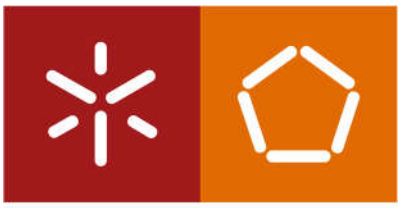


Universidade do Minho
Escola de Engenharia

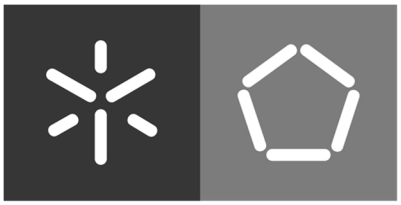
João Manuel Pós de Mina Grenhas

Ancestors Note Book: uma aplicação para memorabilia e genealogia, baseada em ontologias

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação do Professor
José João Almeida, UMinho, D. I.
Coorientador: Rui Castro Mendes, UMinho, D. I.

Abril de 2022



Universidade do Minho
Escola de Engenharia

João Manuel Pós de Mina Grenhas

Ancestors Note Book: uma aplicação para memorabilia e genealogia, baseada em ontologias

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação do Professor
José João Almeida, UMinho, D. I.
Coorientador: Rui Castro Mendes, UMinho, D. I.

Abril de 2022

Agradecimentos

Agradeço à minha esposa todo o apoio e motivação, tornando exequível esta longa jornada.

Agradeço à Módula C toda a tolerância horária, concedendo o tempo necessário a este projeto (e a outros do mesmo mestrado).

Agradeço aos orientadores toda a disponibilidade e aconselhamento, todas as ideias, toda a tolerância da agenda, e toda a inteligente abertura em respeitar caminhos de desenvolvimento por vezes inesperados.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

O tema desta dissertação de mestrado desenrolou-se no âmbito das humanidades digitais, pretendendo desenhar e construir um *toolkit* de apoio ao registo e processamento de histórias de família, micro-história, documentos, fotografias, elementos genealógicos e histórias de instituição, com especial interesse na inferência de indivíduos. Foi desenvolvido um protótipo de aplicação web de uso pessoal, sobre Flask/Python, configurável (e reutilizável, salvo ajustes), como prova de conceito a suportar uma ontologia OWL2 sobre o domínio do problema.

ANB = Ancestors Note Book =
Ontologia AncestorsNB, para
Memorabilia e Genealogia, Relações humanas e Ambiente
+ Filosofia Wiki
+ Inferência Interativa e Dinâmica
+ SPARQL
+ Geração de formulários

O domínio do problema induziu naturalmente uma arquitetura de ontologia, à qual foi dada persistência num ficheiro de texto em sintaxe Turtle, carregada pela aplicação num grafo RDFLib. A ontologia é expansível, mas já inclui à partida relacionamentos de genealogia de pessoas, com diversos graus de parentesco, e ainda relacionamento no âmbito social, institucional e geográfico, para pessoas, organizações e lugares. Ainda no âmbito da memorabilia, prevê-se o registo e referência de eventos, fotografias, documentos, multimédia, histórias de família, artigos sobre qualquer assunto.

Na aplicação, acolhe-se uma filosofia Wiki, no sentido em que proporciona uma apresentação gráfica (suportada em mardkdown e html), navegação por hiperligações (internas ou externas), e edição. Em virtualmente qualquer classe ontológica, a apresentação *prettyprint* de indivíduos e os formulários de ingestão permitem *templating* e alguma configuração. A apresentação *prettyprint* prevê documentos de variada morfologia, incluindo texto, PDF, markdown e multimédia.

Permite-se ao utilizador a ingestão de informação em lote (na sintaxe Turtle, por edição direta ou via ficheiro), e CRUD interativo (via formulários ou SPARQL), num ambiente operacional rico, com pesquisas de utilizador combinadas com navegação, além de facilitar a inferência sobre indivíduos, sejam pessoas ou não.

A inferência é interativa (i.e., a pedido interativo do utilizador) e/ou dinâmica (por etiquetagem no conteúdo de um campo configurado como elegível para este efeito), jogando com classes, ID de indivíduo, nomes e datas. A inferência pode ter posicionamento genealógico: basta que seja encontrado um grau de parentesco. Tanto a inferência como os motores de pesquisa interativa se baseiam num tratamento rico de nomes e moradas em Português, em que a normalização prevê grafias antigas, além de acentos e partículas de ligação (de, da, do, etc.).

Palavras-Chave: Humanidades digitais, ontologias OWL2, genealogia, inferência, prototipagem de aplicações web para ontologias, geração de formulários html, RDFLib, processamento de linguagem natural.

Abstract@en

The theme of this master's dissertation is born in the field of digital humanities, intending to design and build a toolkit to support the recording and processing of family histories, micro-history, documents, photographs, genealogical elements and institution histories, with special focus on the inference of individuals. It was developed a web application prototype of personal use, over Flask/Python, configurable (and reusable, except on adjustments), as a proof of concept to support an OWL2 ontology over the problem domain.

ANB = Ancestors Note Book =
Ontology for Memorabilia and Genealogy, Human Relations and Environment
+ Wiki Philosophy
+ Interactive and Dynamic Inference
+ SPARQL
+ Form generation

The problem domain naturally induced an ontology architecture, which was given persistence in a Turtle syntax text file, loaded by the application in an RDFLib graph. The ontology is expandable, but it already includes genealogical relationships of people, with different degrees of kinship, and also relationships in the social, institutional and geographic scope, for people, organizations and places. Also within the scope of memorabilia, it is foreseen the registration and reference of events, photographs, documents, multimedia, family stories, articles on any subject.

The application embraces a Wiki philosophy, in the sense that it provides a graphical presentation (supported in mardkdown and html), navigation via hyperlinks (internal or external), and editing. In virtually any ontological class, the prettyprint presentation of individuals and the ingest forms allow for templating and some configuration. The prettyprint presentation provides documents of varying morphology, including text, PDF, markdown and multimedia.

It allows the user to ingest information in batch (in Turtle syntax, by direct editing or via file), and interactive CRUD (via forms or SPARQL), in a rich operational environment, with user searches combined with navigation, in addition to facilitating the inference about individuals, whether persons or not.

The inference is interactive (i.e., at the user's interactive request) and/or dynamic (by tagging the content of a field configured as eligible for this purpose), playing with classes, individual ID, names and dates. The inference can have a genealogical position: it is enough to find a degree of kinship. Both inference and interactive search engines are based on a rich treatment of names and addresses in Portuguese, in which the normalization foresees old spellings, in addition to accents and linking particles (de, da, do, etc.).

Keywords: Digital humanities, OWL2 ontologies, genealogy, inference, web application prototyping for ontologies, html form generation, RDFLib, natural language processing.

Índice

Índice de Figuras

1. Introdução

1.1. Contextualização e Motivação

No âmbito da memorabilia, há histórias – orais ou escritas –, documentos, fotografias – possivelmente datadas e anotadas –, que levam a descobertas potencialmente muito ricas, ou simplesmente interessantes/curiosas/cómicas, não só para as famílias relacionadas, como para a própria comunidade, e poderão até conduzir a resultados com interesse histórico ou mesmo literário.

No entanto, a desorganização, antiguidade e ambiguidade dessas informações pode exigir um esforço de investigação e dispêndio de tempo nem sempre disponíveis. Basta pensar que, dentro da família (e na própria comunidade), existe grande repetição de nomes próprios, e mesmo do primeiro e último nome, pelo que, ao serem aflorados ao acaso, não contribuem para a identificação do interveniente, especialmente se for desconhecido do utilizador.

Neste jogo de identificação e relacionamento, ao entrarem datas e lugares, pode ser viável uma identificação, se soubermos mais sobre as pessoas da família e comunidade envolvente, nomeadamente dados pessoais e relacionamentos mútuos, ou mesmo relacionamentos ilustres (pessoas por alguma razão famosas, das quais sempre se recolhe e regista dados diversos). Por exemplo, se dada pessoa participou na Primeira Grande Guerra, mesmo que seja referido simplesmente pelo "José", mesmo que tenha parentes com o mesmo nome, poderá reduzir-se o âmbito de procura dentro do domínio definido.

No âmbito das genealogias, é também viável inferir graus de parentesco através do simples registo dos progenitores, além do cônjuge.

Estes relacionamentos, e outros, poderão enriquecer uma rede que poderá ser muito útil neste contexto de esforço de identificação de pessoas ou outra descoberta de informação, como ao referir "o Joãozinho, filho do tio José", ou "o famoso professor de música da Maria", ou "a casa onde nasceu F".

Todos estes tópicos são elegíveis para um enquadramento por humanidades digitais.

1.2. Objetivos

Em termos de humanidades digitais, podemos pensar, à partida, e sempre num contexto digital, em dicionários enciclopédicos, wiki documentais baseados em hipertexto. Entende-se haver potencial em tentar aliar esses tópicos com a facilidade de carregar informação, e a possibilidade de ter relacionamentos sociais (genealógicos ou não) para facilitar a identificação de indivíduos e contextualizar.

Resumidamente, pretende-se chegar a um sistema que ofereça:

- um tratamento rico de nomes em Português, com um *toolkit* de funcionalidade variada sobre o domínio das humanidades digitais;
- uma ontologia rica para este domínio: ontologia de base (classes e propriedades), que cada instalação poderá enriquecer, em estrutura e povoamento (os indivíduos);
- uma DSL para carregar informação na ontologia, mas que também lhe possa adicionar estrutura e indivíduos, como referido no ponto anterior;
- um motor de inferência (*constraint solver*) para restrições no domínio geográfico, genealógico e temporal.

Esmiuçando, este projeto tem por objetivo o desenho e construção de um *toolkit* de apoio ao registo e processamento de histórias de família, micro-história, elementos genealógicos e histórias de instituição:

- Pretende-se um sistema que, dado um texto ou artigo, ajude a **identificar e associar/inserir pessoas ou entidades** concretas, para o que será necessário:
    - **Desambiguar nomes pela ortografia**: os nomes a analisar poderão estar abreviados (exemplos: JJ, J.J.), ou reduzidos a primeiros nomes (José, José João), ou nomes por que as pessoas não são usualmente tratadas (José Almeida), ou com grafia diferente, seja por erro (Jose Joao, Joes) ou por diferenças ortográficas históricas (Victor x Vítor, Thomaz x Tomás).
    - **Desambiguar pessoas/entidades pelo contexto** de lugares, datas (expressas ou implícitas) e eventos, que possam ser comparados com os conhecidos (nascimento e morte, idade maior, meia idade, idade avançada).
    - O sistema poderá aceitar ***constraints*** para ajudar a refinar soluções. Pretende-se um ***constraint solver*** para restrições no domínio **geográfico**, **genealógico**, **temporal** – no âmbito do período de vida humano (como data de óbito >= data de nascimento, idade para serviço militar em comparação com eventos como batalhas, etc.).
    - Em caso de elegibilidade de várias instâncias (pessoas/entidades/eventos/...) possíveis, o **utilizador** poderá ajudar a escolher, perante uma lista (desejavelmente ordenada por probabilidade), com eventual **identificação de relacionamentos** entre essas pessoas ou entidades, recorrendo à análise de datas, eventos e lugares.
- O utilizador deve poder efetuar compilações, entendidas como **compilações** de documentos e eventos da ontologia, entre outros, e poder inserir anotações, fotos/imagens, ou mesmo uma árvore genealógica gerada pelo sistema.
- O sistema deve facultar a **navegação** pela informação.
- O projeto parece induzir naturalmente, em termos abstratos, uma **ontologia** para:
    - **pessoas** ou organizações, **e seus relacionamentos**:

        - pessoais (exemplo: "Fulano de Tal" conhece "Sicrano, Famoso Músico"),
        - familiares (graus de parentesco)
        - e outros.
        - Pessoas – guardar: nome oficial e variantes, alcunhas, títulos, período de vida, pai, mãe, cônjuge, local onde nasceu (freguesia, concelho, etc.), locais onde residiu (com datas).
    - *wiki* de documentos do tipo genérico **artigo**, podendo incluir **fotografias/imagens** (outra média?) datadas e anotadas, **histórias de vida/família**, **biografias**, **artigos jornalísticos**, **documentos históricos e outros** (história de instituições, de lugares, cronologias, etc.);
    - **eventos** familiares, curriculares, históricos e outros.
- Será útil que o sistema reconheça uma ***Domain Specific Language*** **(DSL) para carregar informação a guardar ou analisar,** a partir de um ficheiro de texto. Esta DSL deve prever o enriquecimento da ontologia, como por exemplo em termos de classes e relações.

1.3. Convenções e Definições

1.3.1 Convenções

Convenciona-se para uso do projeto os graus de parentesco em Português.

1.3.2 Glossário

Dado o potencial volume de termos a definir, optou-se por dividir o glossário em dois, passíveis de consulta no fim deste documento:

- Glossário Genealógico: para termos no contexto da genealogia.
- Glossário Geral: para outros termos em contexto.

1.3.3 Siglas, acrónimos e abreviaturas

ANB	*Ancestors Note Book – tema desta dissertação*
API	*Application Programming Interface*
BD	Base de Dados, o mesmo que DB (*Database*)
Cf.	Confrontar, ver também
CRUD	*Create, Read, Update and Delete* – cf. REST
DSL	*Domain Specific Language*
Endereço IP	Endereço de Protocolo da Internet, *Internet Protocol address (IP address)*
Ex./Exs.	Exemplo, exemplos

HTTP	*Hypertext Transfer Protocol,* Protocolo de Transferência de Hipertexto
ID	Identificador. No contexto atual, tipicamente um ID ontológico, RDF/Turtle.
IRI	*Internationalized Resource Identifier*
MVC	*Model View Controller*, um padrão de arquitetura de software
N/A	Não aplicável
OOP	*Object Oriented Programming*, o mesmo que POO
PF	Programação Funcional
PLN, *NPL*	Processamento de Linguagem Natural, *Natural Language Processing*
POO	Programação orientada a objetos, o mesmo que OOP
RDF	*Resource Description Framework*
REST	*Representational State Transfer*
RGPD	Regulamento Geral de Proteção de Dados – cf. *General Data Protection Regulation – GDPR*.
SGBD	Sistema Gestor de Base de Dados
SPARQL	**S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
Turtle	*RDF 1.1 Turtle, i.e. Terse RDF Triple Language*
UI	*User Interface* – Interface com o utilizador
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier* – recentemente, este termo vem sendo preterido em favor do IRI.

1.4. Factos e Assunções Relevantes

1.4.1 Factos

Tradicionalmente, toda a pessoa humana tem sexo masculino ou feminino, e um pai e uma mãe, biológicos. Algum ou ambos os progenitores podem ser desconhecidos, ou de género indeterminado. A reprodução medicamente assistida, conjugada ou não com mudanças de sexo, podem introduzir complexidade na avaliação de géneros dos progenitores.

O grau de parentesco por consanguinidade, de uma pessoa em relação a outra, nunca se altera ao longo do tempo. Pode alterar-se o cônjuge, mas mesmo alguns relacionamentos por afinidade mantêm-se, nem que seja de um ponto de vista social – suponhamos o caso de uma união/casamento com filhos, que têm os seus avós paternos e maternos: a separação/divórcio não quebra a ligação entre netos e avós, logo por transitividade poderemos afirmar que os relacionamentos por afinidade se mantêm, nem que seja psicologicamente… Por exemplo, novamente o divórcio: em relação aos pais do ex-marido, a ex-esposa passa a ex-nora, e não a "completa desconhecida".

1.4.2 Assunções Relevantes

(Nada a referir até ao momento.)

1.5. Opções e Restrições

Em termos de sintaxe da ontologia, estabeleceu-se como regra geral uma preferência de trabalhar com a sintaxe **Turtle**, por ser habitual no SPARQL e mais legível do que outros formatos, nomeadamente RDF/XML, embora, sendo um subconjunto simplificado de Notation3, esta também se considere interessante (cf. https://en.wikipedia.org/wiki/Notation3). Esta opção assume-se ao mostrar a informação numa perspetiva mais formal, mas, em termos da visualização, poderá ainda amenizar-se termos e IRI, procurando a legibilidade centrada no utilizador.

1.6. Conclusão

O projeto nasce duma necessidade prática de registar, sobre a nossa família ou comunidade, aquilo que provavelmente mais ninguém registará, ou lembrará, e nessa situação será irremediavelmente perdido, com o passar do tempo!

Para responder a esta necessidade, há o desejo duma ferramenta poderosa e versátil, além de agradável.

Dadas as nossas pretensões, defendemos caminhar para a seguinte fórmula exploratória:

> Ancestors Note Book =
>
> Ontologia para Memorabilia e Genealogia, Relações humanas e Ambiente
>
> + Filosofia *wiki*
>
> + Inferência

Lema: Preservar as preciosas memórias da família e comunidade, descobrir todo o tipo de relações.

2. Estado da Arte

Esta secção pretende abordar tópicos do estado da arte do domínio em questão: ontologias, genealogias e humanidades digitais.

2.1. Ontologias

2.1.1 Definição de Ontologia

Interessa-nos, no âmbito desta dissertação, endereçar o significado de ontologia no âmbito das ciências da computação. (Existe outro significado no domínio da filosofia.)

No âmbito das ontologias **relação** é sinónimo de **relacionamento**; **instância** é sinónimo de **indivíduo**.

Definições

1. "*explicit specification of a conceptualization*", i.e., "uma especificação explícita de uma concetualização", Gruber, 1993.
2. "*formal specification of a shared conceptualization*", i.e., "especificação formal de uma concetualização compartilhada", Borst, 1997.
3. "*An ontology is a formal, explicit specification of a shared conceptualization*", Studer et al., 1998; i.e., a fusão de 1 e 2, "**Uma ontologia é uma especificação explícita e formal de uma concetualização compartilhada**".
4. [AH2017]+: Ontologia:

   O = (C, H, I, R, P, A, O),

   onde:
   - **C:** conjunto de classes
   - **H:** conjunto de relações hierárquicas (taxonómicas) entre classes
   - **I:** conjunto de relações entre as classes e as suas instâncias
   - **R:** conjunto de relações não taxonómicas
   - **P:** conjunto de propriedades das classes da ontologia
   - **A:** conjunto de axiomas
   - **O**[1]: conjunto de instâncias das classes (i.e., o povoamento)
5. Definição informal provavelmente questionável: "Pega-se numa panela feita de classes, propriedades, relações e axiomas, junta-se instâncias (indivíduos) e talvez mais relações não taxonómicas, e mexe-se bem com um motor de inferência.
   Poderá ir ao forno numa aplicação web, preferencialmente com persistência sob SGBD orientado a grafos."

[1] Adicionado à definição AH2017.

Objetivos

Uma ontologia tem por objetivo a representação computacional de **conhecimento** de um modo mais simples (a um nível mais próximo do cérebro humano, por conceitos, relações, propriedades e axiomas), e sua **inferência** (*reasoning*), sendo usada em áreas como a **inteligência artificial, web semântica**, engenharia de software e arquitetura da informação.

(Cf. https://pt.wikipedia.org/wiki/Ontologia_(ciência_da_computação), https://en.wikipedia.org/wiki/Ontology_(information_science), entre outras.)

As relações não taxonómicas duma ontologia acrescentam semântica. As queries a uma ontologia estão facilitadas (cf. SPARQL).

Cf. *https://graphdb.ontotext.com/documentation/standard/introduction-to-semantic-web.html,* para uma introdução interessante ao tema.

2.1.2 Linguagens e Ferramentas

Ontologias

Linguagens

Algumas linguagens usadas para descrever ontologias são **XML**, **XML Schema**, **RDF**, **RDF Schema**, **OWL** (1 e 2) e SKOS (Simple Knowledge Organization System). As linguagens para ontologia são geralmente declarativas.

"*The **Web Ontology Language** (**OWL**) is a family of knowledge representation languages for authoring ontologies. Ontologies are a formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains: the nouns representing classes of objects and the verbs representing relations between the objects.*" Origem: https://en.wikipedia.org/wiki/Web_Ontology_Language [03/01/2021]

OWL 2 (Web Ontology Language, Manchester Syntax (Second Edition)) é das linguagens mais usadas. Reconhecem-se algumas limitações às ontologias, em termos de: *property constructs*; no modo como OWL emprega *constraints* (restrições)… cf. Anexo II.

Dada a sua relevância neste domínio, achei por bem reproduzir, ainda no Anexo II, uma lição em Inglês, sobre OWL – cf. secção Lição sobre OWL.

Editores de ontologias

Protégé: Foi o editor usado neste projeto para desenhar a ontologia. A ferramenta **Protégé** permite manusear ontologias (criar, editar, importar, exportar) e possui um *reasoner* considerado muito completo, o HermiT, da Universidade de Oxford. O Protégé suporta OWL 2 e aparenta ser, atualmente, das ferramentas mais usadas e consensuais, a julgar pelas menções encontradas na web.

Outros editores: WebODE, OntoEdit.

Livrarias

Para a fase de codificação, existem livrarias, em particular do universo da linguagem **Python**, que podem ajudar:

- RDFLib (também disponibiliza SPARQL)
- rdflib-web (para Flask)
- Owlready2

DSL

Uma DSL (domain-specific language), sendo tipicamente feita à medida de uma necessidade concreta, fornece uma sintaxe para reconhecer, e possivelmente agir, sobre um conjunto de informação e/ou comandos.

Uma DSL tem a vantagem de fornecer um meio expedito de preparar, *offline*, informação e/ou comandos a processar posteriormente de modo automático. Numa aplicação aberta a todo o tipo de utilizadores, a sintaxe deve ser acessível, seja porque o utilizador tem facilidade em com ela lidar, seja por haver conversores disponíveis (entre um documento do utilizador e a sintaxe destino). Tem a vantagem, em particular, de facilitar o reaproveitamento de trabalho anterior.

Existem atualmente técnicas relativamente ágeis para desenhar e testar **DSL**. Uma das ferramentas em contexto, bastante popular, é o ANTLR.

2.1.3 Software e Projetos Relacionados

É importante ter em conta que há projetos públicos, alguns abertos, cuja especificação e experiência acumulada se poderão aproveitar.

No contexto genérico das ontologias, dá-se apenas estes exemplos:

- **The FOAF Project** – http://www.foaf-project.org/
- **DBpedia** – https://wiki.dbpedia.org/

  "DBpedia (from "DB" for "database") is a project aiming to extract structured content from the information created in the Wikipedia project. This structured information is made available on the World Wide Web. DBpedia allows users to semantically query relationships and properties of Wikipedia resources, including links to other related datasets.

  In 2008, Tim Berners-Lee described DBpedia as one of the most famous parts of the decentralized Linked Data effort."

- **LinkingOpenData** – https://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

  "The Open Data Movement aims at making data freely available to everyone. There are already various interesting open data sets available on the Web. Examples include

Wikipedia, Wikibooks, Geonames, MusicBrainz, WordNet, the DBLP bibliography and many more which are published under Creative Commons or Talis licenses.

The goal of the W3C SWEO Linking Open Data community project is to extend the Web with a data commons by publishing various open data sets as RDF on the Web and by setting RDF links between data items from different data sources.

RDF links enable you to navigate from a data item within one data source to related data items within other sources using a Semantic Web browser. RDF links can also be followed by the crawlers of Semantic Web search engines, which may provide sophisticated search and query capabilities over crawled data. As query results are structured data and not just links to HTML pages, they can be used within other applications."

Projetos Open Source

Também há projetos de código aberto muito interessantes no âmbito específico desta dissertação. Despertaram-me a atenção os seguintes.

OntoWiki – Semantic Data Wiki and Linked Data Publishing Engine

Endereço: http://ontowiki.net/

Este projeto mereceu especial atenção, por parecer promissor.

http://docs.ontowiki.net/: "*OntoWiki is a semantic application as well as a framework which acts as a hardened basement for your application in the Semantic Web context. One of its main purposes is to assist you managing your knowledge. Knowledge means here machine readable data organized as RDF/XML, Notation3, Turtle as well as Talis(JSON). You organize your knowledge using a feature-rich user interface managing classes, properties and resources.*"

Pontos a favor:

- Open source.
- Disponibiliza um navegador da ontologia.
- Suporta vários formatos ontológicos, incluindo **Turtle**.
- Suporta SPARQL 1.0.
- Suporta SGBD como Vituoso ou MySQL, ou outro à escolha (*backend independent)*.
- Linked Data server: permite obter dados adicionais na web ("*fetch additional data from the web*").

No entanto, alguns elementos da sua arquitetura revelaram-se dificultadores:

- **Escrito em PHP** 5.2 (que seria mais uma frente para desbravar), apesar de anunciar que permite extensões escritas noutras linguagens, como JavaScript.
- Precisa do servidor **Apache** (ou nginx), algo pesado.
- A instalação revelou-se difícil sobre Microsoft Windows…

Gramps – Research, organize and share your family tree with Gramps

Endereço: https://gramps-project.org/

"Gramps is a free software project and community. We strive to produce a genealogy program that is both intuitive for hobbyists and feature-complete for professional genealogists. It is a community project, created, developed and governed by genealogists."

Caraterísticas relevantes:

- Open source.
- Escrito em Python.
- Aparência agradável.
- Muito grande.
- Ligação a SGBD SQLite ou BerkeleyDB, ou seja, desvia-se dum SGBD orientado a grafos, como pretendemos nesta dissertação.
- Consta que dificulta a composição de livros.

Zim – A Desktop Wiki

Endereço: https://zim-wiki.org/

"Zim is a graphical text editor used to maintain a collection of wiki pages."

Eis um editor que poderá revelar-se interessante para o projeto, quanto mais não seja para descobrir padrões, a traduzir em requisitos do projeto.

Conclusões

Revelou-se difícil encontrar um projeto aberto já existente que fosse de feição a esta dissertação, i.e., que combinasse: ser escrito em Python ou JavaScript; BD orientada a grafos (com SPARQL); fácil instalação e modificação.

2.2. Genealogias

2.2.1 Definição de Genealogia

O que é a genealogia? Vejam-se algumas definições.

Na Infopédia

Origem: https://www.infopedia.pt/dicionarios/lingua-portuguesa/genealogia [15/11/2020]:

genealogia

nome feminino

1. HISTÓRIA ciência auxiliar da história que estuda a origem e a sucessão das famílias, descrevendo as relações de parentesco entre as gerações
2. obra ou documento onde se expõe cronologicamente a linha de gerações anteriores a um indivíduo, a uma família ou a um grupo

3. filiação, em ordem regressiva, de um indivíduo ou espécie, que determina a ascendência
4. conjunto de relações genéticas, estabelecidas através de gerações, entre seres vivos; história da evolução de uma espécie; filogenia
5. linhagem; estirpe
6. *figurado* origem; procedência

Na Wikipédia

In https://pt.wikipedia.org/wiki/Genealogia [14/11/2020]: «A genealogia ocupa-se da identificação da ligação biológica entre diferentes indivíduos e da reconstituição da sequência ordenada de gerações dentro de um grupo familiar, procurando determinar as origens, a rede de parentescos e a evolução cronológica da família. Numa perspetiva mais abrangente, onde se associa à prosopografia, à história, às ciências humanas e sociais, procura reconstituir o perfil e a história social, política, económica e cultural da família e seus integrantes, as suas associações com outros grupos e o seu papel na sociedade. No âmbito da **história**, a genealogia está centrada no estudo das **famílias**, contribuindo para e sendo subsidiada por outras ciências, como a sociologia, a economia, a história da arte, a genética, a medicina ou o direito.»

2.2.2 Conceitos

Nesta secção discorre-se sobre alguns conceitos importantes em Genealogia.

Graus de Parentesco e Sua Inferência

Para não perder de vista a potencial complexidade das relações de parentesco, sugiro um relance sobre a seguinte imagem.

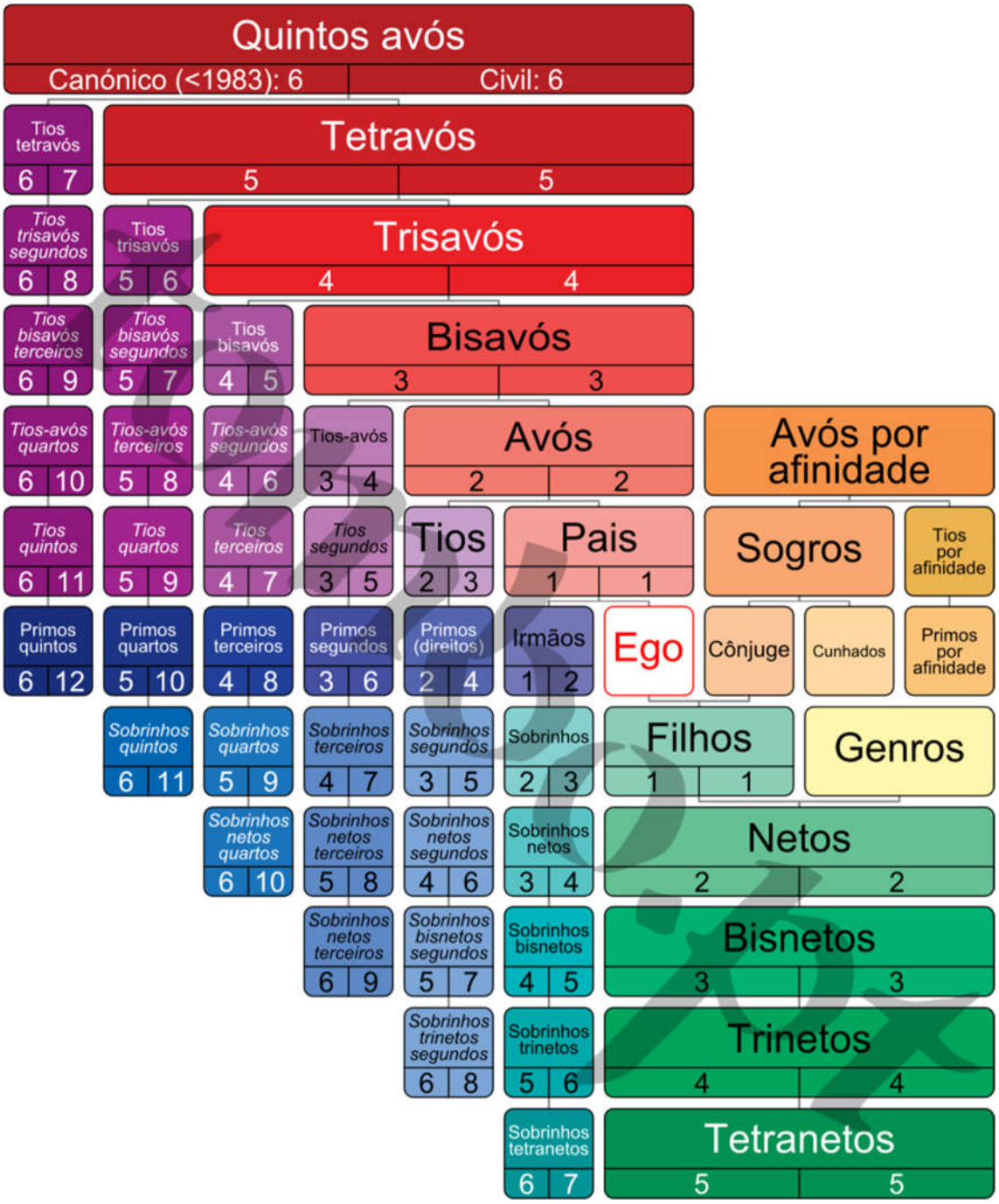


Figura 1 – Graus de parentesco, em https://tombo.pt

A genealogia é uma área rica em inferência. Basta saber os **pais e respetivo género sexual** para inferir graus de parentesco consanguíneos, como irmãos (os filhos dos mesmos pais), tios (os irmãos dos pais), avós paternos (os pais do pai) e maternos (os pais da mãe) e quase todos os restantes. Os que faltam (parentesco por afinidade) inferem-se sabendo o **cônjuge**, como cunhados (um irmão do cônjuge, ou o cônjuge dum irmão) ou meios-irmãos, entre outros. Nalguns casos, para evitar aberrações, poderá ter de se testar caraterísticas como a não reflexividade de uma relação (como por exemplo em referência a irmãos), ou

mesmo o sexo.

Como é sabido, as ontologias permitem definir axiomas, que duma forma simples poderiam, por execução de um *reasoner*, calcular graus de parentesco entre indivíduos concretos. No entanto, constata-se que há problemas nesta abordagem, cuja descrição deixei para o capítulo de Resultados e Discussão, secção Problemas dos Axiomas x *Reasoners*.

2.2.3 Software e Projetos Relacionados

Abreviando uma longa história, a genealogia é praticada há séculos, sobretudo por encomenda de famílias ilustres. No âmbito desta dissertação, estamos, naturalmente, mais interessados no estudo informático desta atividade, particularmente em termos de implementação.

Uma pesquisa rápida na internet deixa-nos com a sensação de que já existe muito software nesta área, com muita usabilidade, proporcionando construções gráficas não só de árvores de ascendentes e descendentes, mas também mapas e serviços de localização de possíveis familiares! Alguns, possibilitam importar e exportar dados em formatos específicos, mas consensuais, como **GEDCOM**.

Em `https://en.wikipedia.org/wiki/Comparison_of_genealogy_software` podemos apreciar algumas marcas. As caraterísticas revelam-se interessantes, muitas das quais (pesquisa em arquivos, geolocalização, etc.) consideramos fora do âmbito desta dissertação por escassez de recursos.

Numa web cada vez mais comercial, nem todas as sugestões encontradas são sinceras: há que ter o cuidado, em qualquer pesquisa, de desconfiar do interesse comercial (ainda que legítimo) de quem aconselha/promove determinado software, não pelas suas qualidades, mas pelo que as marcas poderão ter pago para obter essa publicidade.

Software de Código Fechado

Este tipo de software não despertou muito interesse no âmbito do projeto, valendo pela possibilidade de poder indicar requisitos, em particular ao nível da aparência.

- Legacy 9: Bem aclamado pela crítica. Inclui, entre outras caraterísticas, coisas como geolocalização e pesquisa assistida em arquivos e web sites.

  "Traduzido do inglês - O "Legacy Family Tree" é um software de genealogia para Windows que auxilia os historiadores da família a rastrear, organizar, imprimir e compartilhar a história da família. A edição padrão é distribuída como freeware, sem restrições, exigindo apenas o registro no site da empresa para o download do software. Wikipedia (inglês)"

- MyHeritage https://www.myheritage.com.pt/

  Trata-se de uma plataforma web, que promete ao utilizador descobrir a sua árvore genealógica, em particular via testes de ADN, e pesquisa numa "base de dados

internacional".

- FamilySearch – https://www.familysearch.org/pt/

  "FamilySearch is a nonprofit organization and website offering genealogical records, education, and software. It is operated by The Church of Jesus Christ of Latter-day Saints (LDS Church), and is closely connected with the church's Family History Department. The Family History Department was originally established in 1894 as the Genealogical Society of Utah (GSU) and is the largest genealogy organization in the world. FamilySearch maintains a collection of records, resources, and services designed to help people learn more about their family history. Facilitating the performance of LDS ordinances for deceased relatives is another major aim of the organization. Although it requires user account registration, it offers free access to its resources and service online at FamilySearch.org. In addition, FamilySearch offers personal assistance at more than 5,100 family history centers in 140 countries, including the Family History Library in Salt Lake City, Utah. The Family Tree section allows user-generated content to be contributed to the genealogical database. As of February 2021, there are over 1.3 billion individuals in the tree and the historical records database contains over 5.7 billion digital images, including digitized books, digitized microfilm, and other digital records."

Software Open Source

Este tipo de software desperta mais interesse pela possibilidade de reaproveitar o trabalho feito. Por praticabilidade e interesse pessoal, além de seguir a corrente, procurei projetos que permitam escrever código na linguagem JavaScript ou Python.

Destaco, das minhas pesquisas, os seguintes softwares, incluindo módulos reutilizáveis.

GeneWeb

Eis um software com uma interface visual que considero agradável (na sua versão atual, 7), rico em possibilidades de informação e navegação. Também este software apresenta diversas caraterísticas, mas poderá servir de inspiração. Suporta GEDCOM. Possível barreira: escrito em OCaml.

Sítio oficial: https://geneweb.tuxfamily.org/wiki/GeneWeb: "*GeneWeb is free, open source genealogy software written in OCaml by Daniel de Rauglaudre, a researcher at the Institut national de recherche en informatique et en automatique (Inria).*"

https://en.wikipedia.org/wiki/GeneWeb: "*GeneWeb is a free multi-platform genealogy software tool created and owned by Daniel de Rauglaudre of INRIA. GeneWeb is accessed by a Web browser, either off-line or as a server in a Web environment. It uses very efficient techniques of relationship and consanguinity computing, developed in collaboration with Didier Rémy, research director at INRIA. GeneWeb is used as the engine for several public genealogy websites, including Geneanet, a collection of inter-searchable genealogical databases currently containing references to more than 225 million persons.*

Notable features of GeneWeb include:

- *High capacity.*
- *Web Server: When GeneWeb runs on a computer connected to the internet, it can accept HTTP requests from web clients, generating and serving HTML web pages and linked objects (images, etc.).*
- ***GEDCOM**: GeneWeb supports import and export of GEDCOM files.*
- *UTF-8: GeneWeb supports UTF-8.*
- *Written in **OCaml**.*

Figura 2 – Rainier III GRIMALDI (geneweb.tuxfamily.org)

Gramps

Escrito em Python. https://gramps-project.org/blog/

Projeto muito interessante, com caraterísticas muito ricas.

Contudo, como SGBD usa Berkeley DB, ou SQLite, o que foge aos nossos objetivos.

Dtree

"A library for visualizing data trees with multiple parents, such as family trees. Built on top of D3."

Kingraph

Plots family trees using JavaScript and Graphviz.

Silsilah

A genealogy/family tree application, built with Laravel.

Bonsai

Self-hosted family wiki engine / photoalbum

Genealogy

Laravel 8 and Vue family tree and genealogy data processing website.

Graphview

Flutter GraphView is used to display data in graph structures. It can display Tree layout, Directed and Layered graph. Useful for Family Tree, Hierarchy View.

Treeviz

"Tree diagrams with JavaScript."

2.3. Humanidades digitais

"Humanidades digitais é uma área académica na interseção das tecnologias digitais ou de computação com as disciplinas das humanidades. Inclui o uso sistemático de recursos digitais nas humanidades, bem como a análise da sua aplicação."

Traduzido do inglês em https://en.wikipedia.org/wiki/Digital_humanities, 05/01/2021.

2.4. Inferência

Alguns conceitos em inferência:

Constraint Solvers

Há sistemas que auxiliam a resolver restrições. Alguns, até, escritos em Prolog, conseguem conjugar restrições do domínio temporal, relacionando o período de vida ativa com datas curriculares (nascimento, puberdade, serviço militar, etc.).

Machine Learning

É muito sensível ao contexto. Por ser uma área muito extensa, não foi aprofundada no contexto desta dissertação.

Keyword spotting

From Wikipedia, the free encyclopedia, 06/03/2021.

"Keyword spotting (or more simply, word spotting) is a problem that was historically first defined in the context of speech processing. In speech processing, keyword spotting deals with the identification of keywords in utterances.

Keyword spotting is also defined as a separate, but related, problem in the context of document image processing. In document image processing, keyword spotting is the problem of finding all instances of a query word that exist in a scanned document image, without fully recognizing it."

No trabalho em causa, estamos preocupados especialmente com inferência de indivíduos da ontologia a partir de nomes próprios e datas (ainda que possivelmente reduzidas ao ano), além de graus de parentesco. Racional: Numa história ou fotografia, aparecem, frequentemente, não os nomes completos, mas apenas os nomes próprios, eventualmente associados a um grau de parentesco. Para desambiguar, poderão ser essenciais o grau de parentesco, uma data ou ano, de nascimento ou dum evento. Ex.: "foto do **tio Serafim** tirada em **…**". Isto realça o nosso interesse na genealogia para efeitos de inferência de ID (não tanto para descobrir linhagens).

3. Levantamento e Análise de Requisitos

O domínio em causa parece conduzir-nos naturalmente a uma ontologia rica.

Como se pode derivar dos objetivos do projeto (cf. secção respetiva no capítulo 1), este não pretende focar-se principalmente na genealogia numa perspetiva de linhagens, nem concorrer com outras dissertações desse âmbito, nem tão pouco com aplicações comerciais já no mercado e que levam bastante vantagem. A genealogia servirá, sim, de suporte à inferência, na identificação de pessoas (além de lugares, eventos, etc.) em fotografias e histórias (de família e outras).

Vem-me frequentemente à memória diálogos entre os meus pais, em que, referenciando um deles certo relacionamento pessoal pelo nome próprio (e mesmo com apelidos e alcunhas), o outro não identifica a pessoa até se explicitar que era, por exemplo, irmã/irmão, filha/o, prima/o de alguém, ou seja, até chegar a um relacionamento seu conhecido. Daqui se intui que a parte genealógica referente a graus de parentesco poderá ser determinante.

3.1. Do levantamento de requisitos

A recolha de requisitos partiu dos objetivos do projeto. Ocorreu alguma recolha de impressões de outras aplicações, de *brainstorming*, especialmente com o orientador da tese, e de introspeção (também partilhada com o orientador). Não foram realizados inquéritos públicos.

É claro que pode ter-se chegado, enfim, a um conjunto de requisitos algo subjetivo, mas por outro lado pareceu-me tratar-se de um tema sobre o qual o orientador, Prof. José João Almeida, já leva anos de reflexão e experiência. Deste modo, encontrei no orientador um "cliente" informado e esclarecido sobre o que lhe poderia interessar pessoalmente, o que é mais do que se pode dizer de muitas entrevistas a potenciais clientes, a menos que já trabalhem intensivamente com alguma aplicação… Ao fim de cerca de trinta anos de experiência a trabalhar em engenharia de software, constato que só depois de começarem a trabalhar com a aplicação é que os utilizadores conseguem fazer chegar pedidos de melhoria… Não considero, portanto, que tenha havido necessariamente uma grande perda por falta de diversidade de opinião, até porque muitos requisitos estão ainda por implementar. Ainda assim, deve sempre ter-se a humildade de ouvir opiniões e desejos, mesmo quando se adivinha que serão muito trabalhosos.

3.2. Lista de Entregáveis (*Deliverables*)

3.2.1 ANB Ontology

Ontologia, na perspetiva das humanidades digitais, sobre:

- **Pessoas** singulares e coletivas (organizações), e respetivos **relacionamentos**:
    - **familiares** (graus de parentesco);
    - **sociais** (ex.: "Fulano de Tal" conhece "Sicrano, Famoso Músico"); e
    - **outros eventuais**.
- **Locais de interesse.**
- **Documentos de diversa natureza, a relacionar com indivíduos**.
- **Eventos de diversa ordem: familiar, social e histórica**.

Cf. capítulo Arquitetura e Desenho da Ontologia.

3.2.2 ANB Web App

Aplicação web para gestão CRUD da ontologia, visualização gráfica, carregamento de informação. O carregamento de informação e inferência de nomes estão demarcados no entregável NameTK.

3.2.3 ANB DSL

Módulo para carregar informação na ontologia, seja de indivíduos ou para enriquecer-lhe a estrutura (cf. capítulo 5). O carregamento deve estar disponível a partir de ficheiro, numa sintaxe a definir, ou via *copy-paste*.

3.2.4 ANB NameTK

"Name ToolKit": Módulo de pesquisa de nomes, com inferência de indivíduos da ontologia: inferência e desambiguação de identificadores de instâncias da classe Pessoa, ou Org, guiado por nomes, datas e relacionamentos.

3.3. Técnica de Priorização

Por se considerar um padrão intuitivo, optou-se pela **priorização de requisitos** segundo a técnica **MoSCoW**:

- Must – Tem de ter (requisitos que **têm** de ser considerados);
- Should – Deveria ter (requisitos que **deveriam** ser considerados);
- Could – Poderia ter (requisitos **desejáveis**, mas não necessários);
- Would/Won't – Interessante ter (requisitos que **poderão, ou não,** ser considerados, no futuro).

A etiquetagem foi feita intuitivamente, dando prioridade aos requisitos essenciais para CRUD, simplisticamente falando.

3.4. Requisitos Funcionais ou de Dados

(Nota: Confrontar informação de apoio no anexo *Técnicas de Levantamento de Requisitos*.)

Para apresentação dos requisitos optou-se pelo cartão Volere (descrito nos anexos).

A numeração dos requisitos funcionais será um numeral sequencial com prefixo RF.

O item *Description* do cartão é inscrito no título de cada secção respetiva. Esta opção destina-se a plasmar os requisitos no índice, para facilitar a identificação neste documento.

RF1: O sistema deve permitir CRUD de indivíduos da ontologia

Requirement #: RF1	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Para se poder povoar e gerir a ontologia por via da aplicação.		
Originator: *Brainstorming*; padrão		
Fit Criterion: Como sugestão, em cada área de interação disponibilizada para cada classe da ontologia, o utilizador poderá chamar um formulário para poder acionar CRUD, contexto em que Update e Create permitirão editar. Poderão existir outras formas de realizar estas tarefas – cf. RF6.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. **Implementado** (interativamente e via SPARQL).		

RF2: O sistema deve permitir visualizar listas de dados da ontologia

Requirement #: RF2	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Para conhecer coleções de dados relacionados e permitir a consequente navegação pela ontologia.		
Originator: Introspeção; padrão		
Fit Criterion: Nos contextos elegíveis, será mostrada a informação na forma de tabela, com hiperligações para a origem dos itens listados.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. **Implementado**.		

RF3: O sistema deve permitir visualizar os dados de cada instância da ontologia

Requirement #: RF3	Requirement Type: Funcional	Event/BUC/PUC #:

Rationale: Poder conhecer, de maneira integrada, os dados duma instância e suas relações.		
Originator: Introspeção; padrão		
Fit Criterion: Nos contextos elegíveis, será apresentada a informação na forma duma página web, com toda a informação relevante duma instância, com hiperligações para informação associada. Cada classe poderá ter uma página desenhada à medida das suas necessidades.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. **Implementado**.		

RF4: O sistema deve permitir visualizar a genealogia duma pessoa

Requirement #: RF4	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Para conhecer, de maneira mais abrangente, os relacionamentos familiares duma pessoa, sem descurar outros relacionamentos pessoais.		
Originator: *Brainstorming*; padrão		
Fit Criterion: Nos contextos elegíveis, será mostrada a informação na forma duma página web, com os relacionamentos familiares, e outros pessoais, duma instância, com hiperligações para permitir navegar.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. **Implementado** no âmbito do RF3.		

RF5: O sistema deve permitir navegar pelos relacionamentos entre indivíduos

Requirement #: RF5	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Para conhecer, de maneira mais abrangente, a informação específica das instâncias e seus relacionamentos.		
Originator: *Brainstorming*; padrão hipertexto/web		
Fit Criterion: Nos contextos elegíveis, existirão hiperligações para navegar, pelos relacionamentos, para as páginas específicas das instâncias, senão para navegar pela informação em forma de tabela.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. Implementado.		

RF6: O sistema deve disponibilizar interface SPARQL para CRUD

Requirement #: RF6	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Recorrer ao padrão SPARQL para dispor duma ferramenta versátil, ainda que especializada, de gestão da ontologia.		
Originator: Introspeção		
Fit Criterion: Deverá existir uma página onde se possa introduzir e executar *queries* SPARQL, obtendo os resultados em forma tabular ou de debug, conforme opção do utilizador.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies: RNF1, RNF2	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. Implementado.		

RF7: O sistema deve permitir carregar dados via ficheiro de texto conforme DSL

Requirement #: RF7	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Permitir recolha e preparação *offline* de informação a carregar. Permitir carregar informação com menos interação. Permitir reaproveitar edições de anteriores de ficheiros, que poderão ser facilmente adaptados, evitando trabalho e poupando tempo.		
Originator: *Brainstorming*; padrão		
Fit Criterion: Nos contextos elegíveis, será disponibilizada uma opção para indicar um ficheiro de texto, conforme com uma especificação DSL, para carregar informação na ontologia. O fluxo poderá passar por uma área tampão, onde o utilizador terá de confirmar a operação de inserção e atualização de informação. Na ocorrência de erros, será desejável obter um *log* de mensagens.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies: Especificação DSL	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. **Implementado** sobre a sintaxe Turtle.		

RF8: O sistema deve permitir a definição interativa de hiperligações a indivíduos da ontologia

Requirement #: RF8	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Para permitir, orientado ao contexto, enriquecer a ontologia com hiperligações ou relacionamentos.		
Originator: *Brainstorming*		
Fit Criterion: Em textos de resumo, descrição, transcrição, deverá ser possível selecionar palavras inferir e a estabelecer hiperligações a instâncias. Esta tarefa deverá ser assistida		

pela aplicação, dando a escolher as instâncias mais prováveis. O utilizador marca o texto a analisar e usará algum controlo para pedir uma marcação que será disponibilizada como hiperligação ao utilizador.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Should	Dependencies: NameTK	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. **Implementado** – cf. inferência interativa/dinâmica e âncoras html.		

RF9: O sistema deve permitir adicionar estrutura à ontologia

Requirement #: RF9	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Para permitir enriquecer a ontologia com classes e relações necessárias.		
Originator: *Brainstorming*		
Fit Criterion: Deverá existir um modo de introduzir/modificar triplos no contexto de estrutura de ontologia, como classes, atributos e relacionamentos.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Should	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. **Implementado** – cf. áreas DSL e SPARQL.		

3.5. Requisitos Não Funcionais

A numeração dos requisitos não funcionais será um numeral sequencial com prefixo RNF.

3.5.1 Aparência

(O essencial da aparência da aplicação.)

Seguem-se sugestões. Poderá tentar seguir-se, de um modo geral, o estado da arte da aparência de aplicações web.

Genealogia

A página de genealogia duma pessoa poderá ser organizada como na figura seguinte, preferencialmente com fotografias, nome abreviado e período de vida (nascimento e idade, ou então nascimento, óbito e idade à data de falecimento). Poderão ser selecionadas apenas partes desta interface – cf. o padrão *Pedigree Chart*.

Dada a quantidade de informação, poderá ser útil haver controlos que esperem pelo clique do utilizador antes de desdobrar mais informação.

Poder-se-á ainda considerar o seguinte padrão de *display*, com sete linhas:

1. Pessoa de referência (seja i)
2. Pai
3. Mãe
4. Avô paterno
5. Avó paterna
6. Avô materno
7. Avó materna

Em que linha (i*2) =>pai de i, linha (i*2 + 1) => mãe de i (numeração de Sosa-Stradonitz).

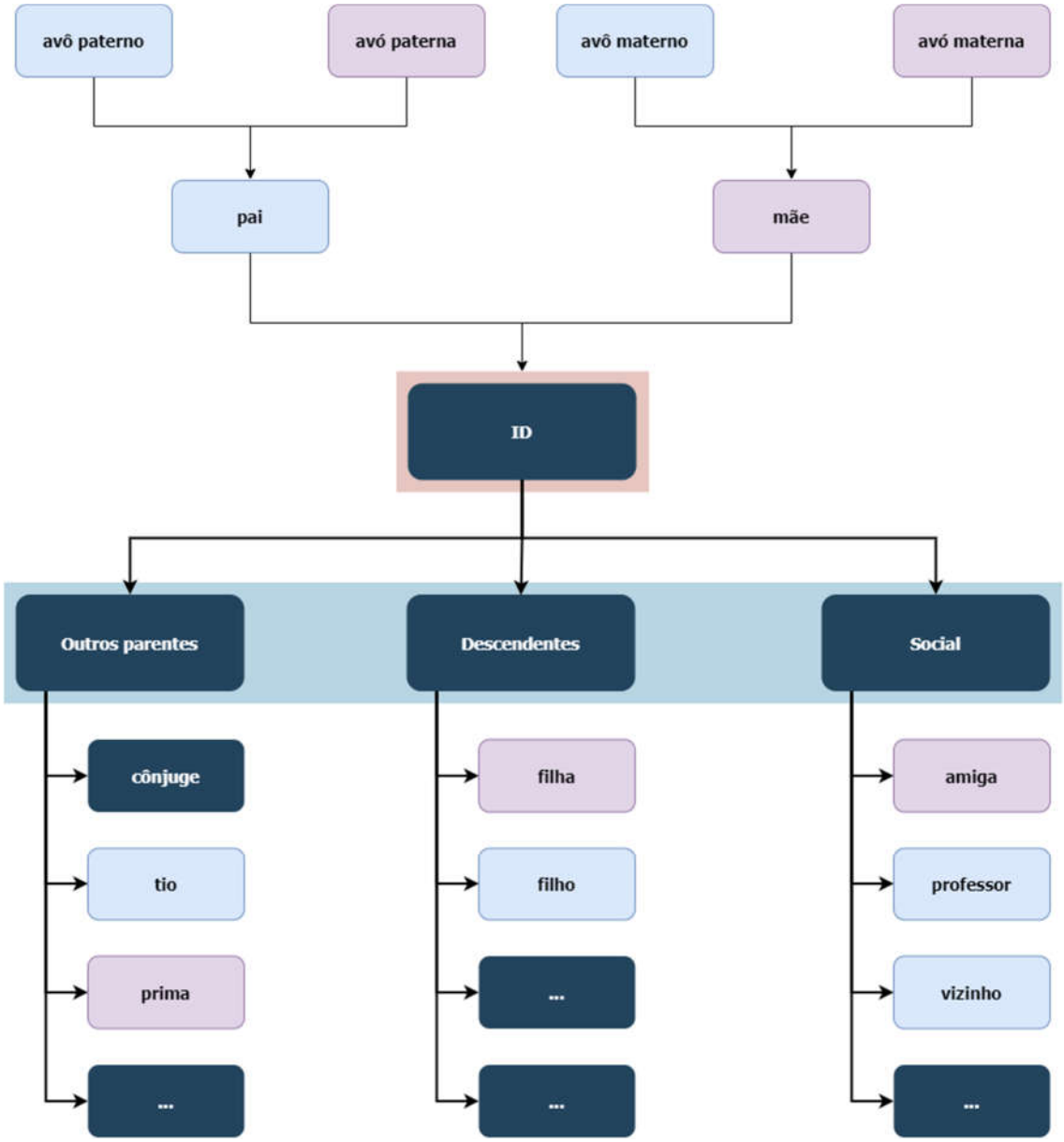


Figura 3 – Pessoa: sugestão de apresentação em árvore

Exemplo de informação acompanhada de fotos:

Stéphanie
1965
56 anos

Alfred
HITCHCOCK
1899-1980
80 anos

Pessoa

A página de informação duma pessoa poderá ser assim:

Nome Completo **Foto**

e outros dados pessoais…

Relacionamentos familiares

Genealogia (hiperligação para a página de genealogia, cf. secção anterior)

Listagem de relacionamentos (listagem de hiperligações)

Relacionamentos pessoais (listagem de hiperligações)

Cronologia (de eventos com que a pessoa está associada por relacionamentos ontológicos)

Completa (todos os eventos associados)

Familiar (eventos de tipo familiar)

Curricular (eventos de tipo curricular)

Histórica (eventos de tipo histórico)

Outros relacionamentos (listagem de hiperligações de relacionamentos não contidos nas opções acima)

3.5.2 Usabilidade

(Facilidade de utilização e outras considerações de usabilidade.)

RNF1: O sistema deve apresentar os dados em formato simplificado

Requirement #: RNF1	Requirement Type: Usabilidade	Event/BUC/PUC #:
Rationale: Numa utilização regular, a facilidade de visualização confere facilidade e eficácia de utilização, melhorando essa experiência. O "ruído visual" dos IRI completos introduzirá complexidade desnecessária e poderá desmotivar o utilizador.		
Originator: *Brainstorming*		
Fit Criterion: Os IRI serão apresentados limpos de prefixos ontológicos.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. Implementado.		

RNF2: O sistema deve permitir visualizar os dados em formato de debug

Requirement #: RNF2	Requirement Type: Usabilidade	Event/BUC/PUC #:
Rationale: Para inspeção rigorosa aos dados.		
Originator: *Introspeção*		
Fit Criterion: Os IRI serão apresentados completos, incluindo prefixos ontológicos. Serão mostrados todos os dados retornados por pedidos ao SGBD.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. Implementado.		

RNF4: O sistema deve permitir interação pela linha de comandos

Requirement #: RNF4	Requirement Type: Usabilidade	Event/BUC/PUC #:
Rationale: Para permitir flexibilidade de utilização.		
Originator: *Brainstorming*		
Fit Criterion: Deverá existir uma interface para comunicar com o sistema via linha de comandos do sistema operativo.		
Customer Satisfaction: 4	Customer Dissatisfaction: 3	
Priority: Should	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021.		

3.5.3 Desempenho

(Quão rápida, quão extensa, quão precisa uma funcionalidade tem de ser.)

3.5.4 Operacionais

(Ambiente de operação do produto, físico ou não, e considerações sobre esse ambiente.)

RNF3: O sistema deve ter a arquitetura duma aplicação web

Requirement #: RNF3	Requirement Type: Operacional	Event/BUC/PUC #:
Rationale: Obter um padrão *state of the art* em termos de usabilidade, aparência, disponibilidade, ferramentas de suporte ao desenvolvimento, entre outros. O sistema poderá ser instalado num servidor.		
Originator: *Brainstorming, padrão corrente.*		
Fit Criterion: A aplicação ficará disponível via navegador web. Será desenvolvida com		

uma *framework tool* de código aberto.		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Should	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. Implementado.		

3.5.5 Manutenção, Portabilidade e Suporte

(As alterações esperadas e o tempo permitido para as efetuar.)

RNF5: O sistema deve ser portável entre diferentes Sistemas Operativos

Requirement #: RNF5	Requirement Type: Portabilidade	Event/BUC/PUC #:
Rationale: Não limitar desnecessariamente a um ambiente específico, tendo em conta que poderá ser instalado localmente.		
Originator: *Brainstorming, padrão corrente.*		
Fit Criterion: A aplicação poderá instalar-se e correr em sistemas operativos que aceitem a framework escolhida (Flask/Python).		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority: Must	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado em jan/2021. Implementado.		

3.5.6 Segurança

(A segurança e confidencialidade do produto.)

Observações: O sistema será desenhado, inicialmente, para um uso privado em dispositivo local, daí que não relevem situações de segurança apreciáveis. Os dados guardados serão de interesse familiar ou público.

3.5.7 Culturais e Políticos

(Requisitos especiais que surgem por causa das pessoas envolvidas no desenvolvimento e operação do produto.)

Cultural: Uma questão que poderá dificultar alguns aspetos da utilização são os novos géneros sexuais, que não foram especificamente atendidos neste projeto.

3.5.8 Legais

(Leis e normas que se aplicam ao produto.)

Dada a utilização privada da aplicação, supõe-se que o RGPD não tenha cabimento, a menos que os dados pessoais venham a ser publicados.

4. Arquitetura e Desenho da Ontologia

4.1. Metodologia Seguida

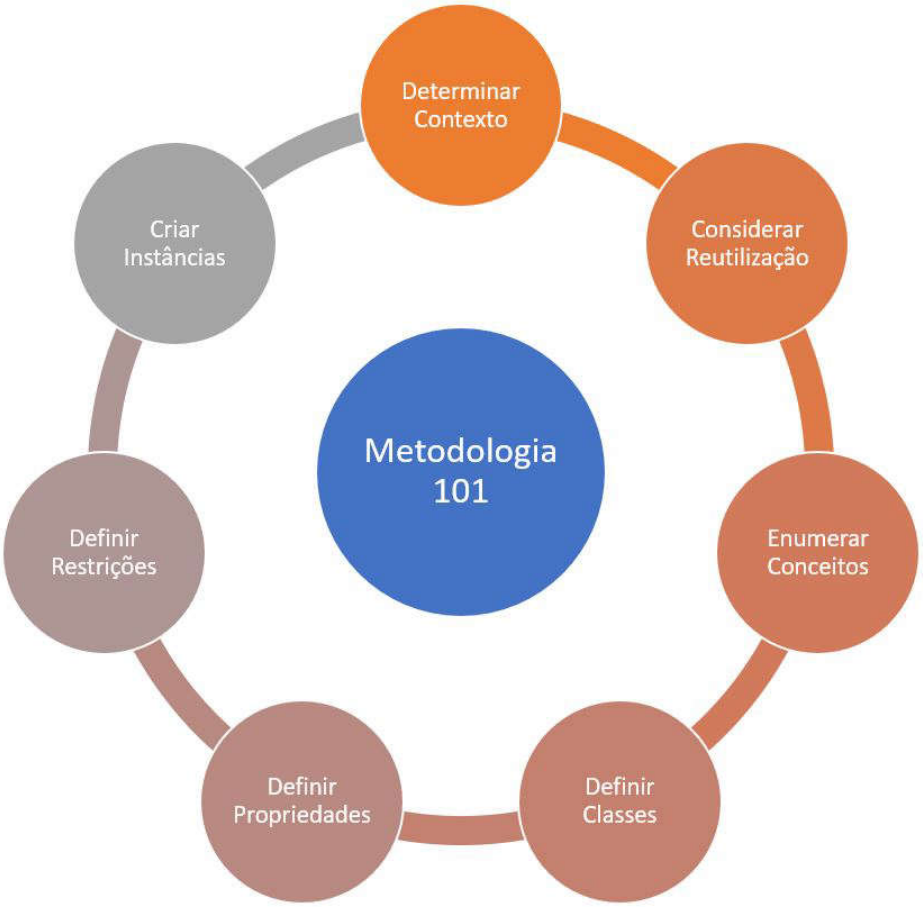


Figura 4 – Esquema da metodologia 101

Esta ontologia foi desenvolvida seguindo uma metodologia equivalente à *Ontology Development 101*[2] da Universidade de Stanford [NM00], por sua vez equivalente à metodologia referida em [AH2017], plasmada em sete passos:

1) Determinar o domínio do conhecimento e o âmbito de aplicação da ontologia. Equivale a responder às seguintes questões:
   a) Qual é o domínio do conhecimento que a ontologia irá cobrir?
   b) Para que vamos usar a ontologia?
   c) A que perguntas deve responder?

[2] "*101 (pronounced ONE-oh-ONE) is a topic for beginners in any area. It has all the basic principles and concepts that are expected in a particular field.*", in https://en.wikipedia.org/wiki/101_(topic) 12/04/2022.

d) Quem vai usá-la e mantê-la?

2) Considerar a reutilização de ontologias. Racional: evitar repetir o trabalho já realizado anteriormente por alguém, para não ter de "reinventar a roda".

3) Enumerar os termos importantes (conceitos, mas ainda sem a preocupação de os classificar como classe ou propriedade).

4) Definir classes e hierarquia de classes – onde, a partir da lista de termos, se eliminam redundâncias e se definem as classes da ontologia (organizadas em hierarquia).

5) Definir propriedades das classes, i.e., atributos e relações entre classes.

6) Definir restrições, como tipos de dados dos atributos, domínio e contradomínio de relações e outras restrições.

7) Criar indivíduos (instâncias das classes).

4.2. Aplicação da Metodologia

1. Domínio e âmbito da ontologia

A ontologia servirá de suporte à aplicação web objeto deste projeto, no domínio das humanidades digitais – cf. objetivos deste projeto, no capítulo 1.

O domínio desenrola-se em torno de histórias de família, pessoas e documentos, e irá responder a perguntas como saber dos familiares e outros relacionamentos de uma pessoa, em que histórias e fotos (preferencialmente anotadas) é referenciada, qual o seu percurso de vida (seja familiar ou curricular), em que eventos históricos participou, em que casa especial viveu.

Quem irá usar a ontologia será potencialmente qualquer pessoa com interesse em gerir informação avulsa sobre pessoas e seus relacionamentos, possivelmente no âmbito familiar, mas não obrigatoriamente.

2. Considerar a reutilização de ontologias

Foi revisitada, com cuidado, a ontologia para genealogia das aulas da unidade curricular PRC 2019/2020, bem como a sua fonte de inspiração, [SSMJ2015] http://owl.cs.manchester.ac.uk/ publications/talks-and-tutorials/fhkbtutorial. Esta última introduz uma complexidade considerável a nível de classes, que considero algo desmotivante, uma vez que, numa fase inicial de desenvolvimento, introduz demasiada complexidade sem um objetivo claro. Em particular, refira-se que muitas ou quase todas as classes podem ser substituídas por uma função, como por exemplo Ancestor (alguém com descendentes), Man (alguém do sexo masculino) e muitas outras referentes a graus de parentesco (note-se que me refiro às classes, não aos relacionamentos).

Foi ainda consultado [Rib2017].

Optando por simplificar classes, emergiu uma outra complexidade, difícil de contornar, e, portanto, aceite: prende-se com os relacionamentos por graus de parentesco, que rapidamente se complicam e desdobram, para mais com subdivisão por géneros sexuais, e parentescos que saem da órbita das relações de sangue, como parentescos por afinidade.

A genealogia está muito acomodada a dois progenitores de sexos diferentes. Poderá ser um desafio, em relação a graus de parentescos, acomodar casais homossexuais, procriação por recurso a bancos de esperma, barrigas de aluguer, divórcios e novos casamentos, adoção, famílias de acolhimento de crianças e jovens em risco.

Os termos ingleses requerem estudo e tradução cuidada para o português. Optei pelos termos portugueses, para um melhor e mais direto ajuste ao nosso contexto e defesa da nossa língua, sem colocar de parte a possibilidade de apresentação multilíngue na aplicação. Outro cuidado potencial a ter será alguma diferença em relação ao português do Brasil. Abundam na internet os termos e tabelas. Escolhi uma tabela que apresento na secção 2.2.2 Conceitos.

3. Enumeração de termos (conceitos)

(Neste passo, pretende-se enumerar os termos importantes, ou seja, os conceitos, mas ainda sem a preocupação de os classificar como classe ou propriedade (atributos e relacionamentos).)

Termos recolhidos, no âmbito do parentesco

Neste âmbito, há muitos termos a considerar, mas felizmente já foram historicamente definidos pelo estudo da genealogia. Há parentescos sanguíneos e outros por afinidade. A título de exemplo: pai, mãe, avô, avó, filho, filha, enteado, enteada, irmão, irmã, esposo, esposa – com alternativas… (marido, mulher) – etc… cf. capítulo «Estado da Arte | Genealogias | Conceitos», figura Graus de parentesco.

No âmbito da nossa ontologia, temos como exemplo:

Grau de parentesco	**Termo ontológico originado**
pai	temPai, éPaiDe (inversa)
mãe	temMãe, éMãeDe (inversa)

Além da necessidade de distinguir sexos, serão necessários termos em que o sexo seja indistinto. Assim, poderão servir nessas situações, entre outros, termos como **progenitor** (pai ou mãe), **avôOuAvó** (progenitor de grau 2, um termo menos conhecido e talvez menos prático menos intuitivo).

Termos recolhidos, extra-parentesco

Nesta secção, optou-se por agrupar os termos extra-parentescos, por tema, de modo a facilitar a sua leitura.

Destacados I:

- **Animal**: Não humano, mas ser vivo multicelular, com capacidade de locomoção e de resposta a estímulos, que se nutre de outros seres vivos.
- **Casa**: Edifício de habitação relevante em termos familiares. Para outros edifícios, considerar Local.
- **Compilação**: Coleção personalizada de indivíduos de qualquer classe (cf. Doc, Evento, etc.). Assim, uma compilação não vazia deve estar obrigatoriamente associada a outros indivíduos, conforme adequado, via relacionamentos temDoc, temEvento, temCasa, etc.

  Doc: Documento num sentido lato: enquadramento digital que descreve, transcreve ou referencia qualquer tipo de documento digital ou físico, tipificado (cf. tipoDoc). Exemplos de documentos:
  - **áudio**: um ficheiro de som;
  - **artigo**: um documento informativo ou enciclopédico; exs.: uma biografia, um artigo de jornal, a descrição de um objeto (exs.: quadro, vaso, etc.), um ensaio;
  - **carta**: correspondência;
  - **certidão**: documento oficial; exs.: certidão de batismo ou casamento, caderneta militar, etc.;
  - **imagem**: imagem digital; ex.: fotografia e respetiva descrição;
  - **narrativa**: uma narrativa, real ou ficcionada; exs.: uma história de família real, um conto;
  - **vídeo**: um ficheiro de vídeo;
  - **outro**: nenhum dos anteriores.
- **Evento**: Acontecimento memorável, preferencialmente datado, do âmbito **Familiar**, **Curricular, Histórico** ou **Outro**. Exemplos: batismo, casamento, graduação académica, serviço militar, exposição relevante, etc.
- **Local**: Elemento tipicamente geográfico ou arquitetónico, de interesse. Um local pode ser algo tão diverso como um país, uma cidade, uma freguesia, um campo, um arruamento, ou mesmo um edifício (igreja, etc.). Para outras situações, cf. Doc ou Casa.
- **Org**: Pessoa coletiva, instituição, etc.
- **Pessoa**: Pessoa humana (homem ou mulher), com:
  - nome completo, género sexual, pai e/ou mãe, cônjuge,

  podendo referir:

  - alcunhas, nomes alternativos, nomes frequentemente gralhados,
  - nomes de família materna e/ou paterna,
  - datas de nascimento, óbito, batismo, casamento,
  - local de nascimento,
  - casas onde viveu,
  - eventos em que participou,
  - documentos onde é mencionado,
  - graus de parentesco de familiares e afins,
  - pessoas com quem socializou.

Nota: Quanto mais rica for a informação acerca de uma pessoa, mais facilmente poderá ser identificada em textos, fotos e outros média.

- **Profissão**: Profissão de uma pessoa.

Destacados II:

- **conhece**: Inspirado em foaf:knows, relaciona indivíduos que se conheçam. O significado é amplo, mas implica alguma reciprocidade na relação.
- **temCasa**: Casa de residência, passada ou presente, duma pessoa.
- **temCasaNasc**: Casa onde nasceu uma pessoa.
- **temLocalNasc**: Local de nascimento de uma pessoa.
- **tipoEvento**: Tipo de evento, com o objetivo de separar cronologias por esta tipificação.
- **tipoDoc**: Enumeração "áudio", "artigo", "carta", "certidão", "imagem", "vídeo", "outro".
- **tipoLocal**: "edifício", "arruamento", "lugar", "freguesia", "região", "país" – conjunto a definir.
- **temProfissão**: Relaciona uma pessoa com a sua profissão.

Anos e datas:

- **ano:** Ano de referência por excelência. Alternativo a data, quando não se souber. Formato: Ano 'aaaa', em que 'a' são dígitos 0-9 e indicam o ano. Exemplos: '2021', '2000'.
- **anoFinal:** Ano em que termina dado evento ou período, ou ano do óbito, ano de demolição, etc. Alternativo ao termo dataFinal.
- **data**: Data de referência por excelência, num indivíduo. Pode ser uma data de início, como fundação, ou construção, a complementar, eventualmente, com dataFinal. Para pessoas, ver dataNasc (e eventualmente dataÓbito).
  Formato: ISO 'aaaa-mm-dd', em que 'a' são dígitos 0-9 e indicam o ano; 'm' dígitos do mês, 'd' dígitos do dia. Exemplo: "2021-01-31".
  Como alternativa, ver os atributos ano e anoFinal.
- **dataFinal**: Data ISO, de fim de dado evento ou período, ou data de demolição de um imóvel. Para pessoas, ver dataÓbito. Como alternativa, usar anoFinal.

Datas especializadas:

- **dataBatismo**: Data de batismo religioso (cristão) de uma pessoa.
- **dataCasamento**: Data de casamento de duas pessoas.
- **dataNasc**: Data de nascimento de uma pessoa.
- **dataÓbito**: Data de falecimento de uma pessoa.

Endereço postal, pela ordem lógica:

- **arruamento**: Designação de um arruamento – nome de rua, avenida, etc., sem números de porta nem andar. Pode conter abreviaturas e numerais. Exemplos: "Avenida da Liberdade"; "Rua Dom Pedro V", "Rua do Regimento de Infantaria 8

Braga".

- **numPorta**: Número de porta (nº de polícia) dum endereço. Pode conter letras.
- **andar**: Andar de um endereço. Pode conter abreviaturas e letras. Exemplos: "r/c", "R/C", "Rés-do-chão", "1º", "1º andar", "1ºB", "1º Esqº", "1º Esquerdo", etc.
- **localidade**: nome duma localidade.
- **códigoPostal**: Código postal propriamente dito, sem designação da localidade.
- **morada** (alternativo aos anteriores): Um endereço postal, ou pelo menos uma localização. Exemplo: Rua Dom Pedro V, 99, 8º Esquerdo Traseiras – São Victor – 4710-374 BRAGA. Poderá também indicar o país.
- **país**: Nome de um país.

Nomes e âmbito pessoal:

- **alcunha**: Alcunha de uma pessoa. Uma pessoa pode ter várias alcunhas.
- **apelidoMaterno**: Apelido(s) herdado(s) da mãe.
- **apelidoPaterno**: Apelido(s) herdado(s) do pai.
- **cônjuge**: Nome completo do cônjuge de uma pessoa.
- **designação**: Designação de um item, humanamente inteligível, descrição curta, mas relevante.
- **nome**: Nome de pessoa, preferencialmente completo.
- **nomeAlternativo**: Nome duma pessoa, alternativo ao de registo, ou como variante do nome completo, que ocorre com frequência. Pode haver vários... Não confundir com alcunha.
- **nomeComGralhas**: Nome de pessoa, gralhado, que é comum ou expectável.
- **nomeMãe**: Nome da mãe.
- **nomePai**: Nome do pai.

Texto, média, anotação:

- **html**: Conteúdo html para apresentação numa secção a isso destinada.
- **imagem**: Nome ou caminho de um ficheiro de imagem no sistema de ficheiros.
    - **versoImg**: Nome ou caminho de um ficheiro de imagem no sistema de ficheiros, tipicamente contendo o verso de uma imagem.
- **json**: Dados em formato JSON, provavelmente uma lista.
- **nota**: Anotação textual específica.
- **obs**: Observações livres. Tipicamente, serão para consumo interno e preferencialmente não impressas.
- **path**: Caminho no sistema de ficheiros. Eventualmente, para um ficheiro que guarda uma digitalização.
- **resumo**: Resumo de um conteúdo ou perfil de um indivíduo, i.e., uma descrição mais pormenorizada do que um mero título, mas dificilmente uma transcrição.
- **sexo**: Género sexual de uma pessoa. "M" ou "F".
- **títuloDignitário**: Título atribuído à pessoa. Denominação honorífica. Designação ou qualificação de uma relação social, de uma função, de uma dignidade.
  Ex.: "Engenheiro", "Dr.", "Doutor".

- **transcrição**: Transcrição, supostamente integral, de um documento.
- **url**: Endereço web por excelência sobre determinado assunto.

4. Definir classes e hierarquia de classes

(Definir classes e hierarquia de classes – onde, a partir da lista de termos, se eliminam redundâncias e se definem as classes da ontologia (organizadas em hierarquia).)

As classes são: **Animal**, **Casa**, **Compilação**, **Doc**, **Evento**, **Local**, **Org**, **Pessoa**.

Nota sobre a classe **Casa**: eventualmente, poderia ser uma subclasse de Local, depois de bem definida esta; no entanto, remete-se para as precauções da Arquitetura de Software, que defende que a criação de subclasses aumenta sempre a complexidade das soluções (e consequente manutenção), sendo preferível começar com duas classes de algum modo relacionadas, em vez de estabelecer à partida uma herança que é "para sempre".)

Durante o desenvolvimento do projeto, novas classes e hierarquias foram surgindo, donde se destaca a seguinte hierarquia (subclasses apresentadas sob indentação):

Doc

Carta (Transcrição e anotação de correspondência.)

HistóriaDeFamília (História de família.)

HistóriaDeInfância (Histórias de infância, por exemplo sobre crianças e jovens.)

Img (Imagens, com desejável descrição. Podem corresponder a fotografias com existência em papel.)

Org

CasaDeFotografia (Casa de fotografia. Autor ou detentor de direitos enquanto pessoa coletiva. Para autores individuais, usar Pessoa.)

5. Definir propriedades das classes

(Definir propriedades das classes, i.e., atributos e relações, específicos das classes.)

Nota: Atributos (*data properties*) e relações (*object properties*) não mencionados terão domínio e/ou contradomínio aberto (i.e., não especificado, extrínseco).

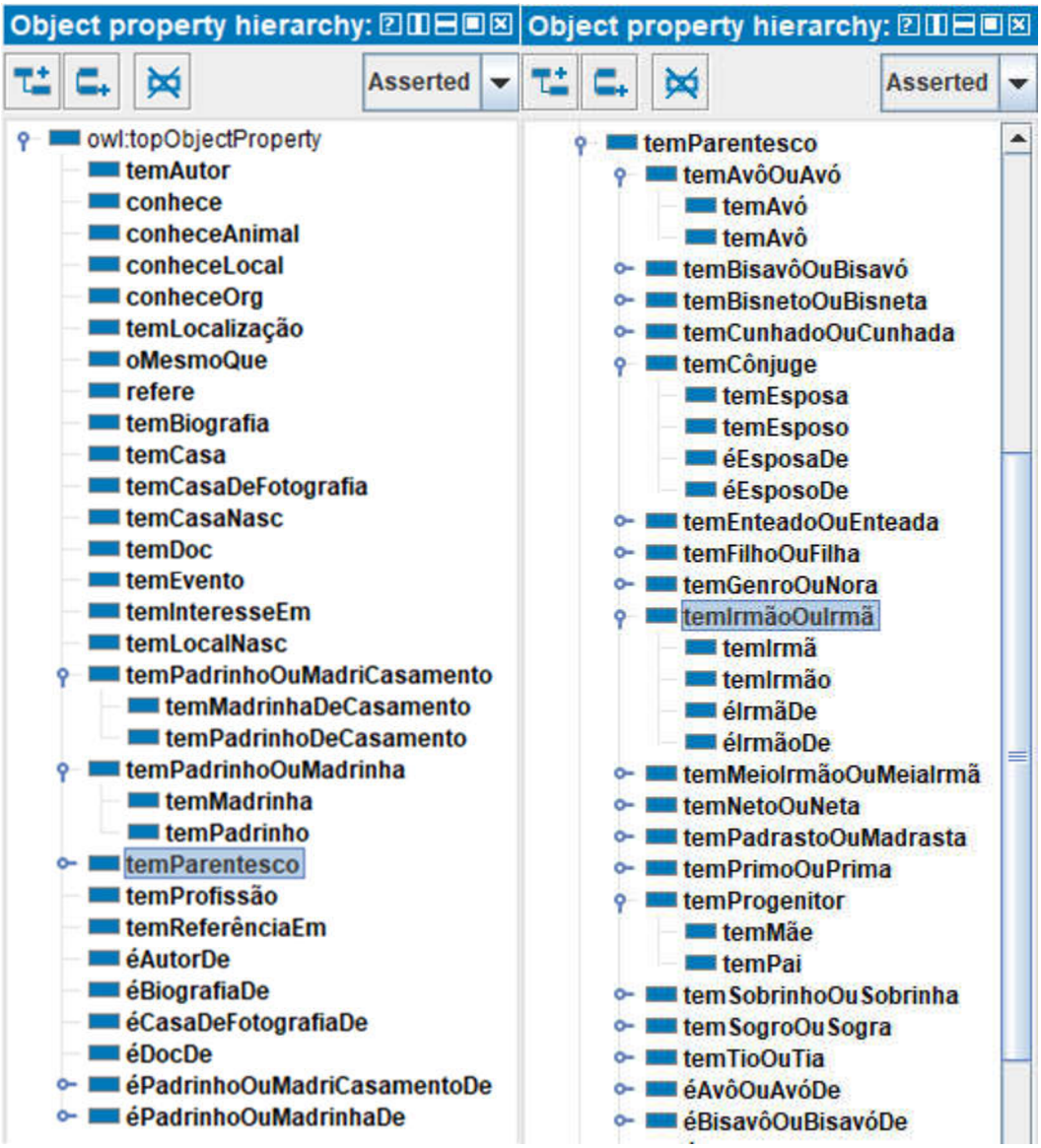


Figura 5 – Vislumbre da hierarquia de relações do ANB

Animal

Atributos: designação, dataNasc, dataÓbito, ano, anoFinal, imagem, resumo, obs, etc.

Relações: conheceAnimal.

Casa

Atributos: designação, data, dataFinal, ano, anoFinal, imagem, resumo, obs, etc.

Relações: temCasa, temCasaNasc.

CasaDeFotografia

Atributos: designação, data, dataFinal, ano, anoFinal, imagem, resumo, obs, etc.

Relações: éCasaDeFotografiaDe, temCasaDeFotografia.

Compilação

Atributos: designação, data, ano, imagem, resumo, obs, etc.

Relações: n/a.

Doc

Atributos: numClichê, tipoDoc, designação, data, dataFinal, ano, anoFinal, imagem, resumo, obs, etc.

Relações: éAutorDe, refere, temBiografia, temReferênciaEm, temDoc.

Evento

Atributos: tipoEvento, designação, data, dataFinal, ano, anoFinal, imagem, resumo, obs, etc.

Relações: temEvento

Local

Atributos: tipoLocal, designação, data ou ano, imagem, localidade, resumo, obs, etc.

Relações: temLocalização, conheceLocal, temLocalNasc.

Exemplo:

- ID: "cp_P_4700"
- códigoPostal: "P-4700"
- designação: "Braga, Portugal"
- localidade: "Braga, centro da cidade"

Org

Organização

Atributos: marca, volumeDeNegócios, url, designação, data ou ano, imagem, resumo, obs, etc.

Relações: conheceOrg.

Pessoa

Atributos: alcunha, apelidoMaterno, apelidoPaterno, biografia, dataBatismo, dataCasamento, dataNasc, dataÓbito, imagem, localNasc, morada, nome, nomeAlternativo, nomeComGralhas, nomeCônjuge, nomeMãe, nomePai, obs, sexo, títuloDignitário, etc.

Relações[3]:

[3] São pelo menos 127 relações com domínio e/ou contradomínio em Pessoa, daí indicar-se apenas algumas mais

- temAvó, temAvô, temCônjuge, temEsposa, temEsposo, …, temFilha, temFilho, …, temIrmã, temIrmão, …, éAvóDe, éAvôDe, …, éTioOuTiaDe;
- oMesmoQue, …, temBiografia, …, temProfissão, temReferênciaEm, …, éPadrinhoOuMadrinhaDe.

Dos atributos extrínsecos, sugere-se genericamente o uso de **data, dataFinal, imagem, obs, resumo, transcrição, url**, entre outros.

6. Definir restrições

(Definir restrições, como tipos de dados dos atributos, domínio e contradomínio de relações e outras restrições. O domínio, ou contradomínio, deve ser aberto/extrínseco ou associado a apenas uma classe – cf. secção Do domínio e contradomínio.)

Legenda: [A] → [B] => A é o domínio (opcional) e B o contradomínio (opcional).

Atributos

- **alcunha**: Pessoa → xsd:string.
- **apelidoMaterno**: Pessoa → xsd:string.
- **apelidoPaterno**: Pessoa → xsd:string.
- **biografia**: Pessoa → xsd:string.
- **data<…>**: → xsd:date. "aaaa" => ano; "aaaa-mm-dd" => data de calendário
- **dataBatismo**: Pessoa → xsd:date.
- **dataCasamento**: Pessoa → xsd:date.
- **dataNasc**: → xsd:date. A usar em Pessoa e Animal.
- **dataÓbito**: Pessoa U Animal → xsd:date.
- **imagem**: → xsd:string.
- **json**: → xsd:string, em formato JSON.
- **localNasc**: Pessoa → xsd:string.
- **marca**: Org → xsd:string.
- **nome**: → xsd:string.
- **nomeCônjuge**: Pessoa → xsd:string.
- **nomeMãe**: Pessoa → xsd:string.
- **nomePai**: Pessoa → xsd:string.
- **obs**: → xsd:string.
- **sexo**: → xsd:string. "M" (masculino) ou "F" (feminino).
- **tipoDoc**: → xsd:string. "áudio" | "artigo" | "carta" | "certidão" | "imagem" | "vídeo" | "outro".
- **tipoEvento**: Evento → xsd:string. Valores permitidos:
  - "F" (evento de âmbito familiar) – exemplo: casamento, divórcio;
  - "C" (evento de âmbito curricular) – exemplo: graduação académica, serviço

exemplificativas.

militar;
    - "H" (evento de âmbito histórico) – exemplo: "25 de abril".
- **tipoLocal**: → xsd:string. "arruamento" | "lugar" | "freguesia" | "país" | … conjunto a definir.
- **títuloDignitário**: Pessoa → xsd:string.
- **volumeDeNegócios**: rdf:type owl:FunctionalProperty; Org → xsd:decimal.

Relações sem parentesco

- **conhece**: Pessoa → Pessoa; Symmetric; Irreflexive
- **conheceAnimal**: Pessoa → Animal
- **conheceLocal**: Pessoa→ Local
- **conheceOrg**: Pessoa → Org
- **oMesmoQue**: → ; Irreflexive
- **refere**: Doc → Pessoa; inversa: **temReferênciaEm**
- **temAutor**: → Pessoa; inversa: **éAutorDe**
- **temBiografia**: Pessoa → Doc; inversa: **éBiografiaDe**
- **temCasa**: → Casa
- **temCasaDeFotografia**: Img →CasaDeFotografia; inversa: **éCasaDeFotografiaDe**
- **temCasaNasc**: Pessoa → Casa
- **temDoc**: → Doc; inversa: **éDocDe**
- **temEvento**: → Evento
- **temInteresseEm**: → ; Irreflexive
- **temLocalização**: → Local
- **temLocalNasc**: Pessoa → Local
- **temPadrinhoOuMadriCasamento**; inversa: **éPadrinhoOuMadriCasamentoDe**
    - **temMadrinhaDeCasamento**
    - **temPadrinhoDeCasamento**
- **temPadrinhoOuMadrinha**; inversa: **éPadrinhoOuMadrinhaDe**
    - **temMadrinha**
    - **temPadrinho**
- **temProfissão**: Pessoa → Profissão
- **temReferênciaEm**: Pessoa → Doc; inversa: **refere**

Relações de parentesco

(Apenas um excerto, remete-se mais uma vez para a ontologia:)

Acima das relações associadas a parentescos foi definida temParentesco, que alberga todo o tipo de relações de parentesco familiar entre pessoas, de sangue ou por afinidade. Não abriga relacionamentos como padrinho ou madrinha, de batismo ou casamento, estes definidos em paralelo.

Os relacionamentos foram divididos por sexo, e agrupados sob um outro, que não

distingue o sexo[4]. Como expectável, o domínio e contradomínio são herdados pelas subpropriedades, já as caraterísticas[5] não.

temParentesco: Pessoa → Pessoa; Symmetric, Irreflexive

- **temAvôOuAvó**: Asymmetric, Irreflexive; inversa: éAvôOuAvóDe
  - **temAvó**: Asymmetric, Irreflexive; inversa: éAvóDe
  - **temAvô**: Asymmetric, Irreflexive; inversa: éAvôDe
- **temCunhadoOuCunhada**: Symmetric, Irreflexive; inversa: éCunhadoOuCunhadaDe
  - **temCunhada**: Asymmetric, Irreflexive; inversa: éCunhadaDe
  - **temCunhado**: Asymmetric, Irreflexive; inversa: éCunhadoDe
- **temCônjuge**: Symmetric, Irreflexive
  - **temEsposa**: Asymmetric, Irreflexive; inversa: éEsposaDe
  - **temEsposo**: Asymmetric, Irreflexive; inversa: éEsposoDe
- **temIrmãoOuIrmã**: Symmetric, Irreflexive, Transitive; inversa: éIrmãoOuIrmãDe
  - **temIrmã**: Asymmetric, Irreflexive, Transitive; inversa: éIrmãDe
  - **temIrmão**: Asymmetric, Irreflexive, Transitive; inversa: éIrmãoDe
- **temProgenitor**: Asymmetric, Irreflexive; inversa: éProgenitorDe
  - **temMãe**: Asymmetric, Functional, Irreflexive; inversa: éMãeDe
  - **temPai**: Asymmetric, Functional, Irreflexive; inversa: éPaiDe

...

Das caraterísticas das propriedades

Caraterísticas, como Irreflexive, são importantes na definição duma relação. Neste projeto, não houve tempo para as validar no código, exceto no que diz respeito a Functional, como descrito na secção owl:FunctionalProperty. Por experiência própria, refira-se que certas caraterísticas, isoladas ou combinadas, poderão levar o *reasoner* a falhar, pelo que poderá ser preferencial evitar defini-las na ontologia e estabelecer validações à medida nas aplicações...

Do domínio e contradomínio

Em relação a *rdfs:range* e *rdfs:domain*, sejam referentes a atributos ou relacionamentos, deve realçar-se que as classes, por essa via associadas, **não passam a ser donas do termo**. Isto deduz-se do seguinte:

---

[4] Exemplo: temAvôOuAvó. A nomenclatura com "Ou" acaba por facilitar, em geral, a deteção e processamento de graus de parentesco, uma vez que não se perde a designação dos graus de parentesco por baixo, ainda que se tenha seguido um caminho diferente com temCônjuge e temProgenitor.

[5] *property characteristics*, no Inglês, que no OWL são mais concretamente escritas como owl:AsymmetricProperty, owl:FunctionalProperty, owl:InverseFunctionalProperty, owl:IrreflexiveProperty, owl:ReflexiveProperty, owl:SymmetricProperty, owl:TransitiveProperty.

Do RDF Schema 1.1[6] retiramos, resumidamente, que:

- «*3.1 (…)* ***rdfs:range*** *is an instance of rdf:Property that is used to state that* ***the values of a property are instances of one or more classes***.»
- «*3.2 (…)* ***rdfs:domain*** *is an instance of rdf:Property that is used to state that* ***any resource that has a given property is an instance of one or more classes***.»

Da documentação do Protégé, em relação às mesmas propriedades, vem que:

- *Domains (Intersection) - The selected property has each class expression listed in this section in its domain. If a given property has a given class in its domain* ***this means that any individual that has a value for the property*** *(i.e. is the subject of a relation along the property),* ***will be inferred to be an instance of that domain class****. For example, if John hasParent Mary and Person is listed in the domain of hasParent, then John will be inferred to be an instance of Person.* ***Note that domain does not mean that, "the class Person has hasParent as a property".***
- *Ranges (Intersection) - The selected property has each class expression listed in this section in its range. If a given property has a given class in its range this means that* ***any individual that is the value for the property*** *(i.e. is the object of a relation along the property),* ***will be inferred to be an instance of that range class****. For example, if John hasParent Mary and Person is listed in the range of hasParent, then Mary will be inferred to be an instance of Person.* ***Note that range does not mean that, "the class Person is a value for the hasParent property".***

Posto isto, faça-se o seguinte teste no Protégé (exemplo na sintaxe Turtle):

```
:temDoc a owl:ObjectProperty ;
        rdfs:domain :Doc , :Pessoa ;
        rdfs:range :Doc .
:doc__001 a :Doc , owl:NamedIndividual .
:pedro a :Pessoa , owl:NamedIndividual ;
        :temDoc :doc__001 .
```

⇨ **inferência** (:pedro a :Pessoa , :Doc .) – ao correr o *reasoner* HermiT (1.3.8.413).

Ou seja, o *reasoner* infere, devido ao triplo (:pedro :temDoc :doc__001.), que *pedro* é também uma instância da classe *Doc*, devido a *temDoc* incluir o domínio *Doc*. E não adianta estabelecer (:Doc owl:disjointWith :Pessoa .): o *reasoner* declara que a ontologia está inconsistente.

Assim, pareceu-me preferível que, se for necessário ter atributos ou relacionamentos associados a uma classe, seja com domínio ou contradomínio, então que sejam exclusivos dessa classe, ou seja, tenham no máximo uma classe em rdfs:domain ou rdfs:range. Por esta razão, de modo a evitar a complexidade decorrente da proliferação de atributos, dependentes da classe, com pequenas variantes no nome, optei

[6] cf. https://www.w3.org/TR/rdf-schema#ch_range e https://www.w3.org/TR/rdf-schema#ch_domain

por deixar diversos *Domain* e/ou *Range* em aberto, na ontologia final.

7. Criar indivíduos (instâncias)

(Criar indivíduos (instâncias das classes).)

Os indivíduos serão inseridos por via da aplicação, seja interativamente ou em lote, via opção DSL (e possivelmente outras, em acomodação a padrões como GEDCOM). Nesta secção, limito-me a dar exemplos de indivíduos da classe Pessoa:

- **ID**: João_M_P_M_Grenhas_d1968;

  alternativas a analisar para ID:

  - nome inteiro: João_Manuel_Pós_Mina_Grenhas_d1968;
  - sem data de nascimento: João_M_P_M_Grenhas_d0000;
  - 1º, 2º e último nome inteiros: Joao_Manuel_P_M_Grenhas_d1968;

  nome "João Manuel Pós de Mina Grenhas"; **sexo** "M"; **dataNasc** "1968-01-19";

  apelidoPaterno "Grenhas";

  apelidoMaterno "Pós de Mina";

  nomeAlternativo "João Manuel Pós-de-Mina Grenhas" (com hífens), "João Manuel Grenhas" (nome incompleto);

  outros nomes "alternativos" poderão ser omitidos por serem uma função:

  - calculados do nome completo, compondo primeiro e último vocábulo, ou dois primeiros e último: "João Grenhas", "João Manuel Grenhas";
  - com iniciais (com ou sem ponto, separadas ou não por espaço): "JGrenhas", "JMGrenhas", "J G", "JG", "J.G.", "J M G", etc.;

  nomeComGralhas "Pós-de-Minas" (donde se deriva "Pós-Minas"), "Pós Minas"; "Granhas", "Grelhas", "Grainhas";

  outras gralhas poderão ser omitidas por serem uma função que acrescente ou retire o plural, ou troque masculino com feminino: Grenhas => Grenha, Grenho, Grenhos; Granhas => Granha, Granho, Granhos.

- ID: jose_joao_antunes_guimaraes_dias_almeida_d0000;

  nome "José João Antunes Guimarães Dias Almeida"; **alcunha** "JJ";

  apelidoMaterno "Antunes Guimarães";

  apelidoPaterno "Dias Almeida".

5. Arquitetura e Desenho da Aplicação

Este capítulo pretende delinear a arquitetura geral da aplicação e adiantar algumas opções de modelação, estas numa perspetiva de dicas que poderão ser úteis num projeto que não se pretende rígido, mas que constitua uma busca de modelos e padrões úteis no domínio em causa.

5.1. Diagrama UML da fronteira do sistema

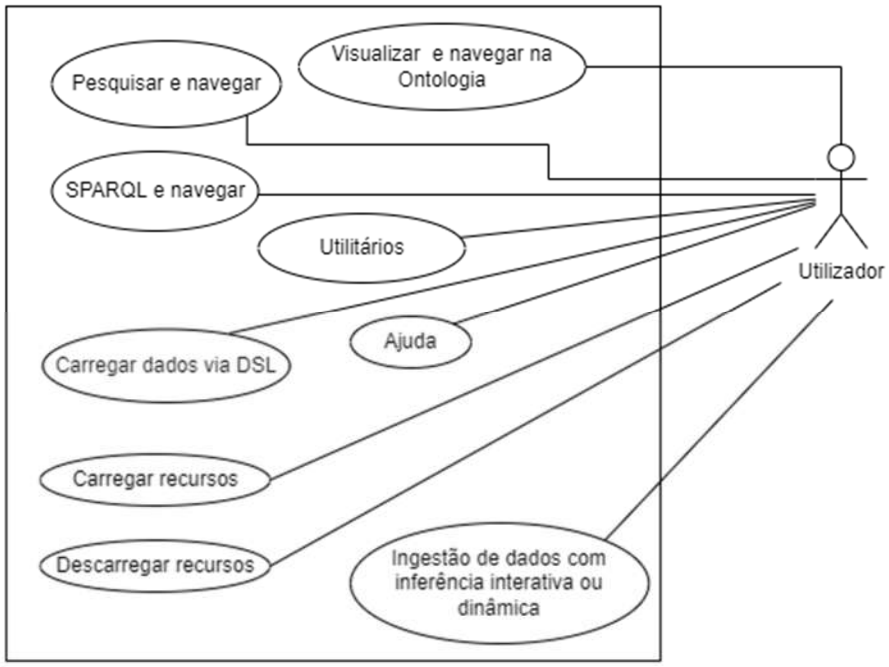
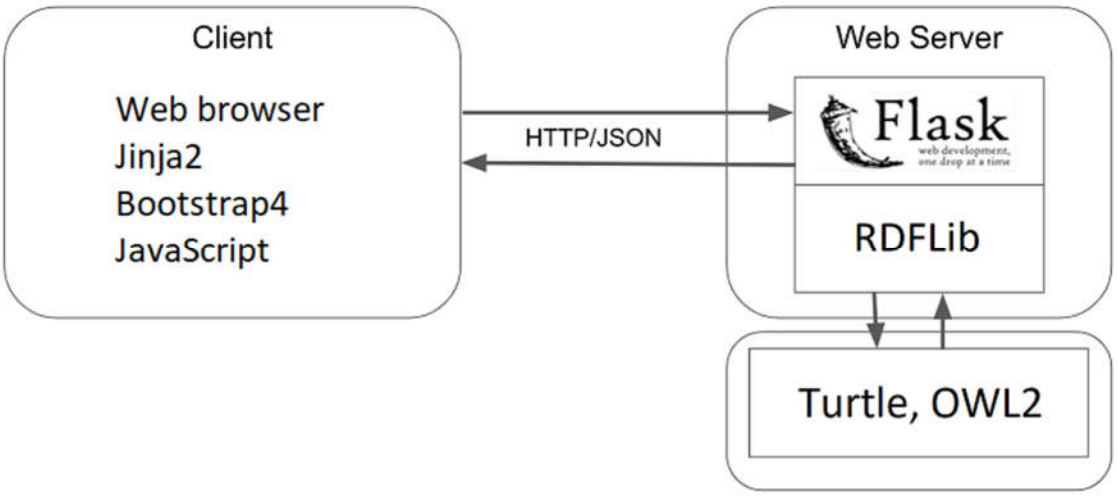


Figura 6 – Diagrama UML de casos de uso da fronteira do sistema

5.2. Opções arquitetónicas



*"Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the **presentation tier, or user interface**; the **application tier**, where data is processed; and the **data tier**, where the data associated with the application is stored and managed."*[7]

ANB será uma aplicação web, com arquitetura a obedecer ao modelo cliente-servidor, nas seguintes camadas sobrepostas:

- **Apresentação** (i.e. *Presentation*): navegador web;
  - Como: as opções de solução vieram a incidir sobre as tecnologias Jinja2 (via Flask/Python), HTML5 (padrão), Bootstrap4, CSS3 (padrão), e JavaScript (padrão);
- **Aplicação** (i.e. *Logic/Business*): servidor REST;
  - Como: *framework* Flask/Python; comunicação com a camada de dados: módulo RDFLib, SPARQL inclusive;
- **Persistência** de dados (i.e. *Data persistence*): Dados guardados em ficheiro de texto.
  - Como: sintaxe Turtle, ontologia OWL2, tecnologia RDFLib Graph para carregar e aceder aos dados num grafo em memória.

5.3. Interface com o Utilizador

Neste capítulo, fornecem-se linhas de orientação para desenhar a UI, na esperança de que, apesar da ambiguidade, ajudem a encontrar caminhos na folha em branco da criatividade.

5.3.1 Aparência e interação

Estabelece-se a preferência por um visual limpo e simples, mas cuidado e uniforme.

Na janela de entrada, mostra-se uma luz solar, que é fonte de vida tal como a família e até a sociedade florescente, quando livre e baseadas em valores humanos, de modo a induzir positivismo no inerente esforço de descoberta de conhecimento.

[7] In https://www.ibm.com/cloud/learn/three-tier-architecture

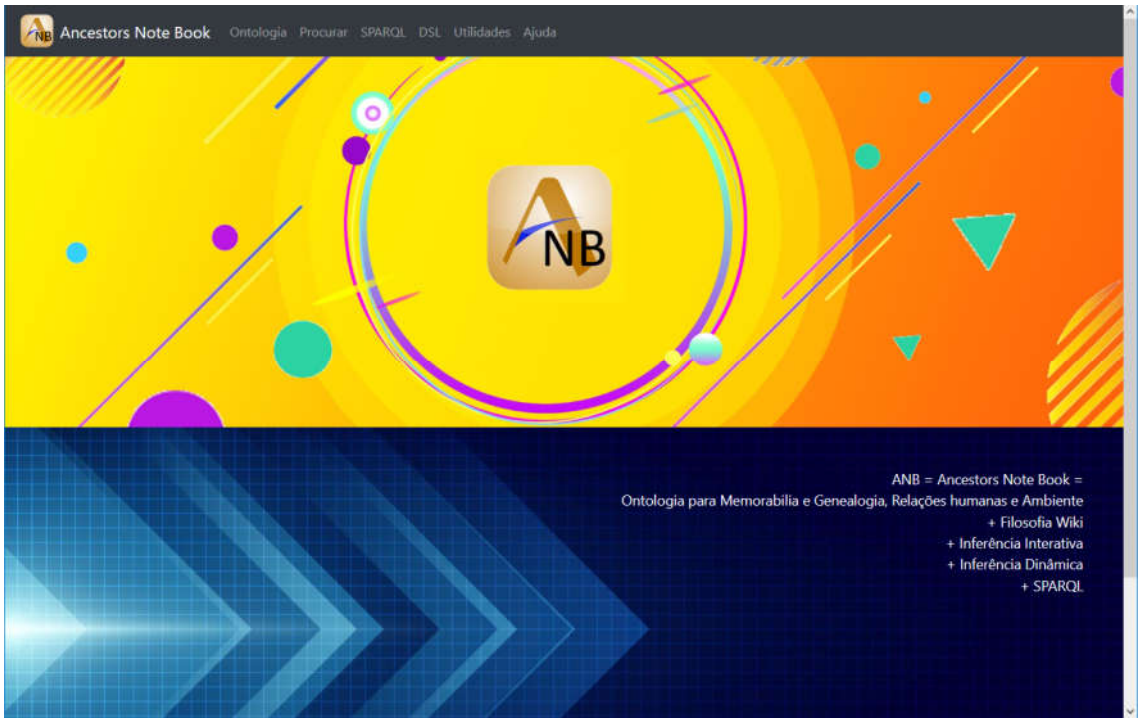


Figura 7 – Janela de entrada da aplicação

No resto da aplicação, tons um pouco mais frios, especialmente nas grelhas da navegação SPARQL, para poupar a vista e energia elétrica.

Perspetiva-se dois ambientes de utilização internos à aplicação, sem descartar a possibilidade de navegar para qualquer endereço internet externo: **navegação SPARQL** e **interface *prettyprint***.

Navegação SPARQL

Neste modo, as hiperligações mantêm-se dentro do mesmo ambiente, exceto a hiperligação "(-i-)". Pretende-se disponibilizar uma área de texto, onde surgem e podem ser editadas as questões SPARQL, sendo os resultados apresentados numa tabela, por baixo.

Um elemento importante será permitir a passagem ao modo *prettyprint*. Assim, onde elegível, disponibilizar-se-á hiperligações denotadas por "**(-i-)**", que ao serem seguidas mudam para esse modo.

Apresentação, por ordem:

- uma área de edição de questões;
- botões de execução;
- zona de resultados.

Visual aprimorado (prettyprint)

Este modo tem por filosofia subjacente uma apresentação mais aprimorada da informação de cada indivíduo (em comparação com outras opções), e as hiperligações

disponibilizadas levam, regra geral, ao mesmo tipo de ambiente, sem descartar a possibilidade de navegar para qualquer endereço internet externo à aplicação, caso em que se poderá abrir a nova página num novo separador do navegador. Enquanto se mantiver no mesmo ambiente, reutiliza-se o separador atual.

No *prettyprint* de indivíduos, campos de conteúdo mais extenso, como transcrições, deverão ter lugar no fim da página. No canto superior direito, poderá ser mostrada uma fotografia associada ao indivíduo, e eventualmente uma segunda com o verso ou outra perspetiva. Na ausência destes recursos, apresentar-se-á um ícone que deverá desejavelmente variar conforme a classe do indivíduo, devendo existir um ícone genérico para evitar o espaço em branco. Os diferentes ícones devem partilhar uma linha estética relativamente uniforme. Onde omisso, pretende-se procurar inspiração nas aplicações do tipo *wiki*, embora haja exemplos já bastante sofisticados e que colocam a fasquiam muito alta para um projeto como o atual.

5.4. Especificação

5.4.1 ANB DSL

Como hipótese de sintaxe, confrontar a DSL do artigo [AM2019].

O sistema poderá aceitar uma simbologia simplificada para marcar dados, seja para os reconhecer ao carregar ou mesmo para os apresentar ao utilizador:

- * (asterisco) => ano/data nascimento. Exemplos: *1968; *1968-01-01; *01/01/1968
- + (mais) => ano/data de óbito
- & ("e" comercial, *ampersand*) => ano/data de casamento
- ~ (til) => ano/data de batismo religioso
- U => ano/data de formação académica, mesmo que não seja universitária
- aaaa - aaaa => período de vida por anos (de ano1 a ano2)
- aaaa-mm-dd - aaaa-mm-dd => período de vida por datas (de data1 até data2)

5.4.2 Inferência

Dos Requisitos

Deve ser definido um conjunto de regras para tentar inferir relações da ontologia a partir de texto avulso, por exemplo em artigos de jornal, cartas, etc.

Exemplo com o grau de parentesco **irmão**:

- Selecionar relação **éIrmãoDe** ao detetar as seguintes situações:
  "é irmão", "irmão de", "é irmão de", "e o seu irmão".
- Facilitadores de escrita convencionados, com seleção da relação **éIrmãoOuIrmãDe**, (superclasse de éIrmãDe e éIrmãoDe), ao detetar as seguintes situações:

"é irm@", "irm@ de", "é irm@ de".

- Selecionar relação **temIrmã**/**temIrmão** ao detetar situações de enumeração:
  "irmão José, irmã Maria" => temIrmão José; temIrmã Maria.

5.4.3 ANB NameTK

Para a app web, pretende-se um módulo de pesquisa de nomes na ontologia, do qual será interessante efetuar um *spinoff*, publicável via PyPI e utilizável em qualquer outro módulo Python. Tipicamente, este módulo fará pesquisa sobre uma lista de *strings* ou sobre um ficheiro.

A pesquisa deverá contornar acentos e maiúsculas, além de grafias antigas de nomes de pessoas. Os nomes de entidades poderão apresentar também alguns desafios, em variantes como "Ld.ª", "Lda.", "L.da", etc. A pesquisa de moradas também, apresentando variantes em abreviaturas dos tipos de arruamento, e números de porta e andar, incluindo a string "N.º".

5.5. Estruturas

Notas: Esta secção foi escrita com o projeto já dado como fechado.

5.5.1 Organização das fontes

A aplicação tem as suas fontes próprias organizadas em três camadas. Em termos de HTML e CSS, é residual a camada de apresentação incluída na lógica. Depois, há ainda a configuração de *templating*, que contempla nomes de atributos e relações, e a sua distribuição por áreas do templating.

Os módulos de terceiros assumidos são: 'flask', 'markdown', 'requests', 'rdflib'. Apesar do Flask não obrigar, optou-se por uma organização em pastas[8] que traduzisse as camadas arquitetónicas, donde resultou o seguinte (subpastas em relação à pasta de instalação; excluem-se módulos de testes unitários):

Apresentação

templates/ – ficheiros de ".jinja2":

- **layout** – Modelo de base a usar em praticamente todos os restantes modelos, por expansão. Carrega as *frameworks* CSS (Bootstrap), jQuery, e o menu principal, além de eventuais controlos como o botão "go top".
- **index** – Página de entrada na app. Expande layout.
- **ajuda** – Página de ajuda da app. Expande layout.

[8] As pastas não podem ter o mesmo nome de algum módulo Python já instalado no sistema operativo.

- **dsl** – Modelo da área DSL. Expande layout.
- **searchEngine** – Modelo da área Procurar. Expande layout.
- **sparql** – Modelo da área SPARQL. Expande layout.
- **utils** – Modelo da área Utilidades. Expande layout.

- **indivTp** – Modelo de apresentação universal das páginas prettyprint de indivíduos. Expande layout.
- **resourceShow** – Modelo de apresentação para imagens e outros, em ponto grande. Expande layout.
- **indivTpIn** – Modelo universal para gerar formulários de dados de ingestão de indivíduos. Expande layout.
- **trecho_*** – Modelos auxiliares a incluir diretamente noutros modelos.

static/ – pasta de recursos públicos da aplicação:

- **images**/ – imagens, como ico e png.
- **javascripts**/ – ficheiros JavaScript carregados pelos templates Jinja2.
- **stylesheets**/ – ficheiros css e auxiliares.

Intermédia

LibOnto/ – módulos próprios de livraria, de ontologia, sobre RDFLib:

- **myNavCel** – Cálculo de hiperligações de navegação para células de tabelas.

Lógica

app.py – módulo de entrada na aplicação; distribui os pedidos pelos roteadores.

env.py – configurações de ambiente.

***.py** – Outros módulos da aplicação (ficheiros .py):

- **geneaGraphSearch** – Módulo de pesquisa e inferência de genealogia.
- **indInference** – Camada de interface sobre a inferência de indivíduos.
- **inference_autotst** – Testes automáticos do módulo de inferência.
- **myNameGraphSearch** – Módulo de pesquisa de nomes, ID, moradas, sobre o grafo.
- **myNames** – Biblioteca de tratamento de nomes de pessoas e empresas: normalização e pesquisa em listas e strings. Foco na grafia portuguesa.
- **myTpFormat** – Módulo de formatação para modelos de visualização aprimorada ("prettyprint templates").
- **SPARQL_rdflib_custom** – Exemplo de "*custom evaluation function*" para o SPARQL do RDFLib.

Lib/ – módulos próprios de livraria, genéricos:

- **jg_base_iter** – Biblioteca básica sobre iteráveis, incluindo strings.
- **myHtmlPlate** – Operadores para construir elementos HTML.

- **myLists** – Módulo auxiliar sobre listas.
- **myMdFormat** – Formatação de texto markdown para apresentação HTML.
- **myStrings** – Biblioteca sobre strings.

LibOnto/ – módulos próprios de livraria, de ontologia, sobre RDFLib:

- **myIdFormat** – Módulo de formatação de ID de ontologia e URI.
- **myRdflib_Engine** – Motor sobre o módulo rdflib para coleção de informação com tratamento para apresentação html.
- **myRdflib_Graph** – Biblioteca de camada sobre rdflib.Graph e os seus metadados.
- **myRdflib_Plate** – Biblioteca de camada sobre o módulo rdflib.
- **mySparql_Nav** – Biblioteca para gerar SPARQL no contexto da navegação na aplicação.
- **mySparql_Plate** – Biblioteca que usa ou retorna SPARQL para situações diversas.
- **mySparql_Regex** – Funções que usam RegEx sobre queries SPARQL.

rout_app/ – roteadores, i.e., módulos que dão resposta a pedidos REST:

- **dsl** – Tratamento das rotas associadas a /dsl.
- **sparql** – Tratamento das rotas associadas a /sparqlExe.
- **searchEng** – Tratamento das rotas associadas aos motores de busca.
- **searchEng_rdflib** – Satisfação de pedidos de rotas das procuras associadas aos motores de busca, na situação de satisfação via módulo rdflib.
- **searchEng_SPARQL** – Satisfação de pedidos de rotas das procuras associadas aos motores de busca, na situação de satisfação via SPARQL.
- **utils** – Tratamento das rotas associadas a /utils.

- **indivTp** – Tratamento das rotas associadas ao template genérico para *prettyprint* de indivíduos.
- **indivTpIn** – Ingestão interativa de dados de indivíduos (inserção, alteração, ...). Roteador da inferência interativa.
- **pessoa_criar** – Criação de indivíduos da classe Pessoa. Criação automática de relações de parentesco.
- **resourceShow** – Router para fornecer e apresentar ficheiros privados.
- **upload** – Tratamento das rotas associadas a *uploads*.

- **docTp** – Router associado ao modelo da ontoclasse Doc.
- **eventoTp** – Router associado ao modelo da ontoclasse Evento.
- **localTp** – Router associado ao modelo da ontoclasse Local.
- **oclasseTp** – Tratamento das rotas associadas a prettyprint de indivíduos owl:Class.
- **ontoTp** – Tratamento das rotas associadas à apresentação da estrutura da ontologia.
- **orgTp** – Router associado ao modelo da ontoclasse Org.
- **pessoaTp** – Router associado ao modelo da ontoclasse Pessoa.

- **sparqlQTp** – Router associado ao modelo da ontoclasse SparqlQuery.

downloadsFolder/ – pasta de ficheiros temporários de *download*.

uploadsFolder/ – pasta de ficheiros temporários de *upload*.

Persistência

ANB.ttl – ficheiro de persistência da ontologia, na sintaxe Turtle. (O nome é configurável em env.py.)

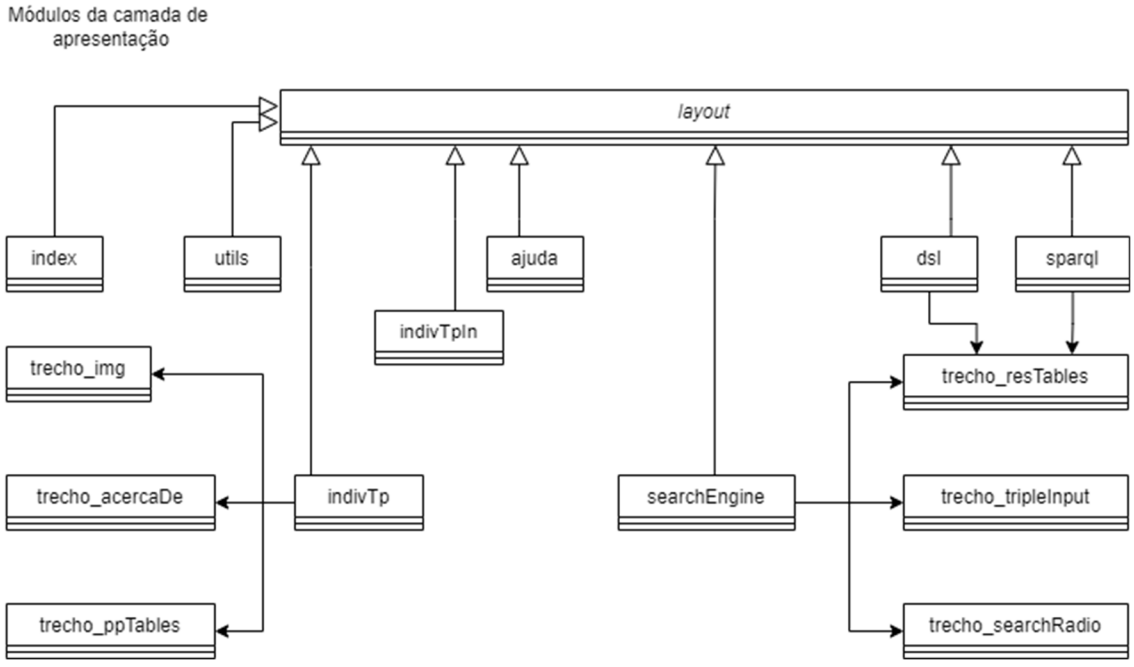
ontoResources/ – ficheiros (imagens, documentos, etc.) carregados pela aplicação, tipicamente referenciados pelo grafo, e organizados nas seguintes pastas em função da extensão do ficheiro:

- **Aud**/ – áudio: mid, midi, mp3, wav.
- **Docs**/ – ficheiros não enquadrados nas restantes pastas, nomeadamente txt, md.
- **Imgs**/ – ficheiros de imagens: gif, jpg, jpeg, png.
- **PDF**/ – ficheiros pdf.
- **Vid**/ – ficheiros de vídeo: mp4, ogg, webm.

5.5.2 Diagramas UML

Apresentação

Os módulos de apresentação organizam-se conforme o seguinte diagrama de classes UML.



Já quanto às restantes camadas, a sintaxe do Python facilita bastante conhecer todas as dependências, plasmadas no cabeçalho de cada módulo.

6. Resultados e Discussão

Este capítulo apresenta, por secção, alguns resultados que se destacaram, ao mesmo tempo que procura elaborar a sua análise crítica.

Da execução do projeto resultou, globalmente, a ontologia AncestorsNB (i.e., ANB, i.e., Ancestors Note Book) e uma aplicação web, homónima, que carrega e permite gerir a referida ontologia, segundo a arquitetura descrita no capítulo respetivo.

6.1. Como iniciar

Sem pretender ser um manual de utilização, esta secção surge da necessidade de abordar tópicos básicos de utilização da aplicação (por relevarem opções filosóficas e padrões), sem prejuízo da consulta à opção Ajuda, na aplicação, aliás transcrita no Anexo VI.

A aplicação espera encontrar, na raiz da pasta de instalação, o ficheiro de ontologia, ANB.ttl, na sintaxe Turtle. O nome é configurável em env.py, logo no início, e depois seguem-se diversas configurações de ambiente.

As áreas Procurar e SPARQL permitem pesquisas diversas. SPARQL permite todo o tipo de questões SPARQL, sejam query ou update – questões malformadas resultarão numa *pyparsing.ParseException*.

Realça-se a apresentação dos resultados, de pesquisas e navegação SPARQL, sob a forma de **tabelas com hiperligações**, onde elegível, que usam sistematicamente rotas da aplicação para navegar dentro da mesma área de interface, produzindo SPARQL, mais ou menos silenciosamente, para obter os dados. Além de proporcionar a facilidade de navegar e consultar informação, torna-se viável editar a qualquer momento a questão SPARQL automaticamente gerada neste contexto (já que é sempre mostrada na zona de edição). Nas referidas tabelas, destaca-se a hiperligação **(-i-)**, lateral aos ID e com a mesma funcionalidade do botão de aparência semelhante na barra de botões de submissão – i.e., acesso direto à página de visualização aprimorada (*prettyprint)* de um ID.

A navegação sistemática por hiperligações via SPARQL foi das coisas mais trabalhosas, dada alguma diversidade de símbolos, entre literais e URI. Assim, todos os URI da ontologia são apresentados como uma hiperligação de navegação, que por meio de SPARQL procura ver onde o URI é usado, à esquerda e à direita dos triplos. URI externos à ontologia, tiveram de ser protegidos nas rotas e inseridos no SPARQL entre carateres menor e maior – exemplo, "<http://example.com>").

Alternativamente, qualquer atributo do tipo string aceitará um endereço HTTP, que reconhecerá como hiperligação. Tirando este, os Literal (ex.: nomes de pessoas, numéricos e outros) não terão hiperligações. Com esta simplificação obteve-se, penso, uma interface mais limpa e mais intuitiva. Esta foi uma opção tomada a meio do caminho, por se ter verificado que as hiperligações nos Literal conduziam a resultados pouco interessantes, porque, se por

um lado poderia ajudar ao clicar no nome do pai/mãe/cônjuge, por outro introduz, curiosamente, alguma complexidade, visto ser necessário evitar que a hiperligação seja demasiado grande e contenha certos carateres, como *newline* e "?", que quebrariam rotas, e outros carateres poderiam exigir *url encoding/decoding*, dificultando desnecessariamente a programação e a apresentação.

Adicionalmente, estamos interessados em que:

1. literais do tipo URL (http://example.com) saltem diretamente para o destino (o que introduziria uma diferença de interface, já que as hiperligações em literais não teriam todas o mesmo comportamento),
2. literais muito grandes sejam apresentados como colapsáveis, para uma aparência mais agradável das tabelas, que por serem responsivas reagem à largura das colunas.

6.2. Do código fonte

Procurou-se escrever código com elevada adaptibilidade evolutiva, a prever evoluções futuras do projeto. Tipicamente, isto leva a estruturar por módulos, a descobrir padrões e *templating*, documentação, e muito mais. Procurou-se desenvolver um sistema que não seja demasiado dependente da ontologia, para que possa ser adaptado a outras ontologias.

Foi nesta atitude que, como sempre devia ser, se tentou recolher inspiração em atributos de qualidade como os que são listados no Anexo I.

Houve algumas preocupações quanto à arquitetura do código (**cf. secção 5.5 Estruturas**):

- Separação em pastas segundo as camadas da arquitetura;
- Separação por módulos;
- Documentação de parâmetros, variáveis e blocos de código;
- Funções pequenas e com poucos parâmetros, sempre que possível;
- Identificadores camelCase, snake_case e combinações, sempre com foco no significado e legibilidade; alterna-se entre o Português e o Inglês, o que tira alguma uniformidade, embora resulte naturalmente de um certo bilinguismo de que sofro, admito…;
- Prefixação de nomes de variáveis e parâmetros segundo o seu tipo, prática comum na comunidade Xbase, mas que considero de uso generalizável e muito útil: cf. Anexo IV e README_desenv.md (que acompanha a instalação da aplicação);
- Testes unitários: Os módulos terminados em "_tst" contêm testes unitários.
- Testes automáticos: Não foram sistemáticos. Mas, especialmente, foi concebido um módulo, inference_autotst.py, para realização de testes automáticos à inferência de ID, por esta se ter revelado muito sensível a pequenas modificações ao código. Estes testes, refira-se, são dependentes dos dados no grafo (para cada caso, testa-se o resultado esperado). O módulo não é parte integrante da aplicação, mas, ainda assim, necessita do contexto normal da aplicação, que além de ser Flask tem um estado, o

que obrigou a "descobrir" as seguintes linhas de código para que pudesse correr separadamente:

```
app = Flask(__name__)
with app.app_context():
    inference_autotst().
```

Como correr: dentro do editor de texto, botão "Run Python File", ou na linha de comandos, com «Python inference_autotst.py».

Finalmente, logo que oportuno seria interessante correr um analisador de métricas de código.

6.3. Motores de busca

Um motor de busca, ou pesquisa, é extremamente importante na obtenção rápida de informação cuja forma pode não ser bem conhecida. É verdade que o SPARQL é poderoso, mas muito técnico para uso direto por um utilizador leigo, e, quando se pretende contornar acentos, por exemplo, começa a tornar-se muito verboso e trabalhoso.

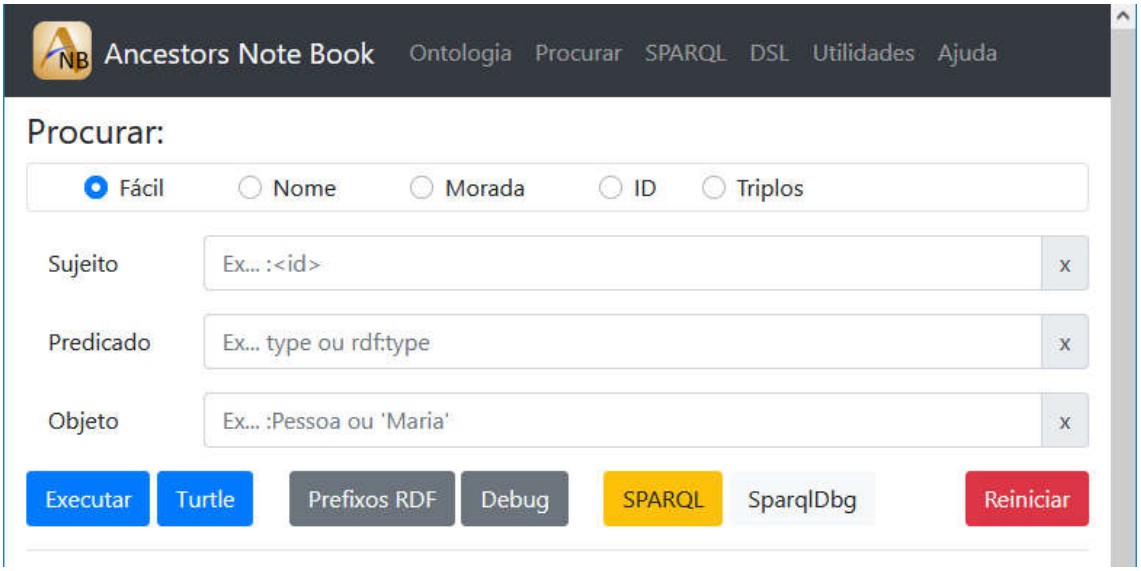


Figura 8 – Motores de busca da opção Procurar

Assim, foram sendo desenhados e implementados vários motores de busca, alguns dos quais algo especializados, que são disponibilizados na opção Procurar:

- **Fácil**: Permite, de modo bastante abrangente, a pesquisa por sujeito, predicado e/ou objeto, com determinadas facilidades, especialmente em termos de acentos e maiúsculas/minúsculas, em nomes. Prevê a pesquisa posicionada genealogicamente, por exemplo procurar Predicado = "mae", "mãe" ou "temMãe", "éMãeDe", sendo nestes dois últimos casos *case sensitive* para ajudar a distinguir casos como "avô" e "avó". Para uma pesquisa de triplos mais orientada à sintaxe Turtle, usar a procura *Triplos*.

- **Nome**: Procura orientada a nome, designação, alcunhas, nome com gralhas, nome em grafia antiga, entre outros. Por exemplo, procurar "Ruy" equivale a procurar "Rui", ou "rui", e vice-versa.
- **Morada**: Procura orientada a moradas. Como, alternativamente a "morada", existem atributos desagregados para "arruamento", "numPorta", "andar", "localidade", "códPostal", quando encontrados serão mostrados nesta pesquisa pela ordem lógica (o que também sucede nas páginas prettyprint de indivíduos).
- **ID**: Procura orientada a ID, em particular de Pessoa, que pode incluir a data de nascimento. Permite pesquisar anos de nascimento entre parêntesis, com os últimos dois dígitos opcionais, que serão adaptados para pesquisar nos ID de Pessoa. Neste âmbito, seria útil permitir ainda intervalos (99xx – 99xx).
  Exemplos: 1) Fulano de Tal (18xx), encontrando pessoas com esse nome que tenham no ID uma data ou ano de nascimento em toda a gama do século 19. 2) Maria da Atouguia (196x), neste caso na gama da década de 60.
- **Triplos**: Também permite a pesquisa por sujeito, predicado e/ou objeto, mas exige mais rigor na introdução de dados, como modo de restringir a pesquisa. Aspas e maiúsculas importam em literais.
  Exemplos: 1) Predicado = :alcunha; Objeto = "Xpto".
  2) Objeto = ObjectProperty.
  Mais tarde, esta pesquisa poderá eventualmente ser incluída na pesquisa Fácil, de modo a reduzir a profusão de opções. Foi sobrevivendo por ter sido a primeira a ser implementada…

Os diversos motores, conforme o caso, aceitam pesquisa de subcadeias de carateres e elementos Turtle. Em certos casos, pode ser mostrada uma segunda tabela, por baixo da primeira, com outros resultados (via iniciais da cadeia procurada), menos assertivos, mas que poderão ajudar quando a pesquisa mais exigente não estiver a ajudar.

Os três botões à esquerda permitem executar a questão e apresentam os resultados na zona inferior da janela, numa tabela com hiperligações:

- **Executar**: apresenta os resultados numa filosofia *prettyprint*, mais legível.
- **Turtle**: apresenta os resultados na sintaxe Turtle.
- **Prefixos RDF:** mostra os IRI completos, com prefixos RDF (começados por "http…", ex.: " http://www.semanticweb.org/grenhasMEI/AncestorsNB#Pessoa").

Sobre os restantes botões na base do formulário:

- **Debug**: executa a pesquisa e mostra os resultados numa janela de texto, apresentando a lista de objetos retornada pelo RDFLib.
- **SPARQL**: muda para a área de pesquisa SPARQL, tentando gerar uma questão tão aproximada quando possível segundo os padrões do SPARQL.
- **Descarregar**: Quando mostrado, dá a opção de descarregar os triplos encontrados, na sintaxe Turtle.
- **Reiniciar**: Volta a entrar pela rota inicial da área atual.

6.4. SGBD Orientado a Grafos

A opção pela orientação, da aplicação, a guardar a informação em grafos (por contraponto a outros SGBD, particularmente SGBDR) foi motivada pelo facto de este paradigma:

- acrescentar **inovação**, a nível de base de conhecimento, e logo uma oportunidade de aprender e ganhar mais experiência neste domínio;
- estar nitidamente a emergir e a assistir-se a uma **crescente utilização**, talvez pelo motivo da sua adequação a certos problemas;
  ser especialmente adequado a carregar **ontologias** e respetivas instâncias (há quem defina que **`ontology + data = knowledge graph`** [9]).

A aplicação carrega o ficheiro ANB.ttl num grafo RDFLib, em memória, mas gravado para ANB.ttl sempre que ocorram modificações. Quando se verifica ser necessário aceder ao grafo, é testada a *timestamp* do ficheiro, e se tiver mudado volta a carregar-se em memória – isto faz-se não apenas se ANB.ttl for mais atual (possivelmente por gravação via processador de texto), mas também se for mais antigo, porque pode ter-se reposto uma cópia de segurança.

Embora não seja materialmente impossível implementar ontologias com SGBDR, a opção por um grafo, implícito no ficheiro ANB.ttl e explícito no objeto RDFLib em memória, é consensualmente a mais desejável para gerir ontologias, dada a teoria de grafos subjacente e a facilidade de efetuar queries SPARQL.

Com RDFLib 5 e Python 3.8, tive a perceção de ser bastante mais eficiente usar os métodos RDFLib orientados ao grafo, em vez de SPARQL. A experiência de utilização era em muitos casos evidentemente mais fluida com *queries* não SPARQL. Depois de atualizar para RDFLib 6 e Python 3.10, a diferença percecionada de eficiência parece mais dependente das queries, mas carecia de métricas mais rigorosas.

Dada a génese dos grafos, pode ser preciso fazer várias travessias e acessos a um grafo para obter um conjunto de informação específico, o que acaba por se refletir na eficiência. Então qual é a vantagem de um grafo? É a sua especial adequação a processos de inferência, que também foram explorados nesta dissertação, como descrito na secção de Inferência de ID.

Uma coisa que se revelou bastante trabalhosa foi a recorrente necessidade de conversão de URI, necessária entre os seguintes formatos:

- **objetos RDFLib** (ex.: «rdflib.term.URIRef('http://www.semanticweb.org/AncestorsNB#temMãe')»),
- **RDF** (ex.: 'http://www.semanticweb.org/AncestorsNB#temMãe'),
- **Turtle** (ex.: ':temMãe') – por vezes referido na literatura como exemplo de um formato abreviado,

[9] https://enterprise-knowledge.com/whats-the-difference-between-an-ontology-and-a-knowledge-graph/

- ***prettyprint*** (ex.: 'tem mãe', em grelhas de dados, ou ainda na página aprimorada, onde se procura (ID, rdfs:label), ex.: 'mãe^').

O formato *prettyprint* nunca será reversível, já os restantes estão em permanente correspondência e conversão, conforme as necessidades. Para mecanizar esta necessidade, criaram-se módulos de recolha de prefixos e *namespaces*, guardados em variáveis, complementados com configurações que indicam a chave de prefixo da nossa ontologia (tipicamente ":"), a lista de prefixos e *namespaces* padrão (como 'owl:', 'rdf:', 'rdfs:', 'xsd:', 'xml:'), etc.

Neste âmbito, procurei nomenclatura desambiguadora, especialmente ao nível da codificação:

- *abbreviated uri* – ex.: "rdf:type"
- *prefix key* (ou *prefix name)* – ex.: "rdf"
- *prefix* – ex.: "rdf:"
- *namespace* – "ex.: http://www.w3.org/1999/02/22-rdf-syntax-ns#"
- *rdf-uri* – ex.: "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"

6.4.1 SPARQL na aplicação

Além das travessias ao grafo RDFLib, via respetivas funções e métodos, usou-se amiúde o SPARQL, seja para *queries* mais complexas, seja para *queries* que neste paradigma resultam mais simples e legíveis, por contraponto a travessias múltiplas.

A passagem a um SGBD de grafos, como GraphDB, Virtuoso ou Neo4J, poderá ser desafiante. Não é que deixe de ser viável usar os métodos RDFLib, mas o subgrafo pretendido (mesmo que não seja todo o grafo) terá de ser carregado em cada episódio, pelo que a eficiência dependerá do tempo desse carregamento. A opção futura por um SGBD como os referidos terá de ser cautelosamente avaliada, com justificação apenas na quantidade de informação ou nalguma vantagem competitiva do SGBD em questão.

Por avaliar, ainda, ficou o uso de RDFLib sobre grafos externos, via HTTP, o que também se poderá aplicar a SGBD de grafos disponibilizados necessariamente como API.

Área SPARQL

A aplicação disponibiliza, ao utilizador, uma opção para inscrever questões SPARQL, sejam do tipo *query* ou *update*. Deste modo, o utilizador pode agir sobre o grafo com todo o poder do SPARQL, permitindo contornar alguma lacuna funcional da aplicação.

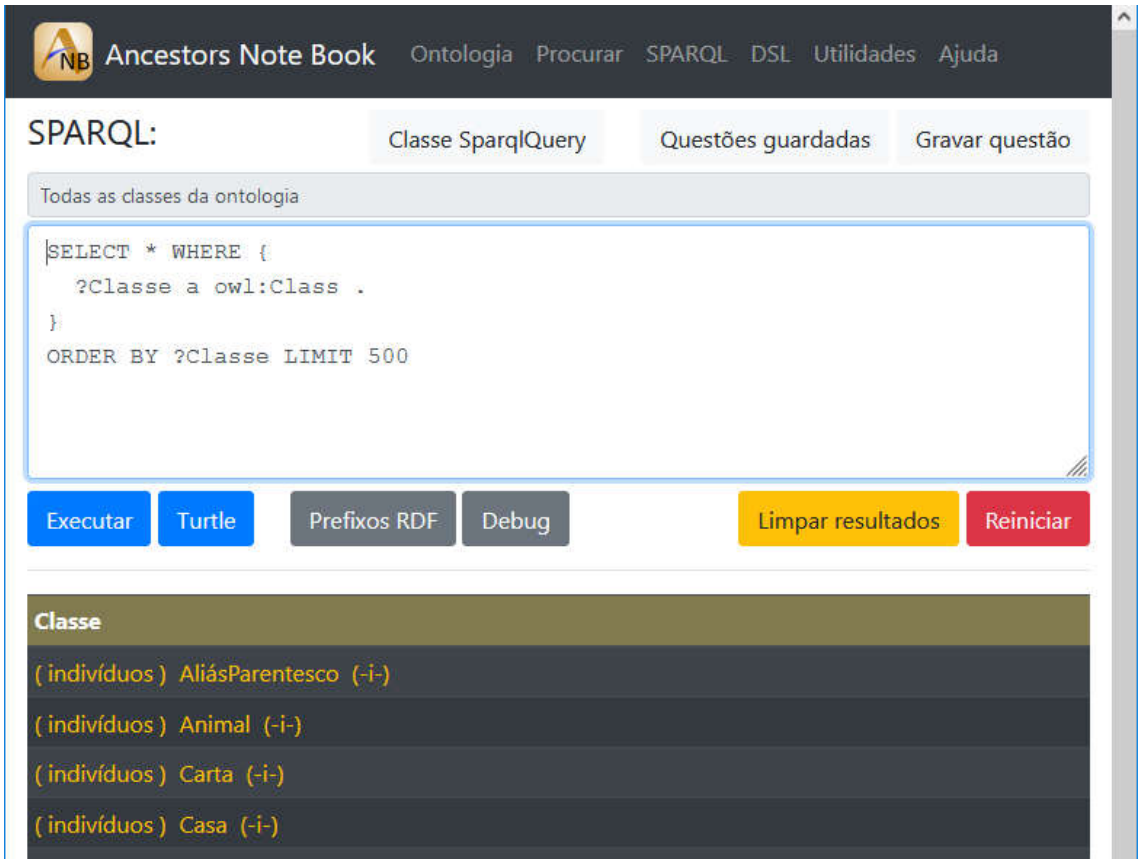


Figura 9 – Área acessível pela opção SPARQL

A questão de arranque tem tratamento especial de acesso aos indivíduos e classes da ontologia. É nessa zona que se pode editar a questão pretendida.

Nos botões de cima, dá-se a hipótese, da direita para a esquerda, de:

- **gravar a questão** (cf. botão homónimo), que pede uma descrição resumida;
- escolher um elemento da lista de **questões guardadas** (cf. botão homónimo), que espera pela seleção do utilizador, carrega a questão no editor e fica à espera de edição ou ordem para executar a questão;
- saltar para a área da **classe SparqlQuery**, com interesse no separador Indivíduos da classe, já que pode ser mais fácil deste modo identificar a questão e mandar logo carregá-la na área SPARQL; ao clicar no ID de um indivíduo específico também se dá a possibilidade de carregar a questão.

Os três botões à esquerda permitem executar a questão e apresentam os resultados na zona inferior da janela, numa tabela com hiperligações:

- **Executar**: apresenta os resultados numa filosofia *prettyprint*, mais legível.
- **Turtle**: apresenta os resultados na sintaxe Turtle.
- **Prefixos RDF:** mostra os IRI completos, com prefixos RDF (começados por "http…", ex.: " http://www.semanticweb.org/grenhasMEI/AncestorsNB#Pessoa").

Sobre os restantes botões na base do editor:

- **Debug**: executa a questão e mostra os resultados numa janela de texto, apresentando a lista de objetos retornada pelo RDFLib.
- **Limpar resultados**: apaga a tabela ou janela de apresentação inferior.
- **Reiniciar**: Volta a entrar pela rota inicial da área atual;
- **(-i-)**: Frequentemente, disponibiliza-se este botão, para acesso direto à página de visualização aprimorada (*prettyprint)* de um ID.

Personalização RDFLib do SPARQL

Sentiu-se necessidade de afinar as questões SPARQL automáticas, de modo a retirarem acentos em procuras de nomes (de pessoas, ruas, etc.), e (uma vez que o padrão SPARQL 1.1 não tem, ainda, funções com esse fim explícito) foi necessário elaborar sobre as funções básicas, o que revelou questões SPARQL um pouco extensas. Chegou-se a questões como a seguinte, para procurar o nome "joao" (já depois de normalizada a cadeia, porque o utilizador poderá procurar "joão" ou "João", etc.):

```
SELECT DISTINCT ?s ?p ?nome WHERE {
  ?s :nome|:designação ?nome .
  ?s ?p ?nome .
  BIND (replace(lcase(str(?nome)),'\\.',' ') as ?nome_low) .
  BIND (replace( replace( replace( replace( replace( replace( ?nome_low, 'ç','c'),
    'ã|á|â|à|ä','a'), 'é|ê|è','e'), 'í','i'), 'õ|ó|ô|ò|ö','o'), 'ú|ü','u') as
    ?nome_norm) .
  FILTER (regex(?nome_norm,"joao",'i'))
}
```

No primeiro BIND, converte-se um nome, elegível, para minúsculas, e retiram-se os pontos, sendo o resultado atribuído a uma variável de conveniência, `?nome_low`. No segundo BIND, procede-se então à retirada de diacríticos, explicitamente indicados, sendo o resultado atribuído a outra variável de conveniência, `?nome_norm`.

Mais tarde, explorou-se a possibilidade que o RDFLib oferece para personalizar o comportamento dos resultados no SPARQL, por exemplo para poder usar uma função da aplicação que retire os acentos e outras partículas.

Devo dizer que achei o caminho um pouco tortuoso, pelas seguintes razões:

- é fácil obter erros difíceis de perceber; para conseguir um ambiente estável, a personalização tende a tornar-se global a todas as queries, tornando-as potencialmente mais lentas;
- a função de personalização lida com objetos RDFLib complexos e que achei mal documentados;
- o exemplo na documentação do RDFLib não parecia ser de feição – cf. https://rdflib.readthedocs.io/en/stable/_modules/examples/custom_eval.html;
- os exemplos disponíveis nos fóruns são escassos – cf. «https://stackoverflow.com/questions/43976691/custom-sparql-functions-in-rdflib», que se revelou bastante útil depois de adaptações.

Apesar das dificuldades inerentes, personalizar o SPARQL significa poder usar nesse contexto todo o poder do lado servidor da aplicação (logo, do Python e de qualquer módulo disponível), o que pode fazer toda a diferença!

Deixo, portanto, no Anexo V (Funções personalizadas no SPARQL do RDFLib), o modo de implementar a personalização do SPARQL RDFLib.

6.5. Ontologia AncestorsNB

6.5.1 Contexto da ontologia

Procurou-se perseguir os objetivos do projeto em termos duma ontologia para Memorabilia e Genealogia, Relações humanas e Ambiente. Sobre este assunto já se discorreu na Contextualização, bem como na Arquitetura e Desenho da Ontologia.

Optou-se conscientemente por simplificar o ponto de partida. Nomeadamente, manteve-se vários relacionamentos com domínio, ou contradomínio, aberto/extrínseco ou associado a apenas uma classe – cf. secção Do domínio e contradomínio.

6.5.2 ANB.ttl

Em consequência da metodologia descrita no capítulo 4 (Arquitetura e Desenho da Ontologia), foi obtida uma ontologia inicial, desenhada sob Protégé e depois exportada, com axiomas, na sintaxe Turtle, obtendo-se um ficheiro de texto ANB.ttl que foi estabelecido para persistência da informação.

Como explicado no capítulo 1.5. (Opções e Restrições): Em termos de sintaxe da ontologia, estabeleceu-se como regra geral uma preferência de trabalhar com a sintaxe Turtle, por ser um padrão muito bem aceite e mais legível do que outros formatos.

A opção por um ficheiro de texto para persistência da informação será adequada enquanto a ontologia não assumir proporções que tornem o carregamento/gravação insuportável. Por outro lado, facilita a prototipagem rápida, porque:

- é muito fácil ajustar a ontologia manualmente, via processador de texto;
- é também viável editar a ontologia via Protégé ou outro editor de OWL2;
- é fácil efetuar cópias de segurança e reposição;
- é muito fácil ensaiar ontologias diferentes, por substituição do ficheiro e eventualmente alguma configuração adicional via env.py e templates.

À medida que o projeto decorria, diversas melhorias e ajustes foram introduzidos. A ontologia final é fornecida no entregável ANB.ttl.

6.5.3 Termos e Axiomas

Axiomas owl:propertyChainAxiom

É possível definir axiomas owl:propertyChainAxiom, a que tipicamente atendem os *reasoner*. A título de exemplo, o seguinte axioma permite inferir os avós do sexo feminino:

SuperPropertyOf(Chain):

temProgenitor o temMãe SubPropertyOf: temAvó

Ou, então, os tios do sexo masculino:

temProgenitor o temIrmão SubPropertyOf: temTio

Foram definidos alguns axiomas deste tipo na ontologia, a título de exemplo, mas evitou-se axiomas que, ao correr um *reasoner,* gerassem aberrações, do tipo "irmão de mim próprio" – cf. secção Problemas dos Axiomas x Reasoners. **Neste âmbito, optou-se por disponibilizar uma opção para inserir relacionamentos de parentesco**, que dispensam o uso de *reasoners* para esse efeito, visto estes não proporcionarem um modo de evitar inconsistências e até *bugs* (como erros de memória, ou até declarar inconsistência sem que o respetivo relatório diga porquê, já para não falar duma certa ineficiência).

No entanto, foi possível aplicar axiomas *hardcoded,* por via duma função codificada à medida, que permite verificar se dois indivíduos estão ligados por uma cadeia de dois relacionamentos, como por exemplo no âmbito de avós ou bisavós, maternos ou paternos. Isto foi possível definindo uma estrutura em forma de lista, em que o último elemento é o número de iterações do penúltimo termo, exemplo:

- ["temMãe", "temProgenitor", 1] => verificar avós maternos.
- ["temMãe", "temProgenitor", 2] => verificar bisavós maternos.
- ["temPai", "temProgenitor", 1] => verificar avós paternos.

Problemas dos Axiomas x Reasoners

A definição de axiomas de inferência, como para calcular graus de parentesco, traz alguns problemas relativamente bem conhecidos, e ainda não resolvidos pelos *reasoner* (os quais por sua vez costumam colocar a responsabilidade na especificação owl).

Refiro aqui algumas constatações a que cheguei:

- **A inferência aberrante "sou irmão de mim próprio".** Um axioma para inferir irmãos (cf. temIrmãoOuIrmã, e sub-relacionamentos temIrmão e temIrmã) pode definir-se como `SubPropertyChain: temProgenitor o éProgenitorDe`. Ou seja: para inferir irmãos, inferem-se todos os filhos/as dos mesmos pais de alguém. Esta inferência acaba por gerar triplos como João temIrmãOuIrmão João.
  Poder-se-ia pensar que a questão se resolvia facilmente definindo a relação como irreflexiva (***irreflexive***), porém verifica-se que o *reasoner* HermiT (1.3.8.413), sob Protégé, entra em "choque" (declara a ontologia inconsistente e poderá mesmo

bloquear; indica no *log* "*non simple property*"; isto acontece mesmo que não seja definido o axioma acima referido). A situação já foi constatada por outros autores: Cf. SSMJ2015, pág. 29 e seguintes, e tem sido justificada com a especificação OWL 2. Esta deficiência acaba por se estender a outros parentescos, como **tios**, que são genericamente os irmãos dos pais, pelo que se os pais forem tidos por irmãos deles próprios, então serão tios dos seus filhos (outra aberração).
Estas contingências resultantes de correr *reasoners*, em conjunção com axiomas problemáticos, obrigariam a afinar as *queries* SPARQL implicadas, de modo a filtrarem todos os casos indesejados…

- Outra questão é a **inferência de meios-irmãos**, que poderemos querer distinguir numa ontologia genealógica. Ou seja, os axiomas podem complicar-se bastante, nesta área.
- Uma dificuldade adicional, agora já na aplicação e em relação ao *reasoner* do SGBD a escolher, será a **remoção interativa**, eventualmente contrariada pelo *reasoner*. Exemplo: o utilizador toma a iniciativa de remover a entrada `João temIrmão João`, por ser indesejável, mas o *reasoner* volta a introduzi-la por ser dedutível dos axiomas…
- Adicionalmente, indivíduos (instâncias) provenientes de **origens inconsistentes** poderão também bloquear ou inviabilizar o uso de um *reasoner*.
- *Reasoner* HermiT (1.3.8.413), sob Protégé: por experiência pessoal, este *reasoner* revela-se muito implicativo com axiomas (cf. owl:propertyChainAxiom), já que não é preciso de mais do que algumas dezenas de triplos para gerar falha de memória (*OutOfMemoryError: Java heap space,* mesmo depois de aplicar a solução https://protegewiki.stanford.edu/wiki/ Setting_Heap_Size), e há situações em que declara a ontologia inconsistente sem conseguir explicar porquê. Remover todos os axiomas parece reconduzir ao regular funcionamento. Combinar axiomas com caraterísticas (*object property axiom*, na especificação do OWL), como *Symmetric*, também costuma redundar em fracasso.

Solução: Evitar os axiomas (`SubPropertyChain`) problemáticos e compensar a perda de inferência via *reasoner* por recurso a uma inferência implementada no código fonte. A inferência poderia ser acionada "a pedido" (por exemplo via questões SPARQL, função VrfAxioma, ou outros métodos…), que faça a ingestão dos triplos pretendidos (nomeadamente, no contexto desta dissertação, triplos relacionados com graus de parentesco). Estas queries terão de correr, automaticamente, a cada novo indivíduo inserido, ou a pedido.

Caraterísticas relevadas

Nas secções seguintes enumeram-se alguns termos ontológicos que receberam especial atenção, na ontologia e/ou na implementação.

owl:FunctionalProperty

Exemplo:

```
:dataNasc a owl:DatatypeProperty, owl:FunctionalProperty ;
```

```
rdfs:comment "Data de nascimento. Ver data." ;
rdfs:domain :Pessoa ;
rdfs:range xsd:date .
```

Neste exemplo da ontologia, define-se `dataNasc` como um atributo (cf. `owl:DatatypeProperty`) com domínio na classe `Pessoa` e tendo como valor uma `xsd:date`, uma data, mas diz-se algo mais: é uma owl:FunctionalProperty, que que significa que cada indivíduo só pode ter uma data de nascimento.

Nada de mais natural, mas foi preciso acomodar esta caraterística na aplicação, no âmbito da introdução de dados interativa e também via ficheiro, na área DSL, facilitando neste caso a fusão ou atualização da informação de indivíduos, porque, se não fosse feito nada, cada nova associação, sendo diferente, iria coexistir com a anterior.

rdfs:comment

Este predicado foi intensivamente usado, com o objetivo de documentar a ontologia e guia para o próprio utilizador, sendo mostrado em várias tabelas de resultados, como tabelas de atributos e relacionamentos de ontologia, e ainda como resumo em páginas *prettyprint* de predicados. Nos formulários, surge ainda ao pairar o rato sobre o *input*.

rdfs:range em atributos

Em atributos (owl:DatatypeProperty) houve uma certa tendência para opção por determinados contradomínios, que receberam atenção especial pela implementação:

- **xsd:string**: incontornável e o mais usado, serve inclusivamente para campos multilinha em formulários, caso particular em que seria útil um tipo específico, que orientasse a UI diretamente para uma *textarea* HTML. Na falta disso, foi preciso, além de proporcionar a configuração de formulários (nos módulos de *templating)*, ter o cuidado de verificar se existem mudanças de linha na string a apresentar em formulário, porque afeta o comportamento, nomeadamente numa alteração de dados, em que é necessário procurar o valor anterior para remover, de modo a evitar lixo, em suma;
- **xsd:date**: útil para criar inputs HTML, com atributo type = "date", que disponibiliza um calendário e valida datas; ainda assim, estes valores são gravados como string, no formato ISO aaaa-mm-dd (ano, mês, dia).
- **xsd:decimal**: determina um número decimal, com tratamento necessariamente diferente, uma vez que não pode ser envolto em aspas – ex.: `127.25`.

owl:sameAs

Defende-se algum cuidado na introdução de triplos com predicados owl:sameAs, visto não haver consenso na comunidade sobre o seu uso, a julgar pelo desacordo na comunidade, notado em sítios como StackOverflow. Neste sentido, há o receio de que possa haver *reasoners* que não deem o resultado esperado.

A este respeito, ver também a secção owl:sameAs versus owl:equivalentProperty.

De modo a evitar "areias movediças", foi decidido introduzir na ontologia o termo substituto **oMesmoQue**:

```
:oMesmoQue a owl:ObjectProperty ;
    rdfs:comment "Associa indivíduos que são o mesmo conceito. Relacionamento de
    cortesia para evitar as prováveis restrições associadas a owl:sameAs, mas com
    o mesmo significado conceptual. Poderia ter as restrições (rdfs:domain
    owl:Thing) e (rdfs:range owl:Thing), mas o sistema teria de ser adaptado a
    essa restrição." .
```

Mesmo assim, tentou atender-se a *sameAs* e *oMesmoQue*, em certos pontos do código, como no cálculo de graus de parentesco ou nos formulários de ingestão (para não impor as mesmas obrigações que ao alterar um indivíduo da mesma classe).

owl:sameAs versus owl:equivalentProperty

Durante o projeto, considerou-se que seria interessante implementar os conceitos associados a owl:equivalentProperty, mas chegou-se à conclusão que estava a produzir muita complexidade no código, além de lentidão. Talvez o uso de um *reasoner* possa colmatar/enriquecer o sistema, neste âmbito, mas sob cautela.

O significado dos termos em questão foi investigado, e feita uma breve análise a partir das especificações OWL. Então, comecemos por olhar para:

«OWL Web Ontology Language Reference – W3C Recommendation 10 February 2004

4.2.1 owl:equivalentProperty[10]

The owl:equivalentProperty construct can be used to state that two properties have the same property extension. Syntactically, owl:equivalentProperty is a built-in OWL property with rdf:Property as both its domain and range.

> *NOTE: Property equivalence is not the same as property equality. Equivalent properties have the same "values" (i.e., the same property extension), but may have different intentional meaning (i.e., denote different concepts). Property equality should be expressed with the* ***owl:sameAs*** *construct. As this requires that properties are treated as individuals, such axioms are only allowed in OWL Full.»*

Assim, podemos sintetizar:

- **owl:sameAs** denota igualdade de conceito, entre indivíduos, ou (=> OWL Full) classes ou propriedades. Este predicado assume importância especial ao combinar diferentes ontologias, em que os mesmos conceitos podem ter IRI diferentes, e podemos querer identificá-los como ontologicamente iguais.
- **owl:equivalentProperty** indica que as propriedades têm a mesma extensão (dão os mesmos resultados), mas podem não ser o mesmo conceito.

[10] https://www.w3.org/TR/owl-ref/#sameAs-def

Exemplos:

- **owl:sameAs**: Sob OWL Full, poderíamos estabelecer com razoabilidade que `:casadoCom owl:sameAs :éEsposoDe` . Ou seja, se um homem "é casado com alguém", pode afirmar-se que o homem "é esposo desse alguém". O conceito é o mesmo. :Lisboa owl:sameAs xpto:cidade_das_sete_colinas.
- **owl:equivalentProperty**: Agora suponhamos os predicados :temNomeMariaRita e :éAPessoaMaisIdosa. Suponhamos que na ontologia **só existe uma pessoa** em ambas as circunstâncias, deste modo:

  :Maria_Rita_d1922 :temNomeMariaRita true.

  :Maria_Rita_d1922 :éAPessoaMaisIdosa true.

  Como retornam o mesmo resultado, podemos afirmar, talvez um pouco abusivamente, que é verdade que

  :temNomeMariaRita owl:equivalentProperty :éAPessoaMaisIdosa .

  Mas obviamente não representam o mesmo conceito.

  Cf. https://en.wikipedia.org/wiki/Extensionality.

6.5.4 Página de estrutura da ontologia

Foi implementada uma página específica de visualização dos termos da ontologia, organizada em separadores colapsáveis, para os seguintes grupos: classes (do tipo owl:Class), atributos (do tipo owl:DatatypeProperty), relacionamentos de parentesco (do tipo owl:ObjectProperty que sejam, direta ou indiretamente, rdfs:subPropertyOf :temParentesco), relacionamentos sem parentesco (do tipo owl:ObjectProperty que não coincidam com os anteriores) e anotações (do tipo owl:AnnotationProperty).

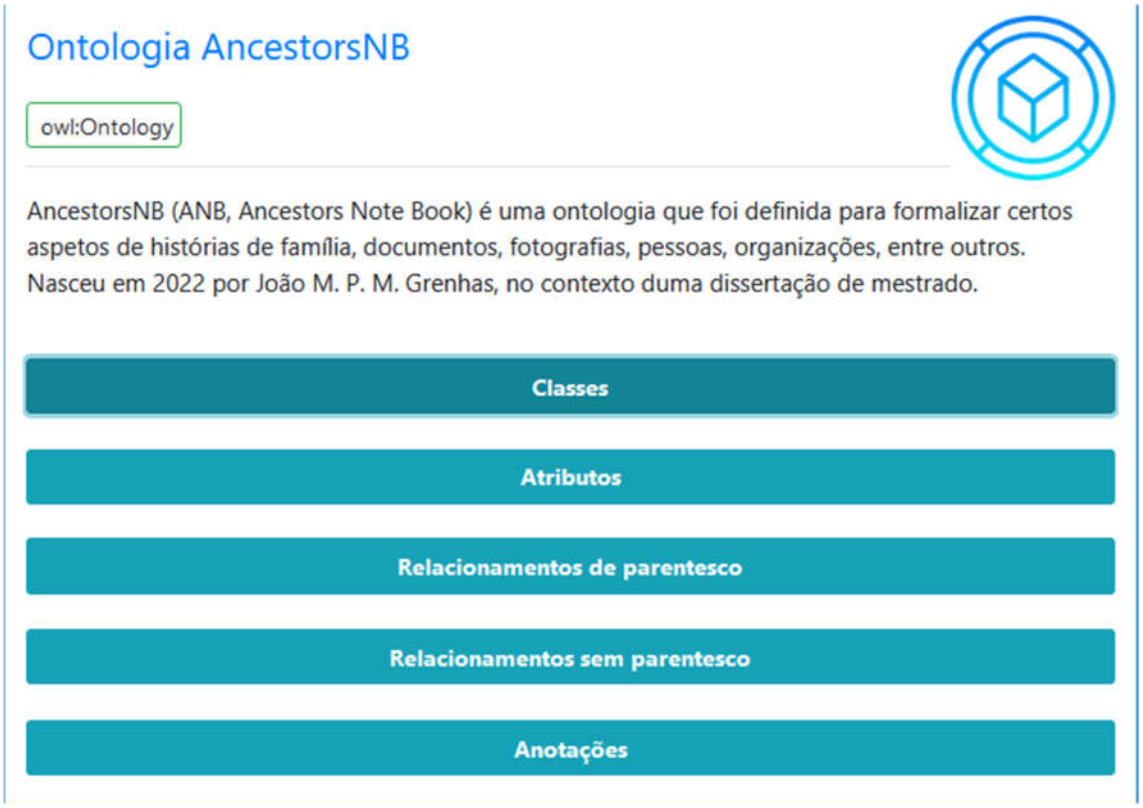


Figura 10 – Página da opção Ontologia

Ao colapsar um grupo, obtém-se uma lista de hiperligações, cada uma para a página *prettyprint* individualizada do termo.

A separação, visual e operacional, dos relacionamentos de parentesco de outros, pareceu-me natural e desejável, por serem muitos, por a mistura poder dificultar a escolha, e porque, naturalmente, quem procura um relacionamento de parentesco provavelmente não está interessado noutra coisa, e vice-versa.

No caso do separador Classes, seria aberto o conteúdo mostrado na figura da secção seguinte.

6.5.5 Das Classes e Subclasses

No momento de escrita desta tese, a árvore de classes anunciava-se com a seguinte hierarquia.

Figura 11 – Página da Ontologia, colapsável das Classes

O sistema, pela sua arquitetura, aceita modificação à ontologia, logo às classes.

É sabido, pelo menos no âmbito da disciplina de Arquiteturas de Software, que a existência de subclasses deve muitas vezes ser bem equacionada, porque aumenta a complexidade por via da herança, que, regra geral, *é para toda a vida*. Uma alternativa aconselhada costuma ser criar uma classe autónoma, com ligação à candidata a superclasse.

Já em ontologias, parece ser costume haver uma opção despreocupada pela adoção de subclasses, que é frequente fazerem todo o sentido. No caso ANB em contexto, faz todo o sentido, por exemplo, ter **Carta** (define cartas, missivas) e **Img** (define imagens, fotografias, etc.) como subclasses de **Doc** (define qualquer documento, genericamente, com possibilidade de tipificação para distinguir). No entanto, ao ter definições e configurações associadas a Doc, pretendemos imediatamente que qualquer subclasse herde essas configurações, se não for definida. Isso implicou implementações em que se começa a procurar pela classe em jogo e, se esta não estiver configurada, se vai subindo até encontrar uma superclasse configurada. Não se implementou a fusão de configurações: se uma subclasse redefinir configurações de apresentação ou ingestão, tem de indicar tudo o que pretende. Pode parecer mais redundante, mas é também mais flexível e fácil de abarcar, senão, ao analisar e configurar uma subclasse teria de se analisar todas as superclasses.

6.6. Página de utilidades

A página de utilidades foi implementada como página acessória, donde se realça:

- Carregar ficheiros de recursos (imagens, documentos, músicas, vídeos, etc.), para a pasta ontoResources, desde que não existam.
- Calcular e inserir automaticamente diversos relacionamentos de parentesco, em duas opções para:
    - Calcular e inserir indivíduos correspondentes a progenitores e cônjuge de cada indivíduo, e respetivos relacionamentos de parentesco com esse indivíduo de partida.
    - Calcular e inserir relacionamentos correspondentes a diversos outros graus de parentesco, em função dos relacionamentos de cada indivíduo com progenitores e cônjuge.

6.7. *Prettyprint*

O *prettyprint* existe a dois níveis no sistema: em menor grau, na **apresentação de tabelas, de resultados** dos motores de busca e da navegação SPARQL; em maior grau, na **apresentação da página de um indivíduo** ou termo da ontologia, e navegação subsequente ao clicar em hiperligações.

No primeiro caso, procura-se uma melhor legibilidade das tabelas: os ID de indivíduos da classe Pessoa são estripados de "_" e o ano de nascimento e sexo apresentados entre parêntesis; os atributos duma morada são apresentados na sua ordem lógica (arruamento, n.º de porta, n.º de andar, localidade e código postal). Os nomes dos termos são convertidos para minúsculas e metido um espaço antes de cada maiúscula – exemplo, ":temMãe" passa a "tem mãe". Esta filosofia também foi aplicada na apresentação referida no segundo caso, salvo ao navegar para páginas de termos de ontologia, apresentadas em Turtle, por se considerar que alterar muito a aparência do termo poderia levar a mais confusão do que vantagem.

No segundo caso, referente à apresentação de páginas de indivíduos e termos da ontologia: para a mostra de termos dá-se preferência a definições `rdfs:label`; adicionalmente, concebeu-se uma **apresentação universal** da página de um indivíduo ou termo, com *templating* opcional orientado à classe (e herdado por subclasses), como explicado nas seguintes secções.

6.7.1 Apresentação Jinja2

O *templating* de apresentação dos dados de um indivíduo assume várias facetas:

- existe um **modelo universal de apresentação**, indivTp.jinja2, por defeito alimentado pelo roteador indivTp_router (cf. indivTp.py), mas está prevista a opção por conceber modelos especializados, do Jinja2 e/ou do roteador;
- a nível do roteador, existe a possibilidade de, facilmente, construir um **objeto de configuração** para definir que atributos se pretende ver, e por que ordem, mesmo continuando a usar o modelo universal indivTp.jinja2 – esta é uma parte do

***templating* de classe**;

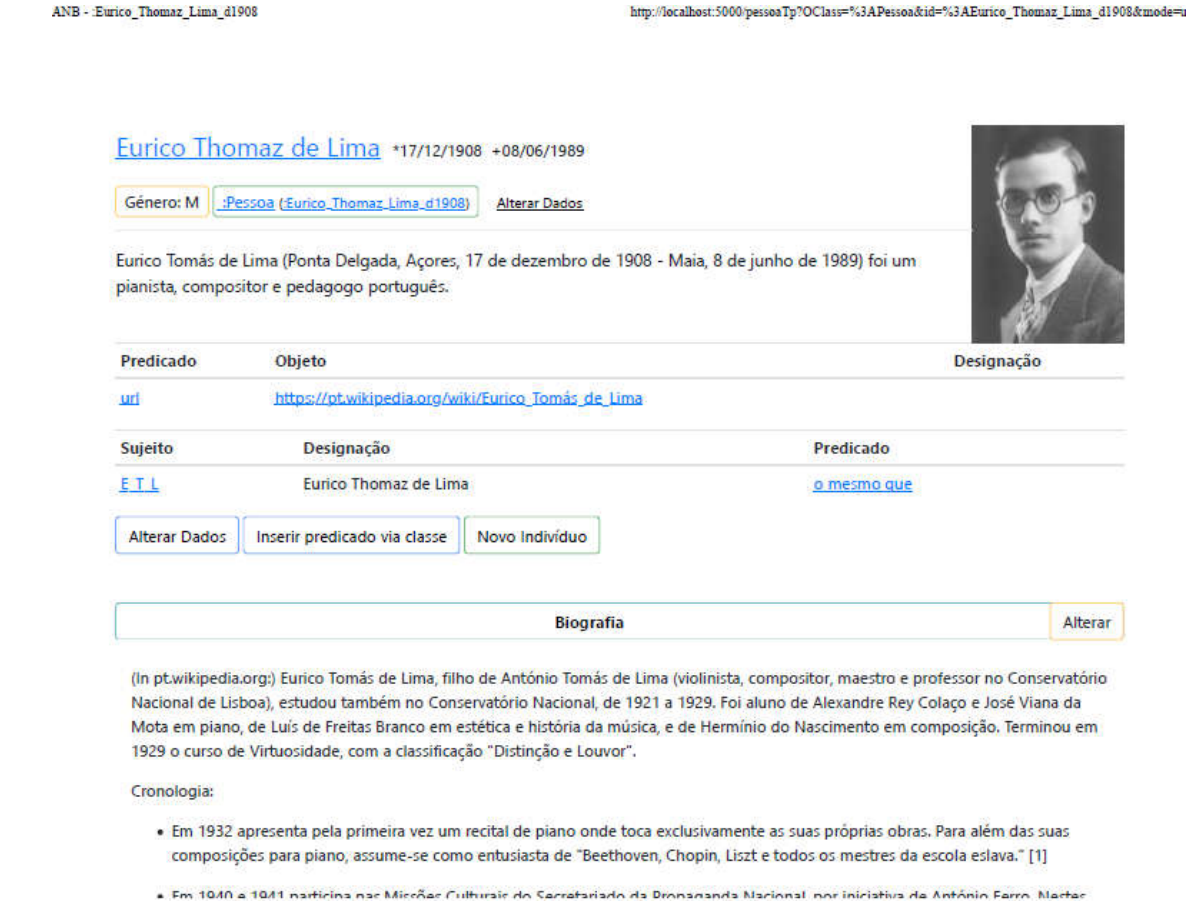
- no roteador anteriormente referido, pode ainda efetuar-se a configuração do formulário de ingestão, mas isso já é assunto para a secção Ingestão de Dados.

Em resumo: conseguiu criar-se um template Jinja2 que pretende, regra geral, visualizar a informação de dado indivíduo de qualquer classe, e ainda de *properties* (DatatypeProperty, ObjectProperty, etc.). Pode refinar-se a ordem de apresentação num roteador, e mesmo assim usar esse modelo genérico para apresentação, ou então construir um específico, o que não chegou a ser necessário.

Figura 12 – Página *prettyprint* de um indivíduo da classe Pessoa

Para **imprimir**, de momento baseamo-nos na função padrão respetiva do navegador web[11], que permite imprimir, inclusive para PDF se existir um controlador de impressora PDF instalado no sistema operativo. É verdade que aparecem os botões de controlo (Alterar Dados, etc.), pelo que seria útil haver, na mesma página *prettyprint,* um controlo para uma rota que mostrasse a página mais "limpa", sem os referidos controlos, que ao imprimir já não iriam sair. Em referência ao exemplo da figura acima, é óbvio que, depois de colapsar Biografia, ou outro que exista no contexto, ao imprimir também serão contemplados. Segue-se um exemplo:

[11] Cf. tecla Ctrl+P, ou menu do navegador.



Destaca-se seguidamente algumas áreas da apresentação padronizada da página de um indivíduo.

Em cima, à direita: ícone principal de um indivíduo (atributo **imagem** da ontologia). Pode haver uma segunda imagem alternativa (atributo **versoImg**, verso duma fotografia ou outra perspetiva, como por exemplo uma imagem de perfil ou o interior de um edifício). Pode clicar-se em cada imagem para ver em tamanho maior. Não sendo indicado, existem ícones padronizados, para mostrar na ausência duma imagem específica.

Figura 13 – Ícones padronizados para indivíduos sem imagem

Da esquerda para a direita, a figura anterior contempla ícones para:

- Termos de ontologia em geral, incluindo classes não personalizadas.
- Classe Pessoa.

- Classe Org.
- Classe Local.
- Classe Doc.
- Classe Evento.

Para a classe Doc, estão ainda previstos ícones para diversos tipos de documentos, quando existir ficheiro associado por via do termo *imagem*: **ao clicar** no ícone existe um tratamento específico da visualização, e descarregamento. Nas imagens (ficheiros de extensão gif, jpg, jpeg, png), mostra-se logo a imagem. Para os restantes, foram selecionados ícones apropriados.

Figura 14 – Ícones padronizados para ficheiros

Da esquerda para a direita, a figura anterior contempla ícones para PDF (ficheiros de extensão pdf), TXT (txt), documentos diversos (nomeadamente markdown), áudio (mid, midi, mp3, wav), vídeo (mp4, ogg, webm).

Voltando à página *prettyprint,* em cima, à esquerda: <Acerca de> (nome ou designação, senão ID); por baixo, subtipo (ex.: sexo ou tipo de documento); depois, o Tipo (a classe) e ID. Por baixo, o resumo.

Depois, informação sobre o indivíduo, em duas tabelas:

1. Atributos e relacionamentos do indivíduo com terceiros, i.e., triplos onde o indivíduo aparece à esquerda, tipicamente a informação mais pretendida.
2. Relacionamentos com o indivíduo, i.e., triplos onde o indivíduo aparece à direita.

Finalmente, botões de ação, e abaixo, quando elegível, separadores colapsáveis associados a campos multilinha, como biografia, transcrição, notas, etc… (conforme configurado no template da classe em causa). Nestes campos pode usar-se a inferência interativa.

6.7.2 *Templating* de classe

Apresentação estática

Como referido na secção anterior, é possível afinar a ordem de apresentação de termos, e até esconder os que se quiser, e continuar a usar o modelo universal de apresentação, indivTp.jinja2, uma vez que o router é que escolhe que modelo usar, ao fazer algo como

```
        return render_template('indivTp.jinja2', …
```

O que é então necessário para este efeito? Tomemos como exemplo o roteador localTp.py (modelação da UI da ontoclasse **Local**). Para afinar a UI, é preciso algo como a seguir:

```
1.  from env import SetFields_PpIns, SetPpTpInfo
2.  from flask import render_template
3.  from LibOnto.myRdflib_Graph import NossoGrafo
4.  from myTpFormat import DadosTp, DaImgIndiv, GetTp_id
5.
6.  yInfoOrder = {
7.      "Principal":
8.          {"PorOrdem": ["nomeAlternativo", "url"],
9.           "OrdemAlfabética": []},
10.     "Secundário":
11.         {"PorOrdem": ["nomeComGralhas", "morada", "arruamento", "numPorta",
12.                       "andar", "localidade", "códPostal", "país", "temDoc"],
13.          "OrdemAlfabética": []},
14.     "EmAnexo":
15.         {"PorOrdem": [], "OrdemAlfabética": []},
16.     "Esconder":
17.         ["data", "dataFinal", "ano", "anoFinal", "designação", "imagem",
18.          "versoImg", "tipoLocal", "resumo", "rdf:type", "transcrição"],
19.     "CollapsibleText":
20.         [["transcrição", {}]]
21. }
22. SetPpTpInfo("Local", yInfoOrder)
23.
24.
25. def localTp_router(request):
26.     jinja_parms = {}
27.     # --- Obter dados específicos a passar ao template Jinja.
28.     DadosTp(jinja_parms, request, yInfoOrder, None, ":designação",
29.             ["", ":tipoLocal"])
30.
31.     cID = GetTp_id(request)
32.     g = NossoGrafo()
33.     # Imagem do verso
34.     DaImgIndiv(g, cID, "profileLoc.png", None, None,
35.                ":versoImg", jinja_parms, "_back")
36.     # Imagem da frente
37.     DaImgIndiv(g, cID, "profileLoc.png", None,
38.                "Imagem indistinta (bolha de localização)", ":imagem",
39.                jinja_parms, "")
40.
41.     return render_template('indivTp.jinja2', **jinja_parms)
```

Cf. linha 6, atribuição de yInfoOrder:

- "Principal" e "Secundário": Duas zonas consecutivas de informação, a pedir listas de termos da ontologia. Serve para selecionar apresentação de atributos e relacionamentos do indivíduo com terceiros, i.e., triplos onde o indivíduo aparece à esquerda, tipicamente a informação mais pretendida.

  "PorOrdem": os termos serão listados pela ordem específica em que surgem na lista.

  "OrdemAlfabética": os termos serão listados pela ordem alfabética.

- "EmAnexo": Usado para selecionar apresentação de relacionamentos com o

indivíduo, i.e., triplos onde o indivíduo aparece à direita.

- "Esconder": Termos a esconder da visualização, tipicamente por serem redundantes. Como exemplo, datas já contempladas no cabeçalho da página apresentada, ou relacionamentos owl:inverseOf.
- "CollapsibleText": Termos a mostrar na zona inferior da página, na forma de separadores colapsáveis – indicar um mapeamento (função parcial) em que cada chave é o termo, e a correspondência prevê configurações futuras ainda não necessárias.

Finalmente, em app.py:

```
1.  SetRoute_Pp("Local", "/localTp")  # Registar a rota prettyprint da ontoclasse.
2.
3.  #Router prettyprint da ontoclasse:
4.  @app.route("/localTp")
5.  def localTp():
6.      return localTp_router(request)
```

Formulários de ingestão

São automaticamente gerados todos os formulários de ingestão de dados, da responsabilidade do roteador indivTpIn_router (cf. indivTpIn.py) e do modelo de apresentação indivTpIn.jinja2, que funcionam como modelos universais. Assim, é possível ter uma ontoclasse sem especificidades de formulário, sendo gerado um formulário padrão – cf. botões de acesso, na secção de Ingestão interativa.

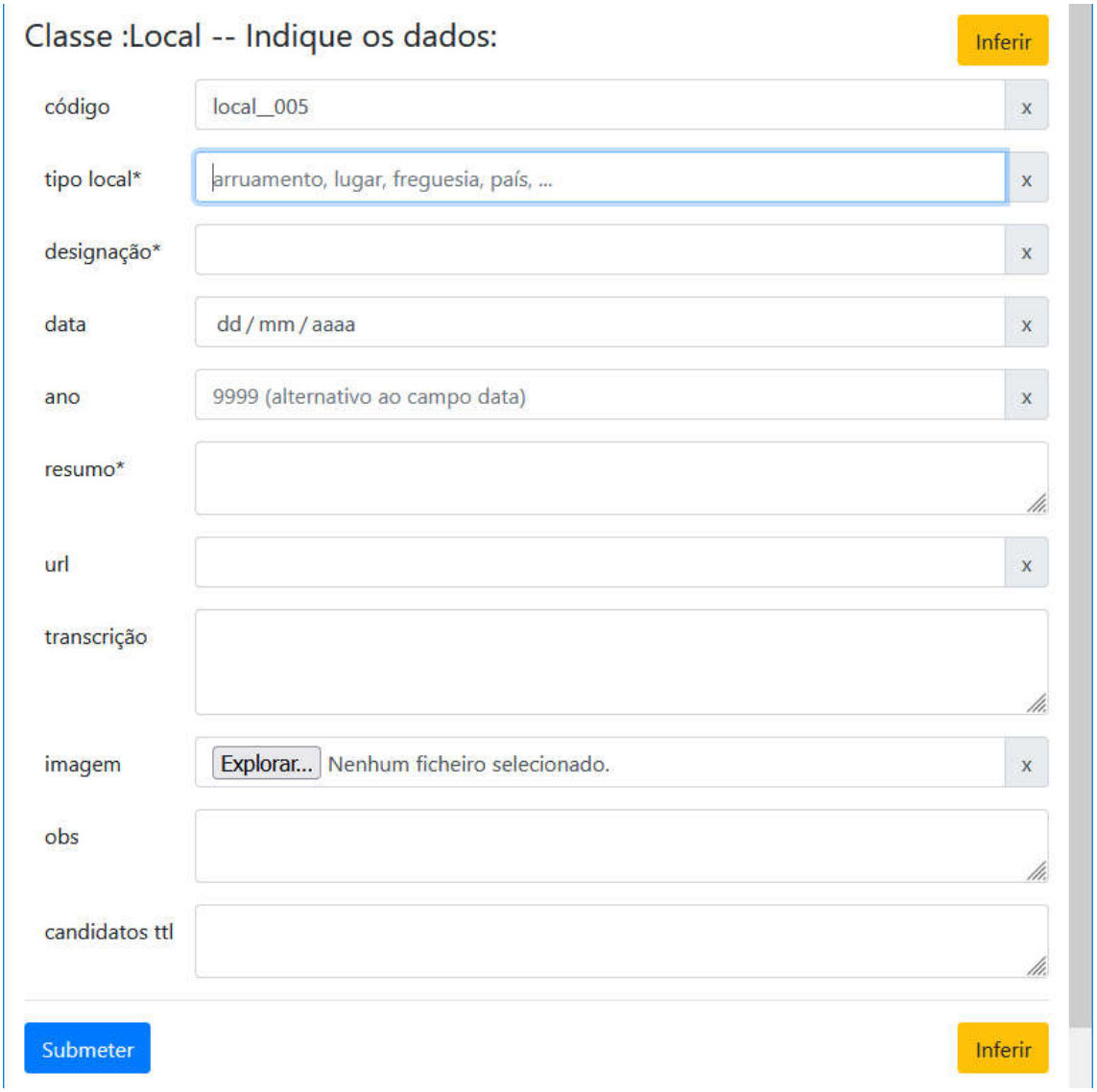


Figura 15 – Formulário de dados automático da classe Local

Mas também os formulários de ingestão podem ser configurados e afinados, com pouco esforço. Continuando com o exemplo do roteador localTp.py (modelação da UI da ontoclasse **Local**), essa afinação consegue-se do seguinte modo:

```
SetFields_PpIns("Local", [
    ["tipoLocal", {'required': True, 'autofocus': True,
                   'placeholder': "arruamento, lugar, freguesia, país, ..."}],
    ["designação", {'required': True}],
    ["data", {'type': "date"}],
    ["ano", {
        'placeholder': "9999 (alternativo ao campo data)"}],
    ["resumo", {'required': True, 'textarea': True}],
    ["url", {}],
    ["transcrição", {'textarea': True, 'rows': "2"}],
    # Prop. 'type' == "file" => botão para Escolher Ficheiro e upload automático.
    ["imagem", {'type': "file"}]
])
```

Para cada termo, temos um objeto/dicionário que associa elementos e atributos html a valores pretendidos ao gerar o formulário. Por exemplo, o termo tipoLocal será um campo unilinear, com *label* "tipo local*", e o devido <input ...>:

```
<input id="tipoLocal" name="tipoLocal" class="form-control form-control-md
rounded-left" onblur="setId(this.id)" required autofocus placeholder="arruamento,
lugar, freguesia, país, ..." title="'edifício', 'arruamento', 'lugar',
'freguesia', 'região', 'país', ...">
```

Handicap: ainda são efetuadas poucas validações aos valores inseridos em formulário pelo utilizador.

Neste âmbito, não é preciso fazer mais registos em app.py. Mas, e se quiséssemos personalizar o roteador indivTpIn_router? Então teríamos de escrever um novo roteador, e fazer o seu registo em app.py, como no exemplo feito para a classe SparqlQuery:

```
1.  # --- Rotas de ingestão de dados. Estas rotas serão atribuídas aos botões do
    template jinja, cf. uso de GetRoute_TpIn().
2.  # Rota de inserção de indivíduos da ontoclasse SparqlQuery.
3.  SetRoute_PpIns("SparqlQuery", "/sparqlQTpIns")
4.
5.  # Rota de alteração de dados da ontoclasse SparqlQuery.
6.  SetRoute_PpAlt("SparqlQuery", "/indivTpIn")
7.
8.  @app.route("/sparqlQTpIns")
9.  def sparqlQTpIns():
10.     return sparqlTpIns_router(request)
```

6.8. Ingestão de Dados

A filosofia de rapidez do desenvolvimento continuou pelo capítulo da ingestão de dados, que se desenrola a diferentes níveis, conforme já abordado e agora completado nas seguintes secções: ingestão automatizada, via opção DSL; ingestão interativa, via formulários html (automaticamente gerados, com possibilidade de configuração a nível da ontoclasse); ingestão automática por inferência, a pedido, via opção Utilidades.

Os campos de texto multilinha aceitam formatação Markdown e/ou HTML. Estes campos distinguem-se na sintaxe Turtle por terem aspas triplas, enquanto que nos formulários por serem um elemento *textarea*.

6.8.1 Área DSL

Esta área proporciona uma ingestão bastante livre, na sintaxe Turtle, por *upload* de ficheiro ou por edição direta, com análise básica de consistência.

É verdade que outros formatos poderão ser desejados (cf. secção sobre trabalho futuro), no entanto realça-se que será sempre possível utilizar previamente algum tipo de conversor, de um desses outros formatos para Turtle, e depois carregar o ficheiro resultante. Deste modo, poderá evitar-se complicar demasiadamente a aplicação, até porque será difícil satisfazer todas as pretensões. Lembre-se que, no âmbito de implementar requisitos e funcionalidades, a obtenção da primeira versão nem sempre é o mais trabalhoso difícil, e até

poderá ser estimulante; no entanto, a crescente montanha de código rapidamente começa a devorar tempo sempre que se pretende efetuar manutenção, que alguns autores referem como levando 80% do esforço total de desenvolvimento, durante a vida do software…

A análise e ingestão não foram muito elaboradas, mas uma caraterística importante é que atendem a coisas como "rdf:Type owl:FunctionalProperty", facilitando a fusão de indivíduos, já que nestes triplos só subsiste o último inserido, sendo o(s) anterior(es) automaticamente descartados.

6.8.2 Ingestão interativa

Por ingestão interativa entendem-se os formulários html com os quais o utilizador interage, seja por inserção de conteúdos em campos, seja por seleção de controlos, como botões e outros.

Acede-se a esta ingestão:

- nas páginas *prettyprint* de indivíduos, via botões **Alterar dados**, **Inserir predicado**, e **Novo indivíduo**, além do botão **Alterar**, ao lado de separadores colapsáveis;
- nas páginas *prettyprint* de ontoclasses, via botão **Inserir novo indivíduo**.

Nestes formulários:

- Os campos obrigatórios veem a sua *label* marcada com um asterisco, exemplo: "nome*".
- Os campos correspondentes a relacionamentos, por pedirem um ID, são marcados com um circunflexo, exemplo: "tem doc^". Neste tipo de campos é possível usar a inferência acionando um botão do formulário (Inferir) ou por uma tecla de atalho.
- As datas abrem um calendário ao clicar.
- Os campos de texto multilinha aceitam formatação Markdown e/ou HTML. Estes campos distinguem-se na sintaxe Turtle por terem aspas triplas, enquanto que nos formulários por serem um elemento *textarea*.
  Depois de selecionar uma parte do texto, é possível usar a inferência acionando um botão do formulário (Inferir) ou por uma tecla de atalho.

6.8.3 Ingestão automática por inferência

A ingestão a que se refere esta secção é efetuada por meio da opção Utilidades, nos respetivos botões. Faculta-se calcular e inserir automaticamente diversos relacionamentos de parentesco, em duas opções para (cf. pessoa_criar.py):

- Calcular e inserir indivíduos correspondentes a **progenitores e cônjuge** de cada indivíduo, e respetivos relacionamentos de parentesco com esse indivíduo de partida. O cálculo recorre aos nomes, com desambiguação pelo ano de nascimento (cf. as funções DaAnoNasc e _diferenca_inconveniente). Nomeadamente, no caso de cônjuges, considera-se inconveniente verificar-se que

```
abs(nAno_sujeito - nAno_testar) > 15.
```

Já ao dirimir pais e filhos, considera-se inconveniente que

```
nAnoFilh_ = nAno_sujeito
lKO = nAnoFilh_ - nAno_testar <= 12 or nAno_testar - nAnoFilh_ > 90.
```

Apesar de haver poucos relacionamentos em jogo, este algoritmo revelou-se bem mais complexo do que o da opção abaixo, devido ao cuidado de não duplicar indivíduos nem introduzir aberrações, devido a nomes iguais. Quando o sistema desconfia, recusa-se e apresenta as suas razões em relatório.

- Calcular e inserir relacionamentos correspondentes a diversos outros **graus de parentesco**, inferido sobre os relacionamentos de cada indivíduo com progenitores e cônjuge. Selecionou-se os parentescos considerados mais relevantes, até bisavós e bisnetos, além de alguns não sanguíneos, como enteados/padrastos/madrastas, meios-irmãos, sogros/genros/noras, cunhados.
  A operação é feita por recurso a diversas questões SPARQL do tipo `insert { … } where { … }`.

6.9. ANB NameTK

A normalização e pesquisa de nomes usada pela aplicação foi acomodada num módulo relativamente estanque (a menos de livrarias auxiliares genéricas), e dele efetuado um *spinoff* que permite procurar sobre uma lista de nomes. Dependendo do contexto, pode a pesquisa poderá ser sensível a nomes incompletos ou com gralhas, recorrendo ao ano ou data de nascimento afixada nos ID de Pessoa para desambiguar.

Futuramente, deseja-se ainda uma *script* para procurar nomes num dado ficheiro de texto.

O módulo está publicado em https://github.com/JMGrenhas/NameSearch e prevê-se que venha a constituir o PyPI (The Python Package Index).

Outra parte importante da funcionalidade prevista para este módulo, a inferência, está contida nos módulos respetivos:

- router: inference_router => indivTpIn.py;
- inferência: indInference.py, geneaGraphSearch.py (destaque de inferência posicionada genealogicamente).

6.9.1 Inferência de ID

A inferência tem sido muito referida neste documento porque recebeu muita atenção no projeto. Nesta secção, pretende-se abordar os resultados alcançados no que se refere à inferência de ID de indivíduos, desdobrada em dois contextos: inferência **dinâmica** ou **interativa**.

Ambas as inferências acabam por recorrer a um núcleo comum, no entanto é diferente o

contexto de cada uma. Para a inferência sobre pessoas (cf. classe Pessoa), é útil que tenham a data ou ano de nascimento preenchidos, em particular no ID (cf. numérico depois de "_d"). O processo tenta primeiro descobrir **graus de parentesco** (e caso encontre infere sobre relacionamentos nesse contexto), depois analisa **ID**, e depois procura por **nome**.

Inferência interativa

A inferência interativa é acessível no contexto duma página *prettyprint*. Como o nome indica, é pedida interativamente pelo utilizador (em campos elegíveis de formulários de ingestão/alteração de dados, via botão *Inferir* ou respetivo atalho *shift+alt+c*), em duas situações:

- num campo referente a um **relacionamento** (i.e., uma owl:ObjectProperty, ex.: :temDoc): como estes campos pedem um ID de indivíduo (ex.: :Eurico_Thomaz_Lima_d1908), permite-se ao utilizador escrever no campo e acionar a referida inferência, que irá retornar um ID ou uma lista de ID, separados por vírgulas, dos quais apenas um deve permanecer, senão a submissão irá ignorar o campo;
- num campo **multilinha** que seja mostrado como colapsável na página *prettyprint* (cf. exemplo na secção Templating de classe, yInfoOrder = { … "CollapsibleText": [["transcrição", …]] }), vai existir à direita do separador colapsável um botão, ***Alterar***, que redireciona para um formulário html, e no campo de edição textarea, respetivo, é possível ao utilizador acionar a inferência, depois de selecionar o texto pretendido; em caso de sucesso da inferência, mostra-se um ID entre chavetas duplas, ou uma lista dessa tipologia: "**{{**<ID>**}}** (, **{{**<ID>**}}**)*". O reconhecimento desta diretiva será efetuado no momento da apresentação *prettyprint*, em que cada para cada diretiva se procura gerar uma hiperligação para a página *prettyprint* de um indivíduo.

Em referência à inferência genealógica, sobre graus de parentesco, tentou abranger-se um grande número de casos especiais, por exemplo:

Texto	**Permite inferir**
avós, ou *av@*	Todos os avós, paternos e maternos, masculinos e femininos.
avós paternos	Todos os avós paternos, masculinos e femininos.
avô paterno	Todos os avós paternos, masculinos.
avô, ou *avó*	Todos os avós, respetivamente masculinos e femininos.
irmãos, ou *irm@*	Todos os irmãos, masculinos e femininos.
irmã, ou *irmãs*	Todas as irmãs.
irmão	Todos os irmãos masculinos.
pais	Ambos os progenitores.
tio Serafim	Indivíduos com parte do nome igual a "Serafim", relacionados via termos temTio.
Serafim, tio de	Indivíduos como o anterior, relacionados via termos éTioDe.
Maria, mãe do João	Indivíduos parcialmente chamados "Maria", relacionados via termos éMãeDe com indivíduos parcialmente chamados "João".

José e filh@	Indivíduos de nome José com filhos de qualquer sexo (via relacionamentos temFilhoOuFilha).
José (1950) e filhos	Indivíduos de nome José nascidos em 1950 e filhos do sexo masculino, relacionados via temFilho.
pais de Maria	Pai e mãe de indivíduos de nome Maria.
as avós do João	Avós do João, do sexo feminino.
Maria, esposa do João	… => éEsposaDe
Maria e o seu marido; ou Maria, cujo marido	… => temEsposo (via ontoclasse :AliásParentesco)
João e a sua esposa Maria	… => temEsposa

Inferência genealógica

A inferência posicionada genealogicamente atua ao reconhecer sensivelmente o seguinte (cf. função Infer_Genea):

inferência := objeto => <pesquisar por ID, senão por nome>
| (parentesco__é_De | parentesco__tem_) objeto **=>** <pesquisar …>
| sujeito (parentesco__é_De | parentesco__tem_) **=>** <pesquisar …>
| sujeito (parentesco__é_De | parentesco__tem_) objeto **=>** <pesquisar …>

sujeito := <ID> | <nome>
objeto := <ID> | <nome>

parentesco__é_De := partPréADescartar* possessivosPré+ grauDeParentesco partPós*
=> <usar relacionamento "é" + **grauDeParentesco** + "De">
parentesco__tem_ := partPréADescartar* partPré+ grauDeParentesco partPós*
=> <usar relacionamento "tem" + **grauDeParentesco**>

partPréADescartar := 'e' | 'a' | 'o' | 'as' | 'os' | 'gémea' | 'gémeas' | 'gémeo' | 'gémeos'
partPré := 'cuja' | 'cujas' | 'cujo' | 'cujos' | 'seu' | 'seus' | 'sua' | 'suas'
possessivosPré = 'de' | 'da' | 'do' | 'com'

grauDeParentesco := <graus de parentesco como subcadeia de relacionamentos definidos na ontologia, tal que
(?relacionamento rdfs:subPropertyOf :temParentesco .)
ou (?id a :AliásParentesco . ?id :parentescoTxt ?grauParentesco .),
ex.: 'pai' | 'mãe' | 'pais' | 'filho' | 'filha' | 'filh@' | …
>

partPós := ‘de’ | ‘da’ | ‘das’ | ‘do’ | ‘dos’

Sintaxe inspirada em expressões regulares. Legenda:

- [ ] – opcional
- | – alternativa
- * – zero ou mais vezes
- + – uma ou mais vezes
- ( ) – agrupamento

Inferência dinâmica

A inferência dinâmica é acionada em todo o campo mostrado como colapsável na página *prettyprint* de um indivíduo (cf. exemplo na secção Templating de classe, yInfoOrder = { … "CollapsibleText": [["transcrição", …]] }).

Este modo reconhece cadeias entre chavetas duplas – {{ <texto> }}. Para cada elemento deste tipo, será gerada máximo uma hiperligação para a respetiva página prettyprint. Em <texto> deve indicar-se tipicamente um ID ou nome completo, opcionalmente prefixado com a classe e o caráter "!", opcionalmente sufixado com um ano entre parêntesis. Esta inferência reage melhor com referências correspondentes a ID de indivíduos, porque são únicos e menos sujeitos a alteração – senão, ao usar nomes e outras informações, torna-se possivelmente menos eficaz por ser sensível a alteração aos dados guardados, e menos eficiente por exigir mais processamento nalguns casos.

Exemplos:

- {{:João_etc_d1969}} <-- ID de indivíduo, a forma mais segura de referência.
- {{ Pessoa!João Alex Alexandre (1911) }} <-- Nome com ano de nascimento (para desambiguar).
- {{Pessoa! M. Joana Pereira}} <-- Nome com inicial explícita.
- {{Pessoa!Maria Albertina Pereira da Silva}} <-- Nome completo.
- {{ Org! Módula C }} <-- Nome da organização; a menos do nome, os espaços na sintaxe são indiferentes.

Se a inferência não tiver sucesso, o texto é apresentado entre {{}} tal e qual: a preto se não foi possível reconhecer a sintaxe, a vermelho se reconheceu mas não unificou com nenhum indivíduo da ontologia.

7. Conclusões e Trabalho Futuro

7.1. Conclusões

Para apresentar resultados inovadores, este projeto teve de ser balizado, dado o recurso tempo e as minhas áreas de especialização. Assim, apesar da ontologia ter no seu âmago uma componente de genealogia e relacionamentos de parentesco, além de sociais e outros, o foco da inovação foi colocado em proporcionar ao utilizador a **ingestão**, completação e **apresentação** de documentos de variada morfologia, incluindo multimédia, num ambiente operacional relevante de **pesquisas** de utilizador e navegação, sem menosprezar a facilidade de **inferência** sobre indivíduos (pessoas ou não).

A aparência não foi descurada, apesar de se tentar manter um visual contemporâneo (simples e tendencialmente minimalista).

A inovação do projeto consiste numa aplicação web:

- apoiada em Flask/Python;
- que disponibiliza um grafo de conhecimento (*knowledge graph*) RDFLib, orientado por uma ontologia na sintaxe Turtle;
- com templating configurável de apresentação e/ou ingestão;
- auxiliada por inferência de indivíduos, motores de busca, editor e executor de SPARQL;
- com navegação sistemática por SPARQL e hiperligações;
- que oferece uma plataforma de prototipagem rápida.

A adoção da arquitetura de ontologia revelou-se útil para a inferência, a vários níveis:

- na inserção automática de triplos de relacionamentos associados a graus de parentesco – via opção na área de Utilidades, com algum apoio no SPARQL mas também cálculos de inferência e validações ao introduzir automaticamente indivíduos correspondentes a pai, mãe e cônjuge;
- na inferência (dinâmica ou interativa) de indivíduos a partir de texto, sejam sobre designações e nomes (incluindo alcunhas, nomes com gralhas, etc.), ID ontológicos – ambos opcionalmente com anos ou datas --, ou mesmo indivíduos relacionados por graus de parentesco.

A inferência não incidiu, apesar do que se desejava, sobre *reasoning de* lugares x datas x eventos, necessariamente algo complexo, mas sim no aproveitamento da data de nascimento e relacionamentos entre indivíduos, de modo a potenciar de imediato a arquitetura de grafo subjacente. A inferência de indivíduos pela designação ou ID ajuda a contornar outras lacunas, em que a desambiguação dos resultados fica disponível ao utilizador.

A adoção da arquitetura da ontologia com persistência num ficheiro de texto relevou-se flexível e prática a vários níveis:

- facilita a prototipagem e correção rápida, por edição direta do ficheiro;
- facilita desenhar a ontologia noutra ferramenta (experimentou-se Protégé) e trazer o ficheiro exportado por essa ferramenta, em sintaxe Turtle, para carregar na aplicação;
- a ontologia é carregada em memória na primeira vez que se pretende o grafo correspondente, e depois automaticamente quando se deteta alteração da *timestamp* do ficheiro. O uso a partir de memória torna o sistema eficiente. Quando a ontologia for muito grande, poderá demorar um pouco mais na primeira vez que é carregada, e ao gravar.

7.2. Trabalho futuro

Durante o projeto, várias ideias houve que acabaram por não ter implementação, essencialmente por falta de tempo, mas poderão ter muito interesse numa abordagem futura, se corresponderem a reais necessidades. As primeiras da lista começaram, inclusive, a tomar a forma de requisito já um pouco mais estruturado, mas por pedirem mais análise não se considerou todas as secções do cartão de Volere. Depois, seguem-se ideias que entendi merecerem registo, ainda que incipientes. Naturalmente, outras poderão emergir como interessantes. Nomeadamente, tudo o que facilite e melhore a experiência do utilizador motiva à utilização do sistema, potenciando o seu futuro desenvolvimento.

Pessoalmente, defendo o desenvolvimento e adoção de ambientes com uma curva de aprendizagem rápida. Tenho constatado que nascem permanentemente aplicações, sistemas, arquiteturas, conceitos, tecnologias, etc., e nem sempre há muito tempo (ou energia anímica) para "tirar mais um curso" como pré-condição de utilização, ou então a devida exploração do sistema revela-se demasiado penosa se não for de manifesto interesse. Um sistema deve facilitar a instalação e uso imediatos, ou quase, ainda que possa, depois, exigir mais formação para uma utilização mais rica e produtiva, mas nessa altura o utilizador já se encontra mais motivado a dedicar-lhe tempo e esforço. Seja como for, poderão futuramente acrescentar-se funcionalidades como explorado nas secções seguintes.

RFf1[12]: Permitir inserção interativa fácil de termos sobre indivíduos

Requirement #: RFf1	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Facilitar a inserção de informação associada a um indivíduo.		
Originator: *Brainstorming*		
Fit Criterion: No contexto da página aprimorada de um indivíduo, deve facilitar-se o acesso interativo a campos para escolher um termo (atributo ou relacionamento) e respetivo valor a introduzir. A escolha do termo deve fazer sugestões em função do que se escreve, baseado na ontologia subjacente. Ex.: ao escrever "avô" (ou mesmo "avo"), deve dar-se a escolher		

[12] O "f" refere-se a "futuro".

termos como temAvô, éAvôDe, entre outros (ou mesmo temAvó, éAvóDe). A escolha poderá basear-se num menu *pop-up*. Ex. – Termo Valor <caixa para escrever><controlo para ver lista de termos> <caixa para valor>

RFf2: Imprimir uma compilação

Requirement #: RFf2	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Facilitar a publicação de assuntos relacionados via um indivíduo da classe Compilação.		
Originator: *Brainstorming*		
Fit Criterion: Poder escolher um indivíduo da classe Compilação e gerar uma página ou documento (ex.: no formato PDF) com os atributos e informação associada via relacionamentos da Compilação. A geração de uma página responsiva poderia permitir, pelo simples ajustar da largura do navegador, ajustar a disposição de elementos conforme pretendido para *output*.		

RFf3: Disponibilizar editor de texto com controlos markdown *ou afins*

Requirement #: RF3	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Aumentar a usabilidade, melhorar a experiência de utilização.		
Originator: *Brainstorming*		
Fit Criterion: Permitir uma configuração na aplicação que sinalize certos campos para recorrerem ao editor wiki, e nas áreas elegíveis da aplicação, permitir abrir o referido editor. Dicas: observar zim-wiki, ontowiki, docuwiki, mediawiki, wikipedia, etc.		

RFf4: Pesquisar iniciais em nomes apenas ao indicar maiúsculas

Requirement #: RF4	Requirement Type: Funcional	Event/BUC/PUC #:
Rationale: Facilitar a afinação da procura, alternando entre procurar ou não com recurso a iniciais de nomes, para limitar a quantidade de resultados.		
Originator: *Brainstorming*		
Fit Criterion: Nos motores Procurar, o utilizador usar Maiúsculas poderia ser tido como indicação de pretender pesquisar pelas iniciais de nomes e ID. Ex.: Pesquisar joão manuel não teria em conta iniciais, já João Manuel, ou mesmo J M, teria. No sistema, por outro lado, já se pode indicar iniciais (maiúscula ou minúscula, seguida de ponto) para restringir a pesquisa – ex.: "J. Manuel" pesquisa nomes com a inicial explícita "J." ou ID com "..._J_...".		

Completar a ajuda

No anexo *Página de ajuda* transcreve-se a ajuda da aplicação, acessível pela opção homónima. A ajuda é sempre pouca e está sempre incompleta. Assim, não será difícil adivinhar que precise de mais conteúdos e exemplos.

Clonar instalações da aplicação

Analisar e disponibilizar um modo fácil de **clonar a aplicação para uso com outras ontologias**. Provavelmente, o nome do ficheiro TTL será diferente, a configurar em env.py. Havendo apenas um ficheiro TTL, a aplicação poderia assumi-lo sem ter de ser configurado.

A respeito de carregar outras ontologias, e conforme já constatado com a famosa ontologia pizza, será necessário reconhecer simultaneamente um prefixo vazio e outro não vazio, como prefixos de raiz da ontologia. Exemplo:

```
@prefix : <http://www.co-ode.org/ontologies/pizza/pizza.owl#> .
@prefix pizza: <http://www.co-ode.org/ontologies/pizza/pizza.owl#> .
```

Melhorar a validação de campos em formulários

Os formulários de ingestão (i.e., inserção ou alteração de dados) beneficiarão da validação de campos, caso contrário o servidor poderá recusar a operação, por vezes de modo silencioso. Assim:

- Quando a validação for efetuada pelo servidor e falhar, deve ser retornado um aviso a mostrar ao utilizador – ex.: um conteúdo mostrado no início da página produzida.
- Quando a validação for efetuada pelo formulário, deve ser emitido um alerta e permanecer na página.

Poder imprimir para PDF

Atualmente, usando as funções do navegador web e um controlador PDF instalado no sistema, já é possível imprimir para PDF. Contudo, seria mais agradável imprimir sem ver todos os controlos da página, como botões de ação (*Alterar Dados*, etc.).

Aceitar Notation3

A sintaxe Notation3/N3[13] poderá ser considerada no futuro, para a persistência da própria ontologia, visto ser um superconjunto de Turtle e o RDFLib permitir carregar aquele formato. Note-se que a opção por uma sintaxe é importante porque, ao carregar a ontologia num grafo, torna-se necessário especificar a sintaxe do ficheiro, e acaba por definir uma parte da interface com o utilizador. Cf. https://en.wikipedia.org/wiki/Notation3.

Poder descarregar informação em diferentes formatos

Poder descarregar informação em diferentes formatos, como PDF, Markdown, Latex, ou mesmo CSV, nos diversos ambientes da aplicação. A este respeito, além de livrarias específicas do Python, como Pandoc, também poderão ser explorados templates Jinja2.

[13] Não confundir com N-Triples.

Poder importar informação noutros formatos

A aplicação prevê importar no formato Turtle. Poderá haver interesse em carregar informação noutros formatos, como **Markdown** ou **CSV**, mas com uma estrutura pré-definida via DSL -- cf. YAML como metadados.

Refira-se que será sempre possível utilizar previamente (via linha de comandos ou ia alguma aplicação) algum tipo de conversor, de um qualquer formatos específico para Turtle, e depois carregar o ficheiro resultante. Deste modo, poderá evitar-se complicar demasiadamente a aplicação, até porque será difícil satisfazer todas as pretensões e respetiva evolução – os requisitos evoluem, logo os formatos pretendidos também irão evoluir, infelizmente para o programador...

Em referência à conversão entre formatos, lembre-se de livrarias específicas do universo Python, como Pandoc.

GEDCOM

Usar uma biblioteca Python para importar e exportar dados segundo GEDCOM, por ser um formato padrão muito difundido, em particular a pensar na área de genealogia da ontologia. Cf., por exemplo: conversores para Turtle; xmlgedcom, entre outros.

Resolver axiomas owl:propertyChainAxiom

Analisar se é útil usar/derivar a função `VrfAxioma`, já aproveitada na inferência interativa, para testar/inferir indivíduos a partir de axiomas definidos na ontologia, como por exemplo :temNeta owl:propertyChainAxiom (:temFilhoOuFilha :temFilha), embora este caso já esteja previsto na geração automática possibilitada pela opção Utilidades -> Genealogia(...).

Analisador de consistência do knowledge graph

Um *reasoner* bem escolhido poderá ajudar nesta tarefa. Recomenda-se cuidado a selecionar reasoners que possam entrar em *deadlock*.

Substituição global de um ID que se revelou insatisfatório.

Na arquitetura atual, esta tarefa pode fazer-se por meio de um simples editor de texto sobre o ficheiro da ontologia, ANB.ttl, com recurso a uma substituição global.

Remoção seletiva de relacionamentos

Pode fazer-se via SPARQL (exemplos):

1. Remoção dos relacionamentos com ID à direita:
   DELETE WHERE { ?s ?p :indiv . ?p a owl:ObjectProperty . }
2. Remoção dos relacionamentos com ID à esquerda:
   DELETE WHERE { :indiv ?p ?o . ?p a owl:ObjectProperty . }

Área DSL

- Havendo dúvidas sobre triplos, poderiam ser automaticamente adicionados a **candidatosTtl**, para análise futura. Ou seja, implementar uma opção **"Inserir mesmo assim"**, que selecionaria os triplos considerados inválidos e os desviaria para candidatosTtl.
  Nesta situação, poderá ter de decidir se gera ID para diferenciar doutro existente ou não (caso em que vai fundir ou reescrever a informação… cuidado!). Lembra-se o termo ontológico oMesmoQue (ou owl:sameAs), que pode usar-se para associar ID com outros já existentes.
  Na introdução de triplos de definição de ontologia, poderá ser necessário cuidados especiais.
- Poder carregar informação numa sintaxe markdown. Como alternativa, será útil **avaliar da existência de conversores de markdown para Turtle.** Ainda assim, deixa-se alguns exemplos:
  - indivíduo com fotografia:
    id: f_55
    ficheiro foto55.jpg
    resumo: <…>
    transcrição: <…>
    <…>
  - carta:
    ---
    id: C044
    type: carta
    from: Olga Pedrário
    from-address: Rua Frederico Eyer, 141, Gávea / Rio de Janeiro, Brasil
    to: ETL
    to-address: Rua Egas Moniz, 6, 1º / Porto / Portugal
    date: 1954-08-20
    transc: jj
    about:
    - estudo
    - berceuse

    Querido Eurico…

Inferência

Reconhecer intervalos temporais, em pesquisas

O sistema já permite indicar casos "(99..)", em expressão regular "\(\d\d..\)" (em particular, aceita-se "x" como aliás para "."). Exemplos: "(1969)", "(196x)", "(196.)", "(19xx)", "(19..)".

Seria ainda mais útil poder indicar intervalos "\(\d\d..-\d\d..\)", de modo a abranger casos

mais ambíguos em termos de viragem de século. Exemplo: "(189x-191x)".

Revisitar objetivos desta tese

Sugere-se revisitar os objetivos da tese, especialmente em termos de inferência, uma vez que o recurso tempo acabou por adiar indefinidamente alguns, como:

- **Desambiguar pessoas/entidades pelo contexto** de lugares, datas (expressas ou implícitas) e eventos, que possam ser comparados com os conhecidos (nascimento e morte, idade maior, meia idade, idade avançada).
- O sistema poderá aceitar ***constraints*** para ajudar a refinar soluções. Pretende-se um ***constraint solver*** para restrições no domínio **geográfico**, **genealógico**, **temporal** – no âmbito do período de vida humano (como data de óbito >= data de nascimento, idade para serviço militar em comparação com eventos como batalhas, etc.).

Implementar a funcionalidade do termo candidatosTtl

Cf.:

```
:candidatosTtl a owl:DatatypeProperty ;
rdfs:comment "Triplos em Turtle, pendentes de análise e inserção." ;
rdfs:range xsd:string .
```

Este termo poderia receber uma lista livre de triplos na sintaxe Turtle, pendentes de análise e inserção, a tratar por uma opção específica, que iria analisar a sintaxe e semântica, desambiguando e pedindo ajuda para os introduzir na ontologia. Numa primeira abordagem, podem considerar-se apenas triplos em que entrem owl:NamedIndividual, ainda que só num dos lados do triplo. Posteriormente, poderá pensar-se na definição de termos da ontologia.

Qualquer dos termos pode não existir na ontologia (e induzir inserção automática), ou ser substituído por um já existente.

Exs.:

- (:João :frequenta :Eduardo .), sendo :João o cliente do barbeiro :Eduardo. Eventualmente, *frequenta* poderia ser substituído por *conhece*.
- (:João :éPaiDe :Maria .) Como :éPaiDe já existe, deduz-que que :João e :Maria são da classe :Pessoa, podendo ser automaticamente inseridos se não existirem.

Ainda assim, lembre-se que existe no sistema uma opção, **DSL**, que permite, na hora, analisar e introduzir na ontologia triplos de toda a natureza. A referida análise não é ainda a ideal e pode ser melhorada.

Níveis de utilização

Implementar diferentes níveis de utilização, por razões de segurança. Assim, determinados níveis não poderiam remover informação, por exemplo.

Bibliografia

[AM2019] José João Almeida, Rui Castro Mendes (2019). ***Hunting Ancestors: a unified approach for discovering genealogical information***

[AH2017] Cristiana Araújo, Pedro Henriques (2017). ***Introdução às Ontologias***, apresentação de diapositivos, unidade curricular de Gramáticas na Compreensão de Software 2017/2018.

[FWCT2013] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, Elias Torres (2013). ***SPARQL 1.1 Protocol, W3C Recommendation 21 March 2013.*** Cf. https://www.w3.org/TR/sparql11-protocol.

[HP2012] M. Horridge and P. F. Patel-Schneider (2012). ***OWL 2 Web Ontology Language Manchester Syntax (Second Edition).*** W3C Working Group Note 11 December 2012. Cf. https://www.w3.org/2012/pdf/NOTE-owl2-manchester-syntax-20121211.pdf.

[MPP2012] Boris Motik, Peter F. Patel-Schneider, Bijan Parsia. ***OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition).*** Eds. W3C Recommendation, 11 December 2012, http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. *Latest version available at* http://www.w3.org/ TR/owl2-syntax/.

[NM2000] Natalya F. Noy and Deborah L. McGuinness (2000?). ***Ontology Development 101: A Guide to Creating Your First Ontology***, Stanford University.

[Ram2020] José Carlos Ramalho (2020). ***Tutorial: Protégé-OWL (parte II)***, apresentação de diapositivos.

[Rib2017] Frederico Moreira Ribeiro (2017). ***Especificação de uma Ontologia para Genealogia***, Master dissertation, Master Degree in Computer Science, Universidade do Minho.

[SSMJ2015] Robert Stevens, Margaret Stevens, Nicolas Matentzoglu and Simon Jupp (November 25, 2015). ***Manchester Family History Advanced OWL Tutorial***, Edition 1.1. Cf. http://mowl-power.cs.man.ac.uk/fhkbtutorial/resources/FHKB-tutorial_v1_1.pdf

[Stur2020] Thomas F. Sturm (2020). ***The genealogytree package Manual for version 2.01 (2020/07/28).*** Cf. http://mirrors.ctan.org/macros/latex/contrib/genealogytree/genealogytree.pdf

Glossário Geral

Nota: Para termos no contexto da genealogia, recorrer ao Glossário Genealógico.

entidade

(latim medieval entitas, -atis)

nome feminino

1. Tudo o que é concreto.

2. Ente; ser; indivíduo.

3. Instituição, organismo ou outra pessoa jurídica com funções específicas (ex.: entidade bancária; entidade privada; entidade pública).

4. [Informal] Importância.

"entidade", in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2020, https://dicionario.priberam.org/entidade [consultado em 03-02-2021].

indivíduo

inglês: ***individual***

No âmbito desta tese e das ontologias, um indivíduo corresponde *grosso modo* a uma instância duma ontoclasse, com associação por via de um identificador (ID), que neste trabalho se considera globalmente único. Exemplo muito resumido, na sintaxe Turtle:

```
:M_Isilda_S_Mendes a :Pessoa, a owl:NamedIndividual.
```

Cf. https://en.wikipedia.org/wiki/Ontology_components [consultado em 27-11-2021].

memorabilia

|memòràbílià|
(palavra latina, plural neutro substantivado de *memorabilis, -e*, memorável) *nome feminino plural*

1. Conjunto de coisas ou acontecimentos memoráveis.

2. Conjunto de objetos.

"memorabília", in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2020, https://dicionario.priberam.org/memorabilia [consultado em 24-01-2021].

ontoclasse

No âmbito desta tese, refere-se a uma classe da ontologia.

(Porquê um termo novo? Porque, em programação e arquiteturas de software, classes há-as de muitas naturezas… e se com um simples termo conseguimos desambiguar o contexto, porque não um termo novo?)

reasoner

"A **semantic reasoner**, **reasoning engine**, **rules engine**, or simply a **reasoner**, is a piece of software able to infer logical consequences from a set of asserted facts or axioms."

Origem: https://en.wikipedia.org/wiki/Semantic_reasoner [02/01/2021]

taxonomia

nome feminino

Teoria ou nomenclatura das descrições e classificações científicas. = TAXINOMIA "taxonomia", in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2020, https://dicionario.priberam.org/taxonomia [consultado em 24-01-2021].

nome feminino

1. conjunto de princípios e métodos de classificação dos diversos elementos de uma área científica; sistema de categorização

2. BIOLOGIA ramo da sistemática que, considerando a semelhança e dissemelhança de características dos seres vivos, os descreve e agrupa em categorias organizadas e hierarquizadas (como o tipo, a família, o género, a espécie, etc.); biotaxia

https://www.infopedia.pt/dicionarios/lingua-portuguesa/taxinomia [24-01-2021]

wiki

"Um **wiki** é uma publicação com hipertexto, colaborativamente editada e gerida pela sua própria audiência, diretamente via *web browser*. Num wiki típico, o texto é escrito com uma linguagem de marcação e frequentemente editado com a ajuda de um editor de texto enriquecido. Um wiki é executado por um software wiki, um tipo de sistema de gestão de conteúdos, mas diverge da maioria de outro tipo de sistemas, inclusive software de *blog*, no sentido em que o conteúdo é criado sem qualquer dono ou líder definido. Os wikis possuem pouca estrutura inerente, o que permite que a estrutura seja melhorada de acordo com as necessidades dos utilizadores."

https://pt.wikipedia.org/wiki/Wiki e https://en.wikipedia.org/wiki/Wiki [26/01/2021].

xsd

"**XML Schema Definition**, commonly known as **XSD**, is a way to describe precisely the XML language. XSDs check the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language."

https://www.tutorialspoint.com/xsd/index.htm [26/01/2021]

Glossário Genealógico

Incluem-se nesta secção alguns termos específicos do âmbito. Não se pretende ser exaustivo, mas abordar alguns termos eventualmente menos conhecidos. Abreviaturas nesta secção: poderá ser necessário consultar a origem da informação, referida ao cabo de cada definição.

agregado familiar

Conjunto de pessoas que vive em comunhão de bens e habitação.

ancestral

an·ces·tral
(francês ancestral; inglês: ***ancestor***)

adjetivo de dois géneros

1. Dos antepassados.

2. Muito antigo.

3. Avito.

adjetivo de dois géneros e nome de dois géneros

4. Que ou quem pertence a uma geração anterior. = ANTECEDENTE, ANTEPASSADO

Origem: https://dicionario.priberam.org/ancestral [15/11/2020].

ascendente

as·cen·den·te

(inglês: ***ancestor)***

adjetivo de dois géneros

De quem se descende.

nome de dois géneros

Antepassado.

ascendentes

nome masculino plural

Geração ou gerações anteriores. = AVÓS, ASCENDÊNCIA, ASCENDENTES

Origem: https://dicionario.priberam.org/ascendente [15/11/2020].

nome de 2 géneros

pessoa de quem um indivíduo descende; antepassado; progenitor

Origem: https://www.infopedia.pt/dicionarios/lingua-portuguesa/ascendente [15/11/2020].

cônjuge

côn·ju·ge
(latim *conjux, -ugis ;* inglês: ***partner, consort, spouse***)

nome masculino

Pessoa casada com outra, em relação a esta (ex.: *cônjuge feminino, cônjuge masculino*). = CONSORTE, ESPOSO

Confrontar: conjugue.

Origem: https://dicionario.priberam.org/cônjuge [15/11/2020].

costado

... 4. cada um dos avôs e avós de um indivíduo

Origem: https://www.infopedia.pt/dicionarios/lingua-portuguesa/costado [15/11/2020].

cunhado

cu·nha·do

(latim *cognatus, -a, -um*, parente, relacionado, ligado, semelhante; inglês: ***brother-in-law, sister-in-law*** (cunhada)***, siblings-in-law*** (cunhados))

nome masculino

1. Irmão de um cônjuge em relação ao outro cônjuge.

2. Marido da irmã ou do irmão de determinada pessoa, em relação a essa pessoa.

Origem: https://dicionario.priberam.org/cunhado [15/11/2020].

enteado

en·te·a·do
(latim *ante*, antes + latim *natus, -a, -um*, nascido; inglês: ***stepson, stepchild***)

nome masculino

1. Filho de uma relação anterior do cônjuge ou companheiro, em relação ao seu padrasto ou madrasta.

Palavras relacionadas: filhastro, não-filho, parequema.

Origem: https://dicionario.priberam.org/enteado [15/11/2020].

género sexual

Na sociedade, **identidade de género** refere-se ao género com que a pessoa se identifica, i.e., se ela se identifica como sendo um homem, uma mulher, ou pessoa não-binária, mas pode também ser usado para referir-se ao género que certa pessoa atribui ao indivíduo tendo como base o que tal pessoa reconhece como indicações de papel social de género (roupas, corte de cabelo, etc.).

Origem: https://pt.wikipedia.org/wiki/Identidade_de_género [15/11/2020].

irmão

ir·mão
(latim *germanus, -a, -um*, que é da mesma raça, irmão; inglês: ***sibling***)

nome masculino

1. Filho do mesmo pai e da mesma mãe.

2. Filho do mesmo pai que outro, mas não da mesma mãe, ou da mesma mãe, mas não do mesmo pai.

3. Membro de uma mesma irmandade, confraria ou comunidade religiosa.

4. Título de fraternidade que os homens podem dar-se mutuamente.

5. Pessoa muito amiga de outra.

irmão bilateral, carnal, consanguíneo ou germano • Filho dos mesmos progenitores.

irmão de leite• Pessoa que, em relação a outra, foi amamentada com o mesmo leite, sem serem irmãos verdadeiros. = COLAÇO

irmão de pai • Aquele que é filho só do mesmo pai.

irmão unilateral • Irmão que tem o mesmo pai que outro, mas não a mesma mãe, ou que tem a mesma mãe, mas não o mesmo pai. = MEIO-IRMÃO

irmão uterino • Aquele que é filho só da mesma mãe.

Feminino: **irmã**. Plural: **irmãos**.

Origem: https://dicionario.priberam.org/irmão [15/11/2020].

madrasta

ma·dras·ta
(latim vulgar *matrastra*, do latim *mater, -tris*, mãe; inglês: ***stepmother***)

nome feminino

1. Esposa ou companheira do pai, ou da mãe em casais do mesmo sexo, em relação aos filhos por eles tidos em relacionamento anterior.

Palavras relacionadas:

avodrasta, madrasto, avodrasto, não-filho, filhastro, enteado.

Origem: https://dicionario.priberam.org/madrasta [15/11/2020].

numeração de Sosa-Stradonitz

A **numeração de Sosa-Stradonitz** (ou **sistema de numeração de Eyzinger-Sosa-Stradonitz**), é um sistema de numeração dos antepassados nas genealogias ascendentes, também conhecido por **método Stradonitz** ou **Ahnentafel**. Este método atribui o n.º 1 ao indivíduo cuja genealogia se estuda (o sujeito, chamado *de cujus* ou *probandus*), o n.º 2 ao seu pai e o n.º 3 à sua mãe. Os seus avós paternos terão os números 4 e 5, e os avós maternos terão os números 6 e 7. Cada homem tem um número que é o dobro do número do seu filho (2n) e o número de cada mulher é o dobro do número do seu filho mais 1 (2n + 1).

Origem: https://pt.wikipedia.org/wiki/Numeração_de_Sosa-Stradonitz [15/11/20209].

padrasto

pa·dras·to
(latim patraster, -tri; inglês: ***stepfather***)

nome masculino

1. Marido ou companheiro da mãe, ou do pai em casais do mesmo sexo, em relação aos filhos por eles tidos em relacionamento anterior.

Palavras relacionadas: não-filho, avodrasto, filhastro, avodrasta, enteado.

Origem: https://dicionario.priberam.org/padrasto [15/11/2020].

parentesco / grau de parentesco (#1)

nome masculino

1. **vínculo que une duas pessoas**, em consequência de uma delas descender da outra ou de ambas procederem de um progenitor comum; consanguinidade
2. relação de **pessoas por vínculo de casamento; afinidade**

Origem: https://www.infopedia.pt/dicionarios/lingua-portuguesa/parentesco [15/11/2020]

parentesco (#2)

pa·ren·tes·co |ê| (inglês: ***relation, relationship)***

nome masculino

1. Condição dos que pertencem à mesma família.

2. Consanguinidade, afinidade.

Origem: https://dicionario.priberam.org/parentesco [15/11/2020].

Notas: O parentesco pode ser consanguíneo (relações de sangue) ou social (por

afinidade), e dentro destes pode inserir-se num de vários graus.

progenitor

pro·ge·ni·tor |ô| (inglês: ***parent***)

nome masculino

1. Aquele que gera. = ASCENDENTE, PAI, PROCRIADOR

progenitores

nome masculino plural

2. Conjunto das gerações anteriores; pais, avós ou antepassados. = ASCENDÊNCIA

3. Conjunto formado pelo pai e pela mãe.

Origem: https://dicionario.priberam.org/progenitor [15/11/2020].

Anexos

Anexo I. Modelo de Qualidade ISO 9126

A norma 9126 foca-se na qualidade do produto de software, propondo **Atributos de Qualidade**, distribuídos em seis caraterísticas principais e respetivas subdivisões, conforme se pode ver na figura abaixo (cf. https://pt.wikipedia.org/wiki/ISO/IEC_9126).

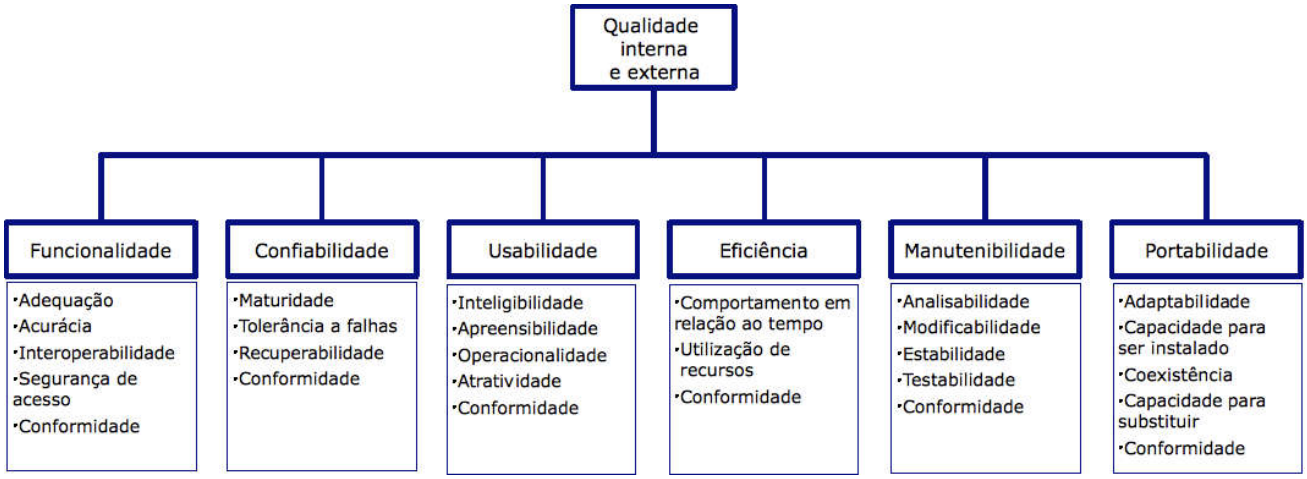


Figura 16 – Atributos de Qualidade da norma ISO 9126

Anexo II. Particularidades das Ontologias

Este anexo foi reservado para excertos de artigos e referências sob o tema em epígrafe.

OWL Constraints

Limitations of Ontologies, in https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/, 05/01/2021:

"While ontologies provide a rich set of tools for modeling data, their usability comes with certain limitations.

*One such limitation is the available **property constructs**. For example, while providing powerful class constructs, the most recent version of the Web Ontology Language – OWL2 has a somewhat limited set of property constructs.*

*Another limitation comes from **the way OWL employs constraints**. They serve to specify how data should be structured and prevent adding data inconsistent with these constraints. This, however, is not always beneficial. Often, data imported from a new source into the RDF triplestore would be structurally inconsistent with the constraints set using OWL. Consequently, this new data would have to be modified before being integrated with what is already loaded in the triplestore.*

*A novel alternative to using ontologies to model data is using the **Shapes Constraint Language (SHACL)** for validating RDF graphs against a set of constraints. A shape specifies metadata about a type of resource – how it is used, how it should be used and how it must be used. As such, similarly to OWL, SHACL can be applied to validate data. Unlike OWL, however, SHACL can be applied to validate data that is already available in the triplestore."*

Lição sobre OWL

Reproduz-se aqui, pela sua relevância, uma lição, em **Inglês**, sobre OWL, obtida de https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/owl-references-humans/:

Classes and Resources

One area in which OWL goes significantly beyond RDFS[14], is that it allows you to construct some fairly complex, but useful, relationships among classes. Some of the most common building blocks for doing so are listed below.

[14] https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/rdfs-vs-owl/

Classes and Resources		
Property	**Used to say that…**	**Example**
intersectionOf	…any instance of the first class is also an instance of all classes in the specified list	:Mother owl:intersectionOf (:Woman :Parent)
unionOf	…any instance of the first class is an instance of at least one of the classes in the specified list	:Parent owl:unionOf (:Mother :Father)
complementOf	…the first class is equivalent to everything not in the second class	:Parent owl:complementOf :NonParent
disjointWith	…the first class and second class have no members in common	:Man owl:disjointWith : Woman
equivalentClass	…the first class and the second class contain all the same members	:AdultFemaleHuman owl:equivalentClass :Woman
sameAs	…the first resource refers to the exact same thing as the second resource	:JimFromWork owl:sameAs :MyNeighborJim
differentFrom	…the first resource refers to something different from the second resource	:BobFromWork owl:differentFrom :MyNeighborBob

Properties

As with RDFS, properties in OWL are used to link things together. OWL provides a rich and complex vocabulary for saying things about these links.

Basic Property Types			
Kind of Property	**Used to say…**	**Example**	**Explanation**
DatatypeProperty	…that this property links to simple data values	ex:hasBirthday	This property links to a date, which is a simple data value
ObjectProperty	…that this property links to another resource	ex:hasSpouse	This property links to a person, which is another resource

Logical Relationships			
Kind of Property	**Used to say…**	**Example**	**Explanation**
TransitiveProperty	…that if this property links A to B, and B to C, then it also links A to C.	ex:tallerThan	If Ann is taller than Bob, and Bob is taller than Chuck, then Ann is taller than Chuck
SymmetricProperty	…that if the property	ex:hasSpouse	If Ann is Bob's

	relates A to B, then it always relates B to A as well.		spouse, then Bob is Ann's spouse too
AsymmetricProperty	...that if the property relates A to B, then it never relates B to A.	ex:tallerThan	If Ann is taller than Bob, then Bob can't be taller than Ann
ReflexiveProperty	...that this property always links something to itself.	ex:livesWith	Everybody lives with themselves
IrreflexiveProperty	...that this property never links something to itself.	ex:hasSpouse	Nobody is their own spouse
FunctionalProperty	...that this property only ever links to at most one thing.	ex:hasBirthday	You only have one birthday
InverseFunctionalProperty	...that the subject of this property is uniquely identified by the value of this property.	ex:hasDLNumber	I am the only person with my driver's license number

Properties Linking Properties		
Property	**Used to say that...**	**Example**
inverseOf	...the two properties are the inverse of each other. For example, if Ann's child is Bob, then Bob's parent is Ann.	:hasChild owl:inverseOf :hasParent
equivalentProperty	...two properties are exactly the same	:hasBirthPlace owl:equivalentProperty :hasBirthLocation

Restrictions

In RDFS, you could impose constraints on properties simply by specifying the domain and range. For example, if you asserted the range of :hasBirthday is xsd:date, then all statements using :hasBirthday should have an xsd:date as their object.

OWL lets you do this too, but it also introduces the concepts of restrictions, enumerations, and dataranges which are much more powerful.

(Note: In the Turtle RDF syntax, these constructs are usually specified using the bracketed blank node syntax, which we'll use below.)

Restrictions and Enumerations			
Parameter	**Used to say...**	**Example**	**Explanation**

cardinality min-cardinality max-cardinality	…that the property can have a certain number of values (objects).	:Automobile owl:equivalentclass [rdf:type owl:Restriction ; owl:cardinality "4"^^xsd:int ; owl:onProperty :hasWheel] .	All automobiles have 4 wheels (e.g., as opposed to a bicycle).
oneOf	…that all instances of a class come from the specified list	:BobsChildren owl:equivalentClass [rdf:type owl:Class ; owl:oneOf (:Bill :John :Mary)] .	The class 'BobsChildren' has the three items: Bill, John, and Mary
hasValue	…that all objects of that property have the specified value	:BobsChildren owl:equivalentClass [rdf:type owl:Restriction ; owl:onProperty :hasParent ; owl:hasValue :Bob] .	Each instance of BobsChildren has 'Bob' as the object of its :hasParent property.
someValuesFrom	…that at least one object of that property is a member of the specified class.	:Parent owl:equivalentClass [rdf:type owl:Restriction ; owl:onProperty :hasChild ; owl:someValuesFrom :Person] .	Any instance of the 'Parent' class has at least one child that is a Person
allValuesFrom	…that all objects of that property are members of the specified class	:Vegetarian owl:equivalentClass [rdf:type owl:Restriction ; owl:onProperty :eats ; owl:allValuesFrom :NonMeat] .	The class 'Vegetarian' is equivalent to the class of things that only eat non-meat.

Anexo III. Técnicas de Levantamento de Requisitos

O **Cartão de Volere** (em inglês, *Volere requirement shell)*.

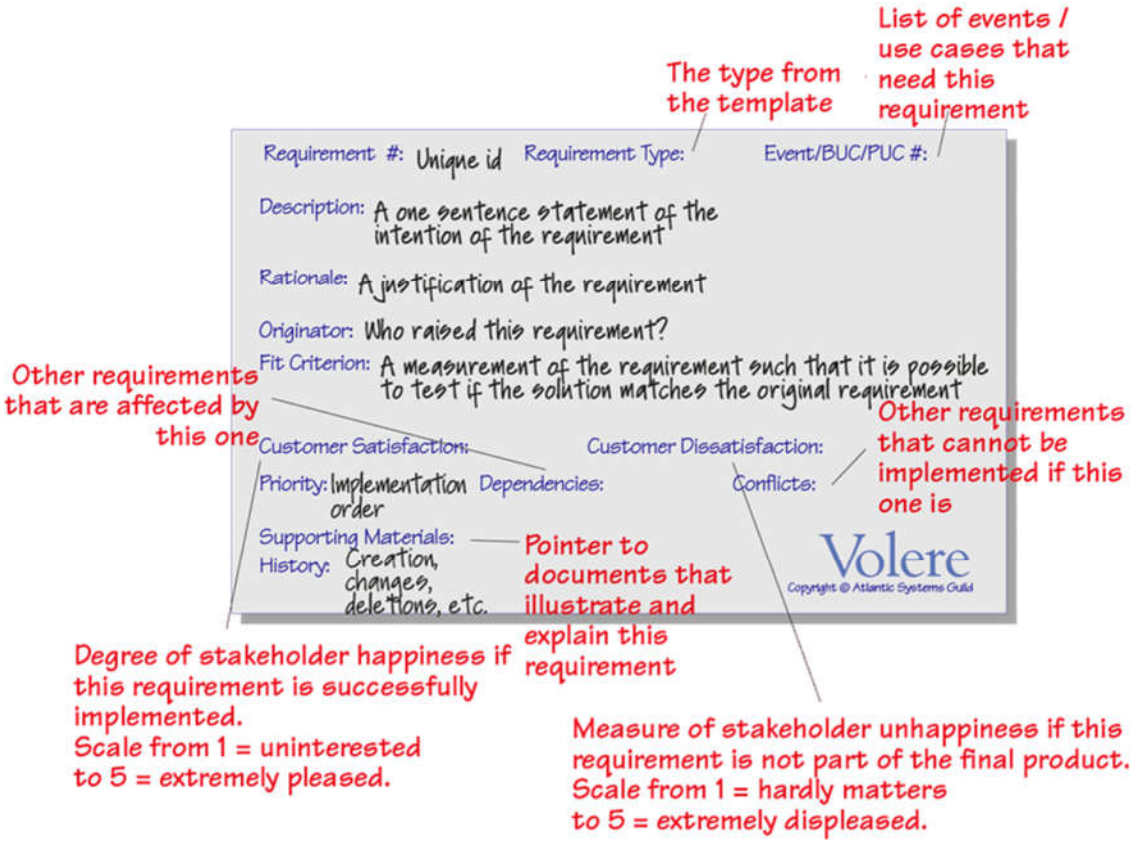


Figura 17 – Cartão de Volere comentado em Inglês

No contexto deste relatório, elaborou-se a seguinte representação deste cartão. Por uma questão de adequação a equipas com membros de diferentes línguas, optou-se pelo modelo com etiquetas em Inglês, ainda que neste documento o cartão seja preenchido em Português.

Requirement #:	Requirement Type:	Event/BUC/PUC #:
Description: (Sugestão: colocar como título da subsecção, para se ver no índice.)		
Rationale:		
Originator:		
Fit Criterion:		
Customer Satisfaction:	Customer Dissatisfaction:	
Priority:	Dependencies:	Conflicts:
Supporting Materials:		
History: Criado a <data>. [Implementado a <data>.]		

Requirement Type: Tipo de requerimento. Colocar em relação com as secções da lista de requisitos deste documento. Por exemplo, abreviando:

- Funcional
- Dados
- Aparência
- Usabilidade
- Desempenho
- Operacional
- Manutenção/Portabilidade/Suporte
- Segurança
- Cultural/Político
- Legal

Description: Descrição resumida, mas objetiva. Poderá ser colocada nomo título da secção para ser mostrada no índice do documento.

Valores para **Originator**:

- Estado da arte (padrão de facto que se impõe)
- Brainstorming (resultante da discussão em grupo)
- Introspeção (quando nasce da nossa imaginação individual)
- Entrevista com S...
- Sr./Sra. F...
- Padrão

History: Parece útil ir afixando uma etiqueta "Implementado", quando o requisito tiver fechado o seu ciclo de desenvolvimento. Informação obrigatória: se o requisito foi descartado, e nesse caso se originou outro(s).

Anexo IV. Nomenclatura de identificadores

Prefixos usados em variáveis, numa inspiração com origem nas práticas do Xbase:

- Simples:
  - o: "object". Ex.: oMyObject.
  - a: "array", tipicamente uma lista.
  - c: "character", "carateres" <=> string, str no Python. Exs.: cString, cNome.
  - l: "logic", "lógico" (booleano). Exs.: lSucesso, lOK.
  - n: "numeric", "numérico", seja inteiro ou real. Ex.: nValor.
  - t: "tuple" (cf. Python).
  - y: Prefixo para "dictionary" (cf. Python); se estabelecermos um paralelo com os objetos JavaScript, "o" também serviria.
  - x: "expression", qualquer tipo, o mesmo que "Unknown" no Python.
- Compostos (exs.:):
  - ac: Prefixo para "array of character string" <=> lista de strings; ex.: acNomes.
  - aa: *Array of arrays*.
  - oc: "object" ou "string".

Anexo V. Funções personalizadas no SPARQL do RDFLib

É possível, com o RDFLib, indicar uma função para personalizar cálculos durante as questões SPARQL. A documentação chama-lhes algo como *custom evaluation function,* e exemplifica (em https://rdflib.readthedocs.io/en/stable/_modules/examples/custom_eval.html), mas, como referido no capítulo 6, secção Personalização RDFLib do SPARQL, este exemplo pode não ser o mais profícuo, pelo que serve o presente anexo para exemplificar como personalizar o RDFLib SPARQL para normalizar uma string para pesquisa, convertendo para minúsculas e retirando diacríticos.

Lembra-se que a alternativa seria elaborar questões como a seguinte, para procurar "joao":

```
SELECT DISTINCT ?s ?p ?nome WHERE {
  ?s :nome|:designação ?nome .
  ?s ?p ?nome .
  BIND (replace(lcase(str(?nome)),'\\.',' ') as ?nome_low) .
  BIND (replace( replace( replace( replace( replace( replace( ?nome_low, 'ç','c'),
    'ã|á|â|à|ä','a'), 'é|ê|è','e'), 'í','i'), 'õ|ó|ô|ò|ö','o'), 'ú|ü','u') as
    ?nome_norm) .
  FILTER (regex(?nome_norm,"joao",'i'))
}
```

Ao invés, teremos uma questão assim:

```
PREFIX custom: <//custom/>
SELECT DISTINCT ?s ?p ?nome WHERE {
  ?s :nome|foaf:name|:nomePai|:nomeMãe|:nomeCônjuge|:email|:designação ?nome .
  ?s ?p ?nome .
  BIND(custom:SPARQL_NormNome(?nome) AS ?o_norm)
  FILTER (regex(?o_norm,"joao",'i'))
}
```

Para que a questão possa correr com sucesso, é necessário ter o seguinte código:

Em searchEng_SPARQL.py:

```
import rdflib.plugins.sparql
from SPARQL_rdflib_custom import SPARQL_NormNome

rdflib.plugins.sparql.CUSTOM_EVALS['SPARQL_NormNome'] = SPARQL_NormNome
```

E algures mais tarde a questão será executada… Mas, para funcionar, falta ainda o seguinte módulo, seja SPARQL_rdflib_custom.py:

```
# Módulos personalizados de normalização
from Lib.jg_base_iter import StdStrNorm
from myNames import NormGrafiaNome_Str

# Importações necessárias
from rdflib.namespace import Namespace
from rdflib.plugins.sparql.evaluate import evalPart
from rdflib.plugins.sparql.evalutils import _eval
from rdflib.plugins.sparql.sparql import SPARQLError
from rdflib.term import Literal, URIRef
```

```
# --- Criar antecipadamente o IRI para poupar processamento. Usado abaixo.
# Namespace('//custom/') é o namespace que convencionámos para esta técnica.
# Nas questões invoca-se via «custom:etc».

SPARQL_StdStrNorm_IRI = URIRef(Namespace('//custom/') + 'SPARQL_StdStrNorm')
SPARQL_NormNome_IRI = URIRef(Namespace('//custom/') + 'SPARQL_NormNome')
# ---

def SPARQL_NormNome(ctx, part) -> object:
    '''
    Função de personalização exemplo para uso numa questão SPARQL do RDFLib.

    :param ctx:     <class 'rdflib.plugins.sparql.sparql.QueryContext'>
    :param part:    <class 'rdflib.plugins.sparql.parserutils.CompValue'>
    :return:        <class 'rdflib.plugins.sparql.processor.SPARQLResult'>
    '''
    # Esta parte trata a implementação básica de adicionar novas funções:
    if part.name == 'Extend':
        cs = []
        # A informação é obtida e guardada, e passada por um gerador:
        for c in evalPart(ctx, part.p):
            lDefaultBehaviour = True

            if hasattr(part.expr, 'iri'): # Existe um IRI.
                # Isto irá verificar se foram mencionadas funções personalizadas.
                if part.expr.iri == SPARQL_NormNome_IRI:
                    # Cálculos personalizados.
                    # Primeiro, obtemos os parâmetros passados em variáveis,
                    # por exemplo ?nome.
                    argument1 = str(GetCustomParm(1, ctx, part, c))  # 1º parm

                    # Este objeto será guardado como valor de saída.
                    evaluation = Literal(NormGrafiaNome_Str(
                        argument1, lMaisNorm=False))

                    lDefaultBehaviour = False

                elif part.expr.iri == SPARQL_StdStrNorm_IRI:
                    argument1 = str(GetCustomParm(1, ctx, part, c))
                    evaluation = Literal(StdStrNorm(argument1))

                    lDefaultBehaviour = False
                else:
                    raise NotImplementedError()

            if lDefaultBehaviour: # Comportamento padrão.
                # Entra aqui em linhas em que não existe a função/funções em causa.
                evaluation = _eval(part.expr, c.forget(ctx, _except=part._vars))

                if isinstance(evaluation, SPARQLError):
                    raise evaluation

            cs.append(c.merge({part.var: evaluation}))
        return cs

    # Tratamento padrão de erro e retorno.
    raise NotImplementedError()


def GetCustomParm(nParmNumber, ctx, part, c):
    '''
    Camada que obtém um objeto com o valor do parâmetro nParmNumber.
    '''
    return _eval(part.expr.expr[nParmNumber-1],
```

```
                    c.forget(ctx, _except=part.expr._vars))
```

Anexo VI. Página de ajuda

Este anexo transcreve a página de ajuda da aplicação.

Índice

Introdução

Como iniciar

A aplicação baseia-se numa ontologia, carregada em memória num grafo. Para aceder à informação duma classe: opção Ontologia e escolher a classe, ou opção SPARQL e clicar no (-i-) ao lado.

A aplicação tem dois ambientes de navegação:

- por listagem (área SPARQL, área dos motores de busca da opção Procurar, e outras áreas),
- ou por apresentação aprimorada (como prettyprint duma coleção de triplos da ontologia).

As hiperligações permitem a navegação no grafo subjacente. A marca **(-i-)** permite passar ao ambiente de apresentação aprimorada (*prettyprint*) de um indivíduo ou classe, e depois mantém-se nesse ambiente (salvo exceções).

As questões SPARQL guardadas (cf. classe SparqlQuery) são acessíveis de vários modos:

- na área SPARQL: botão *Questões guardadas* e escolher uma, ou botão *Classe SparqlQuery* e depois separador *Indivíduos da Classe*;
- através da visualização duma classe da ontologia (como ao ir pela opção Ontologia, escolher a classe e depois acionar o separador *Questões da classe*, se houver questões expressamente associadas).

  A hiperligação "->[" permite levar a questão para a área SPARQL. Neste contexto, a questão será logo executada, a menos que seja do tipo *update* – ex.: *delete* ou *insert*

– estas questões não são imediatamente executadas por precaução, uma vez que agem sobre a base de dados.

Para inserir um indivíduo, aceder à informação da classe e usar o botão respetivo.

A remoção de triplos faz-se no formulário de alteração, ao limpar o conteúdo do campo e submeter.

A remoção de indivíduos tem templates na área SPARQL, nas questões guardadas. Não deve ser removido um ID que esteja referenciado, a menos que se planeie voltar a inseri-lo de novo.

Alguns botões mais usados têm teclas de atalho: basta pairar o rato por cima do botão para obter ajuda.

Lembre-se: Ctrl+clique abre a hiperligação num novo separador.

Apresentação aprimorada (*prettyprint*)

- No canto superior esquerdo, vem o nome ou designação (ou o ID, em último caso), seguido, caso existam, das datas de início, fundação ou nascimento (assinalado com '*' para distinguir) e fim ou óbito (assinalado com '+').
- Os nomes apresentados para propriedades (sejam atributos ou relacionamentos) passam por um processo que visa facilitar a leitura, daí poderem diferir do nome concreto dos termos na ontologia.

Formulários

Os nomes dos campos podem conter os seguintes sinais:

- * (asterisco): preenchimento obrigatório;
- ^ (acento circunflexo): assinala um relacionamento, logo requer como conteúdo o ID de um indivíduo da ontologia; nestes campos é possível usar a inferência dinâmica para calcular ID's (escrever no campo e selecionar a tecla shift+alt+c).

Campos com inferência

Para obter ID de indivíduos e poder mais tarde gerar as respetivas hiperligações para a suas páginas prettyprint, certos campos em formulários permitem solicitar a inferência **interativa** (como campos correspondentes a relacionamentos, que precisam de guardar um ID), enquanto outros permitem, adicionalmente, a inferência **dinâmica** (como campos correspondentes a atributos de texto multilinha, por exemplo Transcrição, sendo a inferência aplicada ao apresentar a informação na página prettyprint):

- Modo **interativo**. Na página de visualização aprimorada, ao lado de um separador colapsável elegível (ex.: Transcrição, Biografia, Notas), existe outro, **Alterar**, que além da edição da informação permite solicitar interativamente a inferência do sistema (no campo em foco, selecionar texto e clicar no botão Inferir). O sistema retorna etiquetas de navegação para todos os indivíduos inferidos. Pode inferir-se sobre ID de indivíduos, nomes (prefixados e sufixados como descrito abaixo), e nomes associados

via graus de parentesco – exs.: "tio Serafim", "Maria, mãe do João", "José e filh@" (filhos de ambos os sexos), "José e filhos" (filhos do sexo masculino), "pais de Maria".

- Modo **dinâmico**. Este modo reconhece cadeias entre chavetas duplas – {{ <texto> }} –, sendo de preferência um ID ou nome completo, opcionalmente prefixado com a classe e o caráter "!", opcionalmente sufixado com um ano entre parêntesis. Esta inferência reage melhor com referências correspondentes a ID de indivíduos – senão, ao usar nomes e outras informações, torna-se possivelmente menos eficaz por ser sensível a alteração aos dados guardados, e menos eficiente por exigir mais processamento nalguns casos. Este modo retornará no máximo uma hiperligação. Exemplos:
    - {{:João_etc_d1969}} <-- ID de indivíduo, a forma mais segura de referência,
    - {{ Pessoa!João Alex Alexandre (1911) }} <-- Nome com data ou ano de nascimento (para desambiguar),
    - {{Pessoa! M. Joana Pereira}} <-- Nome com inicial explícita.
    - {{Pessoa!Maria Albertina Pereira da Silva}} <-- Nome completo.
    - {{ Org! Módula C }} <-- Nome da organização; os espaços na sintaxe são indiferentes.

  Se a inferência não tiver sucesso, o texto é apresentado entre {{}} tal e qual: a preto se não foi possível reconhecer a sintaxe, a vermelho se reconheceu mas não unificou com nenhum indivíduo da ontologia.

Inferência interativa

Para a inferência sobre pessoas (cf. :Pessoa), é útil que tenham a data ou ano de nascimento preenchidos, em particular no ID (cf. numérico depois de "_d"). A inferência tenta primeiro descobrir graus de parentesco, depois analisa ID, e depois procura por nome.

Dicas para alguns casos especiais, como exemplo.

Escrever	**Para inferir**
avós, ou *av@*	Todos os avós, paternos e maternos, masculinos e femininos.
avós paternos	Todos os avós paternos, masculinos e femininos.
avô paterno	Todos os avós paternos, masculinos.
avô, ou *avó*	Todos os avós, respetivamente masculinos e femininos.
irmãos, ou *irm@*	Todos os irmãos, masculinos e femininos.
irmã, ou *irmãs*	Todas as irmãs.
irmão	Todos os irmãos masculinos.
pais	Ambos os progenitores.

Procurar

Esta opção permite diferentes contextos de procura. Mesmo nas procuras Fácil e Triplos não é obrigatório preencher todos os campos (mas cada campo vai procurar na respetiva posição dos triplos).

Fácil: É a mais abrangente de todas, procurando cadeias normalizadas e subcadeias. No predicado, se prefixar "!" a pesquisa é mais exata sobre essa posição dos triplos. Tem particularidades de pesquisa genealógica. Atenção a camelCase, porque "PEssoa" irá procurar "p" + "essoa" – experimente a opção Debug. Não obtendo aqui o resultado esperado, tente a procura Triplos.

Nome: Ao indicar <letra>"." a procura atende especificamente a nomes com essa abreviação literalmente explícita. Ex.: Existem "João M. Silva" e "João Manuel Silva" => procurar "joao m. silva" só encontra o 1º.

Morada: a procura, além do arruamento, analisa ainda a localidade e código postal; para procurar na classe Local, usar a procura Fácil, ou Nome, e prefixar «Local!» – ex.: «Local!Antunes Guimarães».

ID: Pesquisa sobre ID de indivíduos. Pode efetuar procuras Nome'_d'ano4dígitos (neste, os últimos 2 dígitos são opcionais e podem ser substituídos por "." ou "x"). Exs.: josé_m_silva_d1969, "josé_m_silva_d196.", josé_m_silva_d196x, josé_m_silva_d19xx

Triplos: É a pesquisa mais literal de todas. Ao procurar cadeias de carateres, como nomes, poderá ser necessário envolver com aspas.

SPARQL

Esta opção permite executar queries SPARQL e navegar na ontologia. Sugere-se a sintaxe Turtle.

Arranca com a listagem de classes da ontologia (ontoclasses). À esquerda, uma hiperligação para listar os indivíduos (instâncias) da classe. A hiperligação (-i-), sempre que surja, salta para uma visualização mais aprimorada.

Gravar questão

Associar a uma classe: Se a descrição do utilizador contiver no início um nome de classe da ontologia (':'<nome da classe> – exemplo :Doc), a questão será associada a essa classe, e depois passível de listagem no template /oclasseTp, no separador "Questões da classe", onde poderá ser mandada executar (cf. caso da classe :Doc).

DSL

Esta opção permite carregar ficheiros para ingestão de dados na ontologia. Primeiro, deve escolher um ficheiro a Carregar. Depois, mandar Analisar, e só então poderá pedir para Gravar os novos triplos. A aplicação assume que todos os indivíduos têm ID diferentes.

Ontologia

Esta opção apresenta a página de estrutura da ontologia, com separadores colapsáveis e depois hiperligações de acesso. Disponibiliza-se as árvores de Classes (owl:Class), Atributos (owl:DatatypeProperty), Relacionamentos (owl:ObjectProperty) e Anotações (owl:AnnotationProperty). São apresentadas classes referenciadas, mesmo que não definidas na ontologia (ex.: foaf:Person).

Utilidades

Esta opção apresenta uma página para disponibilizar utilidades e utilitários diversos.

Realça-se os botões Genealogia, que inserem automaticamente, em lote, indivíduos e triplos de genealogia, baseado nos pais e cônjuge de cada pessoa – são inseridos diversos graus de parentesco, em particular muito úteis à inferência.

Acerca de Ancestors Note Book

Um protótipo configurável e reutilizável, como prova de conceito sobre uma ontologia para pessoas, documentos, locais, e suas derivações.

ANB = Ancestors Note Book =

Ontologia para
Memorabilia e Genealogia
Relações humanas e Ambiente,
+ Filosofia Wiki
+ Inferência Interativa
+ Inferência Dinâmica
+ SPARQL

Aplicação web, sobre Flask, monoposto.

A *framework* de apresentação HTML é i Jinja2, de CSS é o Bootstrap 4.

A arquitetura de ontologia efetua persistência num ficheiro de texto em sintaxe Turtle, e é carregada num grafo em memória, internamente gerido via SPARQL e rdflib. Permite-se interligação com ontologias externas, ao definir os respetivos prefixos. A ontologia é expansível.

As relações humanas incluem genealogia de pessoas, graus de parentesco e relacionamentos sociais.

O ambiente previsto é social, institucional e geográfico, incluindo pessoas, organizações e lugares, e respetivos relacionamentos.

Memorabilia e História prevê o registo e referência de eventos, fotografias, documentos, histórias de família, artigos sobre qualquer assunto.

Acolhe-se uma filosofia Wiki, no sentido em que proporciona uma apresentação gráfica (suportada em mardkdown ou html), navegação por hiperligações, edição.

A inferência é interativa e/ou dinâmica, sobre classes, ID, nomes e datas.

Créditos

- Autor: João Manuel Pós de Mina Grenhas, entre 2020 e 2022.
- Orientador: Professor José João A. G. Dias de Almeida, Dep. de Informática, UMinho.
- Coorientador: Professor Rui Mendes, Departamento de Informática, UMinho.

Ícones e imagens de fundo

- Incluindo o ícone principal, diversas imagens foram obtidas da internet e modificadas, mas ainda assim poderão estar sujeitas a direitos de autor.
- srip from www.flaticon.com.
- Dimitry Miroliubov from www.flaticon.com.
- MD Delwar Hossain.
- free background photos from pngtree.com.