

**HYBRID ONTOLOGY MAPPING INTERFACE**

**Projeto Final de Curso**

Licenciatura em Engenharia Informática e Computadores

Ana Carolina Baptista

[41487@alunos.isel.ipl.pt](mailto:41487@alunos.isel.ipl.pt)

960314580

Eliane Almeida

[41467@alunos.isel.ipl.pt](mailto:41467@alunos.isel.ipl.pt)

960271968

**Relatório de Progresso**

**Orientadores:**

Cátia Vaz, ISEL, [cvaz@cc.isel.ipl.pt](mailto:cvaz@cc.isel.ipl.pt)

José Simão, ISEL, [jsimao@cc.isel.ipl.pt](mailto:jsimao@cc.isel.ipl.pt)

Alexandre P. Francisco, IST, [aplf@ist.utl.pt](mailto:aplf@ist.utl.pt)

Abril de 2018

## Índice

|   |    |
|---|----|
| Lista de Figuras.....                                 | 3  |
| Lista de Tabelas .....                                | 4  |
| 1 Introdução.....                                     | 5  |
| 2 Descrição do problema.....                          | 6  |
| 2.1 Apollo .....                                      | 6  |
| 2.2 Protégé .....                                     | 6  |
| 2.3 Swoop.....  | 6  |
| 3 Chaos Pop .....                                     | 8  |
| 3.2 API .....   | 8  |
| 3.1 Armazenamento de dados .....                      | 10 |
| 3.3 Exemplo de utilização.....                        | 11 |
| 4 Arquitetura .....                                   | 14 |
| 4.1 Descrição.....                                    | 14 |
| 4.2 Hybrid Ontology Mapping Interface (H.O.M.I) ..... | 15 |
| 4.2.1 Tecnologias .....                               | 15 |
| 4.2.2 Base de dados.....                              | 16 |
| 5 Progresso do projeto.....                           | 17 |
| 6 Referencias.....                                    | 18 |
| 7 Bibliografia.....                                   | 18 |

## Lista de Figuras

|  |    |
|--|----|
| Figura 3.1 - Representação das dependências de um ficheiro que descreve uma ontologia..... | 10 |
| Figura 3.2 Exemplo de ficheiro de dados semiestruturados.....                              | 11 |
| Figura 3.3 Exemplo de árvore gerada pela ferramenta ChaosPop .....                         | 12 |
| Figura 3.4 Caso concreto da ontologia após mapear o ficheiro semiestruturado .....         | 13 |
| Figura 4.1 – Arquitetura da aplicação .....  | 14 |

## Lista de Tabelas

|   |    |
|---|----|
| Tabela 2.1 - Comparação entre várias ferramentas e a nossa aplicação .....            | 7  |
| Tabela 3.1 Endpoints disponíveis no path iniciado por /individualMappingManager ..... | 8  |
| Tabela 3.3 Endpoints disponíveis no path iniciado por /mappingManager .....           | 9  |
| Tabela 3.4 Endpoints disponíveis no path iniciado por /nodeManager .....              | 9  |
| Tabela 3.5 Endpoints disponíveis no path iniciado por /ontologyManager .....          | 9  |
| Tabela 5.1 - Calendarização atual do projeto .....                                    | 17 |

# 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 aspectos: a criação de novas instancias através de uma ontologia bem como o mapeamento de um caso concreto escrito noutra linguagem numa instancia de uma ontologia. (permita a transformação de dados semiestruturados. )

Desta forma, de modo a ajudar os utilizadores – como por exemplo, os biólogos - começamos a 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.

## 2 Descrição do problema

Com o crescimento da popularidade de OWL para a descrição de dados, naturalmente apareceram também alguns softwares para a edição e manipulação de OWL.

Neste capítulo iremos falar sobre alguns destes programas e sobre as diferenças entre eles. Por fim iremos explicar como o nosso projeto de insere nesta área bem no que o difere destes programas que já existem.

Iremos comparar 3 programas populares: Apollo, Protege e Swoop.

### 2.1 Apollo

Apollo é um modelador de fácil utilização para o utilizador, de utilização gratuita e local. Nesta aplicação, um utilizador pode modelar ontologias com noções básicas de ontologias (classes, relações, instâncias, etc). Também é possível criar novas instancias a partir das classes presentes nessas ontologias. Alguns dos pontos fortes deste software é o seu corretor 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 interface gráfica intuitiva, ao contrario dos seus concorrentes. Apollo carece também de extração de informação em web e da possibilidade de dar uma experiencia multiutilizador aos seus utilizadores, para trabalhos em colaboração com mais do que um utilizador.

### 2.2 Protégé

Protege é um editor e modelador de ontologias, de utilização gratuita e tem uma vertente local bem como online (WebProtégé). 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.

### 2.3 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 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.

Durante a nossa pesquisa sobre outras ferramentas e bibliotecas que já existem nesta área, encontramos também, para além de outros editores com características diferentes dos apresentados acima, alguns mapeadores de XML para OWL (Ontmalizer <sup>1</sup> e JXML2OWL<sup>2</sup>)

A nossa ferramenta visa juntar os pontos fortes destas aplicações numa só aplicação

Na nossa ferramenta (HOMI) iremos ter uma vertente local assim como uma vertente online. Ambas terão uma visualização simples e fácil para os nossos utilizadores bem como a opção de apenas criar uma nova instância a partir de uma dada ontologia ou então através de um exemplo concreto 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                         | ? probably       | Sim, em modelos HTML | Sim                  | Não           | Não                 |
| Ontomalizer & JXML2OWL    | Sim                         | Kinda            | Não                  | Não contem interface | Não           | Sim                 |
| HOMI                      | Sim                         | Sim              | Sim                  | Sim                  | Sim           | Sim                 |

*Tabela 2.1 - Comparação entre várias ferramentas e a nossa aplicação*

<sup>1</sup> <https://github.com/srdc/ontmalizer>

<sup>2</sup> <http://jxml2owl.projects.semwebcentral.org/index.html>

### 3 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.

#### 3.2 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<br>(name: Type)    | Decription   |
|--------|---------------------------|--------------------------------|--|
| POST   | /createIndividualMapping  | individualMappingTO:<br>Object | Creates a new IndividualMapping in the database.   |
| POST   | /replaceIndividualMapping | individualMappingTO:<br>Object | Replaces an existing IndividualMapping in the database.  |
| POST   | /removeIndividualMapping  | ids: String                    | Removes a list of IndividualMapping objects from the database. All ids are separated by “,” (ids). |
| GET    | /getAllIndividualMappings | -                              | Returns all the IndividualMappings stored in the database.   |

Tabela 3.1 Endpoints disponíveis no path iniciado por */individualMappingManager*

| Method | Path           | Request Body<br>(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 separated by “,” (ids). |

Tabela 3.2 Endpoints disponíveis no path iniciado por */fileManager*



| Method | Path           | Request Body<br>(name: type) | Description  |
|--------|----------------|------------------------------|--|
| POST   | /createMapping | mappingTO: Object            | Crates a new Mapping object in the database.   |
| POST   | /removeMapping | ids: String                  | Removes a list of Mapping objects from the database. All ids are separated by “,” (ids). |

*Tabela 3.2 Endpoints disponíveis no path iniciado por /mappingManager*

| Method | Path                     | RequestBody<br>(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 3.3 Endpoints disponíveis no path iniciado por /nodeManager*

| Method | Path                         | Request Body<br>(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 separated 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<br>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<br>owlClass: String | Gets the data properties necessary to a given owl class.  |

*Tabela 3.4 Endpoints disponíveis no path iniciado por /ontologyManager*

### 3.1 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 (Capítulo aqui), uma *ontology* 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.

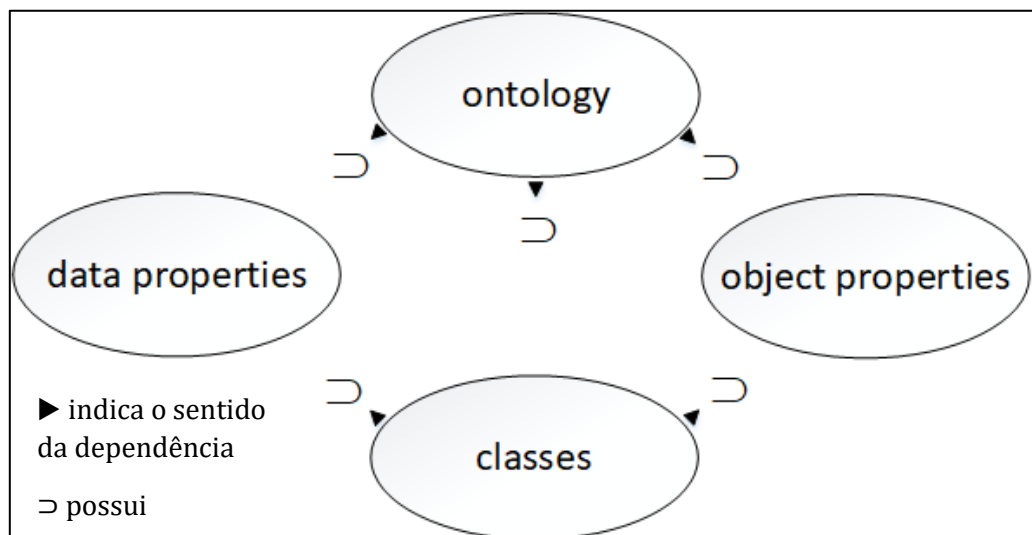


Figura 3.1 - Representação das dependências de um ficheiro que descreve uma ontologia

### 3.3 Exemplo de utilização

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

Por exemplo, se for submetido o ficheiro XML da Figura 3.2, a árvore abstraída a partir dos *nodes* deste será a apresentada na Figura 3.3.

```
<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 3.2 Exemplo de ficheiro de dados semiestruturados

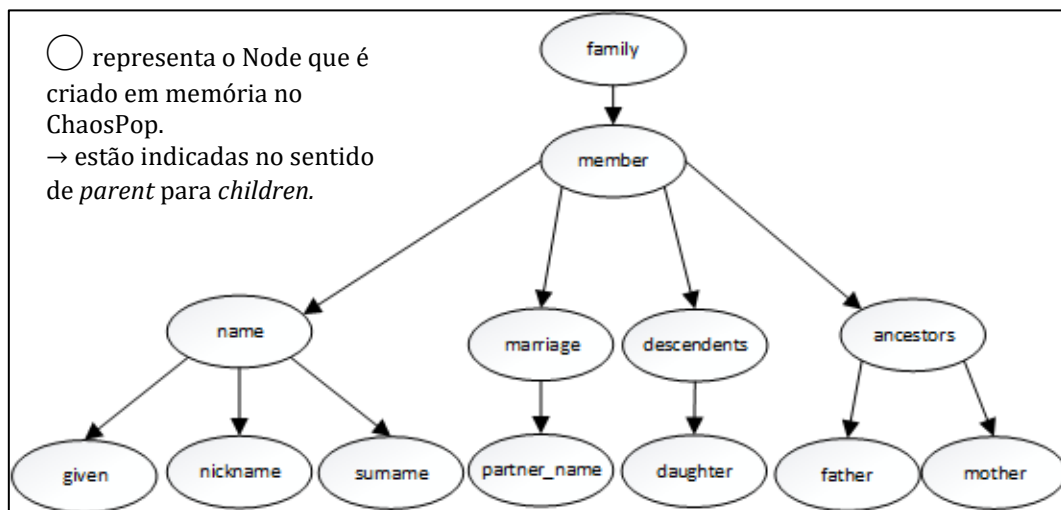


Figura 3.3 Exemplo de árvore gerada pela ferramenta ChaosPop

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

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

Tabela 3.6 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 3.4.

```

<?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 3.4 Caso concreto da ontologia após mapear o ficheiro semiestruturado

## 4 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.

### 4.1 Descrição

A Figura 4.1 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 a API Chaos Pop (5). De seguida irá ser gerada uma interface gráfica (4) onde o usuário 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 usuário de guardar os ficheiros de *input* e *output* numa base de dados remota.

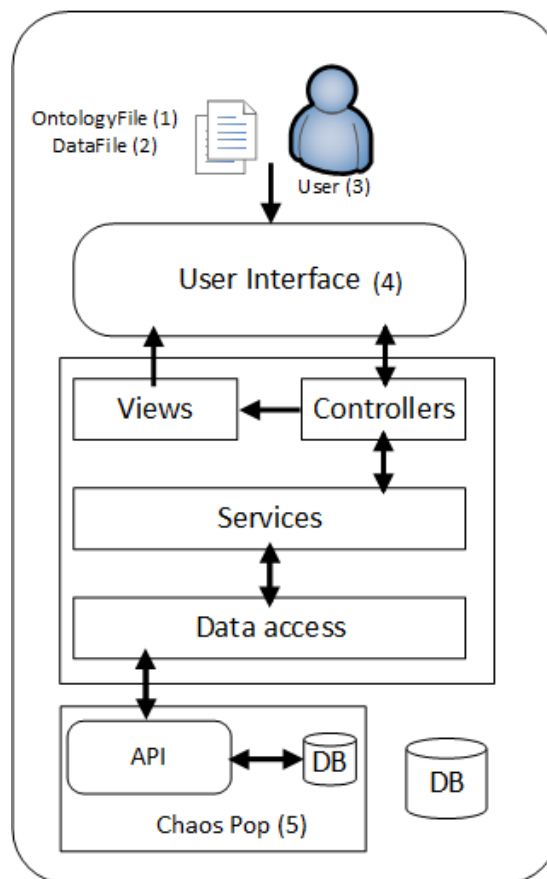


Figura 4.1 – Arquitetura da aplicação

## 4.2 Hybrid Ontology Mapping Interface (H.O.M.I)

A aplicação foi implementada por camadas (Figura 4.1), 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.

### 4.2.1 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 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.

### 4.2.2 Base de dados

O armazenamento dos dados relativos aos ficheiros *OntologyFile* e *DataFile* está sendo realizado 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.



## 5 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 (4.2.1

Tecnologias). Sendo assim, nos encontramos 2 semanas atrasadas para concluir a implementação da *User Interface* e das camadas.

Na Tabela 5.1 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<br>- Estudo do Chaos Pop  |
| 05/03/2018     | 3-4     | - Estudo da ferramenta Electron<br>- Desenvolvimento da proposta   |
| 19/03/2018     | 5-7     | - Entrega da proposta do projeto<br>- 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 ( <i>controllers, services, data-access</i> )<br>- Início do desenvolvimento da <i>User Interface</i> |
| 30/04/2018     | 11      | - Apresentação individual e entrega do relatório de progresso  |
| 07/05/2018     | 12      | - Continuação do desenvolvimento da <i>User Interface</i> e dos níveis de acesso   |
| 14/05/2018     | 13      | - Definição da descrição de ferramentas em OWL   |
| 21/05/2018     | 14      | - Desenvolvimento da aplicação <i>desktop</i><br>- Criação do cartaz   |
| 28/05/2018     | 15 - 17 | - Entrega do cartaz e da versão beta<br>- Otimização dos módulos   |
| 18/06/2018     | 18      | - Testes de escalabilidade   |
| 25/06/2018     | 19-21   | - Finalização do relatório e entrega da versão final   |

Tabela 5.1 - Calendarização atual do projeto

## **6 Referencias**

[https://www.w3.org/wiki/Ontology\\_editors](https://www.w3.org/wiki/Ontology_editors) [21/04/18]

<http://www.ef.uns.ac.rs/mis/archive-pdf/2013%20-%20No2/MIS2013-2-4.pdf>  
[21/07/18]

<https://www.w3.org/wiki/SemanticWebTools> [22/04/18]

## **7 Bibliografia**