INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

HYBRID ONTOLOGY MAPPING INTERFACE

H.O.M.I.

Projeto Final de Curso

Licenciatura em Engenharia Informática e Computadores

Ana Carolina Baptista

41487@alunos.isel.ipl.pt

960314580

Eliane Almeida

41467@alunos.isel.ipl.pt

960271968

Relatório de Progresso

Orientadores:

Cátia Vaz, ISEL, cvaz@cc.isel.ipl.pt

José Simão, ISEL, jsimao@cc.isel.ipl.pt

Alexandre P. Francisco, IST, aplf@ist.utl.pt

Abril de 2018

Índice

Lista de Figuras

Lista de Tabelas

1 Introdução

Atualmente, com o grande crescimento e propagação de dados na internet, surge a necessidade de que a informação seja descrita e transmitida por meio de uma linguagem *standard*, sendo esta de fácil entendimento tanto para computadores quanto para humanos.

Uma das técnicas de descrição de informação que se está a tornar muito popular é baseada em ontologias [1]. Esta permite especificar explicitamente uma conceptualização ou um conjunto de termos de conhecimento para um domínio particular. Apesar da popularidade das ontologias, há em geral dificuldade em transformar o conhecimento pré-definido num caso concreto.

Na área da bioinformática, existem recursos científicos que necessitam de ser partilhados entre a comunidade científica por meio de ontologias. Sendo as ontologias normalmente definidas através de OWL [2] (*Web Ontology Language*), em várias situações poderá não ser uma tarefa simples para os bioinformáticos representar o seu conhecimento do domínio através das ontologias.

Atualmente existem algumas ferramentas de edição de ontologias que permitem ao utilizador inserir um ficheiro referente a uma ontologia e criar novos dados de acordo com este ficheiro, como por exemplo *Protégé* [3]. Existem também algumas bibliotecas para Java que fazem o mapeamento de XML para OWL, que podem ser utilizadas por *developers*. Contudo, não temos conhecimento da existência de uma ferramenta que combine estes dois aspetos: a criação de novas instâncias através de uma ontologia bem como o mapeamento de um caso concreto escrito noutra linguagem numa instância de uma ontologia.

Desta forma, de modo a ajudar os utilizadores – como por exemplo, os biólogos - o objetivo do nosso trabalho é desenvolver uma aplicação que tenha uma interface intuitiva que permita esta transformação de dados semiestruturados em dados anotados com ontologias definidas em OWL. Nesta interface também teria a possibilidade de anotar valores aos vários conceitos da ontologia ou apenas editar os existentes.

1.1 Sinopse

Este relatório está divido em 8 capítulos.

No Capítulo 2 iremos explicar sucintamente o que é uma ontologia e OWL.

No Capítulo 3 iremos clarificar qual o problema existente nesta área bem como que ferramentas já existem.

No Capítulo 4 iremos descrever a biblioteca Chaos Pop que fornece uma API que iremos utilizar no nosso programa.

No Capítulo 5 iremos descrever a nossa arquitetura, bem como cada componente e como estes se relacionam entre si.

No Capítulo 6 iremos falar sobre a planificação apresentada na proposta e como a estamos a seguir.

2 Ontologias e OWL

2.1 Ontologia

Uma ontologia, na área da ciência da computação, é um modelo de dados que representa um grupo de ideias ou conceitos dentro de um determinado domínio e as relações que existem entre eles. Estas são usadas em várias áreas da ciência da computação, tais como inteligência artificial e semântica web, como uma forma de representar conhecimento sobre essa área, ou sobre um subconjunto dessa área. Uma ontologia descreve indivíduos (objetos básicos), classes (conjuntos, coleções ou tipo de objetos), atributos (propriedades, características ou parâmetros) e relacionamentos (entre objetos). [4]

Na Figura 2.1 podemos observar a representação em grafo de um exemplo de uma ontologia, composta por indivíduos, identificados pelos círculos (“Pessoa”, “Parente”, “Pai”); e por relações, identificados pelos retângulos a tracejado (“éUma”, “parenteDe”, “doTipo”). Podemos também ver que nesta ontologia, o indivíduo “Pessoa” tem uma relação com o indivíduo “Parente”, denominada por “éUma”. Na Figura 2.2 podemos observar uma instância da ontologia ilustrada pela Figura 2.1, onde o indivíduo “João Silva” tem uma relação com o indivíduo “José Silva” denominada “temPai”.

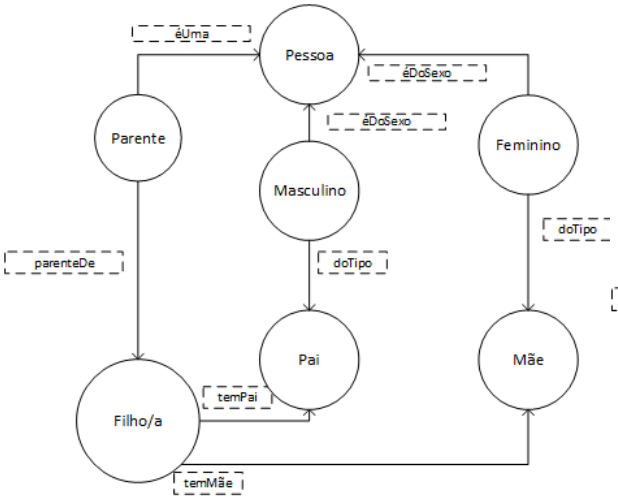

Figura 2.2 Exemplo de uma ontologia

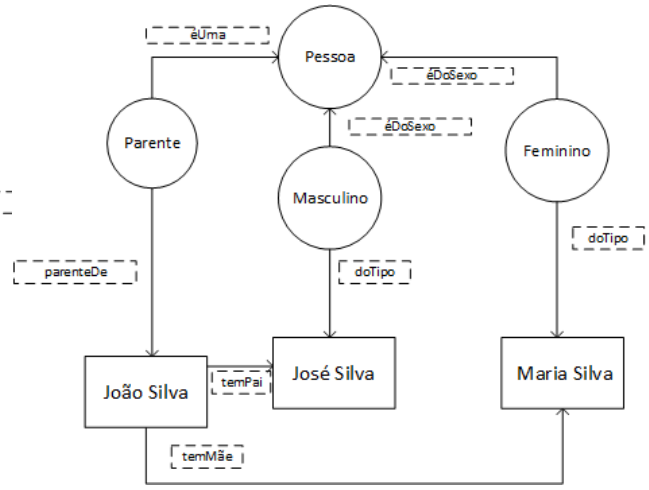

Figura 2.1 Exemplo de uma instância de uma ontologia

2.2 OWL

OWL (*Ontology Web Language*) é uma linguagem utilizada para definir e instanciar ontologias em Web. Em OWL conseguimos definir as 4 propriedades de ontologias descritas acima (indivíduos, classes, atributos e relacionamentos). Esta linguagem foi desenhada para processar informação facilitando assim a sua interpretação por máquinas. Quando comparado com outras linguagens (XML, RDF, etc.), OWL destaca-se por fornecer um vocabulário adicional com uma semântica formal. É a linguagem recomendada da W3C (*World Wide Web Consortium*) para definir ontologias em Web.

Um documento de OWL consiste num cabeçalho da ontologia (ou seja, um *hyperlink* para o documento que contém informação sobre essa ontologia; este cabeçalho é opcional), inúmeras definições de **classe**, **propriedades** e **indivíduos**. Uma **classe** fornece um mecanismo de abstração para o agrupamento de recursos com características similares. Podemos observar um exemplo de uma definição de uma classe na Figura 2.3, onde temos uma classe denominada de "Person", através de <owl:Class (…)>. As **propriedades** em OWL podem ser de 2 tipos: ***Object Properties***, que são referências para indivíduos e ***Datatype Properties*** que referenciam *data values*. Nas Figuras 2.4 e 2.5 podemos observar um exemplo de um *Object Property* e de um *Datatype Property*, respetivamente. Um **indivíduo** de cada classe é denominado de instância dessa classe. [5]

```
<owl:Class rdf:about="http://sysresearch.org/ontologies/family.owl#Person">
    <rdfs:subClassOf>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://sysresearch.org/ontologies/family.owl#DomainEntity"/>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="http://sysresearch.org/ontologies/family.owl#hasFather"/>
                    <owl:someValuesFrom rdf:resource="http://sysresearch.org/ontologies/family.owl#Man"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="http://sysresearch.org/ontologies/family.owl#hasMother"/>
                    <owl:someValuesFrom rdf:resource="http://sysresearch.org/ontologies/family.owl#Woman"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="http://sysresearch.org/ontologies/family.owl#hasSex"/>
                    <owl:someValuesFrom rdf:resource="http://sysresearch.org/ontologies/family.owl#Sex"/>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="http://sysresearch.org/ontologies/family.owl#Sex"/>
</owl:Class>
```

Figura 2.3 Exemplo de uma classe em OWL

```
<!-- http://sysresearch.org/ontologies/family.owl#hasPartner -->

<owl:ObjectProperty rdf:about="http://sysresearch.org/ontologies/family.owl#hasPartner">
    <owl:inverseOf rdf:resource="http://sysresearch.org/ontologies/family.owl#isPartnerIn"/>
</owl:ObjectProperty>

<!-- http://sysresearch.org/ontologies/family.owl#hasChild -->

<owl:ObjectProperty rdf:about="http://sysresearch.org/ontologies/family.owl#hasChild">
    <owl:equivalentProperty rdf:resource="http://sysresearch.org/ontologies/family.owl#isParentOf"/>
    <owl:inverseOf rdf:resource="http://sysresearch.org/ontologies/family.owl#isChildOf"/>
    <rdfs:domain rdf:resource="http://sysresearch.org/ontologies/family.owl#Person"/>
    <rdfs:range rdf:resource="http://sysresearch.org/ontologies/family.owl#Person"/>
</owl:ObjectProperty>

<!-- http://sysresearch.org/ontologies/family.owl#hasParent -->

<owl:ObjectProperty rdf:about="http://sysresearch.org/ontologies/family.owl#hasParent">
    <rdfs:subPropertyOf rdf:resource="http://sysresearch.org/ontologies/family.owl#hasAncestor"/>
    <owl:inverseOf rdf:resource="http://sysresearch.org/ontologies/family.owl#isParentOf"/>
    <rdfs:domain rdf:resource="http://sysresearch.org/ontologies/family.owl#Person"/>
    <rdfs:range rdf:resource="http://sysresearch.org/ontologies/family.owl#Person"/>
</owl:ObjectProperty>
```

Figura 2.4 Exemplos de object properties: hasPartner, hasChild, hasParent

```
<!-- http://sysresearch.org/ontologies/family.owl#alsoKnownAs -->

<owl:DatatypeProperty rdf:about="http://sysresearch.org/ontologies/family.owl#alsoKnownAs">
    <rdfs:subPropertyOf rdf:resource="http://sysresearch.org/ontologies/family.owl#knownAs"/>
    <rdfs:domain rdf:resource="http://sysresearch.org/ontologies/family.owl#Person"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- http://sysresearch.org/ontologies/family.owl#hasFamilyName -->

<owl:DatatypeProperty rdf:about="http://sysresearch.org/ontologies/family.owl#hasFamilyName">
    <rdfs:subPropertyOf rdf:resource="http://sysresearch.org/ontologies/family.owl#hasName"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://sysresearch.org/ontologies/family.owl#Person"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

  <!-- http://sysresearch.org/ontologies/family.owl#hasFirstGivenName -->

  <owl:DatatypeProperty rdf:about="http://sysresearch.org/ontologies/family.owl#hasFirstGivenName">
      <rdfs:subPropertyOf rdf:resource="http://sysresearch.org/ontologies/family.owl#hasName"/>
      <rdfs:domain rdf:resource="http://sysresearch.org/ontologies/family.owl#Person"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
```

Figura 2.5 Exemplos de datatype properties: alsoKnownAs, hasFamilyName, hasFirstGivenName

3 Descrição do Problema

Como referido na introdução, com o avançar da tecnologia e, consequentemente, da propagação de dados leva a que esta informação seja descrita em mais que uma linguagem /estrutura, sendo as mais comuns JSON, XML e OWL. [6] Com isto, nesta área, existe a necessidade de transformar dados de um tipo para outro (por exemplo, de XML para OWL ou de JSON para OWL).

3.1 Ferramentas já existentes

Com o crescimento da popularidade de OWL para a descrição de dados, naturalmente apareceram também alguns *softwares* para a edição e manipulação de OWL. Nesta secção iremos falar sobre alguns destes programas e sobre as diferenças entre eles. Iremos comparar 3 programas que têm um maior número de utilizadores: Apollo [7], Protégé [3] e Swoop [8].

3.1.1 Apollo

Apollo é um modelador *user-friendly*, de utilização gratuita e de instalação local. Nesta aplicação, um utilizador pode modelar ontologias com noções básicas de ontologias (classes, relações, instâncias, etc). Também é possível criar novas instâncias a partir das classes presentes nessas ontologias. Alguns dos pontos fortes deste software é o seu validador de tipos que mantem a consistência de tipos durante o processo, bem como o armazenamento das ontologias (em ficheiros). Contudo, este programa é relativamente antigo e carece de uma visualização de dados em grafo, ao contrário dos seus concorrentes. Apollo carece da possibilidade de dar uma experiência multiutilizador aos seus utilizadores, para trabalhos em colaboração com mais do que um utilizador. [9] [10]

3.1.2 Protégé

Protégé é um editor e modelador de ontologias, de utilização gratuita e tem uma vertente local bem como online (WebProtégé) [11]. Tem uma arquitetura baseada em *plug-ins* o que deu origem ao desenvolvimento de inúmeras ferramentas relacionadas com semântica web. Implementa um conjunto de estruturas modeladoras de conhecimento e ações que suportam a criação, modelação e manipulação de ontologias, complementadas com inúmeras formas de visualização desses dados. A customização proporcionada aos seus utilizadores é uma das características que torna esta aplicação numa das mais populares na área. [9] [10]

3.1.3 Swoop

Swoop é um *browser* e também um editor *open-source*, de utilização gratuita e local. Esta ferramenta contém validadores de OWL e oferece várias formas de visualização gráfica de ficheiros em OWL. É composto também por um ambiente de edição, comparação e fusão entre múltiplas ontologias. As suas capacidades de *hyperlinks* proporciona uma interface de navegação fácil aos seus utilizadores. Um utilizador pode também reutilizar dados ontológicos externos ao colocar os *links* para a entidade externa ou importando a ontologia completa, pois não é possível importar ontologias parciais, mas é possível realizar pesquisas por múltiplas ontologias. [9] [10]

Durante a nossa pesquisa sobre outras ferramentas que já existem nesta área, encontrámos também, para além de outros editores com características diferentes dos apresentados acima, algumas bibliotecas que mapeiam XML para OWL - Ontmalizer[1] e JXML2OWL[2]. Estas bibliotecas foram desenvolvidas em Java e estão disponíveis para *developers* usarem também em Java.

Contudo, apesar da nossa pesquisa, não encontramos nenhuma aplicação que oferecesse todos os pontos mais relevantes das aplicações descritas acima, tais como: versão online como local, mapear XML para OWL[3], etc.

3.2 Como HOMI se compara com estas ferramentas

A nossa ferramenta visa juntar os pontos fortes destas aplicações numa só aplicação. Na nossa ferramenta (H.O.M.I.) iremos ter uma vertente local assim como uma vertente *online*. Ambas terão uma visualização simples, fácil e intuitiva, ou seja, que funcione da forma que o utilizador espera que funcione, bem como a opção de apenas criar uma nova instância a partir de uma dada ontologia ou então, através de ficheiro semiestruturado descrito em XML, realizar o mapeamento de conceitos presentes na ontologia em questão.

Característica - Programa	Criação de novas instâncias	Fácil Utilização	Armazenamento	Interface Intuitiva	Versão Online	Mapeia XML para OWL
Apollo	Sim	Sim	Sim, em ficheiros	Não	Não	Não
Protégé	Sim	Sim	Sim	Sim	Sim	Não
Swoop	Sim	Sim	Sim, em modelos HTML	Sim	Não	Não
Ontomalizer & JXML2OWL	Não	Sim	Não	Não contem interface	Não	Sim
HOMI	Sim	Sim	Sim	Sim	Sim	Sim

Tabela 3.1 Comparação entre várias ferramentas e a nossa aplicação

[1] https://github.com/srdc/ontmalizer
[2] http://jxml2owl.projects.semwebcentral.org/index.html
[3] Mapear XML para OWL refere-se ao processo de realizar uma transformação de uma representação em XML para um documento OWL válido, através da associação de *tags* presentes em XML com conceitos de OXL.

4 Chaos Pop

Chaos Pop é uma biblioteca que fornece uma API que foi desenvolvida para permitir mapear ficheiros semiestruturados de acordo com a descrição de uma ontologia. Para que isto seja possível são indispensáveis dois ficheiros: um OWL referente à uma ontologia (*OntologyFile*) e outro que representa um caso concreto da ontologia em questão (*DataFile*). Neste momento, os ficheiros que são suportados como *DataFile* são de extensões JSON e XML.

4.1 API

De modo a usufruir das funcionalidades existentes no Chaos Pop, seja para submeter ficheiros, obter dados referentes aos ficheiros submetidos ou mapear os vários conceitos é necessário ter conhecimento dos *endpoints* existentes. Todos os *endpoints* apresentados na tabela abaixo iniciam com a URL do servidor do Chaos Pop: **http://chaospop.sysresearch.org/chaos/wsapi**

Method	Path	Requet Body (name: Type)	Decription
POST	/createIndividualMapping	individualMappingTO: Object	Creates a new IndividualMapping in the database.
POST	/replaceIndividualMapping	individualMappingTO: Object	Replaces an existing IndividualMapping in the database.
POST	/removeIndividualMapping	ids: String	Removes a list of IndividualMapping objects from the database. All ids are sepparated by "," (ids).
GET	/getAllIndividualMappings	-	Returns all the IndividualMappings stored in the database.

Tabela 4.1 Endpoints disponíveis no path iniciado por /individualMappingManager

Method	Path	Request Body (name: type)	Description
POST	/addFile	file: InputStream	Uploads a file from the cliente, processes it and stores in the database.
GET	/listDataFiles	-	Returns all the DataFiles stored in the database
POST	/getFile	id: String	Gets a DataFile object when given its id.
POST	/removeFile	ids: String	Removes a list of DataFile objects from the database. All ids are sepparated by "," (ids).

Tabela 4.2 Endpoints disponíveis no path iniciado por /fileManager

Method	Path	Request Body (name: type)	Description
POST	/createMapping	mappingTO: Object	Crates a new Mapping object in the database.
POST	/removeMapping	ids: String	Removes a listo f Mapping objects from the database. All ids are sepparated by "," (ids).

Tabela 4.3 Endpoints disponíveis no path iniciado por /mappingManager

Method	Path	RequestBody (name: type)	Description
POST	/getAllNodesFromDataFile	id: String	Gets all the nodes in a DataFile Node tree.
POST	/getNode	id: String	Gets a Node object.

Tabela 4.4 Endpoints disponíveis no path iniciado por /nodeManager

Method	Path	Request Body (name: type)	Description
GET	/listOntologyFiles	-	Returns all the OntologyFiles stored in the database.
POST	/getOntologyFile	id: String	Gets a OntologyFile object.
POST	/removeOntologyFiles	ontologyIds: String	Removes a list of OntologyFile objects from the database. All ids are sepparated by "," (ontologyIds).
POST	/getOWLClasses	ontologyId: String	Gets all the OWL Classes for a given Ontology.
POST	/getObjectProperties	ontologyId: String	Gets the Ontology's Object Properties.
POST	/getObjectPropertiesForClass	ontologyId: String owlClass: String	Gets the object properties necessary to a given owl class.
POST	/getDataProperties	ontologyId: String	Gets the Ontology's Data Properties.
POST	/getDataPropertiesForClass	ontologyId: String owlClass: String	Gets the data properties necessary to a given owl class.

Tabela 4.5 Endpoints disponíveis no path iniciado por /ontologyManager

4.2 Armazenamento de dados

Quando são submetidos os ficheiros *DataFile* e *OntologyFile*, a informação presente nestes é guardada numa base de dados remota.

No que diz respeito ao *DataFile* os dados são guardados numa estrutura interna em formato de *nodes,* onde cada *node* pode conter *children* e *parent,* formando assim uma árvore.

Por outro lado, quando é submetido um *OntologyFile,* a informação que contém neste é mantida em base de dados da seguinte forma:

- Como mencionado em Ontologias e OWL, uma ontologia pode ter *classes, data properties* e *object properties*, sendo assim é possível obter estes dados referentes a uma ontologia com o identificador da mesma;
- Uma *class* pode ter *data properties* e *object properties,* logo é possível aceder a esta informação com base no identificador da ontologia na qual a classe pertence e no identificador da classe em questão.

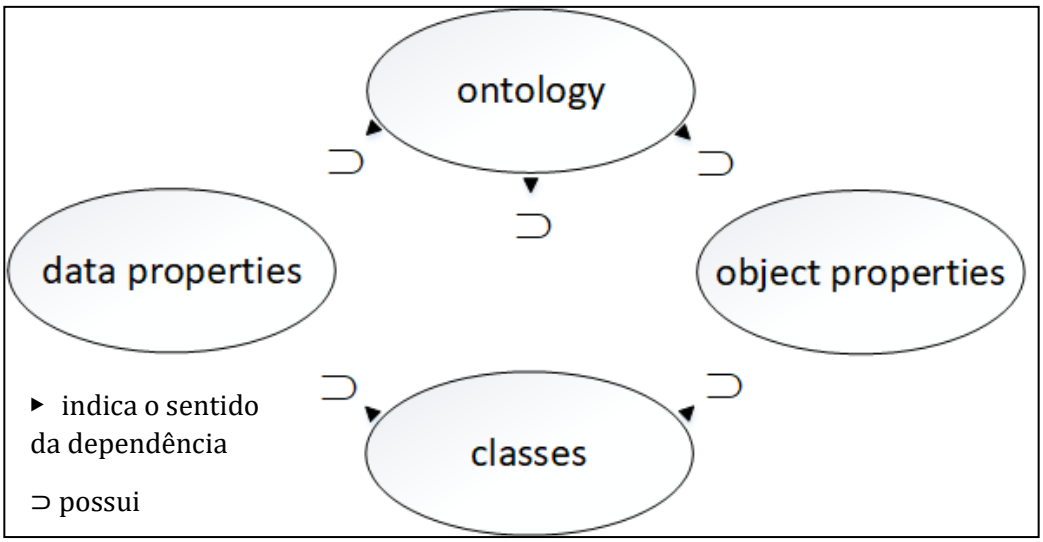

Figura 4.1 Representação das dependências de um ficheiro que descreve uma ontologia

4.3 Exemplo de utilização

Quando se pretende realizar o mapeamento de dados, é necessário primeiro obter as estruturadas de dados que representam os ficheiros que se tenciona mapear, sejam eles um ou mais *DataFile*s e *OntologyFile*s.

Por exemplo, se for submetido o ficheiro XML da Figura 4.2, a árvore abstraída a partir dos *nodes* deste será a apresentada na Figura 4.3.

```
<family>
    <member>
        <name>
            <given>João Antero</given>
            <surname>Cardoso</surname>
            <nickname>Pai</nickname>
        </name>
        <marriage>
            <partner_name>Maria Goretti</partner_name>
        </marriage>
        <descendants>
            <daughter>Ana Rita Cardoso</daughter>
        </descendants>
    </member>
    <member>
        <name>
            <given>Maria Goretti</given>
            <surname>Fernandes</surname>
            <nickname>Mãe</nickname>
        </name>
        <marriage>
            <partner_name>João Antero Cardoso</partner_name>
        </marriage>
        <descendants>
            <daughter>Ana Rita Cardoso</daughter>
        </descendants>
    </member>
    <member>
        <name>
            <given>Ana Rita</given>
            <surname>Cardoso</surname>
            <nickname>Rita</nickname>
        </name>
        <ancestors>
            <father>João Antero Cardoso</father>
            <mother>Maria Goretti Fernandes</mother>
        </ancestors>
    </member>
</family>
```

Figura 4.2 Exemplo de ficheiro de dados semiestruturados

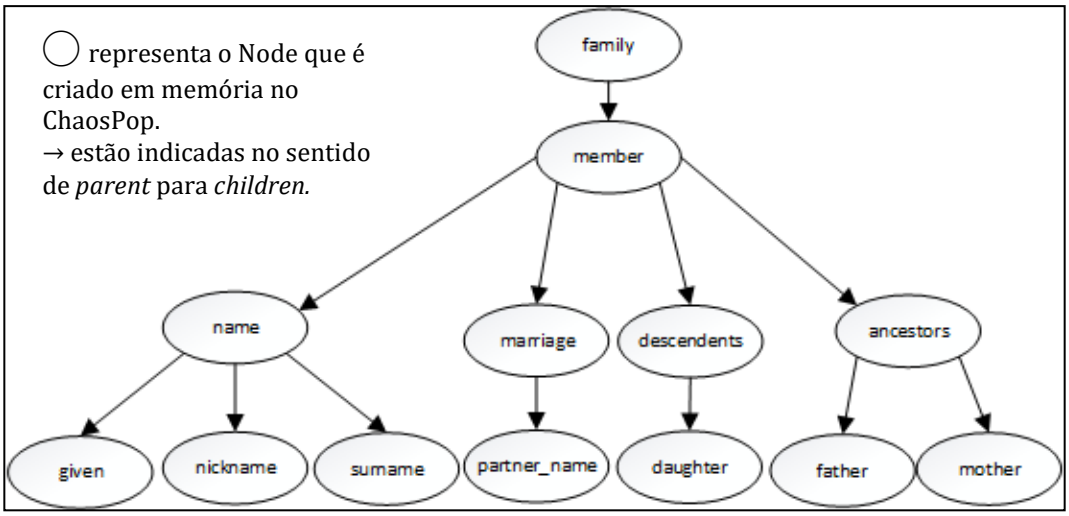

Figura 4.3 Exemplo de árvore gerada pela ferramenta ChaosPop

Num segundo momento, o utilizador deve realizar as correspondências dos vários termos existentes no ficheiro semiestruturado aos conceitos da ontologia (2.2 OWL). Alguns dos mapeamentos a serem realizados são:

Semiestruturado termos	Ontologia conceitos
member	Person
given	hasFirsGivenName
surname	hasFamilyName
nickname	alsoKnowAs

Tabela 4.6 Alguns mapeamentos do ficheiro semiestruturado para ontologia

Após realizar todos os mapeamentos do ficheiro semiestruturado, irá ser gerado o ficheiro do caso concreto da ontologia representado na Figura 4.4.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://sysresearch.org/ontologies/cardosofamily.owl#"
     xml:base="http://sysresearch.org/ontologies/cardosofamily.owl"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:family="http://sysresearch.org/ontologies/family.owl#">


    <owl:Ontology rdf:about="http://sysresearch.org/ontologies/cardosofamily.owl#">
        <owl:imports rdf:resource="http://sysresearch.org/ontologies/family.owl#"/>
    </owl:Ontology>

    <owl:Class rdf:about="http://sysresearch.org/ontologies/family#Person"/>


<owl:NamedIndividual rdf:about=
    "http://sysresearch.org/ontologies/cardosofamily.owl#AnaRitaCardoso">
    <rdf:type rdf:resource="http://sysresearch.org/ontologies/family#Person"/>
    <family:hasParent rdf:resource=
        "http://sysresearch.org/ontologies/cardosofamily.owl#MariaGorettiFernandes"/>
    <family:alsoKnownAs rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">Rita</family:alsoKnownAs>
    <family:hasFamilyName rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">Cardoso</family:hasFamilyName>
    <family:hasFirstGivenName rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">Ana Rita</family:hasFirstGivenName>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about=
    "http://sysresearch.org/ontologies/cardosofamily.owl#JoãoAnteroCardoso">
    <rdf:type rdf:resource=
        "http://sysresearch.org/ontologies/family#Person"/>
    <family:hasChild rdf:resource=
        "http://sysresearch.org/ontologies/cardosofamily.owl#AnaRitaCardoso"/>
    <family:hasPartner rdf:resource=
        "http://sysresearch.org/ontologies/cardosofamily.owl#MariaGorettiFernandes"/>
    <family:alsoKnownAs rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">Pai</family:alsoKnownAs>
    <family:hasFamilyName rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">Cardoso</family:hasFamilyName>
    <family:hasFirstGivenName rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string">João Antero</family:hasFirstGivenName>
</owl:NamedIndividual>

    <owl:NamedIndividual rdf:about=
        "http://sysresearch.org/ontologies/cardosofamily.owl#MariaGorettiFernandes">
        <rdf:type rdf:resource=
            "http://sysresearch.org/ontologies/family#Person"/>
        <family:hasChild rdf:resource=
            "http://sysresearch.org/ontologies/cardosofamily.owl#AnaRitaCardoso"/>
        <family:hasPartner rdf:resource=
            "http://sysresearch.org/ontologies/cardosofamily.owl#JoãoAnteroCardoso"/>
        <family:alsoKnownAs rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string">Mãe</family:alsoKnownAs>
        <family:hasFamilyName rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string">Fernandes</family:hasFamilyName>
        <family:hasFirstGivenName rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string">Maria Goretti</family:hasFirstGivenName>
    </owl:NamedIndividual>

</rdf:RDF>
```

Figura 4.4 Caso concreto da ontologia após mapear o ficheiro semiestruturado

5 Arquitetura

Neste capítulo descreveremos detalhadamente cada módulo da aplicação, assim como a forma como estes interagem entre si e as tecnologias utilizadas na implementação de cada um.

5.1 Descrição

A Figura 5.1 mostra a arquitetura da aplicação, na qual está representada a interação entre as camadas existentes. Esta interação inicia-se quando o utilizador (3) insere um ficheiro com a definição de uma ontologia (1) e, opcionalmente, um segundo ficheiro (2). Estes ficheiros irão ser submetidos à API Chaos Pop (5). De seguida irá ser gerada uma interface gráfica (4) onde o utilizador poderá anotar novos dados aos vários conceitos presentes no ficheiro que descreve uma ontologia (1) ou mapear os conceitos do ficheiro semiestruturado (2) com os termos do ficheiro da ontologia (1). No final deste processo, é gerado um novo ficheiro OWL que contém um caso concreto dos dados descritos no *Ontology File*. Iremos também dar a opção ao utilizador de guardar os ficheiros de *input* e *output* numa base de dados remota.

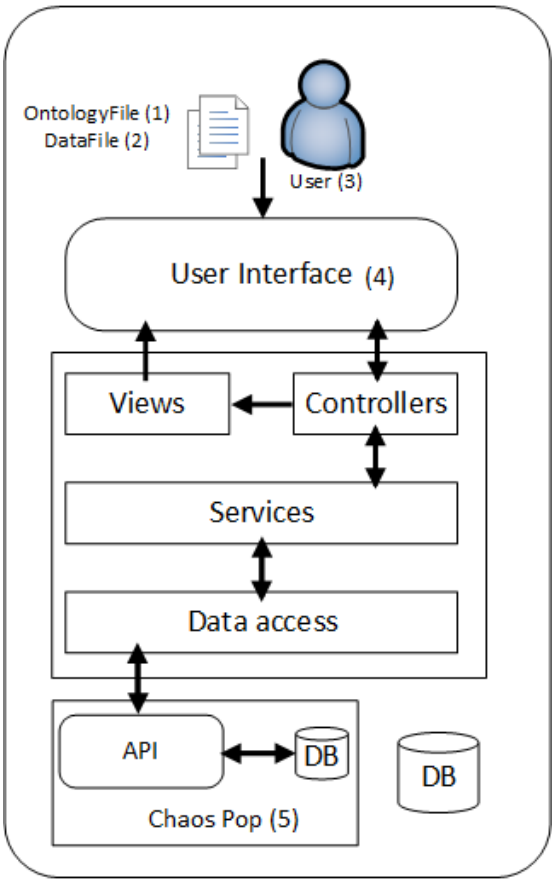

Figura 5.1 Arquitetura da aplicação

5.2 Hybrid Ontology Mapping Interface (H.O.M.I)

A aplicação foi implementada por camadas (Figura 5.1), na qual cada camada apenas comunica com a superior ou inferior. Esta implementação proporciona uma maior facilidade na manipulação das funcionalidades.

As camadas existentes são:

- ***controllers:*** contém todos os *endpoints* possíveis de serem acedidos através da *user interface* e comunica com *views* e *services*;
- ***services:*** camada intermédia entre *controllers* e *data access*, contém toda a lógica necessária para a execução das operações disponíveis;
- ***data access:*** engloba todo o acesso aos dados. Seja este acesso na API Chaos Pop ou na nossa própria base de dados;
- ***views:*** inclui todas as representações visuais utilizadas em *user interface.*

Em cada uma das camadas existem diferentes módulos, nestes estão implementadas funções usufruindo de algumas tecnologias das quais temos conhecimento.

5.2.1 Tecnologias

Node.js

É um *runtime* de JavaScript, que pelo facto de processar o código JavaScript desvinculando-o do browser, possibilita o desenvolvimento de aplicações estáveis e rápidas.

Uma vez que já tivemos experiências em outras unidades curriculares com esta tecnologia e concluímos que fornece uma maneira fácil de construir uma aplicação, escolhemos esta para desenvolver todo o projeto.

D3.js

Consiste numa ferramenta para JavaScript que associa os dados ao *Document Object Model(DOM)* e permite manipular estes gerando gráficos usando diretamente padrões web como o HTML e o CSS.

Decidimos optar por tal pelo facto de suportar comportamentos dinâmicos de interação e animação e grandes conjuntos de dados. Outra característica que nos chamou a atenção foi com facilidade obter gráficos bonitos visualmente. Por isto, utilizamos esta na apresentação dos dados referentes aos ficheiros de entrada, de modo a obter uma representação intuitiva e de fácil entendimento destes por parte do utilizador.

Electron

É um *framework* utilizado para criar aplicações multiplatformas desktop com tecnologias web (HTML, JavaScript e CSS).

Como a nossa aplicação será desenvolvida na tecnologia JavaScript e esta é a suportada pelo Electron, decidimos utilizá-lo para assim não ter a necessidade de aprender a usar uma nova tecnologia no desenvolvimento do sistema desktop.

Express

Equivale a um *framework back-end* em Node.js que cria rotas, *middlewares*, entre outras para facilitar a criação de API's. Este cria e obtém dados a partir do servidor, independentemente da linguagem que irá utilizá-los.

Por já termos experiências com esta tecnologia em outras alturas e a sua utilização ser fácil, decidimos implementas todas as rotas disponíveis na camada *controllers* com base nesta.

5.2.2 Base de dados

O armazenamento dos dados relativos aos ficheiros *OntologyFile* e *DataFile* está a ser feito na base de dados do ChaosPop.

Pelo facto de desejarmos ter a informação referente ao identificador do último ficheiro inserido e, posteriormente, restringir o acesso aos ficheiros a cada utilizador, tencionamos alterar isto de modo a que tenhamos a nossa própria base de dados.

6 Progresso do projeto

A calendarização definida anteriormente na proposta foi criada baseada numa arquitetura por módulos. Após uma análise concluímos que a aplicação seria mais bem-adaptada numa implementação por camadas, o que por consequência causou alterações na arquitetura e calendarização inicial.

As alterações referentes a calendarização não foram apenas devido ao facto de mudarmos a arquitetura, mas também na divisão das semanas para cada tarefa específica. Isto ocorreu porque gastamos mais semanas do que definido inicialmente para ter acesso à API do Chaos Pop, estudá-la e usá-la em alguns exemplos. Neste momento estamos a mapear os conceitos das ontologias e a utilizar a tecnologia D3.js (5.2.1 Tecnologias). Sendo assim, encontramo-nos 2 semanas atrasadas para concluir a implementação da *User Interface* e das camadas.

Na Tabela 6.1 está apresentada a nova calendarização do projeto com as mudanças necessárias.

Data de início	Semana	Descrição
19/02/2018	1-2	- Compreensão da necessidade da ferramenta nos dias atuais - Estudo do Chaos Pop
05/03/2018	3-4	- Estudo da ferramenta Electron - Desenvolvimento da proposta
19/03/2018	5-7	- Entrega da proposta do projeto - Esclarecimentos sobre a API Chaos Pop e utilização do mesmo em alguns exemplos
09/04/2018	8-10	- Início da implementação dos níveis de acesso (*controllers, services, data-access*) - Início do desenvolvimento da *User Interface*
30/04/2018	11	- Apresentação individual e entrega do relatório de progresso
07/05/2018	12	- Continuação do desenvolvimento da *User Interface* e dos níveis de acesso
14/05/2018	13	- Definição da descrição de ferramentas em OWL
21/05/2018	14	- Desenvolvimento da aplicação *desktop* - Criação do cartaz
28/05/2018	15 - 17	- Entrega do cartaz e da versão beta - Otimização dos módulos
18/06/2018	18	- Testes de escalabilidade
25/06/2018	19-21	- Finalização do relatório e entrega da versão final

Tabela 6.1 Calendarização atual do projeto

7 Referências

[1] "Ontology," [Online]. Available: https://www.w3.org/standards/semanticweb/ontology. [Acedido em 19 03 2018].

[2] "OWL," [Online]. Available: https://www.w3.org/OWL/. [Acedido em 09 03 2018].

[3] "Protege," [Online]. Available: https://protege.stanford.edu/. [Acedido em 15 03 2018].

[4] "Ontologias," [Online]. Available: https://pt.wikipedia.org/wiki/Ontologia_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o). [Acedido em 22 04 2018].

[5] "OWL-ref," [Online]. Available: https://www.w3.org/TR/owl-ref/. [Acedido em 24 04 2018].

[6] "Meta Formats," [Online]. Available: https://www.w3.org/standards/webarch/metaformats. [Acedido em 26 04 2018].

[7] "Apollo," [Online]. Available: http://apollo.open.ac.uk/. [Acedido em 22 04 2018].

[8] "Swoop," [Online]. Available: https://github.com/ronwalf/swoop. [Acedido em 23 04 2018].

[9] "Editores 1," [Online]. Available: https://www.w3.org/wiki/Ontology_editors. [Acedido em 21 04 2018].

[10] "Artigo sobre editores de ontologias," [Online]. Available: http://www.ef.uns.ac.rs/mis/archive-pdf/2013%20-%20No2/MIS2013-2-4.pdf. [Acedido em 21 04 2018].

[11] "WebProtege," [Online]. Available: https://protege.stanford.edu/products.php#web-protege. [Acedido em 23 04 2018].

[12] "Editores 2," [Online]. Available: https://www.w3.org/wiki/SemanticWebTools#Semantic_Web_Development_Tools:_Introduction. [Acedido em 21 04 2018].

8 Bibliografia

Jamie Taylor, Colin Evans, Toby Segaran. (2009). Programming the Semantic Web.

Jim R. Wilson. (2013). Node.js the Right Way: Practical, Server-side JavaScript that Scales.