Ruben Miguel Pereira da Fonseca

Licenciatura em Engenharia Informática

Integração de informação histórica georreferenciada

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientadora: Armanda Rodrigues, Professora Auxiliar, Universidade Nova de Lisboa

Co-orientador: Nuno Correia, Professor Catedrático, Universidade Nova de Lisboa

Júri

Presidente: Doutora Susana Maria S. Nascimento M. Almeida

Vogais: Doutora Ana Paula Pereira Afonso

Doutora Maria Armanda Simenta Rodrigues Grueau

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Fevereiro, 2018

Integração de informação histórica georreferenciada

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Resumo

A necessidade de representar e partilhar informação na Internet tem vindo a aumentar ao longo dos anos com diversas entidades, como universidades, cada vez mais a usarem repositórios para cumprir tal objetivo. Não só estas instituições beneficiam com o uso de repositórios, mas também os investigadores que utilizam estes dados em trabalhos científicos. Mas problemas existem, com cada repositório a seguir o seu próprio formato de dados e com a sua própria interface, obrigando investigadores a conhecerem cada um deles para os usar.

Nesta dissertação foi desenvolvida uma plataforma que permite acesso uniforme a repositórios de dados históricos e sobre herança cultural através de uma interface que pode ser integrada por aplicações de terceiros, que para além de simplificar o processo de pesquisa em múltiplos repositórios, converte a informação para um formato comum e deteta relações entre eles. Todo isto com o objetivo de facilitar a pesquisa de informações pelo investigador.

Palavras-chave: Repositórios, Serviços Web, Dados Georreferenciados, Arqueologia, Repositórios Institucionais, Metadados, Integração de Repositórios.

Abstract

The need to represent and share information on the Internet has been increasing over the years, with various entities, such as universities, increasingly using repositories to achieve this goal. Not only do these institutions benefit from the use of repositories, but also researchers who have access to this data for their research. But problems do exist, with each repository following its own data format and its own interface, forcing researchers to know each of them to use them.

In this master's thesis, a platform was developed that allows uniform access to historical and cultural heritage data repositories, through an interface that can be integrated by third parties, that, in addition to simplifying the search process in several repositories, converts the results to a common format and detects relationships between them. All of this in order to facilitate the search of information by the researcher.

Keywords: Repositories, Web Services, Georeferenced Data, Archaeology, Institutional Repositories, Metadata, Repository Integration.

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Capítulo 1

Introdução

O número de repositórios de dados tem vindo a expandir-se nos últimos anos. Estes fornecem acesso a um manancial de informação de grande valor e são, por isso, considerados como fontes de dados para variados tipos de aplicações. No caso da informação histórica e, especificamente dos artefactos arqueológicos existem repositórios para países, áreas, sítios arqueológicos e museus que disponibilizam informação sobre os artefactos em exposição. Atualmente cada um destes repositórios tem a sua própria forma de acesso (interface) e formato de dados, sendo preciso, para o investigador interessado, conhecê-los para conseguir consultá-los e trabalhar com eles. A diversidade de métodos e formatos de acesso a estes repositórios constitui assim um problema, que é necessário estudar.

O mesmo artefacto arqueológico pode encontrar-se representado digitalmente em diferentes repositórios. Estas representações podem seguir formatos de dados diferentes e apresentar diferentes informações sobre o artefacto. Por isso torna-se importante conseguir identificar que os metadados descrevem o mesmo artefacto e desenvolver formas de integração para apresentar um resultado mais completo ao utilizador.

Estes metadados, para além de informação sobre os artefactos, também podem incluir relações para outros artefactos. Para além destas relações, é importante conseguir formar ligações com outros artefactos através da comparação de atributos/campos com o mesmo valor. É ainda preciso ter em conta a semântica destes campos, por exemplo os campos podem apresentar o mesmo valor, mas esse mesmo valor ter significados diferentes, podendo assim induzir em erro, detetando uma relação que não existe.

Um dos pontos importantes para relacionar informação histórica e arqueológica é a sua georreferenciação. A localização geográfica permite a identificação de ligações entre artefactos originários da mesma região na Terra, mas pode ir para além disso. Quando consideramos um artefacto histórico, várias localizações geográficas podem ser relevantes (a sua origem, o sítio arqueológico onde foi encontrado, o local onde se encontra em

exposição, etc.).

O modelo de informação existente por artefacto em cada repositório irá também variar de acordo com os objetos estudados. Os campos utilizados na descrição das diversas peças arqueológicas variam não só consoante a época da peça, mas também a sua tipologia. Por exemplo na catalogação de cerâmica, a arte representada pode ser um fator de extrema importância necessitando de um campo próprio, o que pode não acontecer com bifaces paleolíticos.

Nesta dissertação, pretende-se desenvolver ferramentas de auxílio à pesquisa de dados sobre herança cultural e histórica, facilitando assim o trabalho dos investigadores da área, mas também disseminando a riqueza deste conhecimento. O trabalho desenvolvido no âmbito desta dissertação de Mestrado pretende contribuir para esse objetivo, com estruturas e mecanismos que facilitam a pesquisa e o acesso integrado do investigador, num formato normalizado, aos dados relevantes para o seu trabalho.

1.1 Motivação

Nos tempos atuais, a partilha de dados e informação é um fator importante utilizado para impulsionar o progresso do conhecimento [5]. Devido ao crescente esforço na representação digital da informação e consequente disponibilização online, há cada vez mais a criação de grandes repositórios de informação que atualmente são utilizados como mecanismos de partilha de informação.

Na diretoria de repositórios de acesso livres, Open Doar[1], em 2010 já se encontravam registados 1451 repositórios de acesso livre a informação que abrange as mais variadas áreas.

Uma plataforma que oferece um acesso uniforme a todos os repositórios relevantes numa determinada área de aplicação permite o desenvolvimento de aplicações para diversas audiências, no contexto do tópico em questão, com acesso facilitado aos conjuntos de dados disponíveis. Este tipo de acesso irá ainda facilitar a reutilização de algoritmos desenvolvidos para as aplicações uma vez que não estão intimamente ligados a um repositório, mas sim à plataforma [6]. Novos repositórios poderiam ainda ser adicionados à plataforma de forma transparente para as aplicações suportadas.

1.2 Definição do Problema

Considerando as motivações apresentadas, temos o seguinte problema:

- É possível desenvolver um acesso único a vários repositórios de informação arqueológica, facilitando as atividades de pesquisa e acesso dos utilizadores interessados?

A avaliação do problema leva-nos a analisar um conjunto de problemas mais específicos:

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

- Que tipo de pesquisas disponibilizar?
- Como integrar diferentes repositórios?
- Como distribuir as pesquisas pelos mesmos?
- Como devolver os resultados, detetando relações entre eles?

1.3 Objetivos

Atendendo aos problemas anteriormente enumerados, esta dissertação tem como finalidade o desenvolvimento de uma *framework* que funciona como camada intermédia entre repositórios e aplicações desenvolvidas para os utilizadores, sejam estas disponibilizadas via browsers, aplicações desktop, aplicações móveis etc. Este trabalho foi desenvolvido no âmbito da área da herança cultural e histórica, mas a sua implementação foi o mais independente possível da área de conhecimento para que possa ser implementado em outras áreas. A *framework* contempla:

- Um mecanismo que permite a integração facilitada de acesso a novos repositórios de informação;
- Uma interface normalizada de serviços web para facilitar a utilização por aplicações externas ao projeto;
- A disponibilização dos resultados de pesquisas aos repositórios associados utilizando diversos formatos (XML e JSON);
- Utilizar dados georreferenciados para tentar detetar relações entre os diferentes resultados da pesquisa.

1.4 Contribuições desta Dissertação

A principal contribuição será a *framework* desenvolvida que tem como as principais contribuições:

- A integração de repositórios heterogéneos;
- Deteção de relações entre artefactos arqueológicos;
- Uniformização de acesso a dados arqueológicos (metadados).

Espera-se que esta plataforma auxilie a pesquisa de informação referente ao património cultural e histórico, concedendo o suporte necessário a ferramentas de representação e interação da mesma informação.

1.5 Organização do Documento

Este documento encontra-se estruturado de acordo com os seguintes capítulos:

Capítulo 1 - Introdução: Neste capítulo é apresentada uma introdução à dissertação, tal como a motivação, problemas, objetivos e contribuições da mesma.

Capítulo 2 - Enquadramento: Neste capítulo são apresentados alguns conceitos chave desta dissertação, estando o mesmo dividido em 4 pontos: Repositórios, Serviços Web, Metadados e Dados Georreferenciados.

Capítulo 3 - Trabalho Relacionado: Neste capítulo discutem-se vários trabalhos que, de alguma forma, se assemelham com o trabalho que se pretende desenvolver.

Capítulo 4 - Plataforma de integração de repositórios - Conceção: Apresenta-se o protótipo da *framework* que foi desenvolvida nesta dissertação.

Capítulo 5 - Plataforma de Integração de repositórios - Implementação: Apresentam-se as escolhas de implementação feitas no desenvolvimento desta plataforma.

Capítulo 6 - Avaliação: Neste capítulo vão ser descritas as avaliações realizadas a esta plataforma e os problemas encontrados.

Capítulo 7 - Conclusão e Trabalhos Futuros: É feita a conclusão de todo o trabalho, falando também de algumas ideias para a continuação do desenvolvimento da plataforma.

Capítulo 2

Enquadramento

Neste capítulo são apresentados os conceitos mais importantes para a realização desta dissertação. Este capítulo está dividido em 4 pontos principais:

- Repositórios - O que são e que objetivos servem;
- *Web Services* - O que são *Web Services*, protocolos de comunicação, descritores de serviço e mecanismos de descoberta de serviço;
- Metadados - O que são e para que servem metadados e metadados semânticos;
- Dados Georreferenciados - O que são e como podem ser representados digitalmente.

2.1 Repositórios

Existem várias definições do que é um repositório. Bartlang e Müller[3] definem um repositório como um sistema de alto nível para gerir informação. Por outro lado [17] define repositórios como uma livraria digital ou um arquivo de preservação, que conserva e armazena objetos digitais que podem ter um formato único como e-books ou podem ter uma variedade de formatos como fotografias e mapas. Estes repositórios têm como objetivo principal servir uma instituição ou comunidade, dando suporte às suas ferramentas de acesso e arquivo de informação [4].

Heery e Anderson[14] propõem um conjunto de características que diferenciam repositórios de outras coleções digitais de dados:

- O conteúdo é depositado no repositório, seja pelo criador de conteúdo, proprietário ou por terceiros;
- A arquitetura do repositório permite gerir tanto o conteúdo como os metadados;

- O repositório oferece um conjunto básico de serviços;
- O repositório deve ser sustentável, confiável, bem suportado e bem gerido.

Existem diversos tipos de repositórios[21], no entanto iremos focar-nos apenas em repositórios institucionais, dado que nos interessa facilitar o acesso a informação que tenha sido devidamente verificada e curada. Crow *et al.*[8] define repositórios institucionais como coleções digitais utilizadas para capturar e preservar a produção intelectual de uma ou mais instituições, por exemplo, universidades e museus. Repositórios institucionais são importantes para investigadores, permitindo que os mesmos disseminem os seus trabalhos. A Instituição também beneficia desta disseminação uma vez que esta contribui para o seu prestígio[15].

Nem todos os repositórios estão abertos ao publico, alguns apenas podem ser acedidos por funcionários da instituição ou comunidade a que pertencem, outros apenas permitem o acesso mediante o pagamento de uma subscrição. Como descrito por Chapman *et al.*[7], os repositórios podem apresentar vários formatos diferentes de metadados dentro do mesmo repositório. Isto acontece porque existem diferentes fontes de metadados que alimentam o repositório, podendo os mesmos, por exemplo, ser gerados automaticamente, criados por outros sistemas e integrados, criados pelo autor do artefacto ou pelo funcionário da instituição que mantém o repositório. Devido aos diversos objetivos para que foram criados e às ferramentas que têm de suportar, estes repositórios podem apresentar múltiplos formatos. Todos estes fatores contribuem para a heterogeneidade dos repositórios, dificultando o trabalho de investigadores que, ao depararem-se com os vários formatos de dados, são obrigados a trabalhar sobre eles para os poder organizar ou utilizar numa ferramenta para trabalhar sobre os mesmos[1]. Outro problema que existe é a reutilização de algoritmos que trabalham sobre os dados destes repositórios, uma vez que os mesmos se encontram associados ao formato de um repositório específico[6].

2.2 Serviços Web (*Web Services*)

Segundo o World Wide Web Consortium[26](W3C), *Web Services* são definidos como sistemas que suportam comunicação máquina-máquina através da rede. Um *Web Service* é assim uma interface normalizada de acesso a um recurso ou serviço que pode ser acedida através da Internet ou numa rede privada. Os *Web Services* funcionam em cima de normas abertas (*open standards*) o que permite que a comunicação seja independente da aplicação, da linguagem de programação em que foi construída e do sistema operativo.

Segundo Curbera *et al.*[9], uma framework de *Web Services* consiste em três áreas: protocolos de comunicação (*communication protocols*), descritores de serviço (*service descriptors*) e descoberta de serviço (*service discovery*).

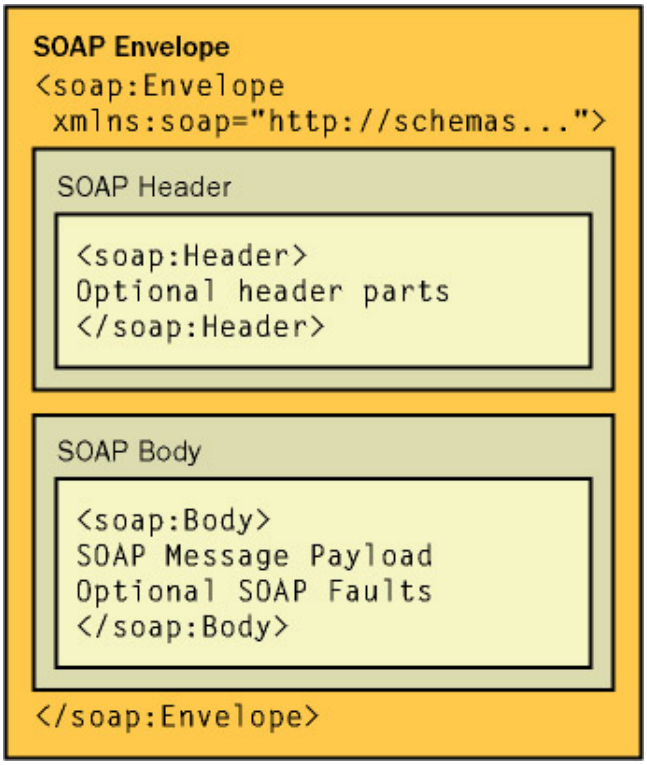

Figura 2.1: Estrutura de mensagens SOAP

(Retirado de: http://letslearn.org.in/post.php?post=120)

2.2.1 Protocolos de comunicação

Protocolos de comunicação são conjuntos de regras e formatos de mensagens que permitem a comunicação entre o remetente e recetor. Existem vários protocolos de comunicação associados a *Web Services* mas iremos focar-nos nos mais utilizados, SOAP e REST.

2.2.1.1 SOAP

Simple Object Access Protocol (SOAP) é um protocolo que utiliza XML para estruturar as mensagens, apresentando assim 4 elementos principais[24]:

- *Envelope* - Este é o elemento raiz da mensagem, define o início e fim da mensagem, contém o *Header* e o *Body*;
- *Header* - É um elemento opcional, que contém informações relacionadas com as aplicações que estão a comunicar como por exemplo, autenticação;
- *Body* - Contém a mensagem que se pretende que seja enviada;
- *Fault* - Sub elemento opcional do *Body* que contém os erros registados ao nível do serviço para aquele pedido, por exemplo, elementos da mensagem em falta.

É possível usar SOAP com qualquer protocolo de transporte, mas a maioria das implementações utilizam HTTP.

2.2.1.2 REST

Representational State Transfer (REST) é um estilo arquitetónico que define um conjunto de princípios para construção de *Web Services*[22].Estes princípios têm como objetivo tornar as aplicações que as usam mais rápidas e escaláveis.

Rodriguez[23] define 4 princípios básicos:

1. Uso de métodos HTTP

   Rest usa de forma explícita métodos HTTP POST, GET,PUT e DELETE relacionando-o 1 para 1 com as operações CRUD (*Create*, *Read*, *Update*, *Delete*) da seguinte forma:

   - POST - Criar um recurso;
   - GET - Obter um recurso;
   - PUT - Atualizar um Recurso;
   - DELETE - Apagar um Recurso.

   Este princípio é importante para definir uma interface que seja o mais generalizável possível.

2. Sem estado (*Stateless*)

   Os pedidos devem de ser independentes entre si, ou seja, devem conter toda a informação necessária para serem processados, de forma que o servidor não precise de guardar estado entre pedidos. O princípio (*stateless*) melhora a performance dos *Web Services* e simplifica o design de componentes do lado do servidor, uma vez que a inexistência de estado remove a necessidade de sincronizar a sessão com a aplicação externa.

3. Identificador Uniforme de Recurso (URI) expostos como diretorias de recursos.

   URIs definem acesso a recursos. Os URIs devem de ser intuitivos o suficiente para que o utilizador consiga adivinhar o URI do recurso que quer, como se fosse uma espécie de interface auto documentada que precisa de poucas ou nenhumas explicações ou referências para que um programador(*developer*) perceba e utilize. Uma forma de conseguir este nível de usabilidade é definindo uma hierarquia estruturada como diretorias para os URIs.

   Por exemplo, imaginemos um *Web Service* para listar sítios arqueológicos com o seguinte URL

   exemplo.com/restapi/sites

   Para listar sítios arqueológicos associados com Portugal já poderia ser algo como:

exemplo.com/restapi/sites/portugal

E ao recebermos os sítios arqueológicos em Portugal queremos aceder a descrição de Tróia:

exemplo.com/restapi/sites/portugal/troia

4. Formato XML, JSON ou ambos

   A interface deve de permitir ao utilizador especificar que tipo de conteúdo melhor se lhe adequa. Assim, serviços devem ter em conta o header HTTP *Accept* do pedido para permitir ao cliente definir o formato da resposta. Isto permite ao serviço ser utilizado por uma maior variedade de clientes.

2.2.2 Descritores de serviço

Descritores de serviço são documentos que descrevem a interface de *Web Services*. Os descritores de serviço providenciam informação sobre os métodos, protocolos e o ponto de acesso de uma interface. Um exemplo de descritor de serviço é o *Web Services Description Language* (WSDL)[1] que usa XML como linguagem para estes documentos e é recomendado pelo World Wide Web Consortium [2]. O uso de um descritor de serviço como o WSDL permite uma mais fácil integração dos *Web Services* por parte dos programados nas suas aplicações uma vez que, atualmente, existe um grande número de ferramentas que conseguem, através deste ficheiro, gerar um cliente para utilizar os serviços, facilitando o desenvolvimento.

O documento WSDL é dividido em 4 elementos principais:

- *Types* - Define os esquemas XML dos diferente tipos de dados utilizados;
- *Message* - Define que tipo de valores serão comunicados entre o cliente e a interface;
- *portType* - Define que operações a interface dispõe;
- *binding* - Define que protocolo é usado pelas diferentes operações.

Listagem 2.1: WSDL Exemplo (retirado de https://www.w3schools.com/xml/xml_wsdl.asp)

```
1 <message name="getTermRequest">
2   <part name="term" type="xs:string"/>
3 </message>
4
5 <message name="getTermResponse">
6   <part name="value" type="xs:string"/>
7 </message>
8
```

[1] https://www.w3.org/TR/wsdl.html
[2] https://www.w3.org

```
<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

2.2.3 Descoberta de serviços

Por descoberta de serviços entende-se como ajudar os utilizadores a descobrir a existência de serviços web que uma empresa tem para oferecer. *Universal Description, Discovery and Integration*(UDDI) é usado com este objetivo. UDDI funciona como um catálogo de serviços web. Entidades registam os seus serviços web no UDDI, usando normalmente um descritor de serviço. Utilizadores podem assim encontrar este serviço ao realizar pesquisas no UDDI.

2.3 Metadados

No seu termo mais geral, metadados são dados sobre dados. Metadatos funcionam como descritores de documentos, artefactos físicos ou até mesmo outros metadados. O *Archaeology Data Service* [2] descreve metadados como sendo informação descritiva de documentos, imagens, dados e outros materiais. O metadados oferecem assim campos e termos para descrever artefactos, facilitando assim a descoberta, a acessibilidade e a usabilidade dos mesmos. Good em, "*A Gentle Introduction to Metadata*" [11], também define que os usos principais dos metadados são: localização de recursos, por exemplo a referência de um artigo para além de ter informação sobre o mesmo, também deverá conter informação sobre onde o mesmo pode ser acedido; e descoberta de recursos, através de um índice de temas de artigos científicos, que é uma coleção de metadados que permite ao investigador descobrir publicações que lhe sejam de interesse através de uma pesquisa por um tema sobre esse índice.

Um exemplo de metadados é a referência dum artigo. Dado que este conjunto de meta informação segue uma estrutura bem definida, podemos dela extrair informação útil como, por exemplo, que a primeira parte é o último nome do autor do artigo. Conhecer a estrutura dos metadados é importante para que possam ser interpretados.

Os metadados são de importância fulcral na internet, pois permitem estruturar recursos por toda a internet, facilitando o trabalho de ferramentas que utilizam estes recursos. Por exemplo, motores de busca usam metadados das páginas, assim como metadados gerados por eles próprios para oferecer ao utilizador pesquisas rápidas pela internet.

2.3.1 Metadados semânticos (Semantic Metadata)

Karimi[16] define *Semantic Metadata* como metadados que descrevem informação contextualmente relevante ou específica de um domínio sobre conteúdo disponível num modelo de metadados partilhado.

Haase [13] identifica como uma das características principais dos metadados a sua precisão. Os metadados necessitam de ser suficientemente precisos para que seja possível distinguir entre dois artefactos e selecionar o correto, detetar se o artefacto é relevante para uma pesquisa, etc; tudo isto sem ser necessário ao utilizador aceder ao próprio artefacto. Metadados mais precisos afetam pesquisas sobre os metadados. Uma imagem descrita como "*German Shepard*" pode não ser encontrada por alguém que pesquisa por "Animais". Os metadados semânticos endereçam este problema relacionando diferentes termos com diferentes relações com o uso de ontologias.

Ontologia é então uma definição de um modelo de dados que representa uma área de conhecimento através do uso de primitivas. As primitivas usadas são normalmente classes, atributos/propriedades e relações [12].

Wang[25] *et al.* define 3 razões para o uso de um modelo com base em ontologias:

- Partilha de conhecimento - O uso de ontologias permite a entidades computacionais interagirem entre si através de um conjunto comum de conceitos sobre um contexto.
- Inferência lógica - com base na ontologia, é possível explorar computacionalmente vários mecanismos de raciocínio para deduzir conhecimento de alto nível a partir de conhecimento de baixo nível.
- Reutilização de conhecimento - Através da reutilização de ontologias Web de diferentes domínios (exemplo: ontologias espaciais e temporais), é possível compor ontologias sem ser preciso começar do zero.

2.4 Dados Georreferenciados

Os Dados Georreferenciados são dados que se encontram referenciados a um espaço físico na superfície do planeta [10]. Estes dados podem ser, por exemplo, leituras realizadas por sensores da precipitação de uma área, ou que associam um artefacto físico a essa área. Estes dados podem apresentar também mais que uma referência a uma posição geográfica. Por exemplo, os metadados de um artefacto arqueológico podem conter uma referência geográfica relativa ao local onde o mesmo foi encontrado e outra relativa a onde se encontra atualmente. MacEachren e Kraak [20] afirmam que cerca de 80% dos dados digitais gerados incluem algum tipo de referenciação geoespacial.

Longley *et al.* [19] identificam dois tipos de métodos de representação digital de dados georreferenciados, raster e vector.

Os dados raster são dados que são estruturados usando um array/matriz de pixéis. Cada entrada da matriz contém um valor que representa um atributo. A combinação

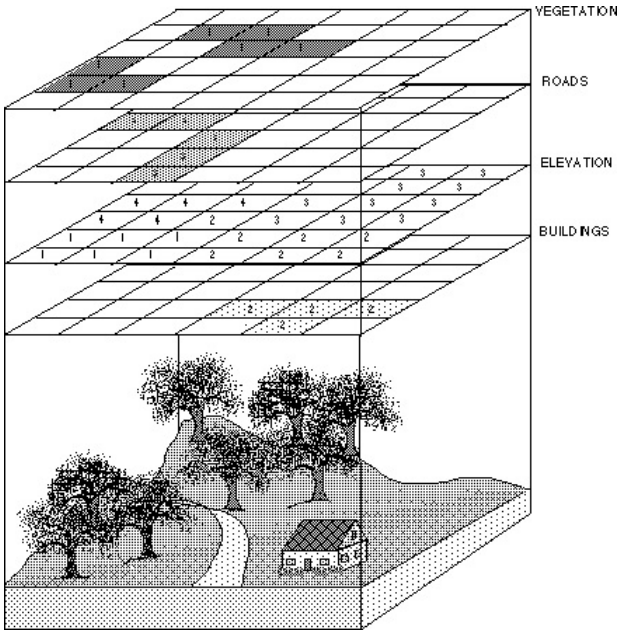

Figura 2.2: Exemplo dados Raster

Retirado de: https://courses.washington.edu/gis250/lessons/introduction_gis/spatial_data_model.html

deste tipo de dados sobre uma área com um atributo específico, como precipitação, é chamado de camada(*layer*)[27], podendo várias camadas existir sobre a mesma área, sendo sobrepostas, como representado na figura 2.2

Os dados vetoriais consistem na representação de informação utilizando vetores e pontos, com os vetores a ser definidos utilizando um ponto inicial e final[27]. Os dados vetoriais suportam a definição de três primitivas: pontos, linhas e polígonos.

Os pontos são a primitiva mais básica. Pontos são usados para representar locais ou entidades não lineares, como hospitais e restaurantes.

As Linhas são representadas por conjuntos de pontos que se conectam através de linhas retas [19], sendo usados para representar características lineares, como estradas ou rios.

Os polígonos funcionam de forma semelhante às linhas sendo representados por pontos que marcam os vértices do polígono[19], sendo usados para representar áreas como cidades, países ou oceanos.

2.5 Conclusão

Neste capítulo foram descritos 4 conceitos importantes para o desenvolvimento da *framework* desta dissertação. Repositórios (secção 2.1) vão ser a fonte informação que vão alimentar a framework, e no qual pretendemos unir os metadados num único output/API para aplicações/clientes. A comunicação entre repositórios-framework e framework-clientes

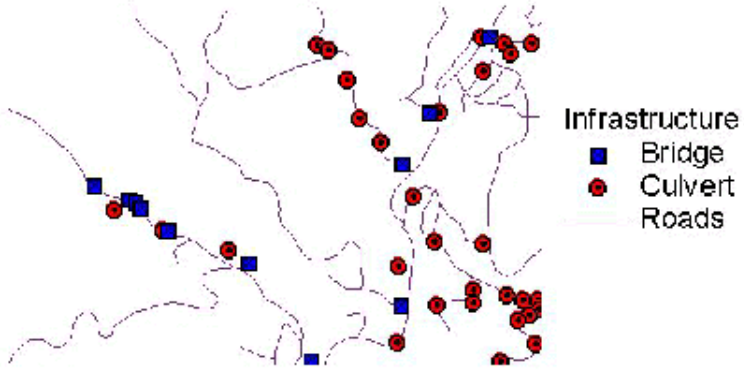

Figura 2.3: Dados Vector: Pontos

Retirado de: https://www.gislounge.com/geodatabases-explored-vector-and-raster-data/

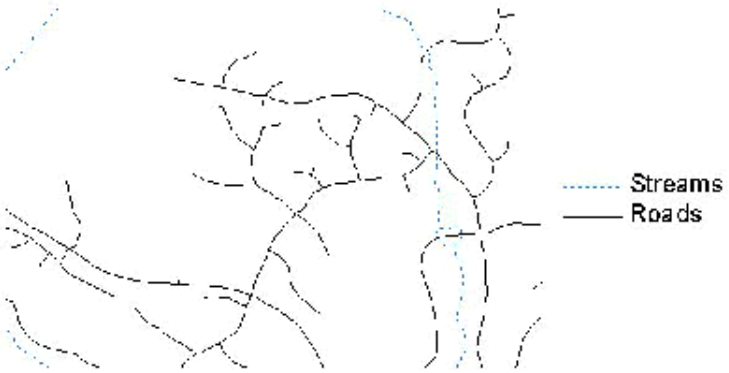

Figura 2.4: Dados Vector: Linhas

Retirado de: https://www.gislounge.com/geodatabases-explored-vector-and-raster-data/

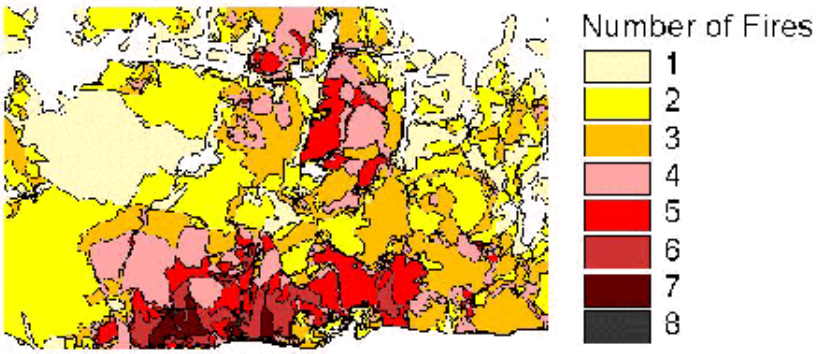

Figura 2.5: Dados Vector: Polígonos

Retirado de: https://www.gislounge.com/geodatabases-explored-vector-and-raster-data/

será realizada através do uso de serviços web (secção 2.2). O uso de serviços web para expor os dados traz um contributo positivo para a framework, que com a sua implementação independente da arquitetura e linguagem permite uma maior variedade de utilizadores. Para tentar detetar relações entre artefactos, a principal informação será a georreferenciação dos dados (secção 2.4). Metadados (secção 2.3) vão estar presentes por toda a framework, e definir uma estrutura é importante para unificar os metadados provenientes dos diferentes repositórios. No futuro seria importante para um sistema deste tipo utilizar uma ontologia para estruturar os metadados, que seria mais difícil de implementar, mas seria uma representação mais fiel da informação, e também, através do uso de mecanismos lógicos sobre a ontologia, as relações detetadas seriam mais e melhores relações entre artefactos.

Capítulo 3

Trabalho Relacionado

Neste capítulo vão ser referenciados vários trabalhos que se encontram relacionados com esta tese.

3.1 HydroDesktop

Ames *et al.* [1] descrevem o software HydroDesktop, que permite pesquisar, aceder e fazer download de dados de observações hidrológicas e climáticas, integrando assim diferentes repositórios de dados pertencentes a diferentes entidades que seguem o *Hydrologic Information System*[1] (HIS). HIS é um projeto desenvolvido pelo *Consortium of Universities for the Advancement of Hydrologic Science, Inc.*[2] (CUAHSI) que fornece *Web Services*, ferramentas, normas e procedimentos que melhoram o acesso a dados hidrológicos.

Os repositórios que seguem o HIS apresentam assim uma interface SOA (arquitetura orientada a serviços) com pelo menos um serviço *WaterOneFlow* para disponibilizar o acesso aos dados numa forma normalizada usando o *Observations Data Model*. *WaterOneFlow* é um grupo de *Web Services* desenvolvido pela HIS com o objetivo de serem o método standard de comunicação e transferência de dados entre os repositórios hidrológicos e os utilizadores. O *Observations Data Model* é um modelo de dados para armazenar e transferir dados hidrológicos.

Estes repositórios encontram-se registados num servidor central. Este servidor central contém uma cópia dos metadados para facilitar a pesquisa, funcionando como um motor de busca.

O HydroDesktop funciona assim como interface que permite fazer pesquisas no servidor central sobre todos os repositórios ou diretamente em repositórios específicos, fazer

[1] http://his.cuahsi.org
[2] https://www.cuahsi.org

download desses dados e guardá-los localmente em formato XML e numa base de dados relacional.

O HydroDesktop assemelha-se ao trabalho que foi desenvolvido nesta dissertação. Existe uma interface que funciona como ponto de união para diferentes fontes de dados que permite ao utilizador aceder aos mesmos de forma transparente. Mas esta aplicação não precisa de lidar com vários problemas que a plataforma que pretendemos desenvolver deve ter em conta. Ao contrário do que foi desenvolvido nesta tese, no HydroDesktop, o acesso aos diferentes repositórios é uniforme devido a todos seguirem o *WaterOneFlow*, não sendo preciso existir uma adaptação para diferentes formas de acesso aos dados. Também não existem dificuldades com os formatos dos dados uma vez que todos os repositórios envolvidos seguem o *Observations Data Model* facilitando a união da informação.

3.2 Generic Data source Management System (GDMS)

Bocher et al. [6] descrevem a heterogeneidade das diferentes fontes de dados e as dificuldades que isso coloca nos Sistemas de Informação Geográfica (GIS), de não terem de se restringir a um formato de dados específico que uma fonte de dados impõe. O GDMS tem como objetivo resolver esse problema, funcionando como uma camada de abstração entre as fontes de informação e estas ferramentas. Para tal, o GDMS integra diferentes fontes de dados através do uso de *Drivers*, e disponibiliza a informação das mesmas, as ferramentas através de uma interface SQL. Esta interface SQL não se encontra diretamente ligada às *Drivers*, uma vez que as mesmas apresentam uma interface de comunicação com as fontes de dados bastante heterogénea o que ia causar problemas na implementação do SQL sobre as mesmas. Para tal existe uma camada sobre estas *Drivers* que disponibiliza uma API homogénea para acesso as mesmas facilitando assim o desenvolvimento da camada SQL Assim a arquitetura do GDMS, como representada na figura 3.1, encontra-se dividida em três partes:

- SQL *Processor*: camada que disponibiliza a interface SQL a ser usada pelas ferramentas.
- *Drivers*: cada driver contém os detalhes de como aceder a uma fonte de dados especifica e de como converter os dados para o modelo interno do GDMS.
- *Datasource*: camada intermédia que lida com heterogeneidade das drivers fornecendo uma interface homogénea para a camada SQL *Processor*.

A implementação desta plataforma resultou numa melhoria na usabilidade de *Spatial Data Infrastructures*, especialmente devido a disponibilização de acesso a todas estas fontes de dados através do uso de uma interface simples e universal como a do SQL.

Esta infraestrutura também apresenta bastantes semelhanças com o nosso trabalho, utilizando SQL como ponto de comunicação homogéneo para com as diferentes fontes de dados, e trabalhando sobre fontes de dados heterogéneas entre si. A diferença encontra-se

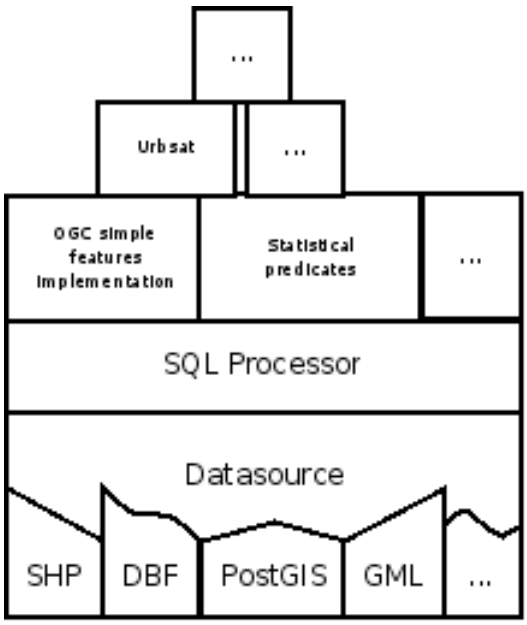

Figura 3.1: Arquitetura GDMS

no acesso às fontes de dados. Na GDMS apresenta-se um acesso total as fontes de dados, uma vez que vai trabalhar diretamente sobre as estruturas que armazenam a informação, por exemplo, base de dados. Isto permite uma maior liberdade de pesquisa de resultados permitindo formas mais complexas de usar os mesmos. Nesta dissertação o mesmo não acontece, sendo o acesso a informação muito mais restrito, estando estes dados acessíveis através de uma API que apenas permite pesquisas bastante simples, como pesquisa por termo.

3.3 Metadata Integration for an Archaeology Collection Architecture

Kulasekaran *et al.*[18] apresentam uma arquitetura para coleções de dados arqueológicos. O desenvolvimento deste trabalho envolveu redesenhar workflows e a definição de duas coleções: a instância de arquivo e a instância de apresentação. Os metadados são parte importante nesta arquitetura, permitindo a integração de dados entre as coleções tal como gerir relações entre artefactos. A arquitetura permite aos investigadores moverem dados de uma coleção para a outra sem a necessidade de redefinir os metadados. Para tal é usado o Dublin Core standard[3] (DC Standard) como base da estrutura dos metadados funcionando assim como uma ponte que permite trocar dados entre as coleções e os workflows, tal como permite criar relações entre artefactos de diferentes repositórios. Na tabela 3.2 estão exemplificados alguns termos que foram mapeados entre a coleção e o Dublin Core.

[3] http://dublincore.org/documents/dces

ARK term	ARK field	Record Keeping Example	DC Term
Short Description	$conf_field_short_desc	Terracotta Figurine	description
File Name	$conf_field_filename	PZ77_725T_b38_p47_f18_M.tif	identifier
Photo Type	$conf_field_phototype	PZ/field/finds/bw	format
Date Excavated	$conf_field_excavyear	1977	date
Date Photographed	$conf_field_takenon	1978	created
Photographed by	$conf_field_takenby	Chris Williams	creator
Area	$conf_field_area	Pantanello	spatial
Zone	$conf_field_zone	Sanctuary	spatial

Figura 3.2: Mapeamento de termos de uma coleção para Dublin Core

Dublin Core Metadata Initiative[4] (DCMI) é um movimento com o objetivo de suportar a inovação e boas práticas no desenho de metadados. Uma das definições de metadados do DCMI é o DC standard. DC standard é uma definição de um vocabulário semântico para descrever a características base de quaisquer objetos digitais, como por exemplo título, criador e descrição.

Também é referido o CIDOC *Conceptual Reference Model*[5](CRM), apesar de a mesma não ser usada no projeto por não ser ideal para o mesmo. CRM é uma ontologia que é construída usando os fundamentos Dublin Core. CRM tem como objetivo facilitar a integração, mediação e troca de informação heterogénea relacionada com património cultural.

Desta arquitetura os dois pontos principais a ter em conta para esta dissertação são o uso da base *Dublin Core* para a estrutura de dados e a ligação entre as duas estruturas de dados das duas coleções. Apesar de neste projeto apenas ter sido feita a ligação entre duas coleções, o uso de uma base como o *Dublin Core* também na estrutura de dados de esta dissertação auxiliou na conversão de dados de diferentes repositórios. Nesta dissertação também não foi utilizado o CIDOC como modelo de dados; a sua complexidade causaria problemas na conversão dos dados, e como a maioria dos repositórios utilizados não apresentam metadados semânticos, esta estrutura não seria usada em todo o seu potencial.

[4] http://dublincore.org
[5] http://www.cidoc-crm.org

Capítulo 4

Plataforma de integração de repositórios - Concepção

Neste capítulo é apresentada e descrita a plataforma desenvolvida no âmbito desta dissertação. Inicialmente, descrevem-se os requisitos a que a plataforma pretende corresponder e uma *User Story*. Apresenta-se a plataforma como um todo e uma simples descrição de todos os seus componentes. Por fim descreve-se a modelação da estrutura de dados.

4.1 Requisitos

Nesta dissertação pretendeu-se desenvolver uma plataforma que funcionasse como intermediário entre repositórios de dados e as aplicações e interfaces que queiram utilizar estes dados, como representado na figura 4.1. Para tal, a plataforma foi desenvolvida tendo em conta os seguintes critérios:

1 - A plataforma terá de disponibilizar uma forma para os diferentes tipos de aplicações comunicarem com ela que possa ser integrável pelas mesmas;

2 - Permitir na plataforma diferentes formas de pesquisar a informação presente nos repositórios;

3 - Tornar a visão dos dados pelos utilizadores/programadores das aplicações independente do repositório do qual os mesmos provêm;

4 - A implementação da plataforma terá de ser independente da área de conhecimento a que se destina nesta dissertação para poder ser usada em outras áreas;

5 - Relacionar a informação proveniente das diferentes fontes com o objetivo de apresentar um resultado mais completo.

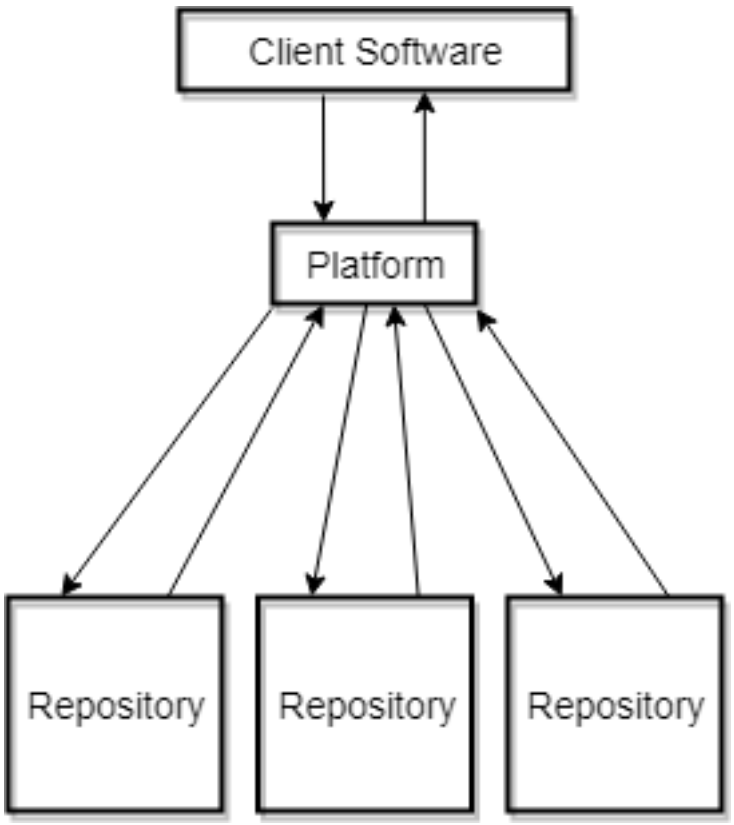

Figura 4.1: Plataforma como camada intermédia entre repositórios e clientes

6 - Possibilitar a integração de novos repositórios de dados sem ser preciso realizar alterações na plataforma.

A necessidade de contemplar os requisitos aqui descritos moldou a arquitetura descrita abaixo.

4.2 Exemplos de utilização

De seguida, apresenta-se um *User Story*:

Como utilizador da Plataforma, eu quero pesquisar informação na plataforma.

Para este *User Story* foram definidos dois cenários usando as duas pesquisas disponíveis; uma de forma geral, outra restringido a repositórios específicos:

Utilizador realiza pesquisa por coordenadas :

1 - Definir área geográfica retangular onde se pretende pesquisar informação;

2 - Chamada ao serviço usando a área definida;

3 - Os resultados de todos os repositórios que permitem uma pesquisa por coordenadas são retornados ao utilizador.

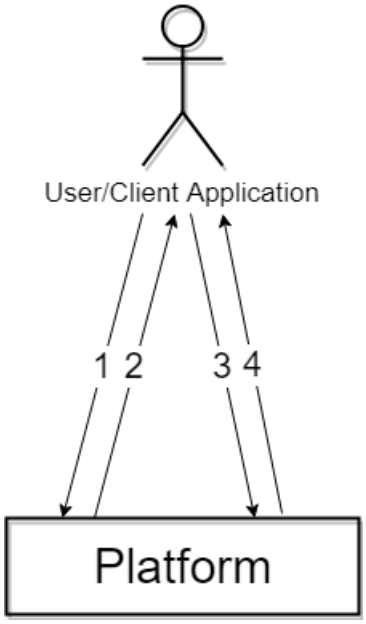

Figura 4.2: User Story

Utilizador realiza pesquisa por termo restringindo a fontes de dados - representado na figura 4.2

1 - O utilizador chama o serviço para listar repositórios;

2 - O utilizador recebe a lista de todos os repositórios e seleciona em quais quer pesquisar;

3 - É feita uma chamada à plataforma por termo com os repositórios selecionados;

4 - O resultado das pesquisas por termo apenas, dos repositórios selecionados são retornados ao utilizador.

4.3 Arquitetura

A plataforma foi desenvolvida seguindo uma arquitetura semelhante à seguida pelo GMDS descrito na secção 3.2, sendo dividida em 4 partes principais(figura 4.3):

- API - Onde se definem as formas de comunicação dos utilizadores com a plataforma;
- *Modelo Interno* - Estrutura de dados que unifica os formatos dos dados provenientes de diferentes fontes.
- *Data Controller* - Onde são detetadas semelhanças entre resultados para unir resultados ou registar relações.
- *Blocos* - Definem as formas específicas de comunicação com cada repositório e a conversão da estrutura dos dados de cada fonte para o modelo interno.

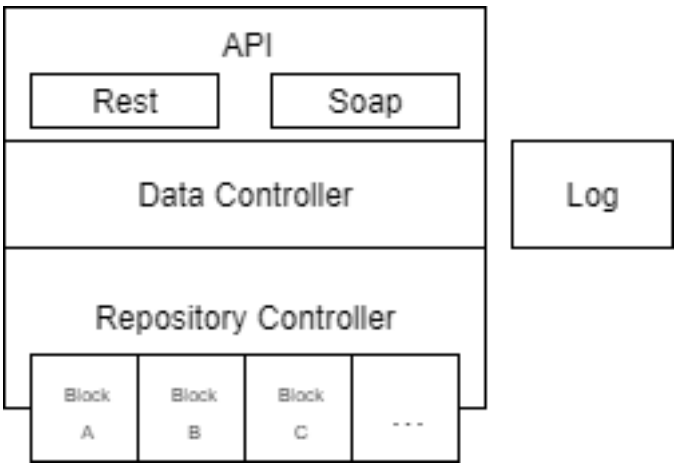

Figura 4.3: Arquitetura da Plataforma

- *Repository Controller* - Controla a comunicação da plataforma com os diferentes repositórios através do uso de blocos, devolvendo os resultados ao *Data Controller*;
- *Log* - Onde são registados os pedidos feitos à plataforma, tal como os passos realizados em cada pedido e os erros detetados em cada passo.

Durante o funcionamento normal da plataforma, a API recebe os pedidos provenientes dos utilizadores, encaminhando esses pedidos para o *Repository Controller*. Os pedidos são distribuídos pelos diferentes repositórios e os resultados devolvidos ao *Data Controller* onde são trabalhados para detetar relações e unir resultados se possível. Por fim os resultados já trabalhados são devolvidos à API, todos os passos realizados para satisfazer este pedido são registados no Log e os resultados são devolvidos ao utilizador.

4.3.1 Comparação com GMDS

Os componentes desenvolvidos apresentam algumas semelhanças com os componentes do GMDS (Figura 4.4):

***Driver* - Bloco** : Em ambas as plataformas estes componentes descrevem como a plataforma deve comunicar com as fontes de dados e como os dados podem ser convertidos para o modelo interno, mas também apresentam pequenas diferenças. As *Drivers* apresentam interfaces heterogéneas entre si sendo preciso uma camada (*DataSource*) para fornecer uma interface homogénea de acesso às mesmas. Blocos seguem uma interface predefinida que é imposta nos mesmos quando são desenvolvidos para poderem ser integrados na plataforma.

Repository Controller* - *DataSource : Estes componentes funcionam como forma de acesso aos Blocos e *Drivers* respetivamente pela plataforma. O *Repository Controller* é a camada que integra os blocos na plataforma e gere as chamadas aos mesmos, enquanto

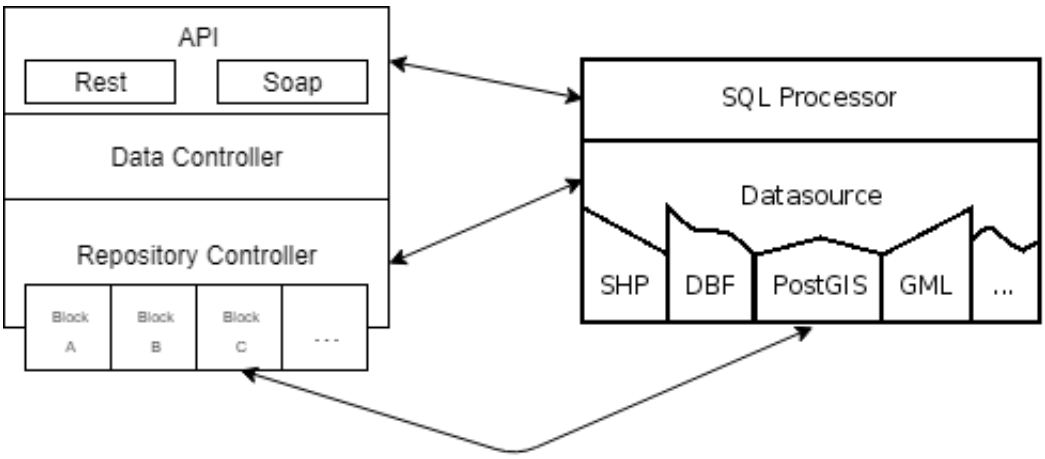

Figura 4.4: Componentes que se relacionam entre a plataforma e GMDS

que a função do *DataSource* é disponibilizar às outras camadas uma interface homogénea de chamadas às drivers, uma vez que as mesmas não o permitem;

API - *SQL Processor* - São os componentes que funcionam como interface de toda a plataforma para que aplicações possam integrar e interagir com a mesma. O componente API fornece uma interface *Web Services*, enquanto que *SQL Processor* fornece uma interface mais complexa através do uso de uma linguagem SQL.

4.4 Modelação do modelo de dados

O Modelo Interno é a estrutura de dados definida para esta plataforma com o objetivo de unificar todos os dados que são acessíveis através da mesma. Para tal, dados provenientes de diferentes repositórios são formatados para seguirem esta estrutura, diminuindo a heterogeneidade dos resultados, facilitando tanto os mecanismos da plataforma como a integração dos dados por uma aplicação cliente, que não têm de lidar com dados seguindo diferentes estruturas. Este modelo foi desenvolvido tendo em conta o Dublin Core, CIDOC CRM e repositórios de informação desta área.

4.4.1 Dublin Core Metadata Initiative

O Modelo Interno começou com a integração dos dois modelos de dados do Dublin Core Metadata Initiative, o Dublin Core e o Dublin Terms, encontrando-se as propriedades dos mesmos com os prefixos Dc e Terms, com o objetivo de mais facilmente os identificar no modelo. Sendo esta uma base de metadados generalista, várias destas propriedades são usadas na maioria dos dados que virão dos repositórios facilitando a sua transformação para o modelo interno. Alguns repositórios como a Europeana também integram na sua estrutura de dados o Dublin Core, como base, não sendo preciso sequer transformação destes metadados.

4.4.2 Modelo Concebido

O Modelo Interno para além do Dublin Core apresenta também propriedades mais especificas para a área da herança cultural para poder integrar informação nesta área. Para tal, foram estudados dados desta área de diferentes repositórios e também o modelo do CIDOC *Conceptual Reference Model*[1], uma ontologia de herança cultural. A mesma não foi totalmente integrada no modelo como o Dublin Core devido a sua complexidade e aos metadados provenientes dos repositórios que serão utilizados não serem dados semânticos. O CIDOC *Conceptual Reference Model* foi analisado tal como os dados dos repositórios com o objetivo de perceber quais as propriedades mais comuns nesta área. Para além destas propriedades mais especificas da área, também existem mais duas propriedades, relationField e relationCoordinates, que servem o propósito apenas de guardar as relações entre resultados detetadas pelos mecanismos.

O Modelo interno atual, representado na figura 4.5, não seria o final para a área de herança cultural; o modelo ainda se apresenta muito simples e teria de ser mais desenvolvido/trabalhado, analisando mais dados para ter a certeza que não fica em falta nenhum pedaço de informação importante quando converter os dados de um repositório para o modelo interno.

4.4.2.1 Propriedades Extra

Para além de todas as propriedades descritas anteriormente existe também uma propriedade para guardar valores que não têm representação direta no modelo, seja porque a propriedade é informação específica do repositório, por exemplo em que coleção se encontram aqueles dados no repositório, seja porque não existe conversão da propriedade em questão para o modelo interno. O objetivo da existência desta propriedade é de não perder qualquer tipo de informação proveniente do repositório, mesmo que esses dados não tenham conversão direta para o modelo interno. Outra vantagem desta propriedade é que ao analisarmos as propriedades extra que vêm nos resultados podemos detetar que propriedades não estão a ser tomadas em conta no modelo interno e adicioná-las.

4.4.3 Outros Modelos

Para utilizar esta plataforma em outras áreas de conhecimento seria preciso desenvolver um esquema diferente com metadados específicos da área em questão. Por exemplo, se fosse para usar na área de conhecimento da biologia, seria preciso uma propriedade para a espécie do animal. Por isso, é importante que a plataforma consiga trabalhar com o modelo interno, mas que esteja presa o mínimo possível a um formato específico do modelo para que o mesmo possa ser alterado. A maior parte do esquema atualmente pode ser alterado, mas mesmo assim existem algumas restrições ao modelo interno, por exemplo que o objeto base se chame Result.

[1] http://www.cidoc-crm.org

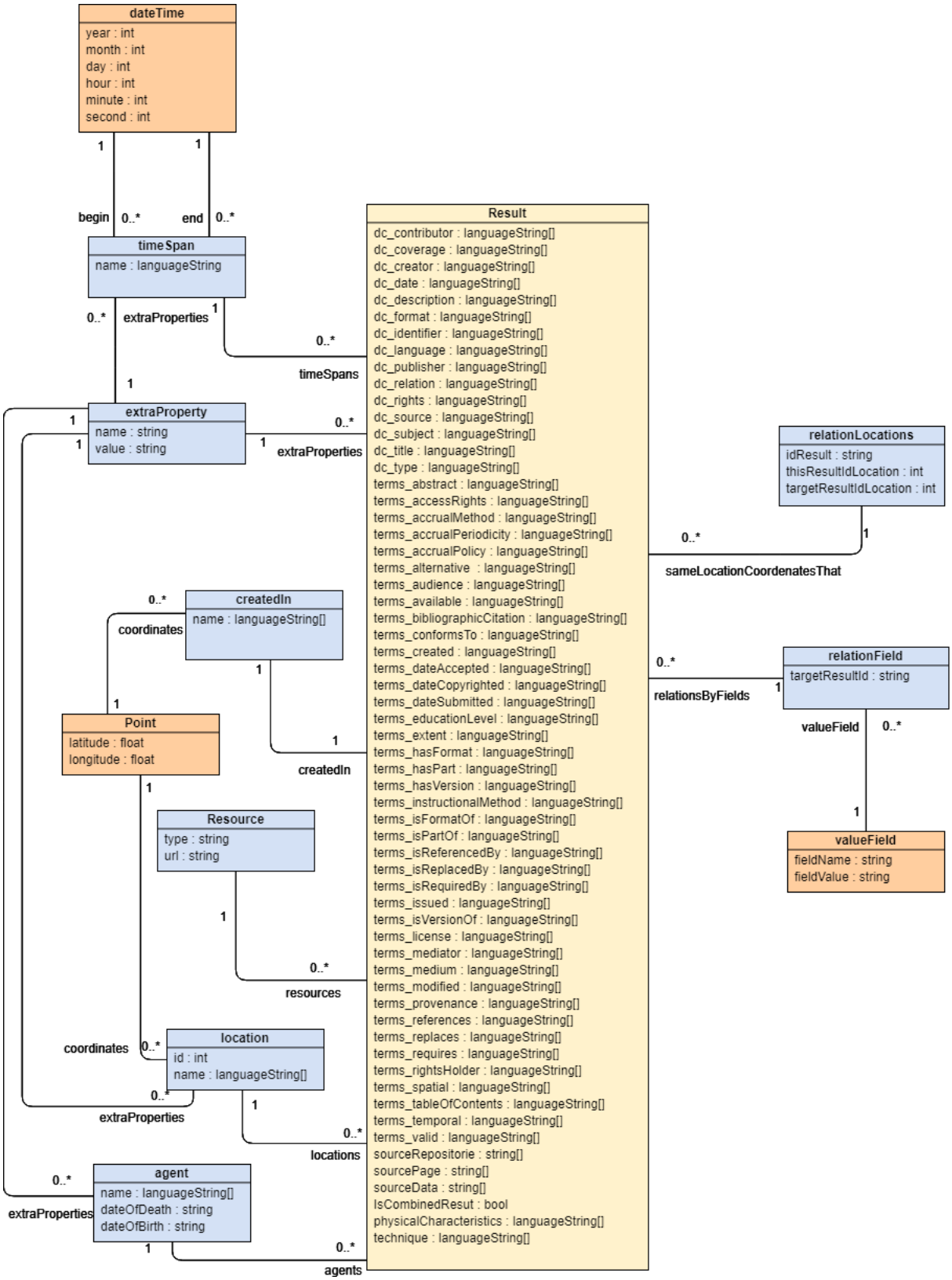

Figura 4.5: Diagrama do modelo interno

4.5 Conclusão

Ao longo deste capítulo foi apresentado o desenho da plataforma desenvolvida e as decisões tomadas em relação à mesma ao longo desta dissertação. Foram descritos os diferentes componentes que constituem a arquitetura com comparação de alguns destes com a plataforma GMDS. Por fim descrevem-se as escolhas de implementação do modelo interno apresentando também a definição final do mesmo.

Capítulo 5

Plataforma de Integração de repositórios - Implementação

5.1 Tecnologias Utilizadas

Nesta secção são descritas as tecnologias que foram utilizadas no desenvolvimento das diferentes componentes da plataforma.

5.1.1 Java

Java[1] é uma linguagem de programação orientada aos objetos, desenvolvida com o objetivo de ser uma linguagem "*Write once, run anywhere*" ou seja, código Java compilado pode correr em todas as plataformas que suportem Java sem ser preciso recompilar. Para tal, programas em Java são tipicamente compilados para *bytecode* que pode ser corrido em qualquer *Java Virtual Machine* independentemente da plataforma.

5.1.2 Spring Framework

Spring[2] é uma *framework open source* em Java usada para desenvolvimento de aplicações. Spring encontra-se dividido em módulos permitindo selecionar apenas as partes da *framework* que são necessárias. A extensão que irá ser utilizada é a *Spring Web Services*[3](Spring-WS). O uso de Spring-WS permite o desenvolvimento de uma API de forma simples, usando uma metodologia *contract first*, ou seja definindo primeiro o ficheiro WSDL, e gerando as classes Java a partir desse ficheiro, para serem usadas no retorno

[1] https://www.java.com
[2] https://spring.io
[3] https://projects.spring.io/spring-ws/

da informação. Estas classes podem ser usadas tanto na API REST como na API SOAP facilitando a uniformização da estrutura dos dados.

5.1.3 Apache Maven

Maven[4] é uma ferramenta para gerir projetos usada principalmente em Java. Baseia-se no conceito de *Project Object Model*, ou seja, utiliza um ponto central de informação, um ficheiro XML, para gerir a compilação e documentação de um projeto. Este ficheiro pode conter dependências, *plugins*, diferentes perfis de compilação etc. O uso de Maven permitirá gerir com maior facilidade o uso de *frameworks* e de *plugins*, que será um grande benefício especialmente para criar blocos, com cada bloco a gerar necessidades de novos *plugins* e dependências.

Um dos *plugins* que vai ser necessário é o JAXB2 Maven[5]. JAXB2 permite gerar classes Java a partir de esquemas XML ou, gerar esquemas XML a partir de classes Java devidamente anotadas.

5.1.4 XML

Extensible Markup Language[6] (XML) é uma linguagem para armazenamento e partilha de informação entre aplicações. XML utiliza *tags*, tal como em HTML, para descrever o seu conteúdo. A estrutura do XML permite que o seu conteúdo seja compreensível tanto para computador como para pessoas.

5.1.4.1 XSD

XML Schema Definition[7] (XSD) é uma definição de uma estrutura para um documento XML. Um documento XSD permite definir um vocabulário com uma lista de elementos e atributos, definir o seu tipo e que restrições se aplicam, bem como documentação sobre os mesmos.

5.1.5 JSON

JavaScript Object Notation[8] (JSON) é um formato para armazenamento e partilha de informação. JSON apresenta duas estruturas principais, coleções de pares chave-valor e listas. Tal como XML, a estrutura do JSON também é legível/compreensível para pessoas

[4] https://maven.apache.org
[5] http://www.mojohaus.org/jaxb2-maven-plugin/Documentation/v2.2/
[6] https://www.w3.org/XML/
[7] https://www.w3.org/TR/xmlschema11-1/
[8] https://www.json.org/

5.1.6 SOAP UI

SOAP UI[9] é uma ferramenta para teste de *Web Services*. Apresenta uma interface gráfica onde permite criar e executar pedidos à API, gerar pedidos através de um ficheiro WSDL entre outros mecanismos.

5.2 Implementação do Modelo Interno

O Modelo Interno encontra-se definido através do uso de um ficheiro XSD. Este ficheiro é posteriormente usado para gerar as classes do modelo interno através do uso do plugin JAXB2 tanto na plataforma como nos repositórios. O ficheiro XSD tambem é utilizado pela plataforma para gerar o ficheiro WSDL que é usado para facilitar a implementação dos serviços SOAP.

Quando o esquema XSD é alterado, o código precisa de voltar a ser gerado para que todas as instâncias que utilizam o esquema se atualizem. Se as alterações do esquema consistirem em adicionar novas propriedades a classes, sem alterar as propriedades que já existem, não é obrigatório atualizar os esquemas nos repositórios se os mesmos não pretenderem usar ou não precisarem das novas propriedades. Por outro lado, se propriedades forem alteradas, por exemplo se uma propriedade passar de *string* para *int*, irão existir problemas entre a plataforma e os repositórios ao nível do formato, causando problemas com o uso do *reflection* do Java, sendo preciso atualizar o esquema em todos os blocos relevantes.

Para usar esta plataforma noutra área de conhecimento, seria apenas preciso desenvolver outra estrutura de dados que fizesse sentido para a área pretendida. Defini-lo num ficheiro XSD, seguindo as restrições que a plataforma impõe, e seguidamente usando o plugin para gerar o código na plataforma e nos blocos. Todos os mecanismos da plataforma, considerando que as restrições foram seguidas, deverão funcionar sem problemas para o novo Modelo Interno.

5.3 Implementação da plataforma

No Apêndice B esta disponível um guia de compilação desta plataforma para um jar executável.

5.3.1 API

A plataforma apresenta uma interface de programação de aplicações (API) para permitir a comunicação e acesso aos serviços que a mesma disponibiliza. A plataforma disponibiliza três serviços principais:

Listar Repositórios: Lista informação dos repositórios presentes na plataforma tal como os identificadores associados a cada repositório. Estes identificadores são utilizados

[9] https://www.soapui.org

nas outras chamadas à plataforma, para restringir a chamada a repositórios específicos, em vez de realizar a chamada a todos os disponíveis.

Pesquisa por Termo: Realiza pesquisas nos repositórios usando um termo definido pelo utilizador. O serviço é por padrão chamado a todos os repositórios que permitem pesquisa por termo, mas o utilizador pode definir repositórios específicos para utilizar na pesquisa, usando os identificadores dos repositórios que pretende utilizar na chamada.

Pesquisa por coordenadas: Realiza pesquisas nos repositórios usando coordenadas definidas pelo utilizador que representam uma caixa/quadrado/área geográfica. Esta caixa é definida através do dois valores de latitude (latitudeFrom,latitudeTo) e dois valores de longitude(longitudeFrom,longitudeTo) como definido na figura 5.1.

A definição dos serviços esta disponível em `https://bit.ly/2DqZh83f`.

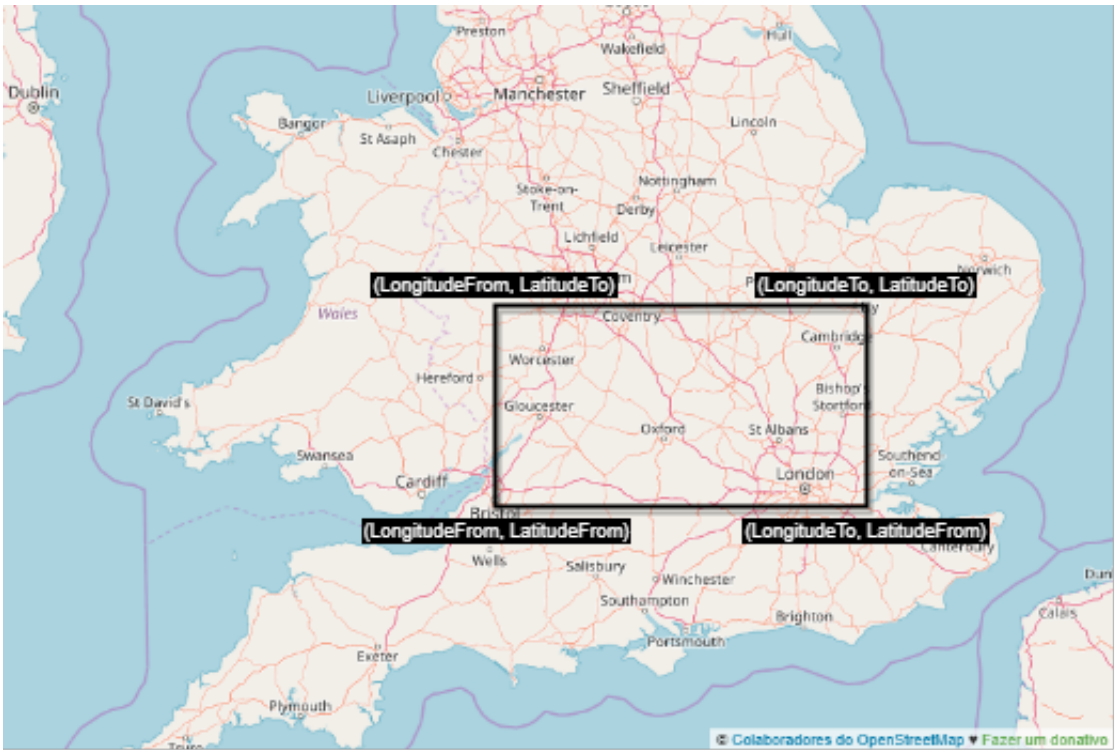

Figura 5.1: Exemplo da definição da box

Para permitir que uma maior diversidade de aplicações possam utilizar esta API, estes serviços encontram-se definidos em formato REST e em formato SOAP, retornando respetivamente a informação em formato JSON e XML, dando a opção ao programador de utilizar os que melhor se adaptam à sua aplicação. Estes dois grupos de serviços apresentam-se em *endpoints* diferentes para ser mais fácil aceder ao formato que se pretende, estando os serviços REST em ip:port/rest/ e os serviços SOAP em ip:port/soap/.

Também, para dar maior liberdade ao utilizador sobre os dados, é possível desligar alguns mecanismos da plataforma para o pedido em questão, caso não seja do interesse do utilizador. O utilizador consegue assim, para os serviços de pesquisa por termo e de

pesquisa por coordenadas, desligar a receção de propriedades extra, desligar o mecanismo para unir artefactos ou desligar o mecanismo para detetar relações.

Todos os serviços são descritos também num esquema XSD e são geradas da mesma forma as classes como no Modelo interno. Estas classes depois são utilizadas para definir o input, output dos serviços tanto SOAP como REST. Este XSD, com os serviços e o modelo interno, também é usado para gerar o ficheiro WSDL, que se encontra disponível na plataforma em `http://10.170.138.201:9200/soap/framework.wsdl` que pode ser usado pelo programador para mais facilmente integrar os serviços SOAP.

5.3.2 Detectar Relações

A plataforma apresenta dois mecanismos para tentar detetar relações entre os diferentes resultados devolvidos ao cliente. Um mecanismo que compara propriedades dos resultados para encontrar semelhanças entre os mesmos, e um que compara coordenadas para detetar proximidade entre os mesmos.

5.3.2.1 Comparação de Coordenadas

As deteções de relações por coordenadas funcionam de duas formas diferentes: caso as localizações do resultado sejam do tipo área ou ponto. Para pontos é definida uma área em torno do ponto e é criada uma relação se algum outro resultado apresentar coordenadas no interior desta área. Por exemplo para um objeto com coordenadas (x,y), é definida uma área retangular com o canto superior esquerdo com coordenadas (x+margem,y-margem) e o canto inferior direito com coordenadas (x-margem,y+margem) e será registada uma relação com todos os resultados que se encontrem totalmente ou parcialmente nesta area (Figura 5.2); caso a localização associada a um resultado seja uma área é apenas verificado que outros resultados estão parcialmente ou totalmente nessa área sem ser preciso definir uma área retangular como no ponto. O valor de margem é um valor float, definido nas propriedades da plataforma sobre a propriedade "CordinatesPointExtraRange" (secção 5.3.5). Esta relação é registada em ambos os resultados na propriedade sameLocationCoordenatesThat, adicionando um objeto relationLocations, aos dois resultados que se relacionam.

5.3.2.2 Comparação de Propriedades

A deteção de relações com propriedades é realizada através de comparação dos objetos LanguageString, verificando, para cada resultado, se para as propriedades selecionadas existem valores iguais para o mesmo idioma. As propriedades que são usadas na comparação são definidas no ficheiro de propriedades da plataforma (ver secção ficheiro propriedades 5.3.5), na propriedade RelationFields.

É possível definir estas propriedades para comparação de várias formas (por exemplo):

RelationFields: DcTitle;

Assim a plataforma compara as propriedades DcTitle de todos os resultados e regista resultados com pelo menos uma entrada igual para o mesmo idioma como uma relação. Para definir caminhos de propriedades, ou seja, propriedades dentro de outras classes, é usado o '-' para separar os vários passos desde a base das classes result até a propriedade que se pretende. Por exemplo para criar relações tendo em conta o Name presente nas Locations define-se:

RelationFields:Locations-Name;

Caso se pretenda que se registem relações tendo em conta várias propriedades basta separar esses campos por ';'. Por exemplo se for pretendido detetar relações tendo em conta DcTitle e DcDate define-se:

RelationFields: DcDate;DcTitle;

Caso se queira criar relações, mas apenas quando ambos os campos se relacionam então define-se vários campos consecutivos separados por '&'. Por exemplo detetar relações quando as propriedades DcTitle apresentam o mesmo valor e, quando as propriedades DcDate e o Name das Location têm valores iguais ao mesmo tempo:

RelationFields: DcTitle;DcDate&Locations-Name;

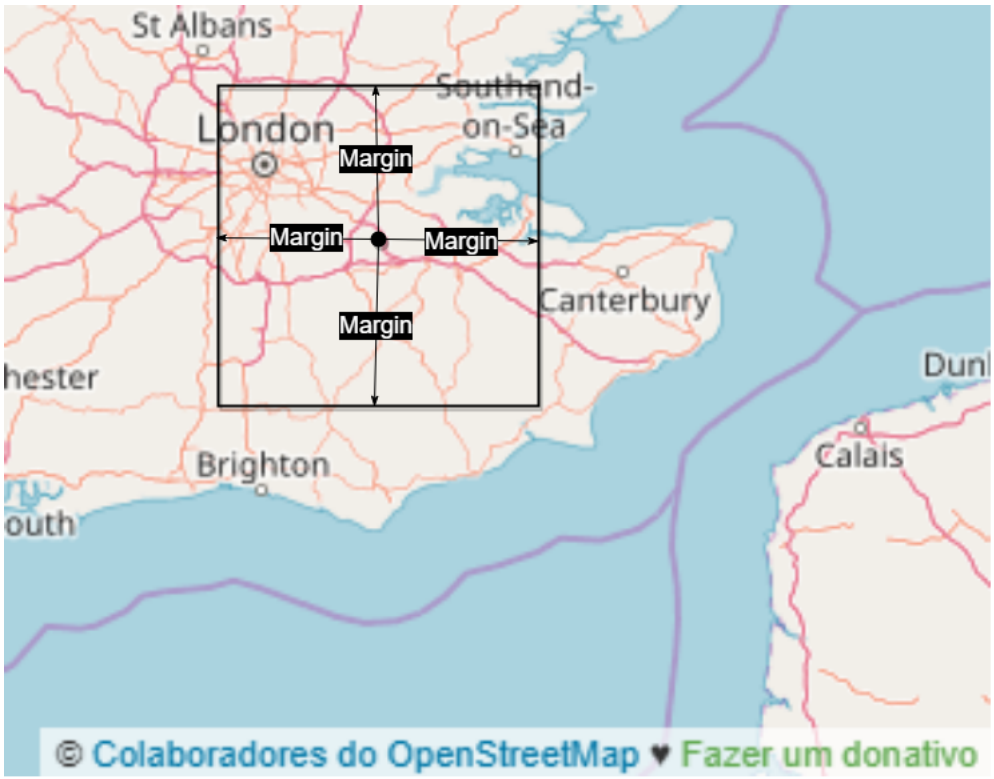

Figura 5.2: Definição da área de relação usando a margem e as coordenadas do artefacto.

5.3.3 Combinar Resultados

A plataforma também permite combinar resultados diferentes em um só resultado. O objetivo deste mecanismo é tentar detetar resultados diferentes, mas que descrevam o mesmo artefacto, por exemplo quando procuramos por um artefacto específico e é retornada informação do mesmo, proveniente de diferentes repositórios. Ao detetar que dois resultados se referem ao mesmo artefacto, estes são unidos num único resultado. Pretende-se com isto, tentar apresentar ao utilizador um resultado mais completo. Para combinar resultados segue-se o mesmo mecanismo descrito na secção Comparação de Propriedades 5.3.2.2, com a diferença que, em vez de criar uma relação quando as propriedades são iguais, os resultados são unidos.

5.3.4 Repository Controller

O *Repository Controller* trata de gerir o acesso aos repositórios presentes na plataforma, carregando-os e distribuindo as chamadas feitas à plataforma pelos mesmos. Os Repositórios são adicionados à plataforma através de jars chamados de blocos. Os Blocos são programados e compilados de forma independente da plataforma, permitindo assim adicionar novos repositórios à plataforma sem ser preciso recompilar a plataforma em si.

5.3.4.1 Blocos

Os Blocos são jars que realizam a comunicação com os repositórios e a conversão dos dados dos mesmos para o modelo interno. Cada bloco é dedicado a uma fonte de dados apenas. É possível colocar um bloco a chamar múltiplos repositórios, mas o *Repository Controller* não conseguiria fazer chamadas individuais a cada um deles, apenas faria a chamada ao bloco em si, que automaticamente chamaria os múltiplos repositórios. Este caso específico poderia ser interessante se os repositórios funcionassem como um complemento da informação de cada um, por exemplo um repositório tem informação de artefactos e o segundo guarda recursos, como imagens desses mesmos artefactos, sendo sempre importante realizar pedidos em conjunto para poder ter as imagens e os dados dos artefactos. Um bloco também pode usar diferentes fontes de dados para além de repositórios, o bloco pode ser desenvolvido para aceder a bases de dados ou ficheiros por exemplo, apesar dos blocos desenvolvidos ao longo desta dissertação, para usar na plataforma, serem repositórios.

Um bloco é desenvolvido seguindo uma interface pré-definida (`https://bit.ly/2NyURkh`). Ao ser seguida essa interface, a plataforma sabe como utilizar o bloco para comunicar com o repositório. A interface contém duas funções, uma função para realizar um pedido por termo, e uma função para pesquisa por coordenadas, descritas respetivamente na tabela 5.1 e 5.2. Estas funções retornam os dados já no formato do modelo interno, retornando assim uma lista de objetos Result.

Interface SearchByTerm		
Argumento	Tipo	Descrição
term	String	Termo de pesquisa a utilizar no repositório
ignoreExtraProperties	Bool	Para ignorar as ExtraProperties do modelo interno

Tabela 5.1: Interface função SearchByTerm.

Interface SearchByBox		
Argumento	Tipo	Descrição
latitudeFrom	int	Valor mínimo de variância da latitude. [latitudeFrom, latitudeTo]
latitudeTo	int	Valor maximo de variância da latitude. [latitudeFrom, latitudeTo]
longitudeFrom	int	Valor minimo de variância da longitude. [longitudeFrom, longitudeTo]
longitudeTo	int	Valor maximo de variância da longitude. [longitudeFrom, longitudeTo]
ignoreExtraProperties	Bool	Para ignorar as ExtraProperties do modelo interno

Tabela 5.2: Interface função SearchByBox.

A classe que implemente esta interface tem de estar no *package repository* com o nome de *Repository*, uma vez que a plataforma vai procurar esse *package* por uma class com esse nome, senão recusa o jar não o adicionando à plataforma.

Uma vantagem do uso de blocos que podem ser desenvolvidos de forma independente é permitir que sejam desenvolvidos por outros programadores que estejam mais familiarizados com o repositório, ou até mesmo pela a própria entidade proprietária do repositório, sem que os mesmos precisem de conhecer ou alterar código da plataforma. Por outro lado, a plataforma não tem conhecimento do que está no bloco, apenas de como o utilizar, não havendo controlo sobre o bloco sendo assim possível introduzir código malicioso na plataforma através dos mesmos. Isto constitui um problema que tem de ser endereçado.

5.3.4.2 Ficheiro Properties do Bloco

Depois de compilado o jar, o mesmo é colocado na pasta "Repositorios" da plataforma, com o ficheiro de properties desse repositório, ambos com o mesmo nome para que a plataforma possa identificar a que jar o ficheiro pertence. O ficheiro apresenta várias informações sobre o repositório, como nome, descrição e quais as funcionalidades que estão disponíveis (Tabela 5.3), que são usadas pelo serviço listar repositórios (Secção 5.3.1) para descrever o repositório ao cliente. Para além destas propriedades, os programadores podem definir outras que considerem importantes para o repositório que estão a desenvolver. Por exemplo, neste protótipo, os blocos associados ao repositório da Europeana e ao repositório da DigitalNZ apresentam uma propriedade extra para guardar a chave de acesso à API do repositório. Como não existe um mecanismo para atribuir identificadores de forma automática aos repositórios, é definido um identificador manualmente no ficheiro. Um exemplo do ficheiro de configuração está presente em `https://github.com/rm-fonseca/Plataforma-Tese/blob/master/Repositorios/example.properties`.

Argumentos do ficheiro de configuração do repositório			
Argumento	Obrigatório	Tipo	Descrição
Name	N	String	Nome do repositório
ID	S	Int	Identificador do repositório usado nos pedidos.
Description	N	String	Descrição do repositório para ser usado no serviço listRepositories
SearchByTerm	N	Bool	Identifica se a funcionalidade SearchByTerm está disponível neste repositório.
SearchByBox	N	Bool	Identifica se a funcionalidade SearchByBox está disponível neste repositório.

Tabela 5.3: Argumentos do ficheiro de configuração do repositório.

5.3.4.3 Controlador

O controlador de repositórios trata de carregar os blocos para a plataforma e de distribuir os pedidos pelos diferentes repositórios. Quando um pedido é feito à plataforma, o controlador divide os pedidos por todos os repositórios, tendo em conta os repositórios selecionados pelo cliente e as funcionalidades que cada um permite, como representado na figura 5.3. Quando nenhum repositório é selecionado pelo cliente, então o pedido é realizado a todos os repositórios possíveis que tenham a funcionalidade associada ao pedido feito, como representado nas figuras 5.4 e 5.5. Os blocos são chamados pelo controlador, cada um na sua própria thread, para que todos os blocos possam trabalhar em paralelo, diminuindo o tempo de espera do cliente.

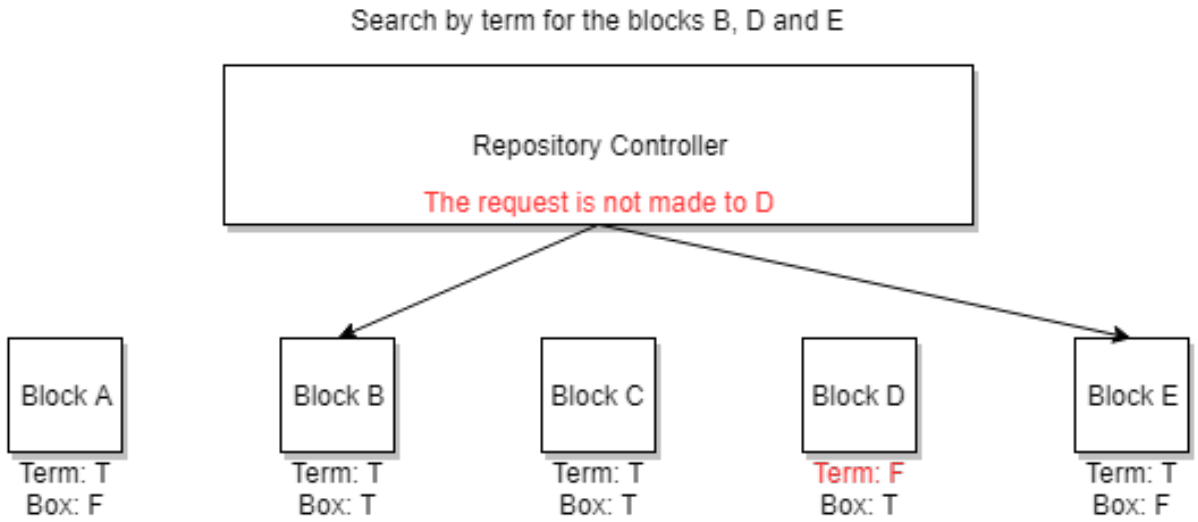

Figura 5.3: Pesquisa por termo com seleção de repositórios.

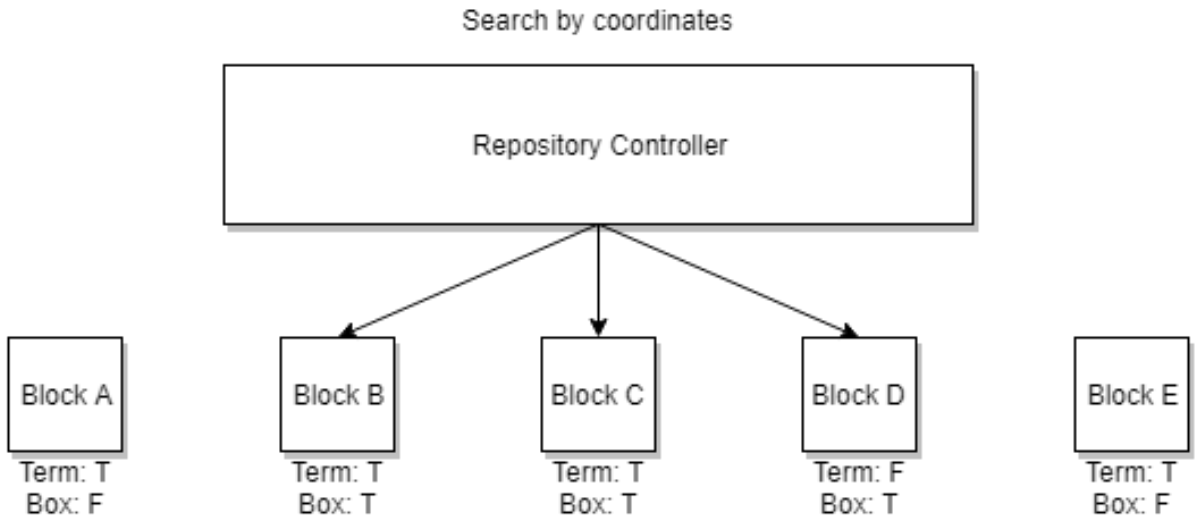

Figura 5.4: Pesquisa por termo.

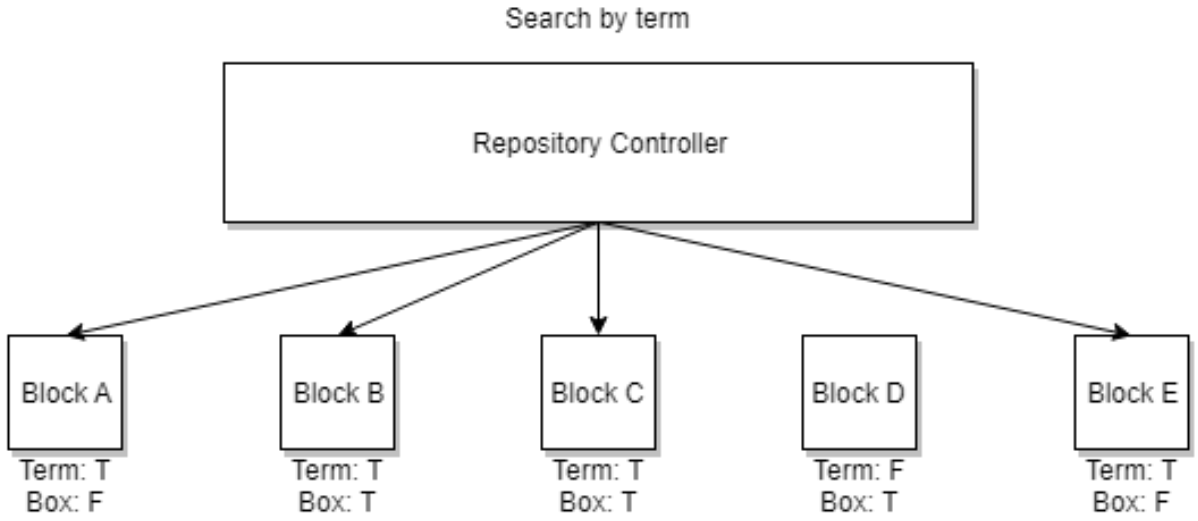

Figura 5.5: Pesquisa por coordenadas.

Ao inicializar a plataforma o controlador carrega todos os repositórios e ficheiros properties presentes na pasta Repositórios para a plataforma e só realiza este processo na inicialização, ou seja, novos repositórios adicionados à pasta apenas seriam integrados na plataforma quando a mesma fosse reiniciada.

Para esta dissertação foram desenvolvidos 4 blocos para quatro repositórios diferentes: Europeana[10], Fitzwilliam Museum[11], Geodados Câmara de Lisboa sobre Casas Religiosas[12] e DigitalNZ[13]. Todos os blocos permitem pesquisas por termo, mas apenas o bloco da Europeana permite pesquisa coordenadas. O código dos blocos está disponível em https://github.com/rm-fonseca/Tese-Repositorio

5.3.5 Ficheiro de Configuração

A plataforma apresenta um ficheiro de configuração com o objetivo de facilitar pequenas configurações/alterações à plataforma para que melhor se adapte à área de conhecimento que se está a trabalhar, sem ser preciso compilar toda a plataforma de novo. Os argumentos do ficheiro de configuração apresentam-se descritos na tabela 5.4.

Para além do ficheiro de configuração da plataforma também existe um ficheiro de configuração do Spring[14] que também pode ser usado para fazer alterações na plataforma, por exemplo definir a porta que é usada para correr os pedidos a API.

[10] https://www.europeana.eu/portal/pt
[11] http://webapps.fitzmuseum.cam.ac.uk/explorer/
[12] http://geodados.cm-lisboa.pt/datasets/cultura-casasreligiosas
[13] https://digitalnz.org/
[14] https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/common-application-properties.html

Argumentos do ficheiro de configuração		
Propriedade	Tipo	Descrição
CordinatesPointExtraRange	float	Valor de margem a ser usado na comparação
RelationFields	String	Propriedades a serem usadas em relações
CombineFields	String	Propriedades a serem usadas em combinações

Tabela 5.4: Argumentos do ficheiro de configuração.

5.3.6 Página Index

No url base da plataforma, existe uma página web para ajudar a orientar os programadores na utilização das funcionalidades da plataforma, disponível na rede da FCT em `http://10.170.138.201:9200`. Apresenta os endpoints REST, SOAP e WSDL da plataforma, e descreve de uma forma simplificada os serviços existentes na plataforma. Também tem uma interface simples para ajudar a construir os pedidos REST com o objetivo de auxiliar o programador a perceber como os utilizar na sua aplicação.

5.3.7 Log

A plataforma apresenta um sistema de logs simples para registar chamadas a mesma e erros encontrados. Quando uma chamada é feita à plataforma é criado um objeto log que regista que pedido foi feito e todos os passos percorridos na plataforma, com um timestamp associado, para satisfazer esse pedido, por exemplo que repositórios foram utilizados, e quais retornaram com sucesso. No final da chamada o log é escrito para um ficheiro. O ficheiro de log apresenta se em formato *Comma-separated values* (CSV) para poder ser visualizado em programas que suportam este ficheiro como por exemplo Microsoft Excel.

5.4 Exemplo de Chamada a plataforma

Na figura 5.7 está descrito o processo de pesquisa por termo:

1 - É realizado um pedido REST pelo utilizador de pesquisa por termo com os identificadores do repositório A, C e D.

2 - É criado um novo objeto log e registado que pedido foi realizado ao repositório.

3 - O termo de pesquisa e os identificadores são encaminhados para o *Repository Controller*.

4 - É lançada uma *thread* para cada repositório selecionado e as mesmas registam no log a que repositório vão aceder.

Figura 5.6: Print da pagina Index.

5 - O termo de pesquisa é enviado para os blocos selecionados.

6/7 - Cada bloco realiza um pedido para o seu repositório e recebe os metadados.

8 - Um erro ocorre durante o pedido ao repositório C e esse erro é retornado para o *Repository Controller*.

9 - É registado no log o repositório que deu erro e o erro associado.

10 - Os metadados são convertidos do formato do repositório para o modelo de metadados interno.

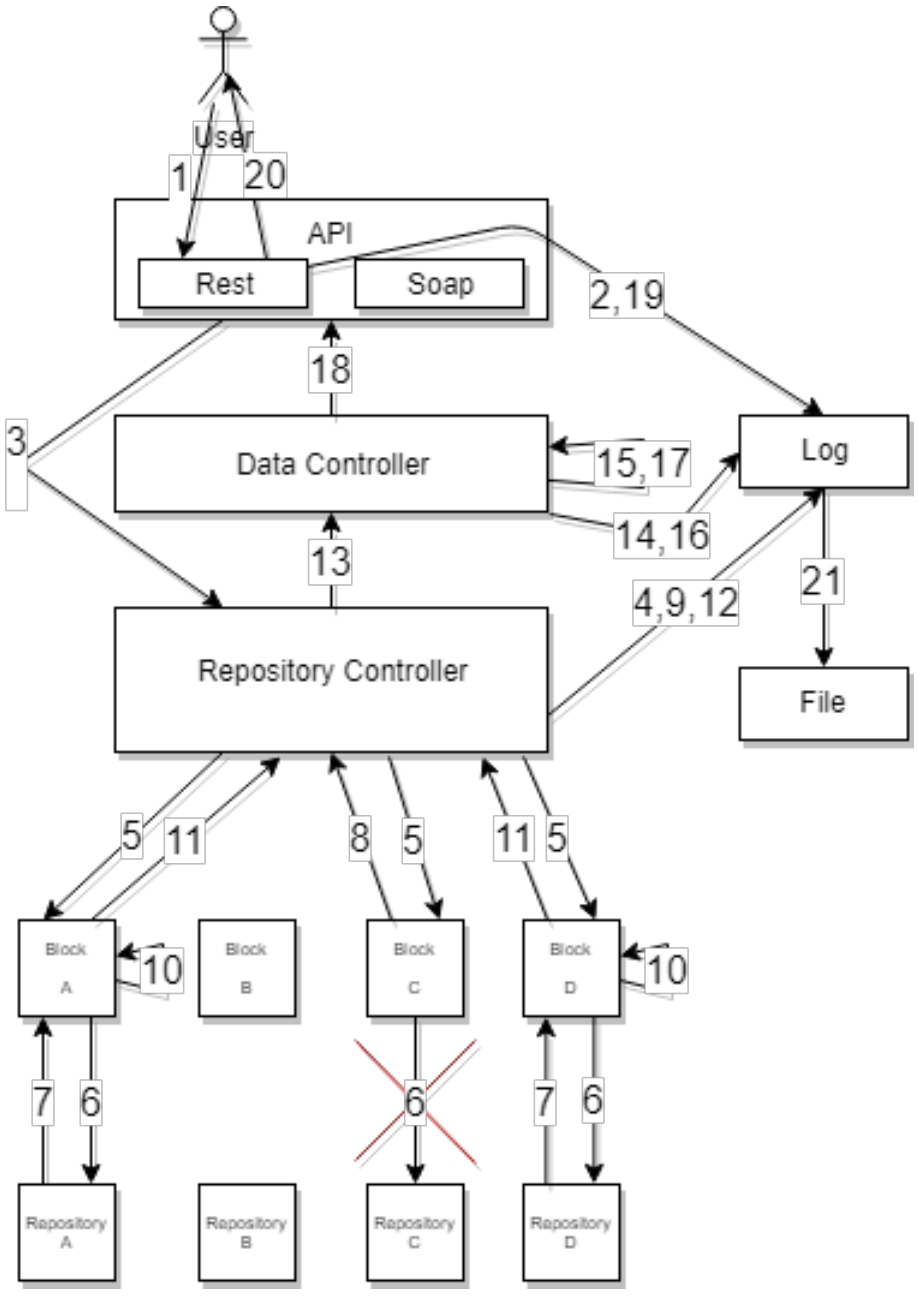

Figura 5.7: Exemplo de um chamada a plataforma.

11 - Os metadados são enviados para o *Repository Controller*.

12 - Regista-se no log os repositórios que responderam com sucesso.

13 - O *Repository Controller* espera pela resposta de todos os blocos e devolve todos os metadados para o *Data Controller*.

14 - A operação de união de resultados é registada no log, assim como que propriedade vai ser usada para realizar a união.

15- Os dados são processados para tentar unir resultados semelhantes num mesmo resultado.

16 - É registada no log a operação de detetar relações entre resultados e que propriedade vai ser usada para detetar relações.

17- Os dados são processados para tentar detetar relações entre resultados.

18 - Os resultados processados são retornados à API.

19 - É registado no Log o fim do pedido e é lançada uma *thread* que irá registar o log no ficheiro de logs.

20 - Os resultados são retornados ao cliente.

21 - A *Thread* regista todo o log da chamada no ficheiro.

5.5 Conclusão

Neste capítulo foram descritas as diferentes escolhas de implementação das componentes apresentadas no capítulo 4. Para o desenvolvimento da plataforma foram escolhidas as tecnologias que melhor se adaptavam aos objetivos ou facilitavam a implementação da mesma. A implementação do Modelo Interno foi realizada através de um ficheiro XSD para mais facilmente o atualizar ou substituir por outro modelo, sem ser preciso realizar alterações ao código da plataforma. Descrevem-se os serviços que a API disponibiliza aos utilizadores, como é que os mecanismos de detetar relações estão implementados no *Data Controller*, a integração dos blocos pelo *Repository Controller*, o sistema de Logs e o ficheiro de configuração. No final uma chamada à plataforma de um ponto de vista mais técnico.

Capítulo 6

Avaliação

Neste capítulo são avaliados os vários pontos da plataforma. Na secção 6.1 iremos falar de quão bem a plataforma segue os requisitos previamente definidos. Em 6.2 é analisado um teste realizado à implementação e integração dos blocos. Na secção 6.3 é feito um teste para confirmar a vulnerabilidade de código malicioso num bloco.

6.1 Cumprimento dos Requisitos

Nesta secção iremos discutir se a plataforma segue cada um dos pontos definidos na secção 4.1.

1 - O uso de *Web Services* permite que as aplicações que usam a plataforma sejam independentes do sistema para o qual são desenvolvidas e da linguagem de programação. A disponibilização de dois tipos de *Web Services*, REST com JSON e SOAP com XML, permite também que os programadores possam escolher que tipo de *Web Services* melhor se integram na aplicação a ser desenvolvida;

2 - As formas de pesquisa presentes na plataforma são: pesquisa por termo e pesquisa por coordenadas. Apesar de estas serem as mais utilizadas nos repositórios, não existe forma de realizar pesquisas mais complexas, como combinar pesquisa por termo e coordenadas, ou pesquisas mais específicas como valores de propriedades, caso os repositórios o permitam;

3 - Através da conversão para o modelo interno dos dados vindos de cada repositório pelo bloco, todos os dados que chegam à plataforma e consequentemente às aplicações são independentes do formato dos repositórios, fornecendo assim uma visão homogénea sobre os dados aos utilizadores;

4 - O modelo interno pode ser quase completamente alterado através do ficheiro do esquema XSD uma vez que a plataforma tenta ser o mais independente possível. Ainda existem algumas especificações que tem de ser seguidas ao desenvolver um modelo interno, mas seguindo essas especificações, podem-se implementar modelos de outras áreas de conhecimento.

5 - O mecanismo de comparação e o de união de resultados funciona como pretendido, o problema encontra-se mais na seleção das propriedades que se pretendem usar tanto na comparação como na união sendo difícil definir que propriedades usar para obter bons resultados ou que sejam interessantes para o utilizador.

6.2 Teste de usabilidade dos blocos

O teste de integração de repositórios tem como finalidade o desenvolvimento de blocos por entidades/indivíduos independentes da plataforma (tabela 6.1). Para tal foi pedido a indivíduos com conhecimentos de programação que implementem blocos da plataforma para acesso a repositórios relevantes. Assim, foi atribuído a cada um, uma fonte de dados diferente, o Museu Victoria[1] e os geodados da Câmara Municipal de Lisboa[2], neste caso dados sobre a Estatuária de Lisboa[3]. Ambas estas fontes de dados estão disponíveis através de uma interface API, mas os geodados da câmara de Lisboa também permitem fazer download dos dados sobre a forma de um ficheiro. Isto permitiu criar um bloco que, em vez de aceder a uma API para receber os dados, utiliza o ficheiro. O objetivo deste repositório é demonstrar que os dados podem vir de diferentes fontes, não sendo obrigatório apenas usar APIs dos repositórios.

Curso	Experiência Profissional	Idade	Fonte de Dados
Licenciatura em Engenharia Informática	2,5 anos Software Engineer	25	Museu Victoria
Licenciatura em Engenharia Informática	2 anos Programador Junior	24	Estatuária de Lisboa

Tabela 6.1: Programadores do teste de usabilidade dos blocos.

[1] https://museumsvictoria.com.au/
[2] http://geodados.cm-lisboa.pt/
[3] http://geodados.cm-lisboa.pt/datasets/estatu%C3%A1ria/data

Para ajudar no desenvolvimento dos blocos foi fornecido aos programadores o guia de desenvolvimento presente no Anexo A.

Inicialmente, foi discutido com um dos programadores o conteúdo do guia, com o objetivo de corrigir qualquer lacuna de informação de forma a tornar o guia o mais compreensível possível.

Os programadores foram acompanhados durante a fase de desenvolvimento para esclarecimento de quaisquer dúvidas ou problemas o que levou a pequenas alterações do guia. Também durante esta mesma fase foram levantados alguns pontos importantes pelos programadores que poderiam melhorar o desenvolvimento de blocos.

- Disponibilizar uma forma de testar os blocos, uma vez que atualmente não existe um conjunto de testes padrão criados para testar o bloco sendo preciso o programador definir os seus próprios testes como, por exemplo, foi feito para o Museu Victoria `https://bit.ly/2DqjRFJ`;

- O desenvolvimento de uma interface que pudesse ser usada para definir as ligações entre a estrutura dos dados de uma fonte e o modelo interno poderia eliminar a necessidade de ter um programador para desenvolver blocos.

Estes pontos serão mais aprofundados na secção 7.2.

Foi realizado um questionário aos programadores (apêndice C) para ajudar a identificar dificuldades com a implementação de blocos. As maiores dificuldades foram apresentadas no Museu Victoria devido a falta de documentação da estrutura de dados do mesmo, o que dificultou a interpretação e, por consequência, dificultou também a definição das ligações entre essa estrutura e o modelo interno.

Depois de desenvolvido o bloco do Museu Victoria foi adicionado à plataforma localmente sem qualquer problema sendo posteriormente, depois de realizadas chamadas de teste, adicionado à plataforma online na rede da FCT.

O Bloco da Estatuária de Lisboa tambem causou alguns problemas. Primeiro existiu a dúvida de onde o ficheiro csv com os dados ficaria. Como não existe uma definição de práticas para estes casos foi definido que o ficheiro estaria na mesma pasta que o bloco e o ficheiro de propriedades do bloco. Outro ponto foi um problema com as dependências. A forma usada para gerar um jar, como descrito no Apêndice A, não adicionava todas as dependências definidas no Maven para o jar. Neste caso as dependências definidas neste bloco para ajudar a trabalhar o ficheiro CSV iam estar em falta quando a plataforma realizasse uma chamada a este bloco não sendo possível utilizar o mesmo. Este problema foi contornado rapidamente utilizando o Maven em vez do eclipse para gerar o jar através do comando "clean compile assembly:single".

6.3 Teste de vulnerabilidades dos blocos

Na secção 5.3.4.1 é referida a vulnerabilidade que os blocos trazem à plataforma, uma vez que a mesma não tem conhecimento sobre o código presente no bloco. Para tal foi pedido a um indivíduo (tabela 6.2) com conhecimento de programação que implementasse um bloco que demonstre esta situação. Para tal foi disponibilizado o mesmo guia que foi fornecido aos programadores da secção 6.2. O objetivo é o individuo ter o mesmo nível de informação sobre os blocos e a plataforma que se tinha no Teste de usabilidade dos blocos, mas tendo um objetivo diferente, neste caso desenvolver código malicioso para a plataforma ou para o servidor, sem causar danos irreversíveis aos mesmos.

Curso	Experiência Profissional	Idade
4º Ano de Mestrado Integrado em Engenharia Informática	—	24

Tabela 6.2: Programadores do teste de vulnerabilidades dos blocos.

O bloco desenvolvido para este teste quando realizada uma pesquisa por termo cria um *spam* de ficheiros na pasta da plataforma, e seguidamente desliga o sistema que está a correr a plataforma. Isso demonstra duas vulnerabilidades principais que os blocos introduzem que podem ser perigosas tanto para a plataforma como para o servidor:

- manipulação de ficheiros;
- manipulação do sistema.

Assim foram discutidas algumas formas para controlar este problema como restrição de permissões dos blocos e através do uso de *sandboxes*. Estes pontos também serão mais aprofundados na secção 7.2.

6.4 Conclusão

Neste capítulo foi avaliada a componente a componente de blocos desenvolvido nesta dissertação. Apesar do número reduzido de programadores em cada teste, é possível perceber a utilidade dos mesmos durante os testes da secção 6.2. Os resultados foram bastante satisfatórios uma vez que os blocos foram integrados na plataforma sem causar problemas, demonstrando uma integração de novos repositórios e de diferentes fontes de dados sem qualquer alteração à plataforma.

Também foram verificados, na secção 6.3, os problemas de segurança que os blocos introduzem na plataforma, uma vez que código malicioso, como demonstrado, pode ser corrido na plataforma através do bloco, uma vez que não existe qualquer controlo sobre o mesmo.

Capítulo 7

Conclusão e Trabalhos Futuros

Neste capítulo apresentam-se as conclusões relativas ao protótipo desenvolvido nesta dissertação e também algumas direções para a continuação do seu desenvolvimento.

7.1 Conclusão

O trabalho realizado nesta dissertação teve como objetivo desenvolver uma plataforma que funcione como intermediário entre aplicações e repositórios de dados, que satisfizesse os objetivos definidos na secção 1.3. Analisando a plataforma tendo em conta estes objetivos podemos afirmar que os mesmos foram cumpridos da seguinte forma:

- **Um mecanismo que permite a integração facilitada de acesso a novos repositórios de informação** - Como demonstrado no teste de usabilidade dos blocos (secção 6.2) o mecanismo dos blocos permite que repositórios sejam adicionados à plataforma e acedidos pelos utilizadores sem ser preciso realizar alterações à plataforma ou à aplicação cliente;
- **Uma interface normalizada de serviços web para facilitar a utilização por aplicações externas ao projeto** - A plataforma apresenta uma API com conjunto de *Web Services* que pode ser integrada por uma grande variedade de aplicações externas. Esta API permite que esta mesma aplicação tenha acesso aos diferentes repositórios de forma uniforme facilitando a sua utilização.
- **A disponibilização dos resultados de pesquisas aos repositórios associados utilizando diversos formatos (XML e JSON)** - A API apresenta dois conjuntos de *Web Services*, REST e SOAP que suportam respetivamente os formatos JSON e XML, permitindo aos programadores escolher aqueles que melhor se adaptam às aplicações a serem desenvolvidas;

- **Utilizar dados georreferenciados para tentar detetar relações entre os diferentes resultados da pesquisa** - O *Data Controller* apresenta mecanismos para detetar relações entre resultados, usando as localizações dos mesmos, mas seria importante realizar um conjunto de testes de chamadas à API por utilizadores, através do uso de uma interface desenvolvida para a mesma, facilitando a realização dos testes.

O desenvolvimento da plataforma iniciou-se com o desenho da arquitetura e dos diferentes componentes que a constituem (secção 4.3). Foi desenvolvido um protótipo com uma API, um *Data Controller* e um *Repository Controller* com ligação ao repositório da Europeana, para testar e escolher as tecnologias (secção 5.1) a utilizar. Posteriormente iniciou-se com o desenvolvimento do modelo interno (secção 5.2), seguido da API (secção 5.3.1) e do *Repository Controller* (secção 5.3.4), onde a Europeana foi removida diretamente do controlador e adicionada a um bloco. Foram desenvolvidos novos blocos e foram feitas pequenas alterações ao modelo interno para estes repositórios, devido à falta de propriedades precisas para guardar os metadados dos mesmos. Por fim foram desenvolvidos o *Data Controller* e o Log.

Tal como descrito na Secção 6.2, para testar a capacidade da plataforma para a integração de novas fontes de dados, foi pedido a programadores que implementassem blocos de acesso a diferentes fontes de dados e analisadas quais as dificuldades que tiveram na implementação.

O protótipo atual apresenta algumas limitações.

- A capacidade de integração de fontes de informação pela plataforma depende da qualidade e amplitude do modelo interno, mas desenvolver o modelo interno não é fácil, sendo preciso ter conhecimento sobre a área e de que dados são importantes nesta mesma área;

- Para integrar uma nova fonte de informação é preciso que o programador conheça a estrutura de dados dessa fonte e do modelo interno, para corretamente fazer a ligação das propriedades;

- Os mecanismos para detetar relações e unir resultados estão dependentes da existência de boas propriedades para fazer comparações.

7.2 Trabalho Futuro

O trabalho desta tese constitui um esforço inicial de implementação que irá ser prolongado no futuro. Como tal existe ainda um longo caminho pela frente para uma plataforma final. Assim, apresentam-se, nesta secção, um conjunto de melhorias e implementações que serão importantes para a continuação do desenvolvimento da plataforma.

Estender a API: atualmente a API apenas apresenta 3 serviços, sendo apenas dois deles para pesquisa em repositórios. Seria importante permitir mais serviços que realizem uma pesquisa mais completa sobre os repositórios, por exemplo pesquisar por termos em propriedades específicas, em vez de só ter uma pesquisa por termo geral. Nem todos os repositórios permitiriam pesquisas complexas. Nos repositórios acessíveis, por exemplo, apenas a Europeana permitiria uma pesquisa mais complexa por termos específicos, mas apesar de, neste caso, só afetar um repositório, pode não ser assim para todos os casos, fazendo com que estejamos a perder capacidades de pesquisa que poderiam ser importantes para o utilizador.

Também seria importante permitir que o utilizador possa alterar quais as propriedades para unir ou detetar relações, permitindo uma maior liberdade às aplicações clientes, para escolher que relações melhor se ajustam as suas necessidades.

Filtros : permitir filtrar os dados diretamente na plataforma também poderia compensar o facto de os repositórios não permitirem pesquisar por valores em propriedades específicas. Assim, essas propriedades seriam verificadas e os resultados seriam filtrados já na plataforma. Não é aconselhável carregar todos os dados do repositório e filtrar na plataforma uma vez que sobrecarregaria a mesma. Tendo isto em conta, o filtro teria de ser usado em combinação com uma pesquisa no repositório, por exemplo pesquisa por termo, para diminuir o número de resultados a serem processados.

Cache : a existência de uma cache permitiria diminuir os pedidos repetidos aos repositórios, melhorando o tempo de resposta para estes casos. Nesta plataforma poderão existir três níveis diferentes de cache: ao nível do bloco, do filtro e ao nível do *Data Controller*. Ao nível do bloco, permitirá guardar pedidos específicos de cada repositório; Ao nível do filtro, guardando os resultados dos repositórios depois de estes passarem pelo filtro, e ao nível do *Data Controller*, guardando os resultados depois dos mesmos passarem pelo filtro e pelo *Data Controller*. Cada uma destas caches apresenta vantagens e desvantagens e tal vai depender de como a plataforma é mais utilizada pelos diferentes utilizadores para saber qual a cache que melhor se encaixa em cada situação. Numa plataforma onde a maioria dos utilizadores apresenta pedidos iguais entre si, uma cache ao nível do Data Controller pouparia bastante tempo em respostas ao utilizador, mas numa plataforma em que os utilizadores realizem a mesma pesquisa com os mesmo termos mas que usem filtros e mecanismos de relação diferentes, a cache ao nível do *Data Controller* não seria tão útil porque os dados guardados na cache apenas poderiam ser reutilizados pela mesma exata chamada. Por outro lado, uma cache ao nível dos blocos poderia guardar os dados da pesquisa por termo de cada repositório, antes de serem processados pela plataforma podendo ser reutilizados para pedidos com diferentes combinações de filtros e mecanismos. Seria assim importante ser possível, através do ficheiro de configurações, definir

que tipo de caches se pretende ativar ou desativar.

Log : É possível melhorar o sistema de logs atual. Usar uma base de dados para guardar os logs e uma interface gráfica para ajudar a visualizar os logs traria grandes vantagens na interpretação dos mesmos, ajudando na interpretação dos erros e podendo facilitar a resolução dos mesmos.

Repository Controller : aqui seria importante melhorar a forma como os blocos são integrados na plataforma. Ao serem adicionados novos blocos à plataforma ou ao serem atualizados blocos já existentes na plataforma, por exemplo para corrigir um bug, o *Repository Controller* detetaria a existência de alterações nos jar dos blocos, e integraria estes blocos na plataforma, sem ser preciso reiniciar a mesma. O objetivo deste mecanismo é diminuir o tempo em que a plataforma se encontra indisponível para os utilizadores cada vez que existe uma alteração aos blocos. Também seria importante colocar a organização dos identificadores dos blocos automáticos. Atualmente os identificadores têm de ser atribuídos manualmente aos blocos. Fazer o controlador realizar esta tarefa automaticamente quando novos blocos são adicionados à plataforma, facilitaria a sua integração e seria mais seguro contra identificadores repetidos, que podem ser inseridos por erro humano atualmente.

Restrições do modelo interno : seria também importante minimizar o máximo possível o número de restrições ao desenvolvimento de novos modelos internos. Atualmente, para utilizar a plataforma para outras áreas de conhecimento que não a de herança cultural e história, é preciso criar uma estrutura de dados para usar como modelo interno, mas essa nova estrutura teria de seguir algumas restrições de estrutura causadas pelos mecanismos atualmente presentes. A diminuição de algumas destas restrições ofereceria uma maior liberdade no desenvolvimento de novas estruturas de dados. Uma das formas para tal seria, por exemplo, remover a restrição causada pelo mecanismo de comparação de coordenadas, que utiliza as coordenadas presentes nos objetos *locations*, obrigando o modelo a apresentar esta propriedade com este objeto para poder ser comparada com as coordenadas. Para tal, os campos deveriam de ser definidos, tal como nos outros mecanismos de relação, diretamente no ficheiro de configuração, dizendo onde se encontram as coordenadas na estrutura ao mecanismo.

Segurança nos Blocos : para melhor controlar o código presente nos blocos, poderia-se restringir as permissões do bloco. Atualmente, a plataforma apenas sabe como chamar o bloco para fazer pesquisas por termo e por coordenadas ao bloco, mas não conhece nem tem controlo sobre o código presente dentro do mesmo. Reduzir as permissões que cada bloco tem para o mínimo necessário para o seu normal funcionamento seria vantajoso como proteção contra código malicioso. Uma outra forma de controlar este problema é através do isolamento do bloco. Poderia-se, para tal, correr os blocos em *Sandboxes*. *Sandboxes* são ambientes para testes de código que

permitem testar código num ambiente isolado contendo quaisquer danos que o mesmo possa fazer dentro da *Sandboxes*, sendo assim possível testá-los em segurança.

Programa para simulação de testes a blocos : para ajudar o desenvolvimento de blocos, deveria de ser desenvolvido um programa que pegaria no jar de um bloco, e realizaria um conjunto de chamadas padrão ao bloco para tentar detetar erros no mesmo, antes de este ser adicionado à plataforma.

Interface de desenvolvimento de blocos : para facilitar o desenvolvimento de blocos poderia ser desenvolvida uma interface gráfica para o desenvolvimento de blocos. A interface permitiria definir os endpoints dos repositórios para realizar pesquisas no repositório e como as propriedades do repositório se interligam com as propriedades do modelo interno. Através desta informação, a interface conseguiria gerar o código do bloco, e consequentemente, o jar para a plataforma.

Esta interface poderia não ser utilizável para gerar os blocos de qualquer tipo de repositórios. Alguns repositórios apresentam APIs bastantes complicadas, como por exemplo a API da Europeana, que pagina os resultados em grupos de 50, e caso existam mais de 50 resultados, várias chamadas têm de ser realizadas para receber todos os resultados. Mas em APIs mais simples, esta interface poderia ser facilmente utilizada, podendo mesmo remover a necessidade de um programador para desenvolver o bloco.

Bibliografia

[1] D. P. Ames, J. S. Horsburgh, Y. Cao, J. Kadlec, T. Whiteaker e D. Valentine. "HydroDesktop: Web services-based software for hydrologic data discovery, download, visualization, and analysis". Em: *Environmental Modelling & Software* 37 (2012), pp. 146–156.

[2] *Archaeology Data Service / Digital Antiquity Guides to Good Practice*. URL: http://guides.archaeologydataservice.ac.uk/g2gp/CreateData_1-2 (acedido em 24/01/2018).

[3] U. Bartlang e J. P. Müller. "A flexible content repository to enable a peer-to-peer-based wiki". Em: *Concurrency and Computation: Practice and Experience* 22.7 (2010), pp. 831–871.

[4] P. A. Bernstein. "Repositories and object oriented databases". Em: *ACM Sigmod Record* 27.1 (1998), pp. 88–96.

[5] M. H. Bhat. "Open access repositories: a review". Em: *Library Philosophy and Practice (e-journal)* (2010), p. 356.

[6] E. Bocher, T. Leduc, G. Moreau e F. G. Cortès. "GDMS: an abstraction layer to enhance Spatial Data Infrastructures usability". Em: *11th AGILE International Conference on Geographic Information Science-AGILE'2008*. 2008.

[7] J. W. Chapman, D. Reynolds e S. A. Shreeves. "Repository metadata: approaches and challenges". Em: *Cataloging & Classification Quarterly* 47.3-4 (2009), pp. 309–325.

[8] R. Crow. "The Case for Institutional Repositories: A SPARC Position Paper". Em: *Scholarly Publishing* (2002).

[9] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi e S. Weerawarana. "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI". Em: *IEEE Internet computing* 6.2 (2002), pp. 86–93.

[10] *Georeferencing*. URL: https://www.ordnancesurvey.co.uk/support/understanding-gis/georeferencing.html (acedido em 24/01/2018).

[11] J. Good. *A gentle introduction to metadata*. 2003.

[12] T. Gruber. *Ontology*. 2008. URL: http://web.dfc.unibo.it/buzzetti/IUcorso2007-08/mdidattici/ontology-definition-2007.htm (acedido em 24/01/2018).

[13] K. Haase. "Context for semantic metadata". Em: *Proceedings of the 12th annual ACM international conference on Multimedia*. 2004, pp. 204–211.

[14] R. Heery e S. Anderson. *Digital repositories review*. 2005.

[15] H. Hockx-Yu. "Digital preservation in the context of institutional repositories". Em: *Program* 40.3 (2006), pp. 232–243.

[16] H. A. Karimi. *Handbook of research on geoinformatics*. IGI Global, 2009.

[17] D. Kichuk. "Loose, falling characters and sentences: The persistence of the OCR problem in digital repository e-books". Em: *portal: Libraries and the Academy* 15.1 (2015), pp. 59–91.

[18] S. Kulasekaran, J. Trelogan, M. Esteva e M. Johnson. "Metadata integration for an archaeology collection architecture". Em: *International Conference on Dublin Core and Metadata Applications*. 2014, pp. 53–63.

[19] P. Longley. *Geographic information systems and science*. John Wiley & Sons, 2005.

[20] A. M. MacEachren e M.-J. Kraak. "Research challenges in geovisualization". Em: *Cartography and geographic information science* 28.1 (2001), pp. 3–12.

[21] H. Pampel, P. Vierkant, F. Scholze, R. Bertelmann, M. Kindling, J. Klump, H.-J. Goebelbecker, J. Gundlach, P. Schirmbacher e U. Dierolf. "Making research data repositories visible: the re3data. org registry". Em: *PloS one* 8.11 (2013), e78080.

[22] *RESTful Services Part II : Constraints and Goals*. URL: https://medium.freecodecamp.org/restful-services-part-ii-constraints-and-goals-530b8f6298b9 (acedido em 12/02/2018).

[23] A. Rodriguez. "Restful web services: The basics". Em: *IBM developerWorks* (2008).

[24] *The structure of a SOAP message*. URL: https://www.ibm.com/support/knowledgecenter/en/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ac55780_.htm (acedido em 12/02/2018).

[25] X. H. Wang, D. Q. Zhang, T. Gu e H. K. Pung. "Ontology based context modeling and reasoning using OWL". Em: *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. Ieee. 2004, pp. 18–22.

[26] *Web Services Architecture*. URL: https://www.w3.org/TR/ws-arch/#introduction (acedido em 12/02/2018).

[27] M. F. Worboys e M. Duckham. *GIS: a computing perspective*. CRC press, 2004.

Apêndice A

Teste de Integração de Blocos

Apresenta-se seguidamente o teste realizado para testar a integração de novos blocos na plataforma.

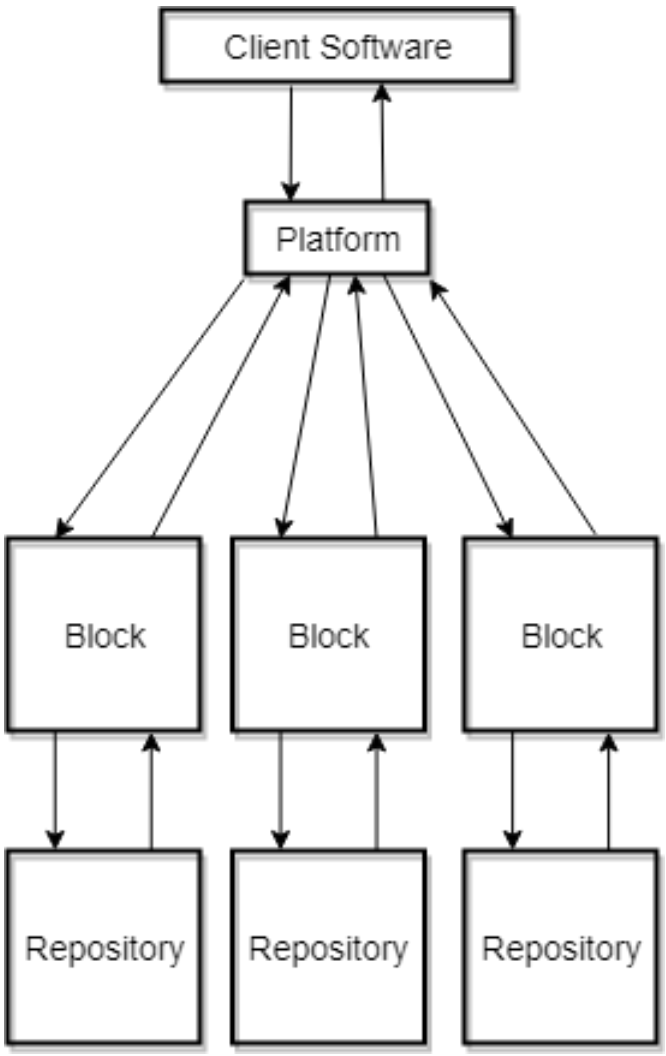

Figura A.1: Blocos como intermediários entre plataforma e repositório.

Teste de Integração de Blocos

Este teste serve para avaliar a capacidade de desenvolvimento dos blocos de forma independente da plataforma.

Este teste consiste no desenvolvimento de um bloco associado a um repositório com o objetivo de adicioná-lo à plataforma atualmente ativa em `10.170.138.201:9200` na rede da FCT. Para tal irá ter de utilizar o projeto com o esqueleto do bloco presente em `https://github.com/rm-fonseca/Tese-Repositorio` como base. Para além disso também existem já outros blocos desenvolvidos que podem ser consultados em caso de dúvida.

O Bloco é um intermediário entra a plataforma e uma fonte de dados como representado na figura A.1. O bloco define uma forma de realizar pesquisas numa dada fonte de dados e converter os dados do formato original para o formato do modelo interno da plataforma, fornecendo uma interface que a plataforma sabe utilizar.

A classe que terá de ser programada será a classe Repository (https://bit.ly/2058wPi) mas classes auxiliares também podem ser desenvolvidas se considerado necessário. Esta classe é utilizada pela plataforma para realizar chamadas aos repositórios. A plataforma procura por uma class chamada Repository dentro de um package repository que implemente a interface RepositoryAbstract (https://bit.ly/2NyURkh). Apesar da interface apresentar duas funções (uma por termo e outra por coordenadas), é apenas preciso desenvolver a função por termo.

Ser-lhe-á atribuída uma fonte de dados e com essa fonte terá de realizar os passos descritos seguidamente:

- Transferir o projecto SkeletonRepository em https://github.com/rm-fonseca/Tese-Repositorio/tree/master/SkeletonRepository.

  Este projeto tem a classe Repository a ser desenvolvida, a interface RepositoryAbstract e as classes auxiliares que definem o modelo interno.

- Realizar pesquisas nessa fonte de dados através do uso de um termo de pesquisa.

  Exemplo com The Fitzwilliam Museum:

  - Api: http://data.fitzmuseum.cam.ac.uk/api/?query=coin
  - Web Interface: http://webapps.fitzmuseum.cam.ac.uk/explorer/index.php?qu=coin

- Receber os dados e convertê-los para o modelo interno, transformando cada resultado do pedido num objeto Result e fazendo a ligação das propriedades para o objeto Result. Na tabela A.1 encontram-se uns exemplos de ligações realizadas entre o modelo do Fitzwilliam e o modelo interno.

  - Modelo Interno descrito em https://github.com/rm-fonseca/Tese-Repositorio/blob/master/Modelo%20Interno.pdf

Modelo de dados Fitzwilliam	Modelo de dados interno
Title	dcTitle
ProductionPlaceName	CreatedIn.Name
Material	PhysicalCharacteristics

Tabela A.1: Exemplos de ligação de propriedades do The Fitzwilliam Museum

- Retornar a lista com todos os objetos result.

- Definir o ficheiro de configuração

  - adicionar chaves extras customizadas se necessário, por exemplo a chave da api (o ficheiro vai-se encontrar na mesma diretoria que o jar).
  - o ficheiro tem de ter o mesmo nome que o jar apenas com extensão diferente.

  Ficheiro de configuração java: https://docs.oracle.com/javase/tutorial/essential/environment/properties.html

  Formato do ficheiro de configuração usado nos blocos: https://github.com/rm-fonseca/Plataforma-Tese/blob/master/Repositorios/example.properties

Depois de desenvolvido o Bloco é preciso exportá-lo para ser adicionado à plataforma. Seguidamente estão descritos os passos a seguir no eclipse para exportar o Jar.

- Ir a File > Export
- Selecionar Java > Jar File > Next (Figura A.2)
- Dar o nome do repositório ao jar para facilitar interpretação > Finish (Figura A.3)
- Por fim é só adicionar o jar e o ficheiro de configuração a plataforma. (Figura A.4)

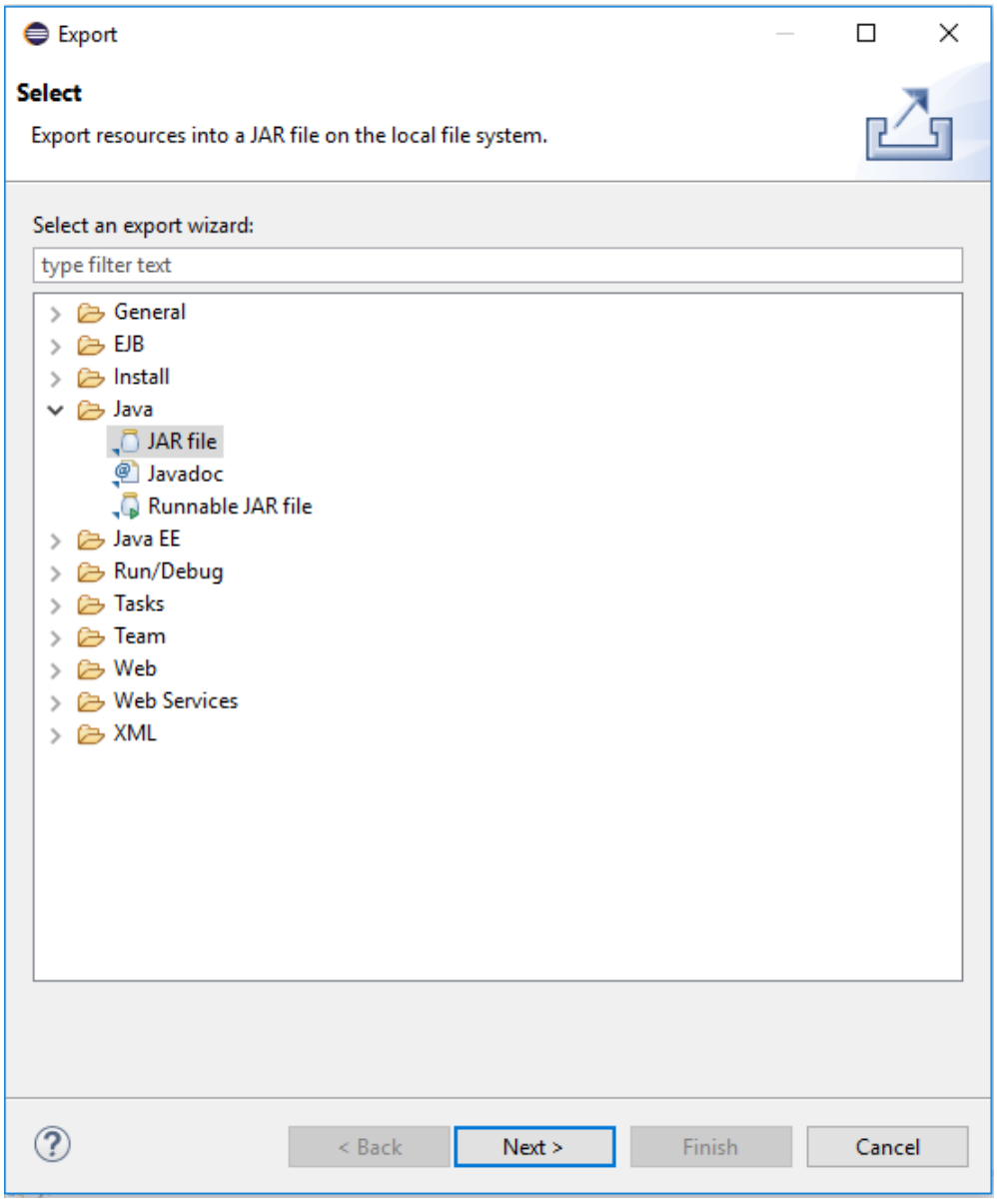

Figura A.2: Exportar como jar

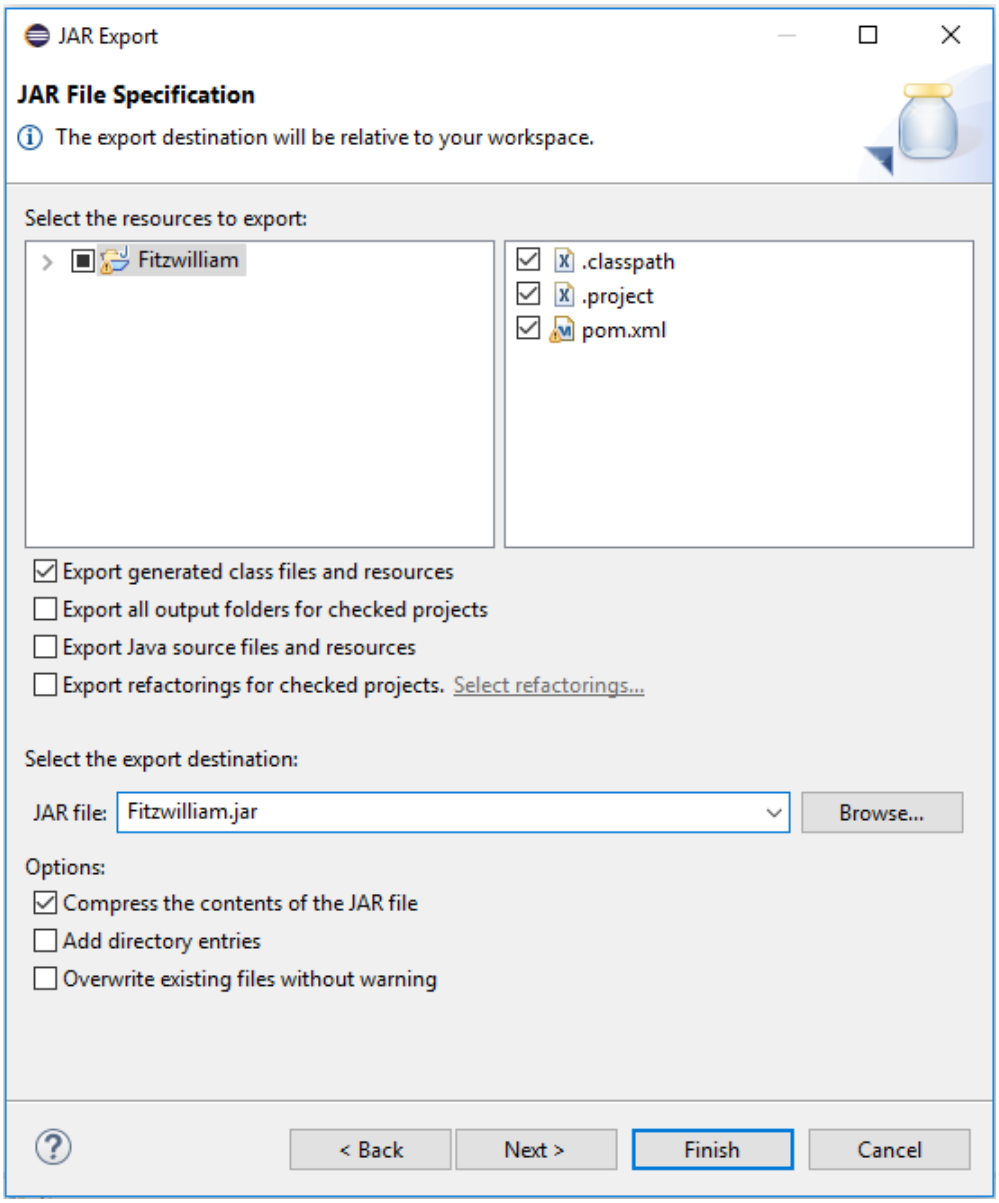

Figura A.3: Exportar como especificações do jar

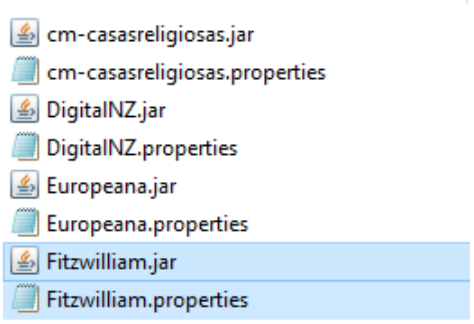

Figura A.4: Pasta repositórios na plataforma

Apêndice B

Guia de Compilação da Plataforma

Lista de passos a seguir para compilar a plataforma para poder ser colocada num servidor.

- Ir buscar o código presente em: https://github.com/rm-fonseca/Plataforma-Tese
- Correr o comando do Maven "install" para fazer download de todas as dependências necessárias.
- Correr o comando do Maven "generate-sources" para gerar as classes do modelo interno através do ficheiro xsd.
- Fazer Export do código como runnable Jar.
- Na pasta base da plataforma (onde esta o jar) criar uma pasta "Repositorios" (https://github.com/rm-fonseca/Plataforma-Tese/tree/master/Repositorios) e adicionar os ficheiros jar dos blocos com os respetivos ficheiros de propriedades.
- Adicionar à pasta base os ficheiros platform.properties (https://github.com/rm-fonseca/Plataforma-Tese/blob/master/platform.properties), application.properties(https://github.com/rm-fonseca/Plataforma-Tese/blob/master/src/main/resources/application.properties) e scheama.xsd (https://github.com/rm-fonseca/Plataforma-Tese/blob/master/src/main/resources/scheama.xsd).
- Correr o jar para ligar a plataforma "java -jar .\framework.jar".

Apêndice C

Questionários Testes de Usabilidade

Resultados dos questionários dos Testes de Usabilidade.

C.1 Museu Victoria

Qual a fonte de dados usada: Museum Victoria
Facilidade na interpretação do modelo interno (1-5): 4
Facilidade na interpretação do modelo de dados da fonte(1-5): 2
Facilidade na ligação de propriedades entre modelos(1-5): 4
Problemas quando a desenvolver o bloco:

Durante o desenvolvimento do bloco sobre a fonte de dados do Museu Victoria foram usadas algumas bibliotecas para facilitar o processamento das respostas aos pedidos HTTP sobre as interrogações colocadas ao endpoint disponibilizado pela API REST do Museu Victoria. Dado que a API do Museu Victoria não é suficientemente documentada ao ponto de que não está disponível a definição dos modelos que a API devolve, tornou-se um pouco complicado a implementação dos modelos, em Java, que vêm das respostas HTTP.

C.2 Estatuária de Lisboa

Qual a fonte de dados usada: Estatuária de Lisboa
Facilidade na interpretação do modelo interno (1-5): 4
Facilidade na interpretação do modelo de dados da fonte(1-5): 5
Facilidade na ligação de propriedades entre modelos(1-5): 5
Problemas quando a desenvolver o bloco:

Para desenvolver o bloco da Estatuária de Lisboa, utilizaram-se bibliotecas para processar o ficheiro CSV, exportado a partir do website. Não existiram problemas ao desenvolver este bloco, uma vez que o CSV estava bem documentado em termos de cabeçalhos, facilitando o desenvolvimento do mesmo.

Ruben Miguel Pereira da Fonseca

Licenciatura em Engenharia Informática

Integração de informação histórica georreferenciada

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Fevereiro, 2018

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Ruben Miguel Pereira da Fonseca

Licenciatura em Engenharia Informática

Integração de informação histórica georreferenciada

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Fevereiro, 2018

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Ruben Fonseca
2018
Integração de informação histórica georreferenciada
2018