de ser respeitantes ao mesmo tema, apenas têm de respeitar as políticas de *tags* e tipos de dados já descritas nesta dissertação.

Depois de o utilizador clicar no primeiro botão "Continuar a Compor" e de ter já duas *queries* simples encadeadas, é-lhe pedido que escolha uma nova *query*, sendo que desta vez ele optou pela *query* "Notas de candidatura por nome de curso, ano e fase". Novamente deve escolher quais os valores para os parâmetros de entrada, sendo que, desta vez, já que existem três parâmetros, o valor de dois deles podem ser inseridos pelo próprio utilizador, não necessitando de ser o *output* das *queries* anteriores. Na selecção da Figura 36, o utilizador optou por escolher como parâmetros para a última *query* da cadeia as colunas "CursoUA", "Fase" e "Ano" resultantes da *query* composta criada até então.

Logo que o utilizador esteja satisfeito com a *query* composta, pode criar a sua representação clicando no botão "Visualizar" inferior da Figura 36.

Figura 37: Representação de dados resultante do encadeamento de *queries* da Figura 34

A representação dos dados devolvidos pela *query* é então efectuada por meio de uma *nested table*, no caso da *query* ser composta, ou de uma tabela simples, no caso contrário. Na Figura 37 pode ser observada a representação de dados resultante da *query* composta criada na Figura 36. Caso a representação criada seja uma *nested table* a aplicação permite expandir ou colapsar cada uma das linhas da tabela conforme a vontade do utilizador, podendo fazê-lo linha a linha por meio dos botões "+" e "-" ou para toda a tabela utilizando os botões "Expandir Tudo" e "Colapsar Tudo".

Ainda na Figura 37 é possível observar os botões "Criar Report", "Criar Gráficos" e "Enviar Hint". Este último, como seria de esperar, desencadeia o processo de criação de uma nova sugestão ou comentário por parte do utilizador. Já o botão "Criar Report" é responsável por iniciar o processo de criação dinâmica de um *report* que será disponibilizado ao utilizador para *download*. No caso de o *report* dizer respeito a uma representação de dados resultante de uma *query* simples, no mesmo figurará apenas a tabela resultante desta *query*. Caso o mesmo derive de uma representação de dados proveniente de uma *query* composta, o *report* será dividido em duas partes distintas: na primeira será representada a *nested table* contendo os dados encadeados, na segunda serão contidas diversas tabelas contendo cada uma delas os resultados de cada *query* simples integrante da *query* complexa. Um exemplo de um *report* criado pela aplicação OD@UA encontra-se no Anexo A2.

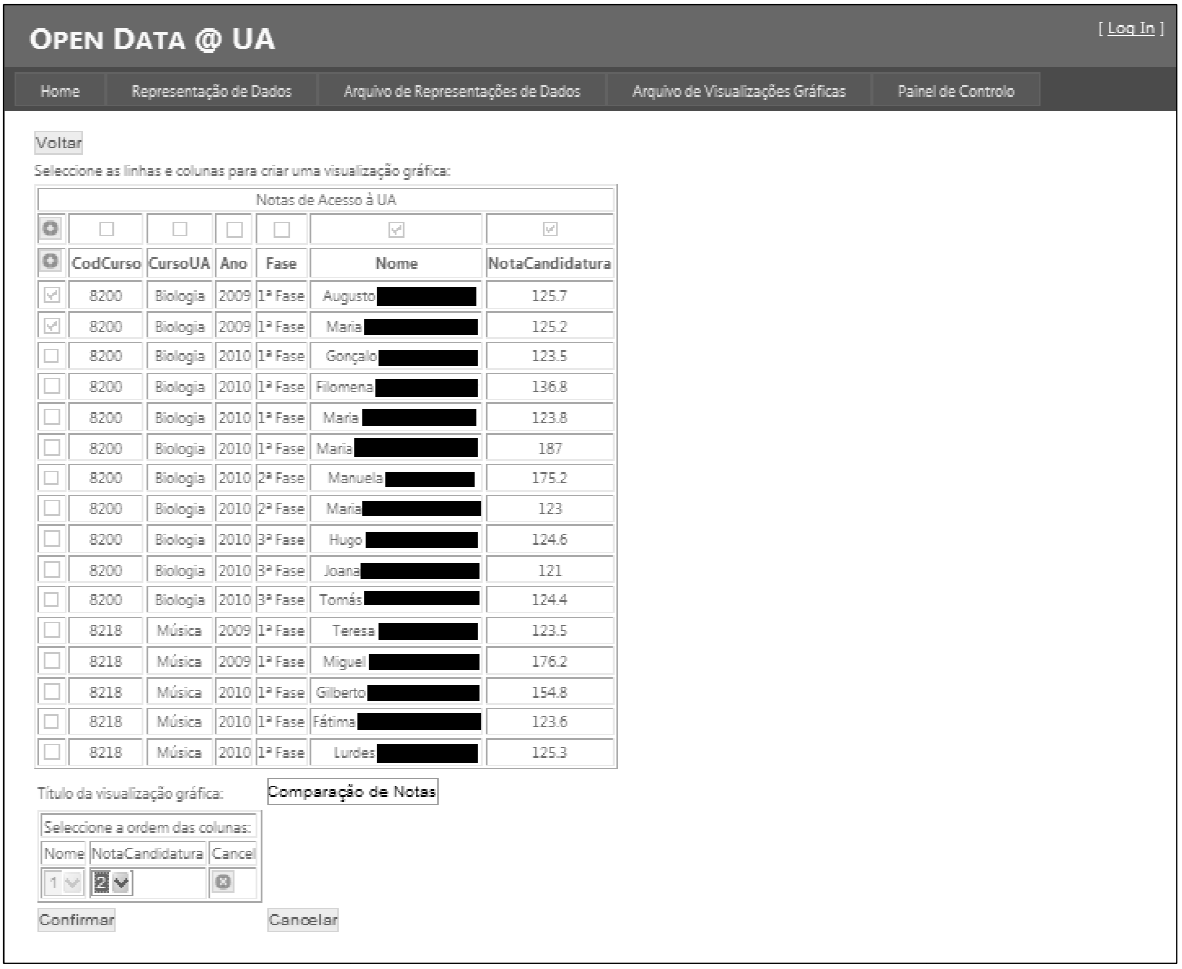


Figura 38: Funcionalidade que permite a criação de visualizações gráficas dos dados na aplicação OD@UA

Finalmente, o botão "Criar Gráficos", ao ser clicado, disponibiliza ao utilizador a funcionalidade representada na Figura 38. É recorrendo à mesma que o utilizador pode criar as visualizações gráficas dos dados. Conforme se pode observar na Figura 38, é disponibilizada uma tabela contendo todos os dados resultantes da *query* seleccionada pelo utilizador no processo representado na Figura 36, seja ela simples ou composta. Nessa tabela ele pode seleccionar quais as linhas e colunas a incluir na visualização, atribuir um título à mesma e escolher a ordem das colunas. No exemplo representado na Figura 38 é possível observar que o utilizador seleccionou as colunas "Nome" e "NotaCandidatura", mantendo esta ordem, escolhendo as primeiras duas linhas da tabela para visualizar. Esta selecção vai criar automaticamente a visualização gráfica representada na Figura 39, sendo que a aplicação escolhe automáticamente qual a escala mais favorável (linear ou logarítmica) e quais os tipos de gráfico que podem representar os dados escolhidos.

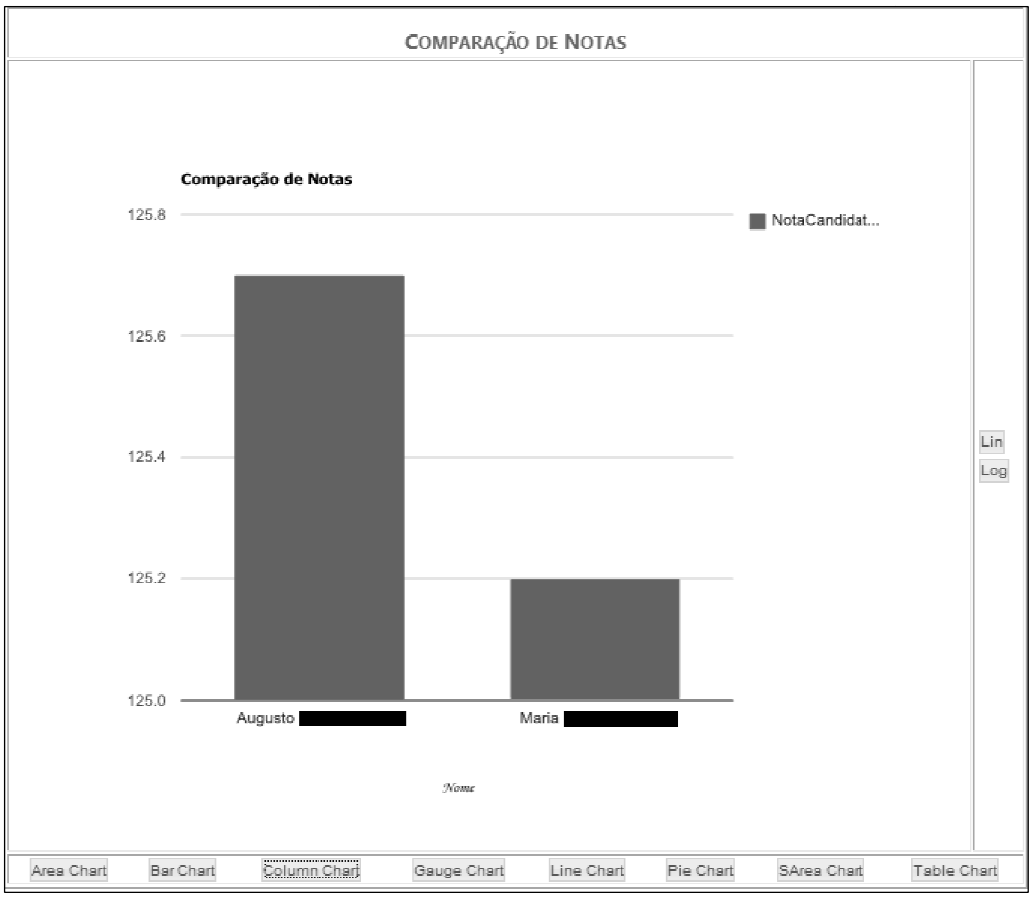


Figura 39: Visualização gráfica criada pela aplicação OD@UA tendo em conta as selecções de dados representadas na Figura 36

De notar que a funcionalidade de criação de visualizações gráficas permite ao utilizador criar diversas visualizações a partir dos mesmos dados para serem visualizadas em simultâneo. No entanto, por restrições espaciais optou-se por mostrar o processo de criação e visualização de apenas uma nesta Figura.

Na Figura 39 é possível observar a visualização gráfica gerada a partir dos dados seleccionados na Figura 38, bem como os botões laterais à visualização ("*Lin*" e "*Log*"), que permitem uma mudança de escala da mesma. Na base da Figura 39, por baixo do gráfico, são ainda visiveis vários botões que permitem a mundança do tipo de gráfico.

As funcionalidades descritas neste ponto são as principais da aplicação, no entanto, esta disponibiliza ainda funcionalidades de armazenamento das representações e visualizações de dados criadas, bem como a sua posterior visualização sem necessidade de repetir o processo de criação das mesmas.

Por forma a melhor compreender como a arquitectura da aplicação OD@UA influencia as funcionalidades disponibilizadas, apresenta-se, na Figura 40, novamente o diagrama da arquitectura da mesma. No entanto, os componentes nesta figura, encontram-se numerados de forma ordenada,

sendo que essa ordem significa a ordem de utilização de cada componente da arquitectura na sequência de funcionalidades que se passará a descrever de seguida.

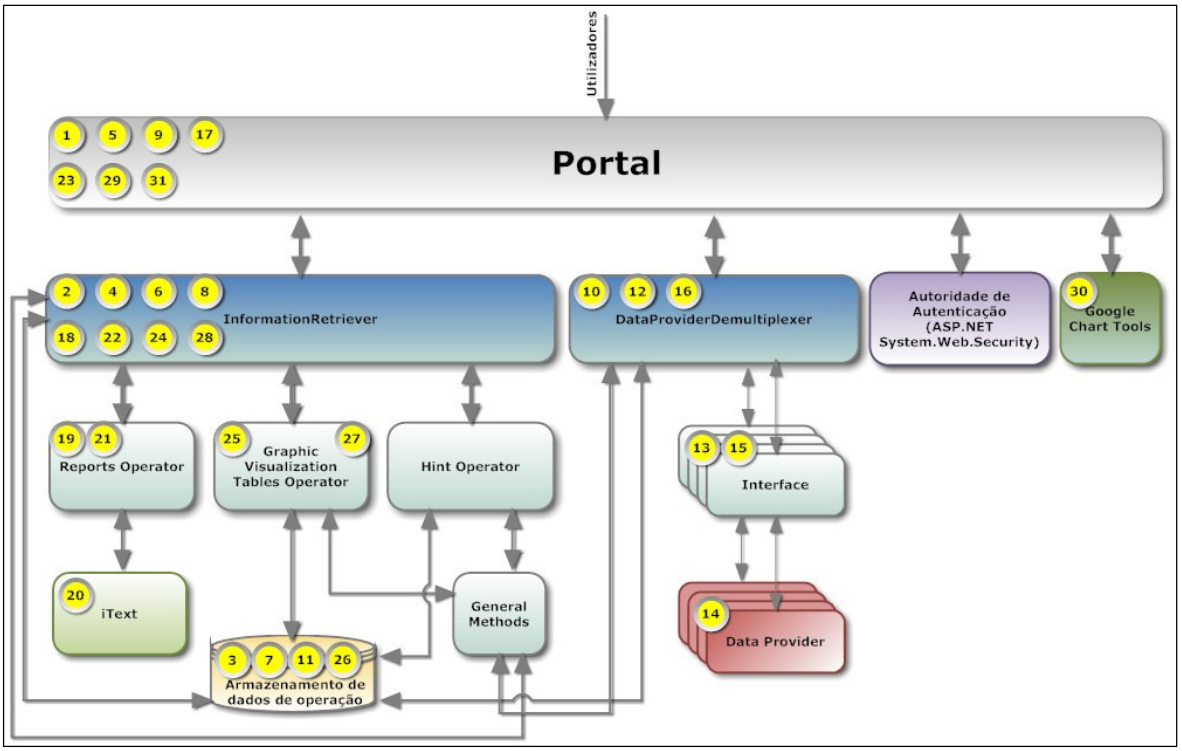


Figura 40: Arquitectura da aplicação OD@UA com componentes numerados por ordem de utilização para um exemplo de sequência de funcionalidades

Um utilizador não autênticado selecciona uma *query* simples,executa-a e vê o seu resultado representado. Logo de seguida, solicita a criação de um *report* sobre a mesma e cria uma visualização gráfica com os dados retornados pela *query*.

Por uma questão de simplificação, na Figura 40, a utilização do componente "General Methods" não está numerada. No entanto, esta componente é amplamente utilizada, sendo essa utilização evidenciada pelo número de outros componentes a ela ligados. Apesar disso, nesta sequência de utilização a sua existência pode ser ignorada, uma vez que ela é utilizada maioritariamente para efectuar conversões de *strings* formatadas em JSON para objectos C#.

Antes de poder seleccionar uma *query*, o utilizador deve seleccionar qual o tema para o qual quer ver as *queries*. O que acontece no número "1" da Figura 40 é que o utilizador, por meio do "Portal" indica à aplicação OD@UA que prentede criar uma nova representação de dados. Imediatamente a aplicação vai procurar saber quais são os temas existentes para os poder mostrar ao utilizador. Por esta razão o "Portal" interage com o *handler* "InformationRetriever" (número "2") que, por sua vez, irá interagir directamente com a base de dados "Armazenamento de dados de operação" (número "3") por forma a determinar quais os temas a mostrar ao utilizador. Logo que

os temas estejam determinados a resposta é enviada de volta para o *handler* "InformationRetriever" (número "4") que por sua vez a reencaminha para o portal e apresenta as opções de temas ao utilizador (número "5"). Após o utilizador seleccionar um tema, é então necessário a aplicação determinar quais as *queries* relaccionadas com o mesmo. Por conseguinte, o "Portal" vai interagir novamente com o *handler* "InformationRetriever" (número "6") para que, dado o tema, este indague junto da base de dados "Armazenamento de dados de operação" (número "7") quais as *queries* que correspondem a esse tema. A resposta será então devolvida para o *handler* (número "8") que a encaminhará para o "Portal" (número "9"), apresentando ao utilizador as *queries* possíveis de executar para o tema seleccionado.

Depois do utlizador seleccionar a *query* que pretende é necessário executá-la. Cabe ao *handler* "DataProviderDemultiplexer" (número "10") determinar qual o "Data Provider" sobre o qual a *query* vai ser executada. Conhecendo o identificador único da *query* na base de dados "Armazenamento de dados de operação", este *handler* vai determinar, servindo-se dessa base de dados, qual o *namespace* e qual a *string* que representa a *query* (número "11"). Esta informação retirada da base de dados é então processada pelo *handler* que, tendo em conta o *namespace* da *query*, irá instânciar a "Interface" (número "12") que permite a interacção com o "Data Provider" correcto. Logo que a interface esteja instânciada, esta é utilizada para enviar a *query* (número "13") para o "Data Provider" correcto. A *query* é então executada no "Data Provider" pretendido (número "14") e logo que a execução esteja concluída os dados resultantes são enviados ao respectivo "Interface" (número "15"). Por sua vez, o "Interface" irá encaminhar os dados para o *handler* "DataProviderDemultiplexer" (número "16") que finalmente os reencaminhará para o "Portal" (número "17").

Uma vez no "Portal", os dados são apresentados ao utilizador sobre a forma de uma tabela simples, uma vez que, neste caso, a *query* é simples.

Logo que o utilizador disponha de uma representação dos dados, à semelhança do que se observa na Figura 37, ele pode optar por criar automaticamente um *report* sobre os mesmos ou criar visualizações gráficas a partir dos dados.

Para criar um *report* o utilizador informa a aplicação desta intenção por meio do botão "Criar Report" visível na Figura 37. Esta acção irá fazer com que toda a informação relativa à *query* que deu origem aos dados apresentados, nomeadamente o seu título e os dados em si, sejam enviados para o *handler* "InformationRetriever" (número "18"). Por sua vez, o *handler* vai iniciar o processo de criação do *report* servindo-se do componente "Reports Operator" (número "19") que se irá servir de funcionalidade disponibilizada pela biblioteca *iText* (número "20") para criar um *report* em PDF. Logo que o *report* esteja concluído e seja armazenado no servidor (número "21"), a sua localização é enviada para o *handler* "InformationRetriever" (número "22") que por sua vez

reencaminha essa informação de volta para o "Portal" (número "23"), permitindo assim ao utilizador fazer *download* do relatório gerado automáticamente.

Para criar uma visualização gráfica a partir dos dados, o utilizador deve indicar essa intenção no "Portal" por meio do botão "Criar Gráficos" visível na Figura 37. Essa acção irá apresentar ao utilizador uma tabela e opções semelhantes às da Figura 38, que lhe permitirão seleccionar quais os dados que pretende incluir nas visualizações gráficas, bem como a ordem das colunas a ser utilizada em cada uma delas e o título que lhes será atribuído. Logo que o utilizador conclua a sua selecção, toda essa infomação é encaminhada do "Portal" para o *handler* "InformationRetriver" (número "24") que, por sua vez a reencaminhará para o componente "Graphic Visualization Tables Operator" (número "25"). Este componente, servindo-se de informação recolhida na base de dados "Armazenamento de dados de operação" (número "26") irá determinar qual a melhor escala (linear ou logarítmica) para cada visualização, bem como os tipos de gráficos (bar, column, etc.) que cada uma pode utilizar (número "27"), retornando esta informação para o *handler* "Information Retriever" (número "28") que por sua vez a reencaminhará para o "Portal" (número "29"). Já o "Portal" irá servir-se da ferramenta GCT ("número "30") e de toda esta informação para poder mostrar as visualizações ao utilizador (número "31").

Importa salientar novamente que a *query* que deu origem à sequência numerica representada na Figura 40 é uma *query* simples. Caso se tivesse optado por representar uma *query* composta a complexidade da sequência seria muito maior, uma vez que cada query teria de ser executada para cada conjunto de parâmetros de *input*, o que implicaria uma maior utilização do *handler* "DataProviderDemultiplexer".

7 Conclusões e trabalho futuro

Esta dissertação descreve o processo de desenvolvimento de um protótipo para uma ferramenta que permite a representação e visualização dinâmicas de dados provenientes de fontes heterogéneas. A mesma especifica os aspectos mais relevantes relativos a esse processo.

Com o desenvolvimento deste protótipo procura-se também compreender as diferenças existentes nas formas de acesso aos dados, seja quando estes são heterogéneos apesar de provenientes de uma só fonte ou quando são provenientes de fontes heterogéneas.

7.1 Conclusões

Uma vez que o conceito OD está cada vez mais difundido entre as instituições públicas e privadas, surgem constantemente novas ferramentas que permitem fazer uso dos dados disponibilizados, transformando os mesmos em conhecimento.

Como facilmente se compreende, qualquer universidade gera diariamente grandes colecções de dados, sendo que a UA não é excepção. Por esta razão, surgiu a necessidade de se criar uma aplicação que se servisse destas colecções de dados e permitisse ao seu utilizador a criação de conhecimento sobre esta instituição.

No decorrer deste trabalho criou-se um protótipo para uma aplicação à qual se deu o nome de OD@UA e que permite a representação e visualização dinâmicas de dados provenientes de fontes heterogéneas. Esta é uma aplicação *web* que se serve da *framework* .NET, tendo sido utilizadas as linguagens de programação C# e *JavaScript* no seu desenvolvimento.

Com esta aplicação ficará disponível à comunidade académica uma nova ferramenta para que esta possa compreender melhor os dados disponibilizados pelos diversos *data sources* dentro da instituição UA. Uma das mais valias desta aplicação é que não é necessária a sua reprogramação sempre que se pretende adicionar um novo *data source*, já que a mesma está preparada para os receber de forma dinâmica. Além disso, podem ser cruzados dados provenientes de diversos *data sources*, por forma a aprofundar ainda mais o conhecimento deles derivado. Os dados seleccionados pelos utilizadores podem depois ser representados em tabelas simples ou *nested tables*, bem como em *reports* e visualizações gráficas, o que confere aos utilizadores a possibilidade de melhor explorarem a informação contida nos dados.

Procurou-se desenhar a ferramenta OD@UA por forma a que utilizadores sem qualquer conhecimento de linguagens de *query* pudessem usufruir dela. No decorrer do uso desta ferramenta, os utilizadores seleccionam as *queries* a partir das quais pretendem retirar dados (*queries* simples ou compostas), sendo que os dados obtidos serão depois representados de uma

forma dinâmica numa tabela simples (quando a *query* que lhes deu origem era também ela simples), ou *nested tables* (quando a *query* da qual provêm era composta). De notar que o processo de execução de *queries* compostas implementado nesta aplicação se pode tornar ineficiente, uma vez que o *input* da *query* seguinte na composição é cada uma das linhas do *output* da *query* anterior. Por outras palavras, a *query* seguinte será executada tantas vezes quantas o número de linhas integrantes do *output* da *query* anterior. Quando o *output* de uma determinada *query* tem demasiadas linhas ou quando estão encadeadas muitas *queries* na composição, a execução da *query* composta pode demorar um tempo não aceitável do ponto de vista do utilizador, já que é necessário executar cada uma das *queries* para cada grupo de parâmetros de *input*.

O dinamismo é transversal a toda a OD@UA, encontrando-se na criação de representações de dados, na criação de *reports* e na criação de visualizações gráficas. Relativamente à primeira, e como descrito anteriormente, o dinamismo inerente à representação de dados significa que é da ferramenta a responsabilidade de gerar automaticamente estas mesmas representações. No que respeita à criação de *reports*, o processo é dinâmico uma vez que o utilizador apenas necessita de indicar à aplicação a intenção de o criar, sendo da responsabilidade da mesma a criação automática do *report*. Quanto à criação de visualizações gráficas, após a obtenção das representações de dados, o utilizador tem a possibilidade de escolher quais as linhas e colunas sobre as quais a aplicação irá gerar automaticamente as visualizações gráficas, tendo estas a particularidade de que é a própria aplicação a decidir qual a melhor escala (linear ou logarítmica) para as apresentar. Além disso, será a aplicação que, de acordo com os tipos de dados seleccionados pelo utilizador, irá escolher quais os tipos de visualizações que a eles se adequam.

A ferramenta OD@UA serve-se, na criação dos *reports*, de funcionalidade disponibilizada pela biblioteca *iText*, a qual oferece várias funções que permitem a criação de ficheiros PDF, facultando igualmente a possibilidade de inclusão de tabelas e texto, entre outros. Optou-se pelo uso desta biblioteca porque, apesar de haver falta de documentação sobre a mesma, a simplicidade de utilização das suas funções permitiu ultrapassar esse obstáculo.

Na criação das visualizações gráficas recorreu-se à ferramenta GCT, uma vez que esta permite o controlo sobre diversos aspectos da visualização (controlo de cores utilizadas, tamanho de letra, entre outros), confere uniformidade no formato de aceitação de dados e permite utilizar diferentes tipos de gráficos nas visualizações.

Foi também tido em conta, no que diz respeito às visualizações gráficas criadas pela aplicação OD@UA, um factor apontado por Yi et al [58] como um dos mais importantes para a extracção de conhecimento a partir de visualizações de dados, sendo este factor a afinidade do utilizador com os dados contidos nas visualizações. Para estes autores [58], uma forma de conseguir esta afinidade é permitir aos utilizadores a interacção com as visualizações. Foi isto que

se procurou na aplicação OD@UA, ao oferecer a possibilidade aos utilizadores de alterar o tipo de gráfico e a escala das visualizações gráficas, por forma a melhor adequar as mesmas às suas expectativas.

Numa tentativa de redução da desordem visual das visualizações gráficas geradas pela aplicação OD@UA, uma das técnicas adoptadas foi a de "alterar tamanho dos pontos", técnica essa cuja maior eficácia se revela apenas em pequenas colecções de dados. No caso de colecções mais extensas, a técnica que proporcionaria uma real redução da desordem visual, seria a de "distorção topológica" [61] que inclui os métodos "zoom" e "fish-eye". No entanto não foi possível implementar tais técnicas devido a restrições temporais.

Outra característica da aplicação OD@UA é sua independência face aos *data providers* de onde obtém os dados para a criação das representações. Esta independência é conseguida, pois para cada *data provider* é necessário criar uma classe que implemente um interface entre este e a aplicação, interface esse que oferece funções que permitem à aplicação compreender os dados provenientes dos *data providers*. Novamente aqui o dinamismo toma uma posição importante na aplicação, visto que a mesma apenas necessita de saber a localização da DLL que define a classe que implementa o interface, bem como o nome dessa classe para dinamicamente dela se servir.

Uma vez que os tipos de dados utilizados pela ferramenta OD@UA são os tipos de dados nativos da linguagem C#, e sendo que cada um dos *data providers* pode utilizar um tipo de dados específico, cabe ao interface proceder à tradução destes últimos para os utilizados pela OD@UA.

Um outro objectivo deste trabalho era compreender qual a diferença no acesso aos dados recorrendo a dois paradigmas distintos: acesso a dados heterogéneos provenientes de uma só fonte e acesso a dados de fontes heterogéneas. Para se estudarem as diferenças criaram-se duas classes que implementam o interface entre a ferramenta e os *data providers*. Essas classes são a "*IndicadoresCubosDataProvider*" e a "*APIServicesUADataProvider*". A primeira representa o paradigma de acesso a dados heterogéneos provenientes de uma só fonte, uma vez que o seu *data provider* é apenas a base de dados UA-BD. A segunda representa o paradigma de acesso a dados de fontes heterogéneas, uma vez que se serve da camada de serviços UA-API tendo vários serviços como *data provider*.

As principais diferenças encontradas são que quando se trata do primeiro paradigma podem ser efectuadas *queries* sobre o *data provider* directamente, sendo que estas *queries* podem ter um elevado grau de detalhe. Já no segundo, uma vez que existem diversos *data providers*, as *queries* não podem ser efectuadas directamente sobre os mesmos. Neste caso, quando a *query* é enviada ao interface implementado pela classe "*APIServicesUADataProvider*", a mesma vai fazer o pedido dos dados que necessita a cada um dos *data providers*. De seguida, processa toda a informação recolhida tendo em conta os resultados que se pretendem obter a partir da *query*, e só então os

disponibiliza. Em suma, a principal diferença entre os dois paradigmas é que, ao contrário do primeiro, no segundo é necessário processar os dados provenientes dos *data providers* de acordo com a *query* utilizada, sendo que o nível de detalhe oferecido estará sempre dependente da forma como foi programado o interface e não da *query* em si. Desta forma, qualquer alteração à complexidade das *queries* neste segundo paradigma requer uma reprogramação do interface, o que não é necessário acontecer no primeiro paradigma descrito.

7.2 Trabalho futuro

Uma vez que a ferramenta criada no decorrer deste trabalho é apenas um protótipo, podem ser incluídas no futuro diversas funcionalidades que permitam aumentar a qualidade da mesma.

Na sua forma actual, a ferramenta permite que qualquer utilizador tenha acesso a todas as representações e visualizações gráficas armazenadas. Seria interessante, no futuro, restringir o acesso às *queries*, representações armazenadas e visualizações gráficas armazenadas, consoante o grupo ao qual pertencesse determinado utilizador. Esta ideia pressupõe que todos os utilizadores da ferramenta necessitariam de estar registados, sendo que cada um deles pertenceria a um ou mais grupos. Qualquer utilizador faria parte do grupo "público". O acesso aos conteúdos ficaria então dependente da pertença do utilizador a um grupo que tivesse autorização de aceder aos mesmos. A título de exemplo, supondo que determinado conteúdo permitia o acesso apenas a membros do grupo "professor" e "aluno", um utilizador pertencente apenas ao grupo "público" não teria acesso.

A materialização desta ideia permitiria também incutir um cariz mais social à ferramenta, tal como na ferramenta *Many Eyes* (capitulo 4.3), já que, assim, com utilizadores registados, poder-se-ia permitir a associação de comentários aos diversos conteúdos, transformando os últimos em artefactos sociais [59].

Os grupos deveriam ser criados pelos gestores da ferramenta, sendo que a autenticação dos utilizadores deveria passar a ser feita com o utilizador universal da UA para facilitar o processo de atribuição de grupo a um utilizador. Desta forma, seria necessário adicionar à base de dados "Armazenamento de dados de operação" (Figura 30) novas tabelas para conter todos os dados necessários. Uma vez que os pormenores da base de dados que contém as informações sobre os utilizadores universais da UA não são públicos, parte-se do princípio que as mesmas estariam armazenadas numa tabela denominada "Universal_User" contendo todos os utilizadores, sendo que a cada um deles estaria associado um identificador numérico. Assim sendo, na base de dados "Armazenamento de dados de operação", seria necessário criar quatro novas tabelas: uma tabela "User_Group" que armazenasse todos os dados relativos a um determinado grupo, nomeadamente o seu nome e identificador; uma tabela "User_Group_Link" que associasse o identificador de um

utilizador universal da tabela "Universal_User" com o identificador de um grupo da tabela "User_Group"; uma tabela "Group_Visualization_Link" que associasse o identificador de um grupo com o identificador de uma visualização armazenada da tabela "GraphicVisualization" (Figura 30); uma tabela "Group_Query_Link" que associasse o identificador de um grupo com o identificador de uma *query* da tabela "InfoQuery" (Figura 30).

Tendo em conta a crescente tendência das instituições disponibilizarem os seus dados segundo um conceito de LOD, seria interessante que a ferramenta OD@UA adoptasse também esta funcionalidade. Por conseguinte, para atingir esse objectivo seria necessário incluir um novo componente na arquitectura da Figura 29. Esse componente estaria ligado ao "Portal" e à base de dados "Armazenamento de dados de operação", sendo que o mesmo teria como função a transformação em *triples* RDF das representações de dados pertencentes ao grupo "público" armazenadas na tabela "DataRepresentation" (Figura 30). Para armazenar os *triples* criados seria necessário criar uma nova tabela na base de dados da Figura 30. Essa tabela poderia adoptar o nome "RDFTriples" contendo todos os *triples* criados sendo que a cada um seria dado um identificador numérico. Além disso, para permitir associar a representação de dados com os *triples* criados a partir dela, deveria também ser criada uma tabela na mesma base de dados para armazenar os dados dessa associação. A mesma estaria ligada à tabela "DataRepresentation" (Figura 30) e à tabela "RDFTriples".

Por forma a simplificar o processo de criação de interfaces com os *data providers*, seria importante, no futuro, investigar de que forma a classe *ServiceDescription* [107] da *framework* .NET poderia permitir a automatização da definição dos métodos de acesso a dados quando os *data providers* são serviços baseados em XML, como por exemplo *web services* SOAP. Assim sendo, não seria necessário ao programador da classe que implementa o interface criar manualmente cada um desses métodos.

No que respeita às visualizações gráficas, seria importante no futuro incluir na aplicação OD@UA as funcionalidades de "zoom" e "fish-eye", o que permitiria uma redução significativa da desordem visual em visualizações criadas a partir de colecções de dados muito extensas. A funcionalidade de "zoom" deveria permitir a expansão ou diminuição do pano de fundo das visualizações consoante o utilizador efectuasse "zoom" *in* ou *out*, aumentando ou diminuindo, respectivamente, o número de pontos de dados mostrados na visualização. Já a funcionalidade de "fish-eye", deveria permitir ao utilizador servir-se como que de uma lupa, sendo que em cada ponto da visualização sobre o qual o mesmo passasse o cursor tendo seleccionada esta funcionalidade, o pano de fundo da mesma estender-se-ia, sofrendo os pontos contidos nesta selecção uma deslocação e aumento de tamanho, diminuindo assim a desordem visual.

Referências

[1] T. TERAOKA, *A Study of Exploration of Heterogeneous Personal Data Collected From Mobile Devices and Web Services*, Multimedia and Ubiquitous Engineering (MUE), 2011 5th FTRA International Conference on (2011).

[2] M. BRYANT. (2011) *Open Data in Europe gets a huge boost from new EU rules*, Em linha http://thenextweb.com/eu/2011/12/12/open-data-in-europe-gets-a-huge-boost-from-new-eu-rules/ (Consultado a 6 de Abril de 2012)

[3] C. BÖHM, M. FREITAG, S. GEORGE, N. HÖFLER, M. KÖPPELMANN, C. LEHMANN, A. MASCHER, F. NAUMANN and T. SCHMIDT, *Linking Open Government Data: What Journalists Wish They Had Known*, I-Semantics '10 Proceedings of the 6th International Conference on Semantic Systems, ACM New York, NY, USA (2012).

[4] *Data.gov*, Em linha http://www.data.gov (Consultado em 29 de Abril de 2012)

[5] *data.gov.uk - Opening up government*, Em linha http://data.gov.uk/ (Consultado em 29 de Abril de 2012)

[6] *data.gov.au*, Em linha http://data.gov.au/ (Consultado em 29 de Abril de 2012)

[7] *Pordata - Estatisticas, gráficos e indicadores de Municípios, Portugal e Europa*, Em linha http://www.pordata.pt/ (Consultado em 29 de Abril de 2012)

[8] *Dados.Gov*, Em linha http://www.dados.gov.pt/pt/inicio/inicio.aspx (Consultado em 29 de Abril de 2012)

[9] S. P. USELTON, *exVis: Developing A Wind Tunnel Data Visualization Tool*, VIS '97 Proceedings of the 8th conference on Visualization '97. IEEE Computer Society Press Los Alamitos, CA, USA (1997).

[10] C. ANDORF, J. BAO, D. CARAGEA, D. DOBBS, V. HONAVAR, J. PATHAK and A. SILVESCU, *Information Integration and Knowledge Acquisition from Semantically Heterogeneous Biological Data Sources*, DILS'05 Proceedings of the Second international conference on Data Integration in the Life Sciences. Springer-Verlag Berlin, Heidelberg (2005).

[11] UNIVERSIDADE, DE AVEIRO, SAPO.PT and PT, INOVAÇÃO. *API Academic Playground & Innovation*, Em linha http://api.web.ua.pt/ (Consultado a 17 de Agosto de 2012)

[12] A. L. MACHADO and J. M. P. D. OLIVEIRA, *DIGO: An Open Data Architecture for e-Government*, Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International (2011).

[13] D. S. SAYOGO and T. A. PARDO, *Understanding the Capabilities and Critical Success Factors in Collaborative Data Sharing Network:The case of DataONE*, dg.o '11 Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times. ACM New York, NY, USA (2011).

[14] A. BRAHAJ and J. HOXHA, *Open Government Data on the Web: A Semantic Approach*, Emerging Intelligent Data and Web Technologies (EIDWT), 2011 International Conference on (2011).

[15] C. BIZER, T. HEATH and BERNERS-LEE, TIM. (2009) *Linked Data - The Story So Far*, Em linha http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf (Consultado a 26 de Abril de 2012)

[16] M. HAUSENBLAS, *Understanding Linked Open Data as a Web-Scale Database*, Advances in Databases Knowledge and Data Applications (DBKDA), 2010 Second International Conference on (2010).

[17] R. CYGANIAK and A. JENTZSCH. *Linking Open Data cloud diagram*, Em linha http://lod-cloud.net/ (Consultado a 26 de Abril de 2012)

[18] T. BERNERS-LEE. (2009) *Linked Data - Design Issues*, Em linha http://www.w3.org/DesignIssues/LinkedData.html (Consultado a 26 de Abril de 2012)

[19] T. BERNERS-LEE, R. FIELDING, L. MASINTER, W3C/MIT, SOFTWARE, DAY and SYSTEMS, ADOBE. (2005) *RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax*, Em linha http://tools.ietf.org/html/rfc3986 (Consultado a 27 de Abril de 2012)

[20] W3C, G. KLYNE, J. J. CARROL and B. MCBRIDE. (2004) *Resource Description Framework (RDF): Concepts and Abstract Syntax*, Em linha http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (Consultado em 28 de Abril de 2012)

[21] W3C, F. MANOLA, E. MILLER and B. MCBRIDE. (2004) *RDF Primer*, Em linha http://www.w3.org/TR/rdf-primer/ (Consultado em 28 de Abril de 2012)

[22] W3C, D. BECKETT and B. MCBRIDE. (2004) *RDF/XML Syntax Specification (Revised)*, Em linha http://www.w3.org/TR/rdf-syntax-grammar/ (Consultado em 28 de Abril de 2012)

[23] T. BERNERS-LEE. (2011) *Notation 3 Logic*, Em linha http://www.w3.org/DesignIssues/Notation3.html (Consultado em 28 de Abril de 2012)

[24] W3C, E. PRUD'HOMMEAUX and A. SEABORNE. (2008) *SPARQL Query Language for RDF*, Em linha http://www.w3.org/TR/rdf-sparql-query/ (Consultado em 28 de Abril de 2012)

[25] W3C, K. G. CLARK, L. FEIGENBAUM and E. TORRES. (2008) *SPARQL Protocol for RDF*, Em linha http://www.w3.org/TR/rdf-sparql-protocol/ (Consultado em 28 de Abril de 2012)

[26] M. ARENAS and J. PÉREZ, *Querying Semantic Web Data with SPARQL*, PODS '11 Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM New York, NY, USA (2011).

[27] *wiki.dbpedia.org : About*, Em linha http://dbpedia.org/About (Consultado a 13 de Outubro de 2012)

[28] *Open Data Albania*, Em linha http://open.data.al/en (Consultado em 29 de Abril de 2012)

[29] A. BRAHAJ, J. HOXHA and D. VRANDECIC, *open.data.al - Increasing the Utilization of Government Data in Albania*, I-Semantics '11 Proceedings of the 7th International Conference on Semantic Systems. ACM New York, NY, USA (2011).

[30] D. AUSTIN, A. BARBIR, C. FERRIS and S. GARG. (2002) *Web Services Architecture Requirements - W3C Working Draft* Em linha http://www.w3.org/TR/2002/WD-wsa-reqs-20020819 (Consultado a 5 de Junho de 2012)

[31] IBM, DEVELOPERWORKS. *New to SOA and web services*, Em linha http://www.ibm.com/developerworks/webservices/newto/service.html (Consultado a 5 de Junho de 2012)

[32] P. BRITTENHAM. (2002) *An overview of the Web Services Inspection Language*, Em linha http://www.ibm.com/developerworks/webservices/library/ws-wsilover/ (Consultado a 5 de Junho de 2012)

[33] J. D. HANEY and C. A. VANLENGEN, *Creating Web Services Using ASP.NET*, Journal of Computing Sciences in Colleges 20 (2004).

[34] R. CHBEIR, E. DAMIANI, G. GIANINI and T. JOE, *SOAP Processing Performance and Enhancement*, Services Computing, IEEE Transactions on (2011).

[35] M. GUDGIN, M. HADLEY, A. KARMARKAR, Y. LAFON, N. MENDELSOHN, J.-J. MOREAU and H. F. NIELSEN. (2007) *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, Em linha http://www.w3.org/TR/soap12-part1/ (Consultado a 6 de Junho de 2012)

[36] C. C. ALBRECHT, *How clean is the future of SOAP?*, Communications of the ACM - Information cities. New York, NY, USA, 47 (2004).

[37] K. CHIU, R. V. ENGELEN, M. GOVINDARAJU, M. J. LEWIS, P. LIU and A. SLOMINSKI, *Toward Characterizing the Performance of SOAP Toolkits*, Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on (2004).

[38] R. B. AHMAD and G. M. WALEED, *Security Protection using Simple Object Access Protocol (SOAP) Messages Techniques*, International Conference on Electronic Design, 2008. ICED 2008. (2008).

[39] R. ELFWING, L. LUNDBERG and U. PAULSSON, *Performance of SOAP in Web Service Environment Compared to CORBA*, Software Engineering Conference, 2002. Ninth Asia-Pacific (2002).

[40] C. KOHLHOFF and R. STEELE, *Evaluating SOAP for High Performance Business Applications: Real–Time Trading Systems*, Procedings of the World Wide Web (WWW) Conference. Budapest (2003).

[41] Y. LAFON and N. MITRA. (2007) *SOAP Version 1.2 Part 0: Primer (Second Edition)*, Em linha http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ (Consultado em 8 de Junho de 2012)

[42] D. CROCKFORD. (2006) *RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*, Em linha http://tools.ietf.org/html/rfc4627 (Consultado a 9 de Junho de 2012)

[43] D. CROCKFORD and JSON.ORG. *Introducing JSON*, Em linha http://www.json.org/ (Consultado a 9 de Junho de 2012)

[44] L. BELLIVEAU, S. DOWNES, M. A. RAHMAN, S. SAMET and R. SAVOIE, *Managing Digital Rights Using JSON*, 2010 7th IEEE Consumer Communications and Networking Conference (CCNC) (2010).

[45] A. AZIZ and S. MITCHELL. (2007) *An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET*, Em linha http://msdn.microsoft.com/en-us/library/bb299886.aspx (Consultado a 9 de Junho de 2012)

[46] G. WANG, *Improving Data Transmission in Web Applications via the Translation between XML and JSON*, Communications and Mobile Computing (CMC), 2011 Third International Conference on (2011).

[47] C. IZURIETA, N. NURSEITOV, M. PAULSON and R. REYNOLDS, *Comparison of JSON and XML Data Interchange Formats: A Case Study*, Department of Computer Science Montana State University - Bozeman, Montana, 59715, USA (2009).

[48] C. SEVERANCE, *Discovering JavaScript Object Notation*, Computer - IEEE Computer Society, 45 (2012).

[49] S. K. MAKKI and A. SUMARAY, *A Comparison of Data Serialization Formats For Optimal Efficiency on a Mobile Platform*, ICUIMC '12 Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication Article No. 48 ACM New York, NY, USA (2012).

[50] D. CROCKFORD. (2006) *JSON: The Fat-Free Alternative to XML*, Em linha http://www.json.org/fatfree.html (Consultado a 9 de Junho de 2012)

[51] K. K. S. SARINDER, L. H. S. LIM, A. F. MERICAN and K. DIMYATI, *Information Retrieval from Distributed Relational Biodiversity Databases*, *Computer Research and Development, 2010 Second International Conference on*, 2010, pp. 432-435.

[52] O. M. PEREIRA, R. L. AGUIAR and M. Y. SANTOS, *Assessment of a Enhanced ResultSet Component for accessing relational databases*, *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, 2010, pp. V1-194-V1-201.

[53] R. A. MCCLURE and I. H. KRUGER, *SQL DOM: compile time checking of dynamic SQL statements*, *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, 2005, pp. 88-96.

[54] Q. ZOU, H. WANG, R. SOULÉ, M. HIRZEL, H. ANDRADE, B. GEDIK and K.-L. WU, *From a stream of relational queries to distributed stream processing*, Proc. VLDB Endow., 3 (2010), pp. 1394-1405.

[55] H. ABDALHAKIM, *Addressing Burdens of Open Database Connectivity Standards on the Users*, *Intelligent Information Technology Application Workshops, 2009. IITAW '09. Third International Symposium on*, 2009, pp. 305-308.

[56] Z. LONGTING, L. ZHIHONG, L. QINGQING and X. WEILI, *Technical methods for direct access to ArcSDE vector feature model in RDBMS*, *Geoinformatics, 2011 19th International Conference on*, 2011, pp. 1-5.

[57] IBM, CORPORATION *Descrição geral do controlador de ODBC do iSeries Access*, Em linha http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=%2Frzaii%2Frzaiiodbcadm.htm (Consultado a 13 de Junho de 2012)

[58] J. S. YI, Y.-A. KANG, J. T. STASKO and J. A. JACKO, *Understanding and characterizing insights: how do people gain insights using information visualization?*, *Proceedings of the 2008 conference on BEyond time and errors: novel evaLuation methods for Information Visualization*, ACM, Florence, Italy, 2008, pp. 1-6.

[59] H. MACDONALD, J. YUILLE, R. STANTON and S. VILLER, *The social life of visualization*, *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7*, ACM, Melbourne, Australia, 2009, pp. 193-200.

[60] H. HAUSER, *Toward new grounds in visualization*, SIGGRAPH Comput. Graph., 39 (2005), pp. 5-8.

[61] G. ELLIS and A. DIX, *A Taxonomy of Clutter Reduction for Information Visualisation*, Visualization and Computer Graphics, IEEE Transactions on, 13 (2007), pp. 1216-1223.

[62] C. FORSELL and J. JOHANSSON, *An heuristic set for evaluation in information visualization*, *Proceedings of the International Conference on Advanced Visual Interfaces*, ACM, Roma, Italy, 2010, pp. 199-206.

[63] GOOGLE, INC. *Google Chart Tools*, Em linha http://developers.google.com/chart/ (Consultado a 21 de Junho de 2012)

[64] E. HARTNETT and R. KOURY, *Using Google Apps Through the Electronic Resource Life Cycle*, Collection Management, 37 (2011), pp. 47-54.

[65] GOOGLE, INC. *Introduction to Using Chart Tools - Google Chart Tools*, Em linha http://developers.google.com/chart/interactive/docs/index (Consultado a 21 de Junho d 2012)

[66] GOOGLE, INC. *Charts Gallery - Google Chart Tools*, Em linha http://google-developers.appspot.com/chart/interactive/docs/gallery (Consultado a 21 de Junho de 2012)

[67] GOOGLE, INC. *Customizing Charts - Google Chart Tools*, Em linha http://developers.google.com/chart/interactive/docs/customizing_charts (Consultado a 21 de Junho de 2012)

[68] GOOGLE, INC. *Handling Events - Google Chart Tools*, Em linha http://developers.google.com/chart/interactive/docs/events (Consultado a 21 de Junho de 2012)

[69] GOOGLE, INC. *Controls and Dashboards - Google Chart Tools*, Em linha http://developers.google.com/chart/interactive/docs/gallery/controls#overview (Consultado a 21 de Junho de 2012)

[70] J. SWEENY, *Creating Your Own SIEM and Incident Response Toolkit Using Open Source Tools*, SANS Institute InfoSec Reading Room (2011).

[71] GOOGLE, INC. *DataTable - Google Visualization API Reference - Google Chart Tools*, Em linha http://developers.google.com/chart/interactive/docs/reference#DataTable (Consultado a 21 de Junho de 2012)

[72] M. ÅSBERG, *Enhancing the Interactivity in the Classroom via Smartphone*, Uppsala University, Department of Information Technology, 2011.

[73] GOOGLE, INC. *Data Queries - Google Chart Tools*, Em linha http://developers.google.com/chart/interactive/docs/queries (Consultado a 21 de Junho de 2012)

[74] GOOGLE, INC. *Query - Google Visualization API Reference - Google Chart Tools*, Em linha http://developers.google.com/chart/interactive/docs/reference#Query (Consultado a 21 de Junho de 2012)

[75] K. ANDREWS and M. LESSACHER, *Liquid Diagrams: Information Visualisation Gadgets*, *Information Visualisation (IV), 2010 14th International Conference*, 2010, pp. 104-109.

[76] S. K. DAS, G. GHIDINI and V. GUPTA, *SNViz: Analysis-oriented Visualization for the Internet of Things*, *Internet of Things 2010 Conference*, Tokyo, Japan, 2010.

[77] HIGHSOFT, SOLUTIONS. *Highcharts - Interactive JavaScript charts for your webpage*, Em linha http://www.highcharts.com/ (Consultado 23 de Junho de 2012)

[78] HIGHSOFT, SOLUTIONS. *Highcharts - Highcharts product*, Em linha http://www.highcharts.com/products/highcharts (Consultado a 23 de Junho de 2012)

[79] HIGHSOFT, SOLUTIONS. *Highcharts - Compatibility*, Em linha http://www.highcharts.com/documentation/compatibility (Consultado a 23 de Junho de 2012)

[80] HIGHSOFT, SOLUTIONS. *Highcharts Demo Gallery* Em linha http://www.highcharts.com/demo/ (Consultado a 23 de Junho de 2012)

[81] HIGHSOFT, SOLUTIONS. *Highcharts - Options Reference*, Em linha http://www.highcharts.com/ref/ (Consultado a 23 de Junho de 2012)

[82] HIGHSOFT, SOLUTIONS. *Highcharts - How To Use*, Em linha http://www.highcharts.com/documentation/how-to-use (Consultado a 23 de Junho de 2012)

[83] IBM, CORPORATION. *Many Eyes*, Em linha http://www-958.ibm.com/software/data/cognos/manyeyes/ (Consultado a 25 de Junho de 2012)

[84] F. B. VIEGAS, M. WATTENBERG, F. VAN HAM, J. KRISS and M. MCKEON, *ManyEyes: a Site for Visualization at Internet Scale*, Visualization and Computer Graphics, IEEE Transactions on, 13 (2007), pp. 1121-1128.

[85] C. M. DANIS, F. B. VIEGAS, M. WATTENBERG and J. KRISS, *Your place or mine?: visualization as a community component*, *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, ACM, Florence, Italy, 2008, pp. 275-284.

[86] IBM, CORPORATION. *Many Eyes: Visualization Options*, Em linha http://www-958.ibm.com/software/data/cognos/manyeyes/page/Visualization_Options.html (Consultado a 25 de Junho de 2012)

[87] ADOBE, SYSTEMS. *Application Programming - Adobe Flash Plataform*, Em linha http://www.adobe.com/flashplatform/ (Consultado a 25 de Junho de 2012)

[88] Z. YING and S. KARMAKAR, *Analysis of a social data visualization web site*, *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, 2010, pp. 172-175.

[89] P. TURLEY and R. M. BRUCKNER, *Microsoft SQL Server Reporting Services Recipes: for Designing Expert Reports*, Wrox Press Ltd., 2010.

[90] M. S. DESOUKI, *Dynamic Reports in Administrative Applications*, *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, 2006, pp. 478-482.

[91] H. YU, Y. YANG, Y. LI and J. WANG, *Research and Design of General Report Form System*, *Machine Vision and Human-Machine Interface (MVHI), 2010 International Conference on*, 2010, pp. 25-28.

[92] MICROSOFT, CORPORATION. (2005) *Integrating Reporting Services into Your Application*, Em linha http://msdn.microsoft.com/en-us/library/aa964126%28v=sql.90%29.aspx (Consultado a 29 de Junho de 2012)

[93] V. TERZIEVA and K. TODOROVA, *Using MS reporting services to promote reuse of learning objects*, *Proceedings of the 2007 international conference on Computer systems and technologies*, ACM, Bulgaria, 2007, pp. 1-6.

[94] S. MISNER, *Microsoft SQL Server 2005 Reporting Services Step by Step*, Microsoft Press, 2006.

[95] SAP, AG. *SAP - SAP Business Objects SAP Crystal Reports SAP Crystal Reports Visual Advantage*, Em linha http://www.sap.com/solutions/sap-crystal-solutions/query-reporting-analysis/sapcrystalreports/index.epx (Consultado a 29 de Junho de 2012)

[96] SAP, AG. *SAP - SAP Crystal Reports: Features & Functions*, Em linha http://www.sap.com/solutions/sap-crystal-solutions/query-reporting-analysis/sapcrystalreports/featuresfunctions/index.epx (Consultado a 29 de Junho de 2012)
[97] B. LOWAGIE, *iText in Action*, Manning Publications Co., 2010.
[98] 1T3XT, BVBA. *iText: what is it, what can you do with it?*, Em linha http://itextpdf.com/itext.php (Consultado a 29 de Junho de 2012)
[99] X. XUEYA and Z. YANMEI, *The research and application of the creation PDF document based on the iTextSharp*, *Robotics and Applications (ISRA), 2012 IEEE Symposium on*, 2012, pp. 108-110.
[100] UNIVERSIDADE, DE AVEIRO. *Sistema de Indicadores da UA*, Em linha http://indicadoressitua.ua.pt/ (Consultado a 20 de Agosto de 2012)
[101] I. SOMMERVILLE, *Software Engineering (9th Edition)*, Addison-Wesley, 2011.
[102] GOOGLE, INC. *Google Chart Tools: Image Charts*, Em linha http://developers.google.com/chart/image/ (Consultado a 21 de Agosto de 2012)
[103] M. FOWLER, *UML Distilled - Third Edition - A Brief Guide To The Standard Object Modeling Language*, Addison-Wesley, 2004.
[104] M. FAHNDRICH, M. CARBIN and J. R. LARUS, *Reflective program generation with patterns*, *Proceedings of the 5th international conference on Generative programming and component engineering*, ACM, Portland, Oregon, USA, 2006, pp. 275-284.
[105] Z. RUIZHEN, W. ZUKUAN, T. SHUGUANG and K. JAE-HONG, *Study on application of .NET reflection in automated testing*, *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, 2010, pp. 797-800.
[106] MICROSOFT, CORPORATION and MSDN. *System.Reflection Namespace*, Em linha http://msdn.microsoft.com/en-us/library/136wx94f (Consultado a 5 de Setembro de 2012)
[107] MICROSOFT, CORPORATION and MSDN. *ServiceDescription Class (System.Web.Services.Description)*, Em linha http://msdn.microsoft.com/en-us/library/system.web.services.description.servicedescription.aspx (Consultado a 6 de Setembro de 2012)

Anexos

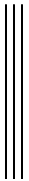
A1 Comandos de criação e inicialização da base de dados "Armazenamento de dados de operação"

```
create database VisualizationsDB
USE VisualizationsDB
go

-- Create database tables

Create Table VisType
(
      ID_VisType int identity not null,
      Symbol nvarchar(20) not null,
      Descript nvarchar(100) not null,
      Constraint pk_vistype_id Primary Key (ID_VisType),
      Constraint un_Symbol_Vis UNIQUE (Symbol)
);

Create Table TableType
(
      ID_TableType int identity not null,
      Symbol nvarchar(10) not null,
      Descript nvarchar(100) not null,
      Constraint pk_tabletype_id Primary Key (ID_TableType),
      Constraint un_Symbol_Tab UNIQUE (Symbol)
);

Create Table Representation
(
      ID_Representation int identity not null,
      ID_VisType int not null,
      ID_TableType int not null,
      Constraint pk_representation_id Primary Key (ID_Representation),
      Constraint fk_vistype_id Foreign Key (ID_VisType) References
VisType(ID_VisType),
      Constraint fk_tabletaype_id Foreign Key (ID_TableType) References
TableType(ID_TableType)
);

Create table Chart
(
      ID_Chart int identity not null,
      ChartIdentificator nvarchar(20),
      Chart_Description nvarchar(50),
      Constraint pk_chart_id Primary Key (ID_Chart),
      Constraint un_ChartIdentificator_Chart UNIQUE (ChartIdentificator)
);

Create table ChartVisualization
(
      ID_Representation int not null,
      ID_Chart int not null,
```

```
      Constraint pk_chartvisualization_id Primary Key (ID_Representation,
ID_Chart),
      Constraint fk_representation_id Foreign Key (ID_Representation)
References Representation(ID_Representation),
      Constraint fk_chart_id Foreign Key (ID_Chart) References
Chart(ID_Chart)
);

Create Table Hint
(
      ID_Hint int identity not null,
      ID_Chart int not null,
      Comment nvarchar(500),
      Checked bit,
      Constraint pk_hint_id Primary Key (ID_Hint),
      Constraint fk_hint_chart_id Foreign Key (ID_Chart) References
Chart(ID_Chart)
);

Create Table InfoQuery
(
      ID_InfoQuery int identity not null,
      ID_Chart int not null,
      QueryString text not null,
      HasParameter bit not null,
      Title nvarchar(200) not null,
      Constraint pk_infoquery_id Primary Key (ID_InfoQuery),
      Constraint fk_infoquery_chart_id Foreign Key(ID_Chart) References
Chart(ID_Chart)
);

Create Table Parameter
(
      ID_Parameter int identity not null,
      Symbol nvarchar(30) not null,
      Name nvarchar(100) not null,
      Parameter_Description nvarchar(150) not null,
      ParameterType nvarchar(30) not null,
      Constraint pk_parameter_id Primary Key (ID_Parameter)
);

Create Table QueryParameter
(
      ID_InfoQuery int not null,
      ID_Parameter int not null,
      Constraint pk_queryparameter_id Primary Key (ID_InfoQuery,
ID_Parameter),
      Constraint fk_queryparameter_infoquery_id Foreign Key(ID_InfoQuery)
References InfoQuery(ID_InfoQuery),
      Constraint fk_queryparameter_parameter_id Foreign Key(ID_Parameter)
References Parameter(ID_Parameter)
);

Create Table Tag
(
      ID_Tag int identity not null,
      Name nvarchar(100),
      Constraint pk_tag_id Primary Key (ID_Tag),
      Constraint un_name_tag Unique (Name)
```

```
);

Create Table QueryTagger
(
      ID_InfoQuery int not null,
      ID_Tag int not null,
      Constraint pk_querytagger_id Primary Key (ID_InfoQuery, ID_Tag),
      Constraint fk_querytagger_infoquery_id Foreign Key (ID_InfoQuery)
References InfoQuery,
      Constraint fk_querytagger_tag_id Foreign Key (ID_Tag) References
Tag
);

Create Table InfoNameSpace
(
      ID_InfoNameSpace int identity not null,
      FullClassName nvarchar(200) not null,
      DLLLocation nvarchar(260) not null,
      INSDescription nvarchar(300) not null,
      IsUserQueriable bit not null,
      Constraint pk_infonamespace_id Primary Key (ID_InfoNameSpace)
);

Create Table NameSpaceConnector
(
      ID_InfoNameSpace int not null,
      ID_InfoQuery int not null,
      Constraint pk_namespaceconnector_id Primary Key (ID_InfoNameSpace,
ID_InfoQuery),
      Constraint fk_namespaceconnector_infonamespace_id Foreign Key
(ID_InfoNameSpace) References InfoNameSpace,
      Constraint fk_namespaceconnector_infoquery_id Foreign Key
(ID_InfoQuery) References InfoQuery
);

Create Table DataRepresentation
(
      ID_DataRepresentation int identity not null,
      Title nvarchar(500) not null,
      JSONExecutionData text not null,
      Constraint pk_datarepresentation_id Primary Key
(ID_DataRepresentation)
);

Create Table QueryDataRepConnector
(
      ID_InfoQuery int not null,
      ID_DataRepresentation int not null,
      Constraint pk_querydatarepconnector_id Primary Key (ID_InfoQuery,
ID_DataRepresentation),
      Constraint fk_querydatarepconnector_infoquery_id Foreign Key
(ID_InfoQuery) References InfoQuery,
      Constraint fk_querydatarepconnector_datarepresentation_id Foreign
Key (ID_DataRepresentation) References DataRepresentation
);

Create Table GraphicVisualization
(
      ID_GraphicVisualization int identity not null,
```

```
      ID_Chart int not null,
      GVTitle nvarchar(500) not null,
      JSONTableVis text not null,
      Constraint pk_graphicvisualization_id Primary Key
(ID_GraphicVisualization),
      Constraint fk_graphicvisualization_chart_id Foreign Key (ID_Chart)
References Chart
);


-- Populate VisType Table

Insert into VisType(Symbol, Descript)
Values ('area','Area chart');

Insert into VisType(Symbol, Descript)
Values ('bar','Bar chart');

Insert into VisType(Symbol, Descript)
Values ('column','Column chart');

Insert into VisType(Symbol, Descript)
Values ('gauge','Gauge chart');

Insert into VisType(Symbol, Descript)
Values ('geo','Geo chart');

Insert into VisType(Symbol, Descript)
Values ('line','Line chart');

Insert into VisType(Symbol, Descript)
Values ('pie','Pie chart');

Insert into VisType(Symbol, Descript)
Values ('scatter','Scatter chart');

Insert into VisType(Symbol, Descript)
Values ('steppedarea','Stepped Area chart');

Insert into VisType(Symbol, Descript)
Values ('tablechart','Table chart');

-- Populate TableType Table

Insert into TableType(Symbol, Descript)
Values ('1cls1ro','1 column string 1 row');

Insert into TableType(Symbol, Descript)
Values ('1clsYro','1 column string Y row');

Insert into TableType(Symbol, Descript)
Values ('1cln1ro','1 column number 1 row');

Insert into TableType(Symbol, Descript)
Values ('1clnYro','1 column number Y row');

Insert into TableType(Symbol, Descript)
Values ('2clsn1ro','2 column string number 1 row');
```



```
Insert into TableType(Symbol, Descript)
Values ('2clsnYro','2 column string number Y row');

Insert into TableType(Symbol, Descript)
Values ('2clcon1ro','2 column country number 1 row');

Insert into TableType(Symbol, Descript)
Values ('2clconYro','2 column country number Y row');

Insert into TableType(Symbol, Descript)
Values ('2clnn1ro','2 column number number 1 row');

Insert into TableType(Symbol, Descript)
Values ('2clnnYro','2 column number number Y row');

Insert into TableType(Symbol, Descript)
Values ('Xclsn1ro','X column string number 1 row');

Insert into TableType(Symbol, Descript)
Values ('XclsnYro','X column string number Y row');

Insert into TableType(Symbol, Descript)
Values ('Xclss1ro','X column string string 1 row');

Insert into TableType(Symbol, Descript)
Values ('XclssYro','X column string string Y row');

Insert into TableType(Symbol, Descript)
Values ('Xclnn1ro','X column number number 1 row');

Insert into TableType(Symbol, Descript)
Values ('XclnnYro','X column number number Y row');

Insert into TableType(Symbol, Descript)
Values ('Xdefault','Default table type. The table does not fit in any
other type');


-- Populate Representation Table

-- 1cls1ro --

Insert into Representation(ID_TableType,ID_VisType)
Values (1,12);


-- 1clsYro --

Insert into Representation(ID_TableType,ID_VisType)
Values (2,12);


-- 1cln1ro --

Insert into Representation(ID_TableType,ID_VisType)
Values (3,6);

Insert into Representation(ID_TableType,ID_VisType)
Values (3,12);
```

```
-- 1clnYro --

Insert into Representation(ID_TableType,ID_VisType)
Values (4,12);


-- 2clsn1ro --

Insert into Representation(ID_TableType,ID_VisType)
Values (5,2);

Insert into Representation(ID_TableType,ID_VisType)
Values (5,4);

Insert into Representation(ID_TableType,ID_VisType)
Values (5,6);

Insert into Representation(ID_TableType,ID_VisType)
Values (5,8);

Insert into Representation(ID_TableType,ID_VisType)
Values (5,9);

Insert into Representation(ID_TableType,ID_VisType)
Values (5,11);

Insert into Representation(ID_TableType,ID_VisType)
Values (5,12);


-- 2clsnYro --

Insert into Representation(ID_TableType,ID_VisType)
Values (6,1);

Insert into Representation(ID_TableType,ID_VisType)
Values (6,2);

Insert into Representation(ID_TableType,ID_VisType)
Values (6,4);

Insert into Representation(ID_TableType,ID_VisType)
Values (6,6);

Insert into Representation(ID_TableType,ID_VisType)
Values (6,8);

Insert into Representation(ID_TableType,ID_VisType)
Values (6,9);

Insert into Representation(ID_TableType,ID_VisType)
Values (6,11);

Insert into Representation(ID_TableType,ID_VisType)
Values (6,12);
```

```
-- 2clcon1ro --

Insert into Representation(ID_TableType,ID_VisType)
Values (7,2);

Insert into Representation(ID_TableType,ID_VisType)
Values (7,4);

Insert into Representation(ID_TableType,ID_VisType)
Values (7,6);

Insert into Representation(ID_TableType,ID_VisType)
Values (7,7);

Insert into Representation(ID_TableType,ID_VisType)
Values (7,8);

Insert into Representation(ID_TableType,ID_VisType)
Values (7,9);

Insert into Representation(ID_TableType,ID_VisType)
Values (7,11);

Insert into Representation(ID_TableType,ID_VisType)
Values (7,12);


-- 2clconYro --

Insert into Representation(ID_TableType,ID_VisType)
Values (8,1);

Insert into Representation(ID_TableType,ID_VisType)
Values (8,2);

Insert into Representation(ID_TableType,ID_VisType)
Values (8,4);

Insert into Representation(ID_TableType,ID_VisType)
Values (8,6);

Insert into Representation(ID_TableType,ID_VisType)
Values (8,7);

Insert into Representation(ID_TableType,ID_VisType)
Values (8,8);

Insert into Representation(ID_TableType,ID_VisType)
Values (8,9);

Insert into Representation(ID_TableType,ID_VisType)
Values (8,11);

Insert into Representation(ID_TableType,ID_VisType)
Values (8,12);


-- 2clnn1ro --
```

```
Insert into Representation(ID_TableType,ID_VisType)
Values (9,6);

Insert into Representation(ID_TableType,ID_VisType)
Values (9,10);

Insert into Representation(ID_TableType,ID_VisType)
Values (9,12);


-- 2clnnYro --

Insert into Representation(ID_TableType,ID_VisType)
Values (10,10);

Insert into Representation(ID_TableType,ID_VisType)
Values (10,12);


-- Xclsn1ro --

Insert into Representation(ID_TableType,ID_VisType)
Values (11,2);

Insert into Representation(ID_TableType,ID_VisType)
Values (11,4);

Insert into Representation(ID_TableType,ID_VisType)
Values (11,8);

Insert into Representation(ID_TableType,ID_VisType)
Values (11,11);

Insert into Representation(ID_TableType,ID_VisType)
Values (11,12);


-- XclsnYro --

Insert into Representation(ID_TableType,ID_VisType)
Values (12,1);

Insert into Representation(ID_TableType,ID_VisType)
Values (12,2);

Insert into Representation(ID_TableType,ID_VisType)
Values (12,4);

Insert into Representation(ID_TableType,ID_VisType)
Values (12,8);

Insert into Representation(ID_TableType,ID_VisType)
Values (12,11);

Insert into Representation(ID_TableType,ID_VisType)
Values (12,12);


-- Xclss1ro --
```

```
Insert into Representation(ID_TableType,ID_VisType)
Values (13,12);


-- XclssYro --

Insert into Representation(ID_TableType,ID_VisType)
Values (14,12);


-- Xclnn1ro --

Insert into Representation(ID_TableType,ID_VisType)
Values (15,6);

Insert into Representation(ID_TableType,ID_VisType)
Values (15,10);

Insert into Representation(ID_TableType,ID_VisType)
Values (15,12);


-- XclnnYro --

Insert into Representation(ID_TableType,ID_VisType)
Values (16,10);

Insert into Representation(ID_TableType,ID_VisType)
Values (16,12);


-- Xdefault --

Insert into Representation(ID_TableType,ID_VisType)
Values (17,12);
```

A2 Exemplo de *report* criado pela aplicação *Open Data* @ UA

Gerado em 01-10-2012 10:26:04

universidade de aveiro

Report da representação de dados "Notas de Acesso à UA"

Representação dos resultados encadeados:

CodCurso	CursoUA
8200	Biologia

Ano	Fase
2009	1ª Fase

Nome	NotaCandidatura
Augusto Santos Pereira	125.7
Marta Gorety Amador	125.2

2010	1ª Fase

Nome	NotaCandidatura
Gonçalo Jesus Teixeira	123.5
Filomena Jardim Salvador	136.6
Maria Tavares Formoso	123.8
Maria Filipa Terra Oliveira	187

2010	2ª Fase

Nome	NotaCandidatura
Manuela Gomes Silva	175.2
Maria Salomé Teixeira	123

2010	3ª Fase

Nome	NotaCandidatura
Hugo Macedo Cunha	124.6
Joana Maria Gonçalves	121
Tomas Valente e Cunha	124.4

8216	Música

Ano	Fase
2009	1ª Fase

Nome	NotaCandidatura
Teresa Martins Vilar	123.5
Miguel Fontes Marques	176.2

2010	1ª Fase

Nome	NotaCandidatura
Gilberto Fonseca Augusto	154.6
Fátima Gonçalves Taborda	123.6
Lurdes Manuel Silva	125.3

1/3

Gerado em 01-10-2012 10:26:04

universidade de aveiro

Resultados da query "Cursos na Universidade de Aveiro":

CodCurso	CursoUA
8200	Biologia
8216	Música

Resultados da query "Fases de acesso anuais por nome de curso":

CodCurso	CursoUA	Ano	Fase
8200	Biologia	2009	1ª Fase
8200	Biologia	2010	1ª Fase
8200	Biologia	2010	2ª Fase
8200	Biologia	2010	3ª Fase
8216	Música	2009	1ª Fase
8216	Música	2010	1ª Fase

Resultados da query "Notas de candidatura por nome de curso, ano e fase":

Ano	Fase	Nome	NotaCandidatura
2009	1ª Fase	Augusto Santos Pereira	125.7
2009	1ª Fase	Maria Gorety Amador	125.2
2010	1ª Fase	Gonçalo Jesus Teixeira	123.5
2010	1ª Fase	Filomena Jardim Salvador	136.8
2010	1ª Fase	Maria Tavares Formoso	123.0
2010	1ª Fase	Maria Filipa Terra Oliveira	167
2010	2ª Fase	Manuela Gomes Silva	175.2
2010	2ª Fase	Maria Salomé Teixeira	123
2010	3ª Fase	Hugo Macedo Cunha	124.6
2010	3ª Fase	Joana Maria Gonçalves	121
2010	3ª Fase	Tomás Valente e Cunha	124.4
2009	1ª Fase	Teresa Martins Vilar	123.5
2009	1ª Fase	Miguel Fontes Marques	176.2
2010	1ª Fase	Gilberto Fonseca Augusto	154.8
2010	1ª Fase	Fátima Gonçalves Taborda	123.6
2010	1ª Fase	Lurdes Manuel Silva	125.3

Gerado em 01-10-2012 10:26:04

universidade de aveiro

(Nota: As células da tabela, que não os cabeçalhos, de cor cinzenta, representam os resultados da query pai da query em questão.)