Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

SDLink

Interfaces com o utilizador para ferramentas baseadas em tecnologias da web semântica: uma aplicação às ciências da vida

Pedro Reis

Relatório de projecto realizado no âmbito de Projecto e Seminário do curso de Licenciatura em Engenharia Informática e de Computadores sob orientação de Ana Teresa Freitas (IST) e Cátia Vaz (ISEL).

Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

SDLink

Interfaces com o utilizador para ferramentas baseadas em tecnologias da web semântica: uma aplicação às ciências da vida

Relatório de projecto realizado no âmbito de Projecto e Seminário do curso de Licenciatura em Engenharia Informática e de Computadores

Setembro de 2010

Autor:

Pedro Miguel Baptista dos Reis, nº 30758

Orientadores:

Ana Teresa Correia de Freitas – IST

Cátia Raquel Jesus Vaz – ISEL

Resumo

Este projecto visa o desenvolvimento de interfaces com o utilizador para sistemas de informação semânticos. A aceitação deste projecto, pela comunidade científica nas áreas das Ciências da Vida, é considerada prioritária nesta implementação. Sendo o *Microsoft Excel*, a ferramenta de eleição destas comunidades, foi utilizada a *Microsoft .NET Framework* para estender o *Excel* e permitir a sua integração como parte de um sistema de informação semântico.

A solução implementada permite que o *Excel* continue a ser utilizado pelos cientistas garantindo a integração dos dados num sistema de informação semântico. Isto permitirá a análise transversal dos dados recolhidos e facilitará a consulta destes últimos por outras instituições.

Esta implementação é dinamicamente adaptável às ontologias criadas no sistema e recorre às normas definidas pelo *World Wide Web Consortium* para a *Web* Semântica. Isto tem a vantagem de permitir adaptar este projecto a qualquer repositório de triplos, com qualquer nível de inferência associado, garantindo assim a independência de implementações concretas no lado do servidor.

Agradecimentos

Gostaria de agradecer:

À professora Cátia Vaz, pela oportunidade de participar neste projecto e pelo apoio, esclarecimento e compreensão nas diversas dúvidas expostas.

À professora Ana Teresa Freitas, pela motivação, mesmo quando as tarefas pareciam herculeanas. Pela confiança depositada e pelo pragmatismo na orientação do meu trabalho.

À Fundação para a Ciência e a Tecnologia e à Dr.ª Susana Vinga pelo apoio e financiamento através projecto DynaMo - PTDC/EEA-ACR/69530/2006.

A todos os professores do ISEL, de quem tive a sorte de ser aluno, e que permitiram a concretização do meu potencial.

À minha esposa, Carla, pelo apoio incondicional nesta viagem cuja constante tem sido a aprendizagem.

Conteúdos

1 Introdução 1

1.1 Motivação 2

1.2 Objectivos 3

1.3 Estrutura do relatório 4

2 Conceitos 5

2.1 RDF 6

2.2 RDFS 9

2.3 OWL 11

2.4 SPARQL 13

3 Sistema de informação semântico 17

3.1 Arquitectura 17

3.1.1 Arquitectura do sistema 17

3.1.2 Arquitectura da solução 18

3.2 Implementação da solução 20

3.2.1 Tipo de solução VSTO 20

3.2.2 Execução assíncrona 21

3.2.3 Representação da ontologia em memória 23

3.2.4 Tecnologias 24

3.2.5 Camadas funcionais 25

3.2.6 Configuração da aplicação 39

3.2.7 Modelo de instalação 40

4 Contribuições adicionais e avaliação 41

4.1 Contribuições adicionais 41

4.1.1 Infra-estrutura de suporte 41

4.1.2 Metodologias 42

4.2 Avaliação da solução 44

4.2.1 Vantagens da *Web* Semântica 44

4.2.2 Vantagens da Implementação 45

4.2.3 Limitações da *Web* Semântica 46

4.2.4 Limitações do *Excel* 51
5 Conclusões e trabalho futuro 55
5.1 Conclusões 55
5.2 Desafios e dificuldades 56
5.3 Trabalho futuro 56

Lista de Figuras

Figura 1 – Pilha de normalizações da *Web* Semântica 5

Figura 2 – Representação em grafo de triplos.....7

Figura 3 – Diagrama de classes UML com taxonomia de seres vivos 10

Figura 4 – Diagrama parcial das extensões entre RDF e OWL 13

Figura 5 – Arquitectura global do sistema.....17

Figura 6 – Arquitectura da solução.....18

Figura 7 – Controlo *MultiPick*.....19

Figura 8 – *Ribbon* implementado para a interface em *Excel*.....20

Figura 9 – Problemas na partilha de ficheiros *Excel*.....21

Figura 10 – Fluxo de execução de tarefas aplicacionais 22

Figura 11 – Exemplo de extensão do formulário FrmAsyncBase 23

Figura 12 – Diagrama de classes UML da representação em memória de ontologias 23

Figura 13 – *Workbench* da *framework Sesame* 24

Figura 14 – Quadro de operações da *RESTFul API* para *endpoints* SPARQL 25

Figura 15 - Quadro de operações da *RESTFul API* para *endpoints* de triplos 26

Figura 16 – Interface e respectiva implementação do analisador sintáctico.....28

Figura 17 – Diagrama de sequência para a construção da ontologia em memória 29

Figura 18 – Primitivas suportadas e excluídas da implementação 30

Figura 19 – Diagrama da classe *Mapping* 33

Figura 20 – Folha de cálculo principal de persistência 34

Figura 21 – Disposição de classes e suas propriedades 34

Figura 22 – Formulário de validação 35

Figura 23 – Utilização de múltiplos livros no *Excel*.....36

Figura 24 – Formulário do *Excel* para criação de regras de validação 37

Figura 25 – Lista de validação para valores booleanos 37

Figura 26 – Resultado normal da directiva para descrição de recursos 38

Figura 27 – Excepção no *triplestore* por incoerência de tipos.....39

Figura 28 – Formulário de configuração 40
Figura 29 – Infra-estrutura de suporte ao desenvolvimento e testes 42
Figura 30 – Solução implementada na versão 2003 do *Excel* 43
Figura 31 – Consulta de propriedades em SQL e SPARQL 47
Figura 32 – Propriedade transitiva em OWL 47
Figura 33 – Visualização de ontologia em *Protégé* através do *OWLViz* 50
Figura 34 - Visualização de ontologia em *Protégé* através do *OntoGraf* 51
Figura 35 – Especificações principais do *Excel* por versão 51
Figura 36 – Expressão da quantidade máxima de referências por célula 52

Lista de Listagens

Listagem 1 - Representação de triplos em RDF 8

Listagem 2 - Exemplo de taxonomia de seres vivos 10

Listagem 3 - Exemplo de condição necessária 12

Listagem 4 - Exemplo de condição necessária e suficiente 12

Listagem 5 - Exemplo de expressão SPARQL 14

Listagem 6 - Utilização da cláusula *union* em SPARQL 15

Listagem 7 – Utilização de nós anónimos em SPARQL 16

Listagem 8 – Exemplo de resultados SPARQL em XML 27

Listagem 9 - Exemplo de incoerências de tipo em RDF 38

Listagem 10 – Ficheiro de configuração 39

Listagem 11 – Excerto da ontologia dos vinhos da W3C 48

Listagem 12 – Exemplo de propriedade sem *range* 48

Listagem 13 – Caso excepcional em condição necessária e suficiente 49

1 Introdução

O conhecimento, ao longo da história da humanidade, tem-se encontrado disperso, segregado e pouco organizado. Estes aspectos são uma consequência directa da evolução natural da espécie humana, dificilmente colmatáveis na antiguidade. Actualmente, graças à evolução das infra-estruturas de comunicação digital e dos sistemas de informação, o impacto da dispersão foi minorado nas empresas, mas continuam a ser notórios os efeitos da segregação do conhecimento nas instituições dedicadas à investigação. Por sua vez, a desorganização é resultante do constante estudo do universo que nos rodeia, sendo perceptível em aspectos como: ausência de definições formais, diversidade de terminologia para o mesmo conceito e não sistematização do conhecimento adquirido.

A *Web* Semântica [1] tem desempenhado um papel importante na interoperabilidade entre disciplinas, quebrando a usual barreira da segregação do conhecimento, um exemplo disto foi referido por Tim Berners-Lee durante uma palestra realizada em 2009 [2]. Para responder à questão: "Quais as proteínas envolvidas na transdução de sinal e que estão relacionadas com neurónios piramidais?" não só foi necessário conhecimento de duas disciplinas, nomeadamente Bioquímica e Neurologia, mas também vencer as barreiras da heterogeneidade dos dados e representação de conhecimento.

O objectivo de criar um sistema de informação global, não é concretizável apenas com a idealização da *Web* Semântica, é necessário resolver o problema da desorganização do conhecimento. A Engenharia do Conhecimento tem exactamente esse propósito e trata da definição de metodologias e concepção de ferramentas para a representação do conhecimento, sendo a criação de ontologias o principal objectivo desta disciplina. As ontologias permitem definir uma estrutura formal para os conceitos existentes num determinado domínio de discurso, sendo a linguagem utilizada para criar ontologias na *Web* Semântica a *Web Ontology Language* (OWL) [3]. Para além disto, as ontologias permitem associar semântica aos dados que estão dispersos pela *Web*, criando um potencial para inquirição que se estende muito além daquilo que é disponibilizado nos principais motores de pesquisa actuais, e.g. *Google*, *Yahoo* e *Bing*. Este potencial surge do facto de ser possível a utilização de operadores matemáticos, bem como, a indicação de quais as propriedades intervenientes na pesquisa.

É comum serem utilizadas as tecnologias que constituem a *Web* Semântica em disciplinas das Ciências da Vida, tais como, Biologia, Genética, Medicina e principalmente na Bioinformática. Isto deve-se ao facto de que o conhecimento nestas áreas estar em constante evolução, o que implica alterações às ontologias, principalmente em matéria de taxonomia de classes. Estas alterações são equivalentes num sistema de base de dados relacional à criação de novas tabelas, resultando na alteração das relações existentes e obrigando à migração dos dados para

o novo modelo. Estes processos são morosos e potenciam o erro humano, como tal, é evitado o uso de sistemas de bases de dados relacionais nestas disciplinas.

Os sistemas de informação semânticos são constituídos por um ou mais repositórios, designados de *triplestore*. Nestes repositórios, os dados encontram-se na forma de afirmações[1] compostas pelos sujeito, predicado e objecto, as quais também se designam por triplos. Isto proporciona alterações taxonómicas a qualquer ontologia, alterando apenas uma afirmação ou triplo e sem implicar a migração dos dados.

Realizar um projecto vocacionado para a compreensão das Ciências da Vida, como é o caso deste projecto, implica necessariamente o recurso às tecnologias da *Web* Semântica. Adicionalmente, para casos em que os dados estão em permanente alteração de tipo e volume, é importante a realização de interfaces com o utilizador que facilitem o processo de carregamento desses dados para o repositório semântico, preferencialmente validando se estão de acordo com as ontologias definidas para os domínios de conhecimento em causa. Como as ontologias nas disciplinas de Ciências da Vida estão em constante evolução, é fundamental que as interfaces com o utilizador sejam adaptáveis às alterações ocorrentes nas mesmas. Esta situação implica necessariamente processos de engenharia reversa que procedam ao mapeamento entre a representação do conhecimento existente nas ontologias e a apresentação gráfica adequada na interface com o utilizador.

1.1 Motivação

Mesmo com a evolução das infra-estruturas de comunicação digital e adopção destas nas instituições de investigação, continua a existir dispersão do conhecimento cujas causas variam de acordo com as regras organizacionais implementadas. No caso das instituições dedicadas ao estudo de disciplinas das Ciências da Vida, o *Excel* [4] é normalmente a aplicação mais utilizada para o armazenamento dos dados resultantes de experiências e na análise desses dados. Isto implica que, estas instituições disponham dos dados, mas não de uma forma centralizada e que permita análises globais aos mesmos, pois estes encontram-se dispersos em múltiplos livros[2] de *Excel*.

Com a evolução dos equipamentos de amostragem destinados a estudos científicos nas disciplinas mencionadas, houve um aumento significativo de dados obtidos por cada experiência realizada, tornando mais gravoso e evidente o problema da dispersão. Adicionalmente, ao

[1] Termo em inglês é *statement*.

[2] Termo em inglês é *workbook* e corresponde a um ficheiro *Excel*.

existirem contextos associados às experiências realizadas, os dados são segregados pelo contexto de cada experiência, tornando mais difícil o inevitável processo de integração de todos os dados institucionais num único repositório.

Foram sendo criados, nestas comunidades científicas, hábitos de trabalho que invalidam algumas mudanças de paradigma na implementação de um sistema de informação. Essencialmente a maioria dos cientistas são pragmáticos no que respeita à análise dos resultados das experiências realizadas, o que determina à partida o *Excel* como ferramenta de trabalho.

Urge então, implementar um sistema de informação que usufrua das qualidades dos sistemas de informação semânticos, mas não obste a utilização do *Excel* como ferramenta de trabalho nas comunidades já referidas. A implementação de um sistema com estas características permitirá, aos seus utilizadores, um substancial contributo na eliminação da dispersão e segregação dos dados institucionais, permitindo a futura obtenção de novo conhecimento com base no conhecimento adquirido, por análise transversal dos dados. Adicionalmente, um sistema com estas características permitirá um avanço significativo naquela que é a visão de Tim Berners-Lee para um sistema de informação global: a *Web* Semântica.

1.2 Objectivos

Este projecto teve como principal objectivo a criação de interfaces com o utilizador para sistemas de informação semânticos, tendo sido considerada prioritária a aceitação da solução por parte dos utilizadores finais. Tal como já referido, os utilizadores alvo desta implementação são Biólogos e Médicos, para os quais o *Excel* é a ferramenta de eleição para manipulação e análise de dados. Desta forma, a implementação adoptada estende as funcionalidades do *Excel* através do uso da *.NET Framework* [5] versão 3.5 e tecnologias de interoperabilidade, como o *Visual Studio Tools for Office* (VSTO) [6] e os *Primary Interop Assemblies* (PIAs) [7].

Este projecto faz parte de um sistema desenvolvido e implementado por um colectivo de pessoas ligadas à investigação na área da Bioinformática. Para o desenvolvimento do sistema, ocorreram um conjunto de actividades para a coordenação da equipa de desenvolvimento, tendo sido também objectivos deste projecto a adaptabilidade e autonomia do autor em função das decisões tomadas nas reuniões de projecto. Neste sentido, objectivos adicionais foram sendo definidos no decorrer de todo o trabalho, dos quais se destacam:

1. Aferição de compatibilidade entre versões actuais e futuras do *Excel*.
2. Determinação e implementação de recursos infra-estruturais de suporte ao desenvolvimento.

3. Definição de métodos organizacionais para a reutilização da solução.
4. Definição dos procedimentos de instalação da solução.
5. Adaptabilidade a múltiplos repositórios de dados na forma de triplos[3].
6. Independência do nível de inferência utilizado no servidor.
7. Suporte e manutenção às versões 2003 e 2007 do *Excel*.

A análise das ferramentas utilizadas neste projecto determinou a divergência entre as funcionalidades inicialmente previstas e as implementadas. Desta forma, os objectivos apresentados reflectem também o impacto das ferramentas no desenvolvimento deste trabalho.

1.3 Estrutura do relatório

Este documento é composto por sete capítulos, sendo o presente capítulo introdutório à problemática da informação e conhecimento, descrevendo as motivações e objectivos que levaram à concretização deste projecto.

Ao longo do presente relatório, os termos "*SDLink*" e "*sistema*" dizem respeito à totalidade do sistema de informação semântico, implementado pela equipa de desenvolvimento que o autor integra. Os termos "solução", "projecto" e "implementação" dizem respeito à solução estudada e implementada pelo autor no contexto da disciplina de Projecto e Seminário. O termo "aplicação" refere-se à conjugação do *Excel* com a solução desenvolvida pelo autor.

O capítulo 2 descreve de forma sucinta os conceitos necessários à leitura dos restantes capítulos e, contextualiza o problema do ponto de vista tecnológico. Aborda as tecnologias que constituem a pilha da *Web* Semântica, recorrendo a exemplos do quotidiano para ilustrar a aplicabilidade dos conceitos.

O capítulo 3 apresenta as arquitecturas do sistema e da solução, detalha o trabalho realizado neste projecto, pormenorizando as decisões tomadas para a implementação. São descritos os pilares funcionais da solução, sendo estes reflectidos na arquitectura adoptada.

O capítulo 4 expõe contribuições organizacionais resultantes da realização deste projecto e refere quais as virtudes e limitações da solução e das tecnologias utilizadas.

O capítulo 5 apresenta as conclusões formuladas com o desfecho do projecto, fazendo uma análise crítica através das principais dificuldades encontradas. Termina com a descrição de actividades que complementam o trabalho realizado, permitindo a continuidade deste projecto.

[3] Termo em inglês é *triplestore*.

2 Conceitos

Para a realização deste projecto foi necessário o estudo de um conjunto significativo de tecnologias que constituem a *Web* Semântica. Estas tecnologias têm como propósito construir uma base de trabalho normalizada para modelar o universo que nos rodeia e inferir novo conhecimento a partir do conhecimento já adquirido. As ferramentas e normalizações que constituem a *Web* Semântica estão organizadas, em forma de pilha, tal como ilustrado na Figura 1.

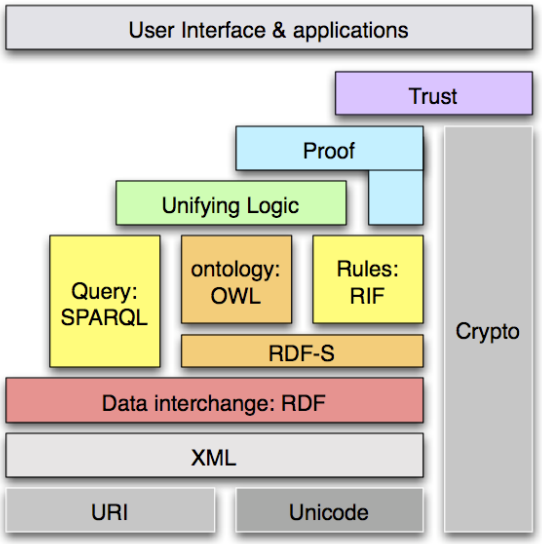

Figura 1 – Pilha de normalizações da *Web* Semântica[4]

Na base da pilha da *Web* Semântica encontra-se o *Unicode* [8] que permite a representação dos símbolos de alfabetos modernos à escala mundial. Adicionalmente, consta da base desta pilha, o mecanismo de *Uniform Resource Identifier* (URI) que permite a identificação de recursos na *Web*. Alguns exemplos de recursos são ficheiros de áudio ou vídeo, fotografias e documentos.

Imediatamente acima na pilha da *Web* Semântica, encontramos a *Extensible Markup Language* (XML) [9]. O uso do XML permite a interoperabilidade entre diferentes aplicações que partilhem dados com formatos distintos. Isto é possível graças à inclusão de metadados[5] conjuntamente com os dados aplicacionais no mesmo ficheiro XML, permitindo que uma determinada aplicação localize no ficheiro as propriedades para as quais pretende realizar determinada acção, ou seja, representação gráfica, persistência em base de dados, entre outras.

[4] Retirado de http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/SemWebStack-tbl-2006a.png

[5] Dados que descrevem outros dados.

Outro aspecto importante do XML é o mecanismo de *namespaces* que permite a desambiguação de termos iguais. Por exemplo, o termo `banco` pode referir-se a uma peça de mobiliário ou a uma instituição financeira. Com o recurso a *namespaces* é possível prefixar o termo de acordo com o domínio de discurso, possibilitando termos como `mob:banco` e `fin:banco`, onde o prefixo `mob` denota mobiliário e `fin` denota o domínio financeiro. Isto é particularmente útil na concepção de ontologias, onde é vulgar encontrarem-se representados termos iguais com significados diferentes.

Uma ontologia pode ser construída recorrendo a diferentes níveis de formalismo na definição de um modelo para representar o universo que nos rodeia. No entanto, na pilha da *Web* Semântica considera-se a camada de ontologia como um modelo formal que descreve indivíduos[6], classes de indivíduos, propriedades de indivíduos, propriedades de classes, associações entre indivíduos, associações entre classes, relações taxonómicas entre classes, relações funcionais e transitivas, e por último, restrições sobre propriedades.

Considerando D o domínio de discurso e R a relação, são apresentadas de seguida as definições formais para relação funcional e transitiva.

- Relação funcional: $\forall\, x, y, z \in D : xRy \wedge xRz \Rightarrow y = z$
- Relação transitiva: $\forall\, x, y, z \in D : xRy \wedge yRz \Rightarrow xRz$

As especificações seguintes ao XML na pilha da *Web* Semântica permitem ir construindo níveis adicionais de formalismo na criação de ontologias, capitalizando nas virtudes das camadas anteriores, sendo estas descritas de seguida.

2.1 RDF

O RDF, *Resource Description Framework* [10], é uma recomendação W3C[7] concebida originalmente para normalizar descrições em recursos *Web*. Verificou-se, no entanto, que o RDF é também apropriado para descrever recursos cuja proveniência não seja a *Web*.

Descrever um recurso é realizar um conjunto de afirmações sobre o mesmo. Estas afirmações são também conhecidas como triplos, devido a serem constituídas por três partes, sujeito-predicado-objecto, podendo ser representadas na forma de um grafo dirigido, em que cada nó representa um sujeito ou objecto e cada arco representa um predicado, sendo o arco

[6] Equivalente à instância de uma classe

[7] *World Wide Web Consortium*

dirigido no sentido sujeito-objecto. A Figura 2 ilustra a representação em grafo de duas afirmações sobre o álbum *The Blue Notebooks*.

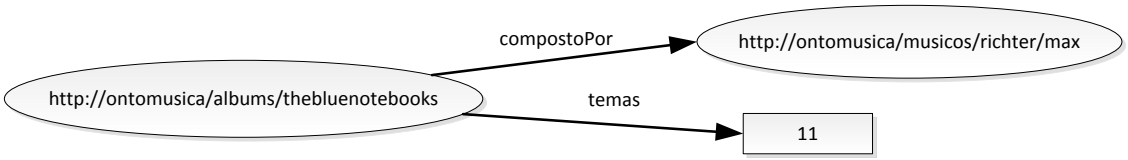

Figura 2 – Representação em grafo de triplos

A partir do grafo ilustrado na Figura 2, é possível determinar que são realizadas duas afirmações, nomeadamente:

1. *The Blue Notebooks* é composto por *Max Richter*.
2. *The Blue Notebooks* tem onze temas.

O rectângulo contendo o valor onze denota um literal, enquanto as elipses denotam recursos. Um factor importante no RDF é a identificação de recursos, que neste exemplo é realizada através de um *Uniform Resource Locator* (URL). Um URL é uma das formas de criar um URI, sendo a forma mais generalizada de identificar e aceder a um recurso na *Web*. Apesar do URL permitir o acesso ao recurso na *Web*, alguns recursos descritos em RDF têm apenas o URL como forma de identificação, não sendo possível o seu acesso. Isto ocorre com frequência em *triplestores*, pois importa apenas identificar o recurso no repositório e o acesso a este último é feito através de interfaces que o repositório disponibiliza. Exemplos de interfaces para *triplestores* são *Application Programming Interfaces* (APIs) em linguagens como o *Java* [11], ou APIs em estilos arquitecturais como o *Representational State Transfer* (REST) [12], designadas de *RESTFul APIs*.

O facto de serem utilizados URLs para identificação de recursos não garante a identificação unívoca desse recurso na *Web*. De facto esta característica é designada como *No Unique Names Assumption*. Esta assunção de não unicidade de nomes, surge por ser impossível garantir na *Web* que todas as pessoas usem o mesmo URI para identificar o mesmo recurso. Através do uso de um *triplestore* é possível existir identificação unívoca de recursos em cada repositório. Neste projecto, é considerada a existência de identificação unívoca de recursos, uma vez que o propósito é um sistema de informação institucional no qual é possível gerir os identificadores para cada recurso contidos num repositório.

Outra forma de representar os triplos é através de notações tais como o XML, *Notation 3* (N3) [13] e *Terse RDF Triple Language (Turtle)* [14]. Estas representações permitem a seriação dos triplos de RDF para armazenamento em ficheiro ou para transferência entre computadores. A notação utilizada neste projecto é XML, cuja sintaxe está especificada em [15]. Esta sintaxe tem

o nome oficial de RDF/XML para a qual é apresentado na Listagem 1 um exemplo que confere com o ilustrado na Figura 2.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:mus="http://ontomusica#">

  <rdf:Description rdf:about="http://ontomusica/albums/thebluenotebooks">
    <mus:compostoPor rdf:resource="http://ontomusica/musicos/richter/max" />
    <mus:temas rdf:datatype="http://www.w3.org/2001/XMLSchema#int">11</mus:temas>
  </rdf:Description>

</rdf:RDF>
```

Listagem 1 - Representação de triplos em RDF

O RDF pode ser interpretado a três níveis distintos:

1. Ao nível da sintaxe, tratando-se apenas de um documento XML, por exemplo, contendo elementos em forma de árvore.
2. Ao nível da estrutura é um conjunto de triplos na forma sujeito-predicado-objecto.
3. Ao nível semântico são grafos dirigidos com semântica pré-definida associada aos nós e arcos.

É usual denominar-se "modelo RDF" ao nível semântico e "documento RDF" ao nível da sintaxe. A razão pela qual se designa de modelo o nível semântico, reside na possibilidade de se construírem ontologias de nível elementar, ou seja, podermos representar relações entre recursos, mas estes últimos poderem não representar necessariamente conceitos. A designação de documento ao nível da sintaxe denota que este é apenas um formato de seriação.

Outros dois aspectos importantes do RDF são os nós anónimos[8] e o mecanismo de reificação. Os nós anónimos são os que conceptualmente não têm identificador associado, embora nos formatos de seriação é possível definir um identificador que é válido apenas no contexto do ficheiro. Regra geral um nó anónimo é reconhecido por utilizar os símbolos sublinhado (_) e dois pontos (:) como prefixo, por exemplo o nó `_:node155vqsc0bx13` é um nó anónimo possível num *triplestore*.

Os nós anónimos são úteis para realizar afirmações sobre algo desconhecido, podendo relacionar o nó anónimo com outros nós que representem recursos ou valores conhecidos. Um exemplo comum ocorre quando alguém tenta identificar um actor de cinema, mas não se recorda do seu nome, referindo apenas filmes em que o actor participou. Nestes casos, os nós

[8] Termo original é *blank nodes*.

anónimos permitem que o sistema de informação contenha dados sobre algo que é desconhecido e que mais tarde pode ser identificado, deixando esse nó de ser anónimo.

O mecanismo de reificação permite fazer afirmações sobre afirmações, evitando conclusões sobre recursos que representem opiniões. Por exemplo, a afirmação: "A Alice diz que o Bruno é amigo do Carlos", não representa necessariamente que o Bruno e o Carlos sejam amigos, pois traduz-se apenas numa opinião da Alice. Como tal, o RDF dispõe das primitivas `rdf:Statement`, `rdf:subject`, `rdf:predicate` e `rdf:object` que permitem construir reificações.

2.2 RDFS

O RDFS, *Resource Description Framework Schema*, é uma extensão ao RDF com a qual podem ser definidos conceitos e taxonomias, criando uma ontologia mais rica. Para isto, são adicionadas ao RDF um conjunto de primitivas, das quais se destacam as que permitem definir classes, subclasses e propriedades com domínio *(domain)* e contra-domínio *(range)*. O domínio refere a classe onde é aplicada a propriedade, o contra-domínio denota os valores possíveis para a propriedade. A Listagem 2 descreve uma taxonomia entre classes que demonstra o uso das primitivas referidas e esclarece a aplicabilidade das primitivas *domain* e *range* em propriedades.

```
<rdf:Description rdf:ID="Animal">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
</rdf:Description>


<rdf:Description rdf:ID="Mamífero">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
  <rdfs:subClassOf rdf:resource="#Animal" />
</rdf:Description>


<rdf:Description rdf:ID="Cão">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
  <rdfs:subClassOf rdf:resource="#Mamífero" />
</rdf:Description>


<rdf:Description rdf:ID="Humano">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
  <rdfs:subClassOf rdf:resource="#Mamífero" />
</rdf:Description>
```

```
<rdf:Property rdf:ID="idade">
  <rdfs:domain rdf:resource="#Animal" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#positiveInteger" />
</rdf:Property>
<rdf:Property rdf:ID="dono">
  <rdfs:domain rdf:resource="#Cão" />
  <rdfs:range rdf:resource="#Humano" />
</rdf:Property>
```

Listagem 2 - Exemplo de taxonomia de seres vivos

O exemplo da Listagem 2 representa uma cadeia hierárquica de relações de generalização e especialização, onde é possível verificar que a propriedade `idade` é aplicada à classe `Animal` e os seus valores possíveis são inteiros positivos. A propriedade `dono` é, por sua vez, aplicada à classe `Cão` e os valores possíveis são instâncias de `Humano`.

A Figura 3 ilustra a representação da Listagem 2 através de um diagrama de classes *Unified Model Language* (UML) [16] que visa a sua aplicação em linguagens de programação como o *C#* [17] ou o *Java*.

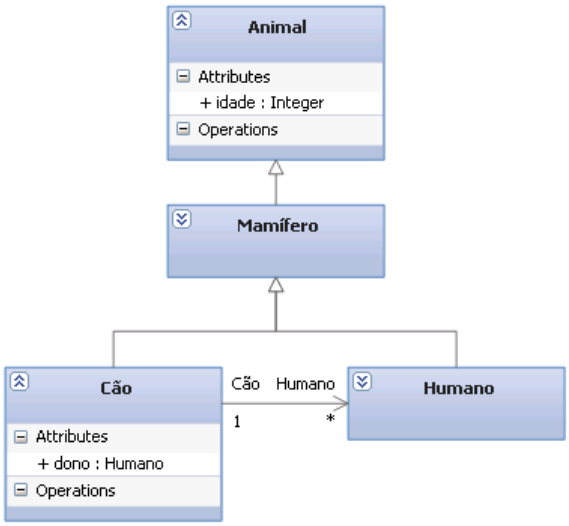

Figura 3 – Diagrama de classes UML com taxonomia de seres vivos

Note-se que o atributo idade da classe animal não pode ser representado como inteiro positivo tal como descrito em RDFS, sendo a melhor aproximação o tipo inteiro. Outro aspecto relevante é que, por omissão, a cardinalidade nas relações entre recursos é de um para muitos[9], constatável na relação entre `Cão` e `Humano`.

[9] No sentido sujeito-objecto.

2.3 OWL

O OWL, *Web Ontology Language* [3], é uma linguagem recomendada pela W3C para a criação de ontologias através do aumento da expressividade existente no RDFS. O OWL está agrupado em três níveis diferentes de expressividade, sendo considerada a linguagem propriamente dita o *OWL Full*. Para além do *OWL Full*, existem também o *OWL Lite* e o *OWL DL* que são consideradas sub-linguagens do OWL. Uma das principais condicionantes à adopção do *OWL Full* é que o seu nível de expressividade torna a inferência computacionalmente complexa, ao ponto de ser considerada uma linguagem indecidível.

Neste projecto, pretende-se usufruir da expressividade do *OWL Lite* incluindo suporte para cardinalidade arbitrária como acontece em *OWL DL*. Em *OWL Lite* as restrições de cardinalidade são existenciais, isto é, só assumem valores do intervalo $\{0,1\}$, ao passo que em *OWL DL* estas podem assumir qualquer valor igual ou superior a zero.

O OWL introduz um conjunto de primitivas que permite definir ontologias que tenham propriedades simétricas, transitivas e funcionais. A definição de propriedades em OWL é dividida em dois tipos, a primitiva `owl:ObjectProperty` refere-se a relações entre instâncias de classes e a primitiva `owl:DatatypeProperty` denota uma propriedade com valores literais. Isto tem a vantagem de identificar a natureza da propriedade sem ter que analisar o tipo definido no *range*. O OWL permite também a definição de restrições, estas não só definem valores possíveis para uma propriedade, mas também permitem ser utilizadas para definir condições necessárias e suficientes. Este tipo de condições permite que o motor de inferência conclua se determinado indivíduo é instância de uma classe. Em *OWL Lite* são definidas através da conjugação das primitivas de intersecção e restrição, podendo ser também definidas directamente através da primitiva de equivalência. Seguem-se dois exemplos que pretendem esclarecer a utilidade das condições necessárias e suficientes, bem como, demonstrar a definição destas últimas através de restrições.

```
<owl:Class rdf:ID="Pessoa" />

<owl:Class rdf:ID="Disciplina" />

<owl:Class rdf:ID="Professor">
  <rdfs:subClassOf rdf:resource="#Pessoa" />
</owl:Class>

<owl:DatatypeProperty rdf:about="nome">
  <rdfs:domain rdf:resource="#Pessoa"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

```
<owl:ObjectProperty rdf:about="ensina">
  <rdfs:domain rdf:resource="#Professor"/>
  <rdfs:range rdf:resource="#Disciplina"/>
</owl:ObjectProperty>
```

Listagem 3 - Exemplo de condição necessária

Na Listagem 3 existem três classes: `Pessoa`, `Disciplina` e `Professor`. A propriedade `nome` pertence à classe `Pessoa`. `Professor` é subclasse de `Pessoa`, possui a propriedade `nome` e a propriedade `ensina`, que tem como valores possíveis instâncias de `Disciplina`.

Esta ontologia define que qualquer instância de `Professor` tem implicitamente os atributos `nome` e `ensina`. No entanto, com esta ontologia, não é possível inferir que um `Professor` é uma `Pessoa` que `ensina` pelo menos uma `Disciplina`. Para construir uma ontologia que permita este tipo de inferência, deve ser aplicada uma restrição ao conceito de `Professor`, sendo esta construção designada de condição necessária e suficiente, como exemplificado na Listagem 4.

```
<owl:Class rdf:ID="Pessoa" />

<owl:Class rdf:ID="Disciplina" />

<owl:Class rdf:ID="Professor">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Pessoa" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#ensina" />
      <owl:someValuesFrom rdf:resource="#Disciplina" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:DatatypeProperty rdf:about="nome">
  <rdfs:domain rdf:resource="#Pessoa"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="ensina">
  <rdfs:domain rdf:resource="#Professor"/>
  <rdfs:domain rdf:resource="#Disciplina"/>
</owl:DatatypeProperty>
```

Listagem 4 - Exemplo de condição necessária e suficiente

Existem um conjunto de ferramentas para criação de ontologias. Estas facilitam o processo de criação e validação das ontologias através de diagramas, tornando desnecessária a produção

manual de ficheiros como o exemplificado na Listagem 4. A ferramenta utilizada neste projecto para análise e criação de ontologias foi o *Protégé* [18], cujo desenvolvimento é realizado pelo centro de Biomédica e Informática da Universidade de *Stanford* em colaboração com a Universidade de *Manchester*.

Sabendo que o OWL é uma extensão ao RDFS e consequentemente ao RDF, é apresentado na Figura 4 um diagrama que ilustra os conceitos do RDF e como são estendidos até algumas das principais primitivas do OWL. Um facto a sublinhar neste diagrama é que uma restrição em OWL descende do conceito de classe de RDFS, tendo especial importância na forma como são construídas interrogações a um *triplestore* para determinar a relação entre uma determinada classe e as suas restrições.

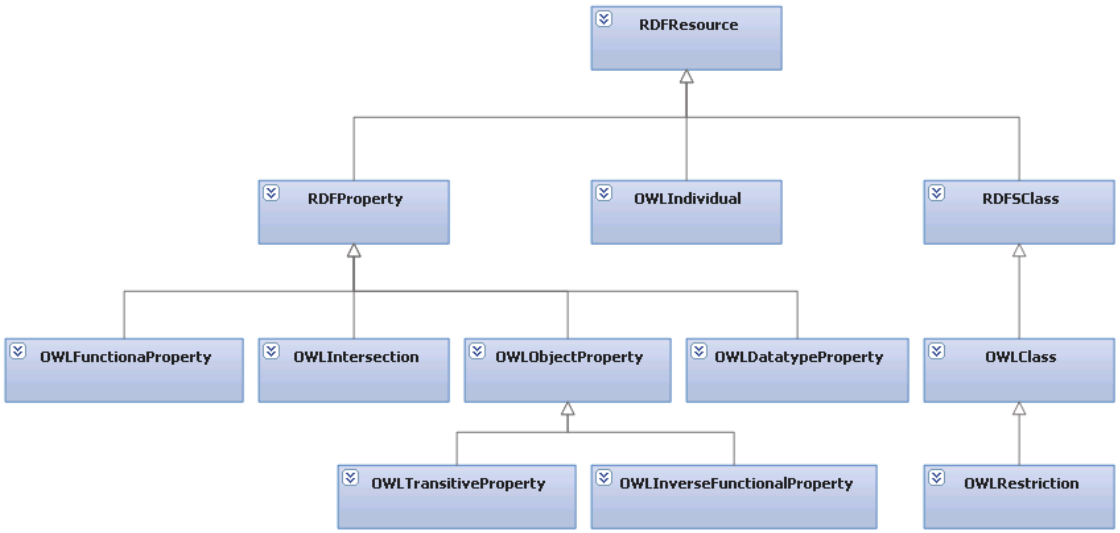

Figura 4 – Diagrama parcial das extensões entre RDF e OWL

Uma restrição OWL é uma das primitivas que origina aquilo que é referido por classe anónima. Isto deve-se ao facto de restrição ser subclasse de `owl:Class` e a sua declaração em OWL não ter identificador associado. Devido a isto, as classes anónimas são representadas como nós anónimos.

2.4 SPARQL

A linguagem de interrogação *SPARQL Protocol and RDF Query Language* (SPARQL) [19], é a recomendação W3C para a camada de inquirição na pilha da *Web* Semântica. A sua sintaxe tem

similaridades com o *Structured Query Language* (SQL)[10], nomeadamente na semântica de cláusulas como `select`, `distinct`, `from`, `union` e `where`.

O SPARQL tem como directivas obrigatórias a `select` e a `where` na construção de expressões. A partir da obrigatoriedade da cláusula `select` é possível concluir que o propósito do SPARQL é apenas realizar consultas, ao contrário do SQL que permite, para além da consulta, realizar manipulação dos dados, sendo estas operações conhecidas por CRUD *(Create, Read, Update, Delete)*. O SQL permite também a criação de modelos físicos de dados através da sua sintaxe *Data Definition Language* (DDL), que no caso da *Web* Semântica é conseguido através de RDF, RDFS ou OWL.

A obrigatoriedade da cláusula `where` determina que consultas num repositório de triplos RDF são sempre restringidas à condição existente nesta cláusula. As condições passíveis de utilização na cláusula `where` do SPARQL diferem substancialmente do SQL, sendo utilizados conjuntos de triplos (grafos) onde são introduzidas variáveis em substituição do sujeito, predicado ou objecto. As variáveis são prefixadas com um ponto de interrogação.

Como um *triplestore* pode conter vários repositórios e, este último pode conter várias ontologias, o mecanismo de *namespaces* para além de ser fundamental para distinguir os termos existentes, é também utilizado para simplificar as consultas através da cláusula `prefix`, sendo apresentada, na Listagem 5 uma expressão SPARQL de exemplo.

```
prefix mus:<http://ontomusica#>

select ?album where {
  ?album rdf:type mus:album ;
         mus:compostoPor <http://ontomusica/musicos/richter/max> .
  ?album mus:temas "11"^^xsd:int
}
```

Listagem 5 - Exemplo de expressão SPARQL

No exemplo ilustrado na Listagem 5 é assumida a existência de uma classe `mus:album`, que consta como objecto no primeiro triplo da cláusula `where`. A declaração do prefixo `mus` permite termos como `mus:compostoPor` em vez de `<http://ontomusica#compostoPor>`, simplificando a expressão SPARQL. O exemplo traduz-se para a questão: que álbuns são compostos por *Max Richter* e têm onze temas?

A cláusula `where`, neste exemplo, é um grafo composto de três triplos, estando os dois primeiros triplos ligados através de um ponto e vírgula. Os dois últimos triplos estão ligados através de um ponto. O ponto e vírgula permite a ligação entre dois triplos sem repetir a

[10] Norma ISO/IEC 9075:2008.

variável usada como sujeito no primeiro triplo. O ponto permite a ligação de quaisquer dois triplos, sendo necessária a repetição da variável no segundo triplo, mesmo que esta seja sujeito em ambos os triplos. No exemplo apresentado, os últimos dois triplos da cláusula `where` estão ligados por um ponto, devido a isto, é necessária a declaração da variável `album` no último triplo, apesar de esta última ser também sujeito no triplo anterior.

Como um repositório pode conter várias ontologias, é importante poder definir em qual das ontologias se pretende realizar uma consulta, sendo isto possível através da cláusula *from*. A recomendação SPARQL define os valores a atribuir à cláusula *from* como sendo um *named graph*, que geralmente corresponde ao URI a partir do qual foi criada a ontologia. Uma analogia possível para *named graph* em SQL é o *schema*. O conceito de *schema* num motor de base de dados relacional corresponde a um conjunto de tabelas, sabendo que uma tabela no modelo relacional equivale a uma classe em RDFS ou OWL, então um *schema* será equivalente a um grafo com uma determinada designação. Nos *triplestores* o *named graph* é geralmente designado por: contexto.

A Listagem 6 representa a expressão SPARQL para a pergunta: quais os álbuns e seus compositores que tenham onze ou doze temas?

```
prefix mus:<http://ontomusica#>

select distinct ?album ?artista where {
  { ?album rdf:type mus:album ;
           mus:compostoPor ?artista ;
           mus:temas "11"^^xsd:int
  }
union
  { ?album rdf:type mus:album ;
           mus:compostoPor ?artista ;
           mus:temas "12"^^xsd:int
  }
}
```

Listagem 6 - Utilização da cláusula *union* em SPARQL

Através do exemplo da Listagem 6, é possível constatar que cláusula `union` é equivalente à cláusula `or` do SQL. Isto deve-se ao facto de que o SPARQL utiliza grafos para definir os padrões a aplicar à cláusula `where` enquanto o SQL utiliza operadores binários.

Outro aspecto a ter em conta no SPARQL é que, como alguns repositórios fazem inferência no carregamento de triplos, é possível ter várias afirmações equivalentes, por exemplo ao carregar uma ontologia em OWL que defina apenas álbum como classe, serão também criados triplos que afirmam que álbum é uma classe em RDFS. Desta forma, uma consulta SPARQL sem a cláusula `distinct` geralmente produz resultados redundantes.

Como já referido, as restrições em OWL são subclasses de `owl:Class`, o que implica que estas sejam representadas como nós anónimos. Assim, existe uma construção em SPARQL que permite realizar consultas usando nós anónimos sem ter que indicar qualquer identificador para esse nó, sendo essa construção denotada pelo parêntesis rectos.

A expressão SPARQL, representada na Listagem 7, permite obter todas as propriedades e respectivos *ranges* de restrições sem explicitamente utilizar a construção `owl:Restriction`, fazendo uso dos parêntesis rectos para referir nós anónimos. A expressão pode ser traduzida para a seguinte questão: quais as propriedades e *ranges* que participam em triplos como objectos e, nesses triplos o sujeito é um nó anónimo e os predicados são `owl:onProperty` e `owl:someValuesFrom`?

```
select distinct ?prop ?range where {
  [ owl:onProperty ?prop ;
    owl:someValuesFrom ?range
  ]
}
```

Listagem 7 – Utilização de nós anónimos em SPARQL

Este tipo de consultas é preferível à utilização de identificadores para nós anónimos na cláusula `where` da expressão SPARQL, isto porque a identificação de um nó anónimo não é garantida como unívoca no sistema de informação. O identificador de um nó anónimo pode variar a cada consulta SPARQL realizada.

3 Sistema de informação semântico

Este capítulo é constituído por duas secções principais: Arquitectura e Implementação.

Na secção referente à arquitectura, são apresentados os principais blocos constituintes do sistema e da solução implementada, através da descrição das respectivas arquitecturas. Como já referido, o sistema é o resultado do trabalho da equipa que o autor integra, sendo a solução o projecto realizado pelo autor.

Na secção referente à implementação da solução, são detalhados os resultados das análises realizadas às tecnologias da *Web* Semântica e das ferramentas de interoperabilidade com o *Excel*, expondo os respectivos impactos na implementação. Adicionalmente, são referidas as principais decisões tomadas e descritas as funcionalidades mais relevantes, agregadas por camadas funcionais.

3.1 Arquitectura

3.1.1 Arquitectura do sistema

A arquitectura do sistema segue o modelo cliente-servidor. A parte cliente, âmbito deste projecto, consiste numa aplicação de carregamento de dados para um repositório semântico, sustentada na extensibilidade do *Excel*. A Figura 5 ilustra a arquitectura do sistema.

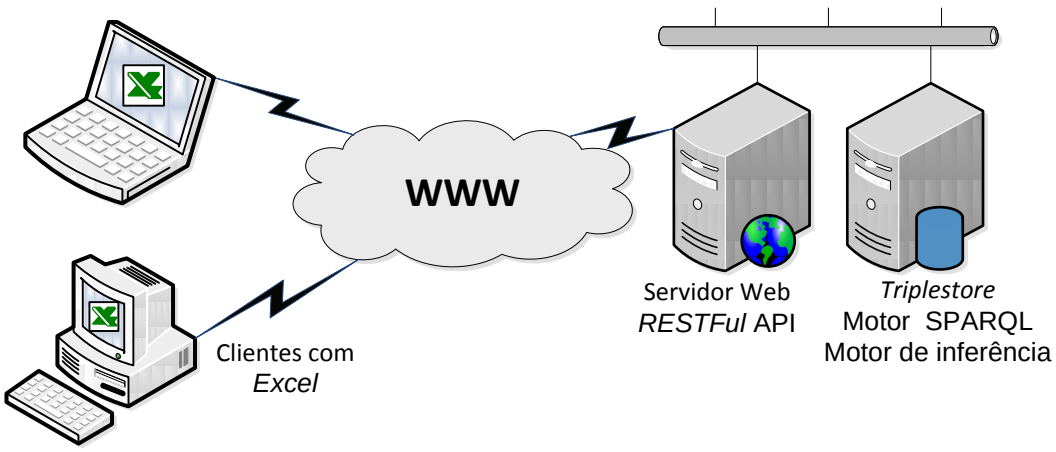

Figura 5 – Arquitectura global do sistema

Enumeram-se de seguida as principais camadas existentes no lado do servidor:

- **Persistência**: consiste no *triplestore Bigdata* [20].
- **Inferência**: podem ser utilizados vários motores de inferência, tais como o *FaCT++* [21], o *Pellet* [22] ou o *OWLIM* [23], entre outros. A escolha irá incidir sobre o que permitir melhor integração com o resto do sistema e desempenho, para o nível de expressividade necessária nas ontologias em implementação.
- **Inquirição**: camada que permite interpretar as interrogações em SPARQL colocadas pelas aplicações cliente e retornar os resultados em notações como o *JavaScript Object Notation* (JSON) [24] ou XML. A notação utilizada neste projecto será a notação XML.
- ***RESTFul* API**: permite a utilização das camadas acima referidas.

3.1.2 Arquitectura da solução

A parte cliente do sistema tem uma forte dependência do *Excel*. Apesar deste aspecto, foi ainda assim, adoptada uma estratégia que permite reutilizar o trabalho desenvolvido, noutros projectos que visem a utilização de um sistema de informação semântico e recorram à *.NET Framework*.

É apresentado na Figura 6, um diagrama que ilustra a especialização em camadas lógicas existentes na implementação deste projecto.

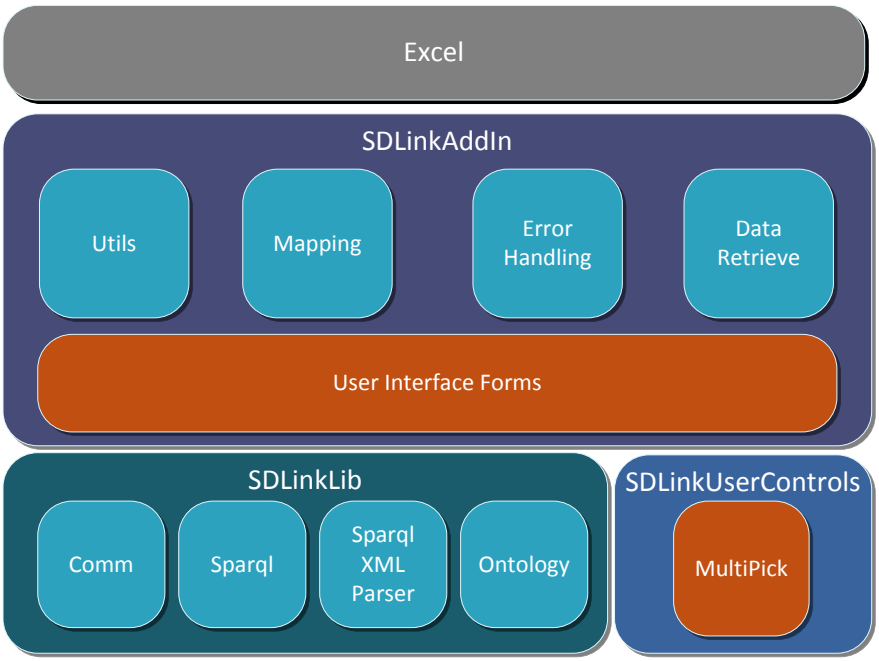

Figura 6 – Arquitectura da solução

A solução é composta por uma camada aplicacional directamente relacionada com o *Excel*, designada de *SDLinkAddIn*, que contém funcionalidades que recorrem directamente à manipulação do modelo de objectos do *Excel*, como por exemplo, a criação de livros, folhas, manipulação de células, entre outras. Nesta camada existem também um conjunto de formulários gráficos de interacção com o utilizador, que dependem não só da camada de manipulação com o *Excel*, mas também do acesso a variáveis globais[11] existentes apenas neste tipo de projectos.

Imediatamente abaixo desta camada, temos a biblioteca *SDLinkLib* que cujas funcionalidades principais são permitir a interacção de qualquer aplicação com *endpoints* SPARQL. Esta biblioteca dispõe de uma representação própria de ontologias, camada de comunicação com servidores *Web* e uma classe especializada em SPARQL que permite a construção da ontologia a partir de interrogações.

Foi também criada uma biblioteca com um controlo gráfico que permite a selecção múltipla de valores e promoção dos mesmos, designado de *Multipick*, ilustrado na Figura 7.

Figura 7 – Controlo *MultiPick*

As camadas *SDLinkLib* e *SDLinkUserControls*, ao contrário da camada *SDLinkAddIn*, são independentes do *Excel*. Em particular a camada *SDLinkLib* está fisicamente representada numa biblioteca[12] e é utilizável em qualquer aplicação que possa referir bibliotecas *.NET*, ao passo que a biblioteca *SDLinkUserControls*, que contém o controlo *Multipick*, só pode ser utilizada em aplicações *Windows Forms*.

[11] Representadas na classe `Globals` de projectos *Visual Studio Tools for Office*.

[12] *Dynamic-link library*.

3.2 Implementação da solução

3.2.1 Tipo de solução VSTO

Para a implementação desta solução foi necessária uma avaliação prévia à utilização do *Excel* enquanto interface gráfica de um sistema de informação. Sabendo que para a implementação dos requisitos é preferível a utilização de uma plataforma como a *.NET Framework*, foi realizada uma avaliação da API de interoperabilidade entre esta e o *Excel*.

Como o *Excel* é uma aplicação *Component Object Model* (COM) [25] os seus objectos são *unmanaged*[13], sendo necessária uma camada de interoperabilidade para os poder utilizar em ambiente *managed*. Para permitir esta interoperabilidade a *Microsoft* disponibiliza um conjunto de bibliotecas designado de *Primary Interop Assembly* (PIA) [7]. Da análise realizada à representação do modelo de objectos do *Excel* no PIA verificou-se que todos estão definidos através de interfaces. Desta forma, é possível garantir que futuras versões do PIA têm que implementar o contrato definido pelas interfaces, permitindo a implementação de uma solução de extensibilidade sem um compromisso directo com versões do *Excel* ou do PIA.

Para o desenvolvimento de soluções de extensibilidade de produtos *Office* [26] a *Microsoft* disponibiliza um conjunto de ferramentas designado de *Visual Studio Tools for Office* (VSTO) [6]. O VSTO permite a criação de projectos do *Visual Studio* com as características necessárias à interoperabilidade com produtos *Office*, por exemplo, a referenciação das bibliotecas de interoperabilidade e definição de métodos para tratamento dos principais eventos. No caso da implementação em *Excel*, versão 2007, é possível usufruir de uma interface gráfica para personalização do *Ribbon*[14]. A Figura 8 ilustra o *Ribbon* implementado na aplicação.

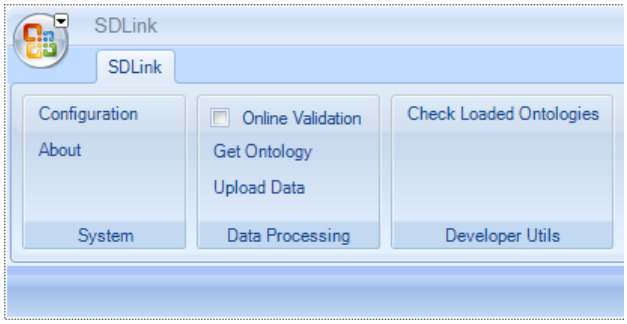

Figura 8 – *Ribbon* implementado para a interface em *Excel*

[13] Objectos que não usufruem das virtudes de um ambiente virtual de execução, como por exemplo gestão automática de memória.

[14] Barra de ferramentas do *Office* que substitui os menus e barras de ferramentas das versões anteriores.

Existem três tipos de projectos VSTO disponíveis: *Workbook*, *Template* e *Add-In*. O tipo adoptado é o *Add-In* por questões de versatilidade organizacional nas instituições que utilizem o *SDLink*. De um modo geral, as folhas de cálculo em *Excel* são criadas por alguém responsável por determinada experiência laboratorial, mas os ficheiros *Excel* resultantes são partilhados entre colaboradores na instituição. Projectos VSTO dos tipos *Workbook* e *Template*, são implementações ao nível do ficheiro *Excel*[15], o que impede a utilização do ficheiro por colaboradores que não tenham a *.NET Framework* ou os PIA instalados no seu posto de trabalho. A Figura 9 ilustra o problema existente na partilha de ficheiros *Excel* em que seja utilizada uma solução de extensibilidade *Workbook* ou *Template*.

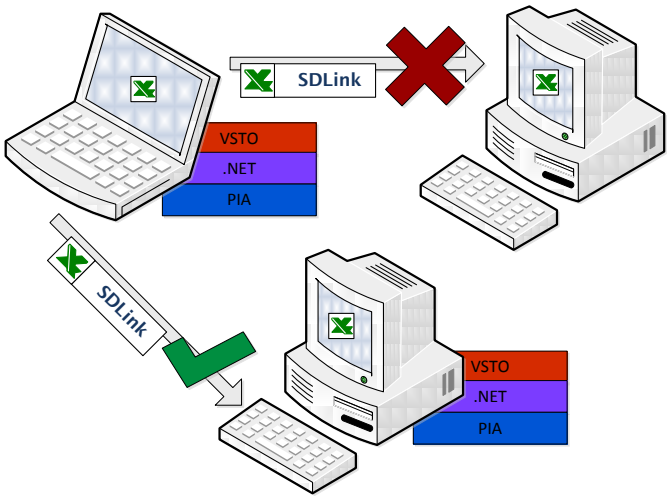

Figura 9 – Problemas na partilha de ficheiros *Excel*

Recorrendo a um projecto VSTO do tipo *Add-In*, a implementação fará parte do posto de trabalho e não do ficheiro *Excel*, permitindo que um colaborador que utilize o *SDLink* possa partilhar os ficheiros produzidos com qualquer outro colaborador que tenha apenas o *Excel* instalado.

3.2.2 Execução assíncrona

As tarefas a realizar pela implementação são demoradas e resultam da interacção do utilizador com a interface gráfica. Esta combinação é propensa a bloqueios na interface com o utilizador, caso se utilize a *thread* que trata os eventos dos controlos gráficos, para também desempenhar as tarefas aplicacionais. De um modo geral, assim que uma tarefa termine, o utilizador deve ser notificado por alterações na interface gráfica. É também desejável que a tarefa seja executada

[15] A implementação é intrínseca ao ficheiro *Excel*, não podendo ser dissociada.

de forma assíncrona, permitindo que a interface com o utilizador fique disponível para notificações de progresso e permita o cancelamento da tarefa. Por conseguinte, existem geralmente duas tarefas a serem executadas por cada acção do utilizador: a tarefa demorada T_1, que deve ser executada assincronamente e, a tarefa T_2 que produz a actualização da interface gráfica após a execução de T_1. Assim, a solução implementa num só formulário todo o suporte à execução assíncrona, com pós-execução síncrona de pares de tarefas $\{T_1, T_2\}$, oferecendo possibilidade de cancelamento de T_1 ao utilizador. Desta forma, é possível conceber formulários seguindo a mesma filosofia comportamental e que reaproveitam a implementação da execução assíncrona com suporte a cancelamento.

A Figura 10 ilustra o fluxo de execução de dos pares de tarefas $\{T_1, T_2\}$, representando as duas *threads* responsáveis pela execução destas tarefas. A `MainThread` é responsável pelo processamento dos eventos na interface gráfica do *Excel*. A `AbortableBackgroundWorkerThread` é a *thread* criada para executar, de forma assíncrona, as tarefas aplicacionais desencadeadas pelo utilizador. O formulário `FrmAsyncBase` é onde reside a implementação para a execução assíncrona de T_1 com pós-execução síncrona de T_2. Este formulário é estendido na solução para suportar as principais funcionalidades da aplicação.

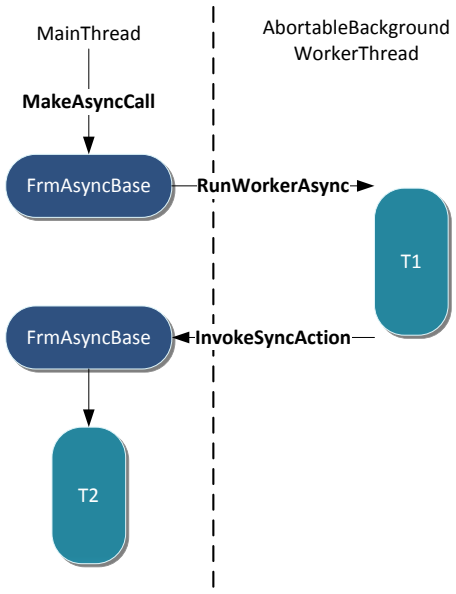

Figura 10 – Fluxo de execução de tarefas aplicacionais

A Figura 11 ilustra o formulário de carregamento de ontologias, onde a tarefa T_1 é a obtenção das ontologias existentes no servidor e a tarefa T_2 é a representação, no formulário, das ontologias obtidas. Este formulário é uma extensão ao formulário `FrmAsyncBase`.

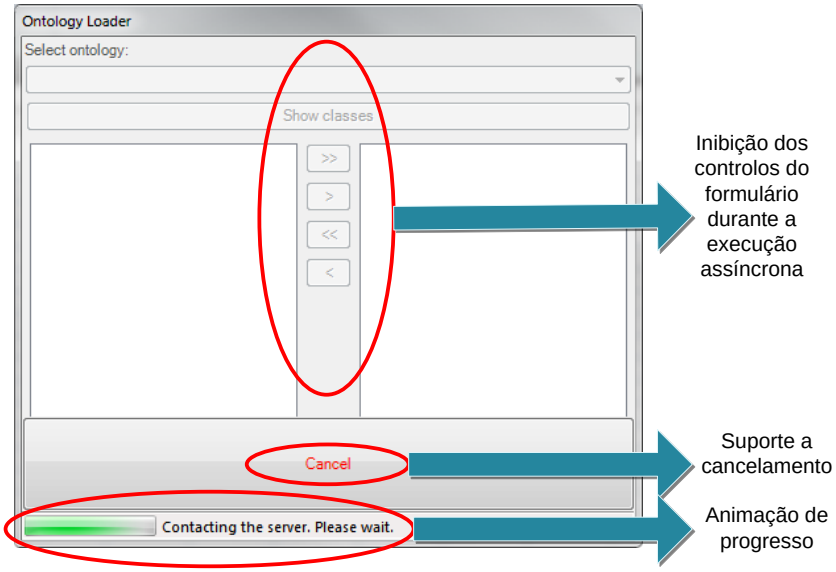

Figura 11 – Exemplo de extensão do formulário FrmAsyncBase

3.2.3 Representação da ontologia em memória

A representação em memória da ontologia diverge da representação em grafo existente em OWL. Os grafos são estruturas indicadas para representar sistemas complexos, permitindo flexibilidade para alterar os nós e arcos existentes. Nesta solução é primordial dispor de eficiência, isto é, ter tempos de acesso muito curtos. Desta forma, a ontologia definida no sistema de informação, para a qual o utilizador pretende realizar o carregamento de dados, é representada em memória através de um conjunto de dicionários, cujos tempos de acesso são próximos de $O(1)$ [27]. A Figura 12 ilustra as classes que suportam a representação em memória da ontologia.

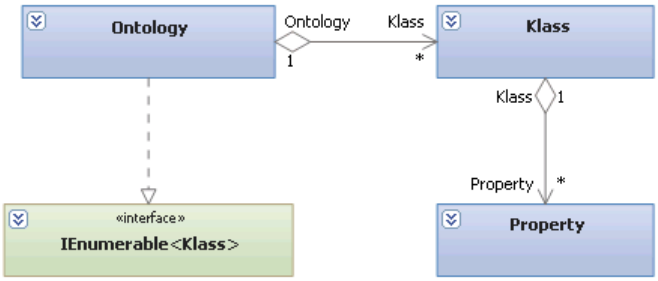

Figura 12 – Diagrama de classes UML da representação em memória de ontologias

A representação da ontologia em memória corresponde a um conjunto de classes (*klass*) que, por sua vez, contêm um conjunto de propriedades. As propriedades contêm a designação, tipo

(literal ou recurso), o seu valor (*range*) e restrições, permitindo assim representar toda a expressividade do OWL, necessária à interface gráfica e interacção com o utilizador.

3.2.4 Tecnologias

Para suportar o desenvolvimento e testes foi utilizada a *framework Sesame* da *openRDF.org* [28]. Esta *framework* contém um *triplestore*, um motor de inferência para RDFS[16] e um *endpoint* SPARQL. Para a execução da *framework Sesame*, foi utilizado o servidor *Web Apache Tomcat* versão 6.

A *framework Sesame* dispõe de uma ferramenta de gestão designada de *Workbench*. Esta ferramenta permite a criação de novos repositórios, execução de interrogações SPARQL e carregamento de ontologias.

Como já referido, um *triplestore* pode conter vários repositórios e um repositório pode conter várias ontologias. Desta forma é possível manter o nível de isolamento considerado necessário entre as ontologias, o que é útil para separar as ontologias de teste das ontologias de produção. A Figura 13, apresentada de seguida, ilustra a existência de oito repositórios num único *triplestore*, através da interface de gestão do *Workbench*.

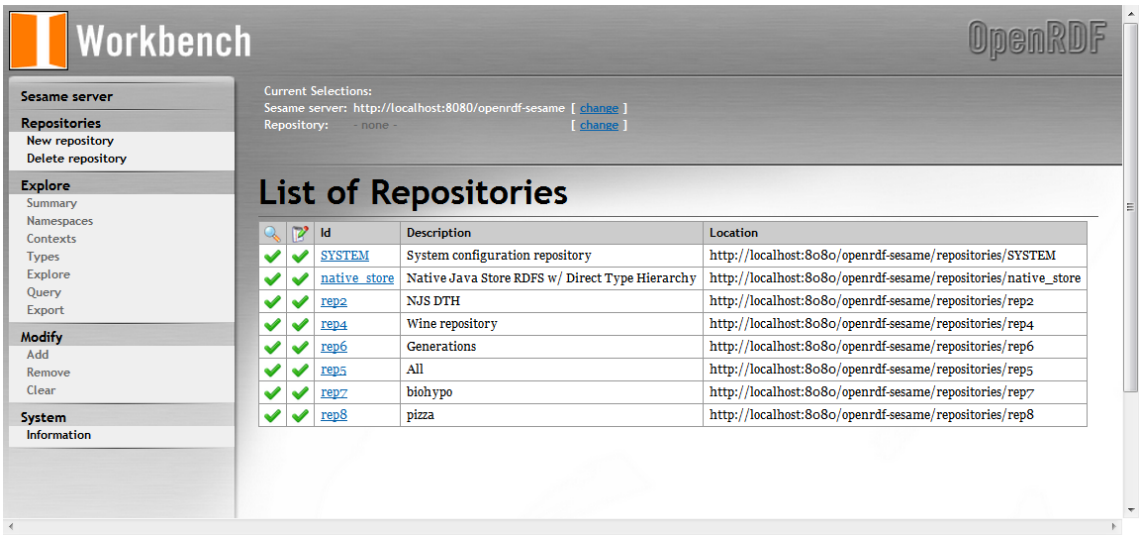

Figura 13 – *Workbench* da *framework Sesame*

[16] Não suportando inferência ao nível de expressividade do OWL.

3.2.5 Camadas funcionais

3.2.5.1 Comunicação

O modelo de objectos do *Excel* não dispõe de funcionalidades de conectividade através dos protocolos existentes na *Web*. Desta forma, o recurso à *.NET Framework* permitiu abreviar o tempo de desenvolvimento desta camada. O protocolo utilizado neste projecto é *Hypertext Transfer Protocol* (HTTP) [29].

Esta camada permite que chamadas de outras camadas sejam transformadas em pedidos HTTP. As respostas são devolvidas aos métodos invocantes sem qualquer alteração. Neste sentido, a camada de comunicação não realiza qualquer interpretação das mensagens que transmite e recebe, limitando-se a reencaminhar pedidos, devolver respostas e tratar eventuais excepções de comunicação.

Esta camada funcional lida apenas com *endpoints* de *triplestores*, podendo no futuro ser adoptada para suportar outras linguagens de inquirição ou novos formatos de representação. De momento esta camada suporta apenas os seguintes formatos de representação:

1. Sparql-results+json
2. Sparql-results+xml
3. X-binary-rdf-results-table (BRTR)

Estas representações são necessariamente enviadas como parâmetro `accept` dos pedidos HTTP, sendo de momento as mais utilizadas no panorama da *Web* Semântica. O formato BRTR foi desenvolvido pela *openRDF.org* e visa a transferência de resultados em formato compacto.

O *endpoint* SPARQL é da forma:

```
http://<host>/<triplestore>/repositories/<repository id>
```

A Figura 14 representa quais as operações realizadas pela *RESTFul* API de *endpoints* SPARQL, de acordo com o verbo HTTP utilizado.

Verbo	Operação
GET	Permite o envio de uma interrogação SPARQL através do parâmetro *query*, retornando os resultados na representação solicitada, ou seja JSON, XML ou BRTR.
POST	Suporta o envio de interrogações SPARQL com mais de 4096 caracteres, utilizando o corpo do pedido para envio do parâmetro *query*.

Figura 14 – Quadro de operações da *RESTFul API* para *endpoints* SPARQL

Esta camada lida também com o carregamento de dados, recorrendo a um *endpoint* com a forma:

`http://<host>/<triplestore>/repositories/<repository id>/statements`

A *RESTFul* API, para este *endpoint* é a representada na Figura 15.

Verbo	Operação
GET	Obtém a totalidade de triplos do *triplestore* na ausência de parâmetros, ou obtém triplos de acordo com os parâmetros indicados no pedido.
PUT	Remove todos os triplos do *triplestore*, substituindo-os pelos novos triplos enviados no pedido.
DELETE	Remove a totalidade de triplos do *triplestore* na ausência de parâmetros, ou remove apenas os triplos coincidentes com os parâmetros indicados no pedido.
POST	Adiciona os triplos enviados no pedido ao *triplestore*.

Figura 15 - Quadro de operações da *RESTFul API* para *endpoints* de triplos

Do quadro anterior, a implementação faz uso apenas do verbo *POST* para o carregamento de triplos. Devido à implementação da *RESTFul* API dos *triplestores*, constatou-se que não era possível realizar as operações habituais de CRUD[17] através desta solução, sendo esta a principal diferença entre os objectivos iniciais do projecto e a implementação final.

O carregamento de dados consiste no envio de uma estrutura em RDF/XML no corpo do pedido *POST*. Para estes pedidos, o servidor envia sempre a mesma resposta: `HTTP/1.1 204 NO CONTENT`.

3.2.5.2 Análise sintáctica

A camada de análise sintáctica[18] é responsável por realizar a tradução dos formatos de resposta do *triplestore* às interrogações SPARQL. O formato eleito para o sistema foi o *SPARQL Query*

[17] *Create, Read, Update* e *Delete*.

[18] Termo em inglês é *parse*.

Results XML [30]. Sendo um formato XML, a implementação recorre à tecnologia *LINQ*[19] *to XML* para simplificar o processo de análise sintáctica e tornar o código mais legível.

O principal objectivo desta camada é converter os formatos de representação vindos do *triplestore* em tabelas onde as colunas são as variáveis da interrogação SPARQL e as linhas os seus respectivos valores. É apresentado na Listagem 8 um exemplo de resposta em formato *SPARQL Query Results XML*.

```
<?xml version='1.0' encoding='UTF-8'?>
<sparql xmlns='http://www.w3.org/2005/sparql-results#'>
  <head>
    <variable name='prop'/>
    <variable name='range'/>
  </head>
  <results>
    <result>
      <binding name='prop'>
        <uri>http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#locatedIn</uri>
      </binding>
      <binding name='range'>
        <uri>http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Region</uri>
      </binding>
    </result>
  </results>
</sparql>
```

Listagem 8 – Exemplo de resultados SPARQL em XML

O exemplo da Listagem 8, demonstra a presença das variáveis `prop` e `range` e respectivos valores, correspondendo ao exemplo da Listagem 7 quando aplicado à ontologia dos vinhos da W3C [31].

Sabendo que existem mais formatos utilizados como resposta a interrogações SPARQL, como o *SPARQL Query Results in JSON* [32], foi definida a interface `ISparqlResultsParser` para permitir futuras implementações de analisadores sintácticos, mantendo a compatibilidade com o resto da implementação. Para a utilização específica no sistema *SDLink*, é fornecida a implementação do analisador sintáctico `SparqlXmlParser`. A Figura 16 ilustra a interface `ISparqlResultsParser` e a sua implementação através do tipo `SparqlXmlParser`.

[19] *Language-Integrated Query*

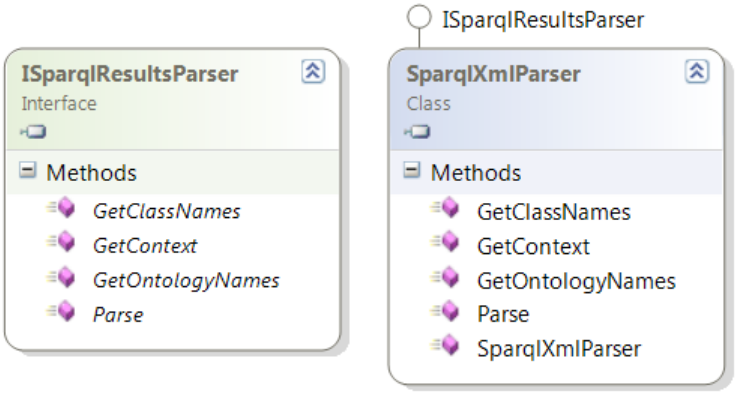

Figura 16 – Interface e respectiva implementação do analisador sintáctico

3.2.5.3 Construção da ontologia em memória

A construção da ontologia em memória é o processo que permite a conversão da representação em OWL da ontologia existente no *triplestore*, para a representação em memória adoptada nesta solução e descrita em 3.2.3. Este processo é constituído por um algoritmo e por um conjunto de interrogações SPARQL.

Esta funcionalidade é suportada em exclusivo pelo tipo `Sparql`, o qual inclui o algoritmo de construção da ontologia em memória e as interrogações SPARQL necessárias à consulta da ontologia existente no *triplestore*. Como o tipo `Sparql` faz parte da biblioteca *SDLinkLib* todas as interrogações SPARQL estão disponibilizadas como métodos públicos, permitindo a sua utilização em contextos similares noutras aplicações que utilizem esta biblioteca.

O algoritmo para a construção da ontologia em memória é composto pelas etapas: Criar Classes, Criar Propriedades, Criar Restrições e Criar Taxonomia. Cada uma destas etapas recorre às camadas de comunicação e análise sintáctica.

Na Figura 17 é apresentado um diagrama de sequência simplificado, que ilustra as principais etapas do algoritmo de construção. Por questões de simplificação só é representada a interacção com as camadas de comunicação e análise sintáctica na primeira etapa do algoritmo, assim como, os nomes apresentados não correspondem aos métodos concretos da implementação.

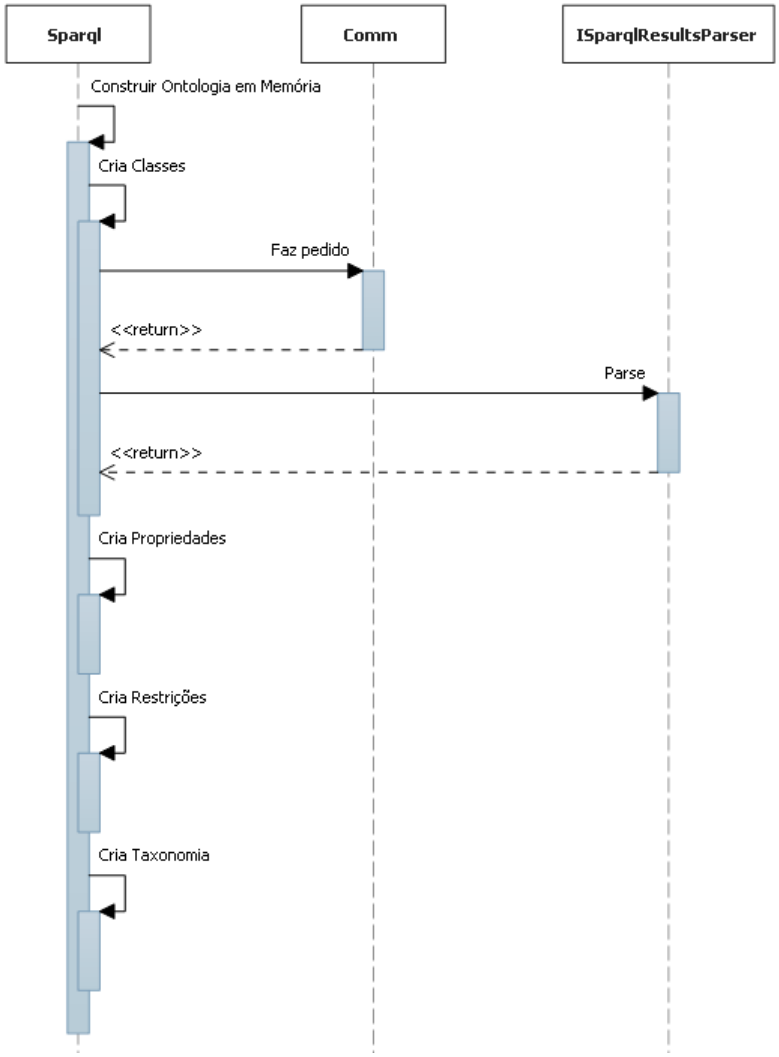

Figura 17 – Diagrama de sequência para a construção da ontologia em memória

3.2.5.4 Mapeamento

O processo de mapeamento exigiu a definição de regras de transformação entre a representação em OWL de uma ontologia e a representação em memória da mesma ontologia, bem como, a sua apresentação em *Excel*.

Para realizar este processo foi necessário definir qual o nível de expressividade necessário para o sistema. Durante as reuniões de projecto, ficou definido que o nível de expressividade mais adequado seria o suportado por *OWL Lite* com suporte adicional para a definição de cardinalidade arbitrária como acontece no *OWL DL*.

Assim, o método de trabalho adoptado consistiu no levantamento de todas as primitivas OWL, assim como, na consulta de bibliografia, especificação do OWL e ontologias exemplo. Isto

permitiu determinar o objectivo de cada primitiva, os seus casos de uso e a utilidade na interface gráfica. A Figura 18 fornece uma visão geral de quais as primitivas de *OWL Lite* suportadas e excluídas na implementação.

Schema	(In)Equality	Properties Characteristics	Property Restrictions
Class (Thing, Nothing)	equivalentClass	ObjectProperty	Restriction
rdfs:subClassOf	equivalentProperty	DatatypeProperty	onProperty
rdf:Property	sameAs	inverseOf	allValuesFrom
rdfs:subPropertyOf	differentFrom	TransitiveProperty	someValuesFrom
rdfs:domain	AllDifferent	SymmetricProperty	
rdfs:range	distinctMembers	FunctionalProperty	
Individual		InverseFunctionalProperty	

Restricted Cardinality {0;1}	Header Information	Class Intersection
minCardinality	Ontology	intersectionOf
maxCardinality	imports	
cardinality		

Versioning	Annotation Properties
versionInfo	rdfs:label
priorVersion	rdfs:comment
backwardCompatibleWith	rdfs:seeAlso
incompatibleWith	rdfs:isDefinedBy
DeprecatedClass	AnnotationProperty
DeprecatedProperty	OntologyProperty

Legenda
Suportada
Excluída

Figura 18 – Primitivas suportadas e excluídas da implementação

Do conjunto de primitivas OWL, ilustradas na Figura 18, são descritas de seguida as justificações para a implementação das primitivas suportadas.

owl:Ontology: Contém o identificador da ontologia na forma de URL, sendo necessário para suportar a distinção de múltiplas ontologias no mesmo repositório. Na implementação permite que o utilizador seleccione a ontologia para a qual pretende realizar o carregamento de dados.

owl:Class: Primitiva que declara uma classe na ontologia. A implementação usa as classes existentes na ontologia não só para a representação da ontologia em memória, mas também para permitir que o utilizador crie instâncias para um subconjunto das classes da ontologia. Uma classe é representada directamente por uma folha de cálculo.

rdfs:subClassOf: Permite a construção de taxonomias entre classes da ontologia. O seu suporte é obrigatório pois classes descendentes têm as mesmas propriedades que as suas ascendentes, implicando a respectiva representação em memória e na folha de cálculo.

owl:ObjectProperty: Propriedade de uma classe que refere outra classe na ontologia, sendo representada na folha de cálculo como o cabeçalho de uma coluna.

owl:DatatypeProperty: Propriedade de uma classe que refere um literal *XML Schema* (XSD), sendo representada como cabeçalho de uma coluna na folha de cálculo.

rdfs:domain: Indica a classe onde é aplicada uma propriedade, o seu suporte é obrigatório para a correcta representação das propriedades em memória, bem como para o correcto mapeamento na folha de cálculo da classe referida por *rdfs:domain*.

rdfs:range: Indica a classe ou tipo XSD cujas instâncias representam valores possíveis da propriedade. A sua representação é um comentário na célula que representa a propriedade indicando qual o tipo definido através desta primitiva.

owl:Restriction: Representa uma restrição aplicada a uma propriedade. Para a sua declaração em OWL é necessária a primitiva *onProperty* conjuntamente com uma das primitivas *allValuesFrom*, *someValuesFrom*, *minCardinality*, *maxCardinality* e *cardinality*. O seu suporte é necessário para alertar o utilizador para restrições existenciais ou de cardinalidade.

owl:onProperty: Permite referir em qual propriedade da classe é aplicada a restrição. O seu suporte está implícito através da primitiva *owl:Restriction*.

owl:allValuesFrom: Indica a classe cujas instâncias podem ser referidas na propriedade com esta restrição. Todos os valores da propriedade, caso existam, devem ser do tipo referido. Esta restrição é implementada como não tendo restrição existencial, nem outra cardinalidade associada.

owl:someValuesFrom: Indica a classe cujas instâncias podem ser referidas na propriedade com esta restrição. Deve existir pelo menos um valor na propriedade que seja do tipo referido. Esta restrição é implementada como sendo de restrição existencial, podendo ser considerada de cardinalidade maior ou igual a um.

owl:minCardinality: Indica qual a cardinalidade mínima de uma propriedade no contexto de uma restrição. A implementação procede à verificação desta restrição durante o processo de *upload*.

owl:maxCardinality: Indica qual a cardinalidade máxima de uma propriedade no contexto de uma restrição. A implementação procede à verificação desta restrição caso o seu valor seja igual a um no formulário de validação. Caso seja definida com outro valor a verificação será imediatamente anterior ao *upload*.

owl:cardinality: Indica qual a cardinalidade exacta de uma propriedade no contexto de uma restrição. A implementação segue as mesmas regras definidas para *owl:maxCardinality*.

owl:intersectionOf: Primitiva que define intersecções restrições e permite inferência por conjugação de recursos da ontologia. Por exemplo, a classe `Professor` pode ser definida à custa da intersecção da classe `Pessoa` com a propriedade `leccionaDisciplina` preenchida com instâncias da classe `Disciplina`, ou seja, através de uma condição necessária e suficiente. O seu uso é obrigatório, pois implica a existência das propriedades da classe `Pessoa` na classe `Professor`, bem como define que a propriedade `leccionaDisciplina` tem como *domain* `Professor` e *range* `Disciplina`.

rdfs:label: Usada para a descrição das classes e propriedades de uma ontologia. É suportada através da sua inclusão como comentário na célula da propriedade a que se refere.

owl:Individual: Primitiva que possibilita a criação de instâncias de classes. A implementação usa a construção de RDF para permitir a utilização por qualquer motor de inferência. A sintaxe em OWL é: `<owl:Individual rdf:ID="identificador"> … </owl:Individual>`, enquanto a sintaxe em RDF é: `<rdf:Description rdf:about="url"> … </rdf:Description>`

O processo de mapeamento termina com a criação das folhas de cálculo, devidamente formatadas de acordo com a representação em memória da ontologia. Geralmente existem várias folhas de cálculo por ontologia, uma vez que uma folha de cálculo representa uma classe. Assim, a operação de criação de folhas de cálculo é cíclica, iterando por cada classe da ontologia, sendo propícia à sua execução em paralelo nos sistemas que dispõem de mais que um núcleo de processamento.

Uma vez que os controlos gráficos em aplicações COM, como é o caso do *Excel*, seguem o modelo *Single Threaded Apartment* (STA) [33], existe apenas uma *thread* a processar as mensagens da interface gráfica. Adicionalmente, não é possível a criação de uma folha de cálculo sem que esta seja representada de imediato na interface gráfica do *Excel*. Desta forma, apesar de existir potencial para execução em paralelo, não é possível tirar partido de múltiplos núcleos de processamento do sistema em operações que recorram ao modelo STA nas aplicações COM.

Esta camada da solução encontra-se implementada na classe estática `Mapping`, cujas principais funcionalidades são: Criação de um livro *Excel*, através da representação em memória de uma ontologia[20] e produção de RDF/XML com base num livro. Adicionalmente, esta classe

[20] Implementada no tipo `Ontology`

implementa também a funcionalidade de persistência descrita no ponto 3.2.5.5. A Figura 19 ilustra a classe estática `Mapping` e os seus principais métodos.

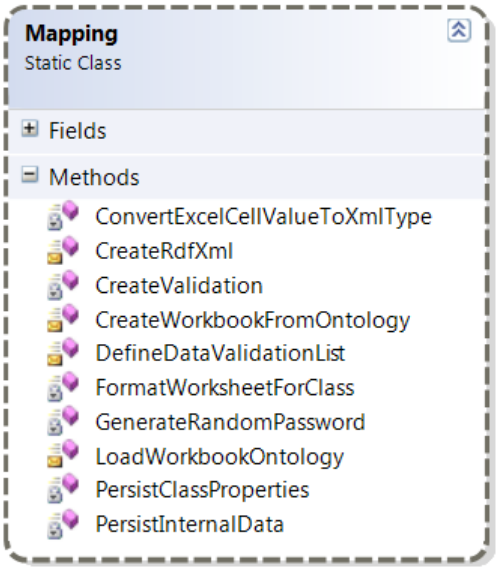

Figura 19 – Diagrama da classe *Mapping*

3.2.5.5 Persistência

Para não obrigar o utilizador a ter que trabalhar ininterruptamente na mesma sessão de carregamento de dados até realizar o *upload* para o *triplestore*, é fundamental persistir a ontologia existente em memória. Isto permite que o utilizador feche o *Excel* ou o livro em que trabalha, para mais tarde, voltando a abrir o respectivo livro, poder continuar a trabalhar na mesma ontologia e com os dados que havia introduzido. Para permitir esta versatilidade de utilização, são criadas um conjunto de folhas adicionais no livro que ficam escondidas do utilizador, sendo estas criadas durante o processo de mapeamento. As folhas escondidas são sempre no mínimo duas, ficando colocadas sempre como as primeiras do livro. A primeira folha contém dados relativos à ontologia, sendo estes:

- *Namespace* da ontologia, armazenado na célula A1;
- Nome do grafo da ontologia, armazenado na célula A2;
- Quantidade de folhas de cálculo de persistência adicionais, armazenada na célula A3;
- Quantidade de classes mapeadas no livro, armazenada na célula A4;
- Identificador do livro, armazenado na célula A5.

Um exemplo destes dados é representado na Figura 20.

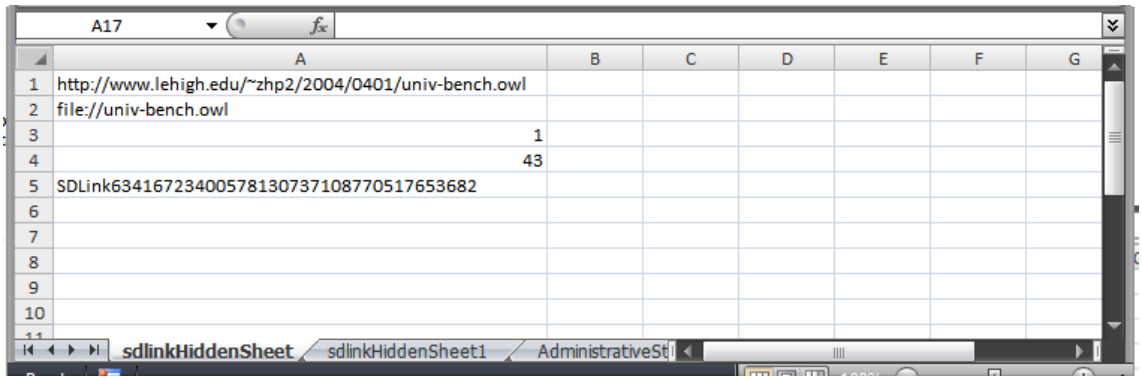

Figura 20 – Folha de cálculo principal de persistência

O identificador do livro é necessário para ser possível o registo de eventos que permite a validação de dados *online*, sendo este pormenor de implementação detalhado em 3.2.5.6.

A decisão do posicionamento das folhas escondidas está relacionada com o desempenho da solução. Quando o utilizador abre um livro no *Excel*, despoleta a execução de um método que analisa se o livro aberto é proveniente do *SDLink*. Este processo de análise deve ser o mais célere possível, como tal, o método limita-se a verificar se o nome da primeira folha do livro é "*sdlinkHiddenSheet*", caso seja, o método realiza o processo assíncrono de carregamento da ontologia para a memória.

As restantes folhas escondidas contêm a descrição de cada classe mapeada no livro, com a disposição de uma classe por topo de coluna e suas propriedades como segunda e demais linhas da mesma coluna, como ilustrado na Figura 21.

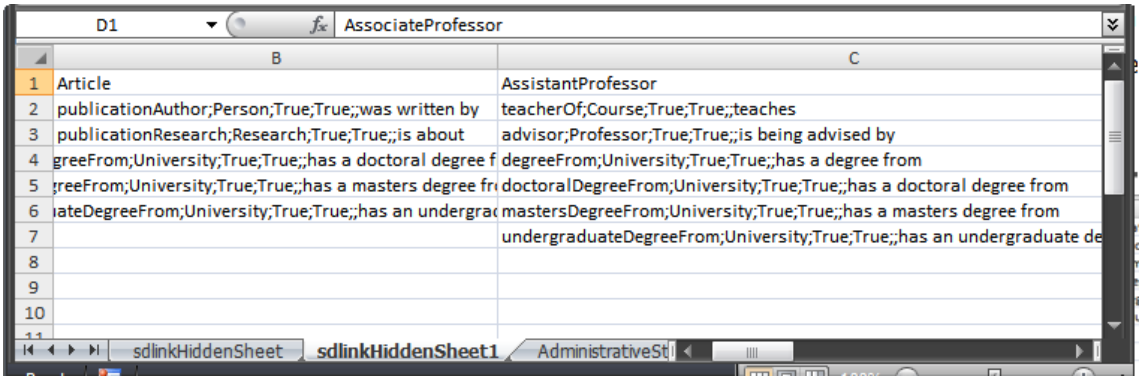

Figura 21 – Disposição de classes e suas propriedades

A quantidade de classes existentes por folha dependerá da versão do *Excel*, sendo esta determinada pela solução. Assim, é possível a criação na quantidade adequada das folhas adicionais para persistência da ontologia.

3.2.5.6 Validação

A validação é um processo composto por várias operações que são complementares. Estas operações ocorrem em diferentes fases da utilização do *Excel*. A validação *online* é uma destas operações, a qual ocorre durante a introdução de valores em propriedades do tipo `owl:ObjectProperty`. Nesta operação é apresentado ao utilizador, o formulário de validação contendo uma lista paginada com as instâncias dos recursos referidos pela propriedade. O formulário de validação está ilustrado na Figura 22.

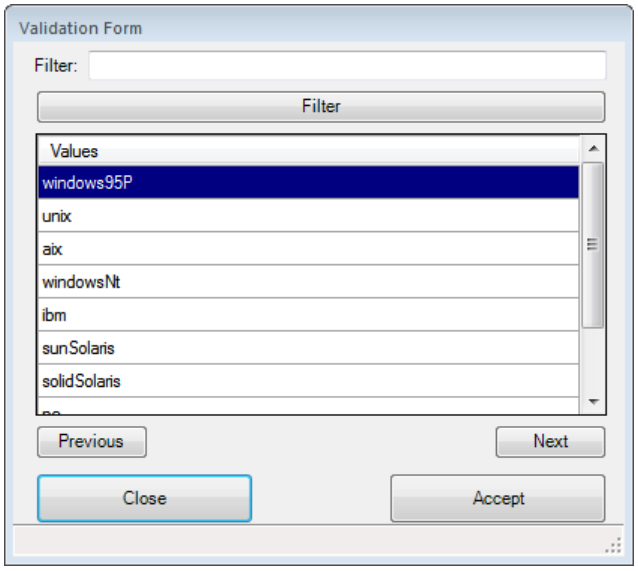

Figura 22 – Formulário de validação

O formulário de validação descende do formulário de execução assíncrona, beneficiando assim do processo de execução assíncrona com pós-execução síncrona e suporte a cancelamento. Adicionalmente suporta filtragem para permitir ao utilizador uma mais rápida obtenção de valores procurados em alternativa à navegação exaustiva nas páginas de resultados.

Devido ao facto da especificação SPARQL não definir uma directiva de contagem[21], a implementação de paginação neste formulário não pode determinar *a priori* a quantidade de páginas navegáveis. Como tal, o mecanismo implementado permite que o utilizador navegue até encontrar a primeira página sem resultados, após isto o formulário impede consequentes avanços no índice de paginação para permitir que o retorno à última página com resultados seja imediato.

Para a implementação da funcionalidade de validação *online* é necessário o registo de um *delegate* [34] no evento `SheetSelectionChange` da folha de cálculo. Este evento ocorre a

[21] Em SQL a directiva de contagem é a *count*.

cada mudança de selecção pelo utilizador nas células de uma determinada folha. Para determinar qual a folha onde ocorreu o evento foi necessária a utilização de um identificador global para cada livro. Esta necessidade surge do facto do *Excel* suportar a utilização simultânea de vários livros, exemplo ilustrado na Figura 23.

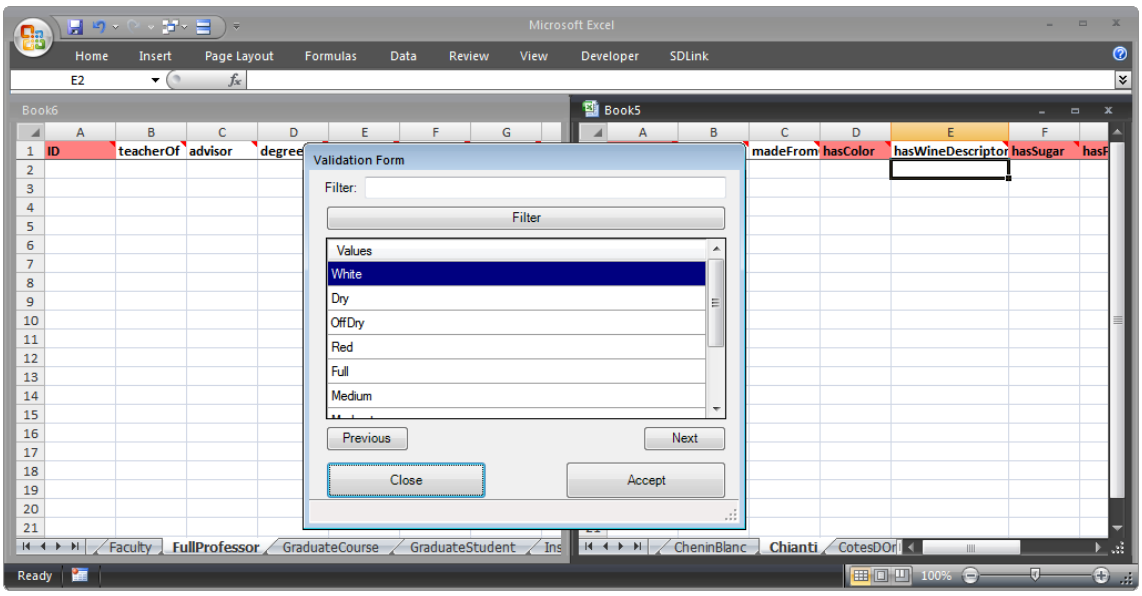

Figura 23 – Utilização de múltiplos livros no *Excel*

Desta forma, é necessária uma estrutura que permita associar a cada livro a respectiva ontologia carregada em memória. Como as referências para os objectos *Excel* são do tipo `TransparentProxy`, estas diferem consoante a acção que as obtém, isto é, o processo de criação de um livro *SDLink* cria uma referência para o livro que difere da referência oriunda do evento `SheetSelectionChange`. Este problema impossibilitava a identificação unívoca dos livros no tratamento do evento, impedindo o correcto funcionamento do processo de validação. A solução encontrada foi criar um identificador global para cada livro tirando partido de valores como o código de *hash* do livro criado durante o mapeamento, a data e hora do sistema convertida para *ticks* e a contagem de *ticks* desde o arranque do sistema. Desta forma é possível diminuir a probabilidade de colisão para $1/10^{35}$, ou seja, um em cem mil quintiliões.

Para além da validação *online*, acima descrita, existem ainda as operações de validação na introdução e validação no *upload*. A validação na introdução consiste em recorrer à validação de dados intrínseca do *Excel*, a qual tem utilidade apenas para verificação de valores inteiros positivos ou negativos e para valores booleanos. Para definir uma validação intrínseca do *Excel*, é necessário fornecer sempre o tipo de validação (inteiros, decimais, data, hora, lista, etc.), o operador matemático e, dependendo do operador, um ou os dois valores do intervalo. Os mesmos requisitos são aplicados quando programaticamente se define uma regra de validação

no *Excel*. Na Figura 24 é apresentado o formulário nativo do *Excel* para criação de regras de validação pelo utilizador, sendo o exemplo escolhido a verificação de inteiros positivos.

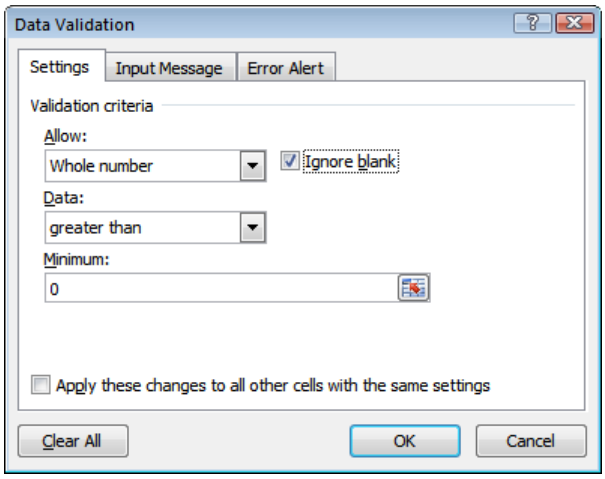

Figura 24 – Formulário do *Excel* para criação de regras de validação

Dos tipos existentes na especificação *XML Schema*, a validação de inteiros positivos e negativos é de directa implementação por ser possível definir um dos valores do intervalo, sendo impossível a implementação de uma regra de validação intrínseca que verifique apenas se um valor é inteiro, pois é sempre necessária a indicação de um operador e a definição de um ou dois valores para o operador seleccionado. Assim, para o caso do tipo `positiveInteger` é definida a regra de validação no *Excel* cujo tipo é `inteiro`, o operador é `maior que` e um dos valores do intervalo é definido com zero, como exemplificado na Figura 24. No caso de valores do tipo booleano é possível definir uma lista de validação com o conjunto de valores $\{true, false\}$ respectivamente. A Figura 25 ilustra a validação de valores booleanos através de uma lista.

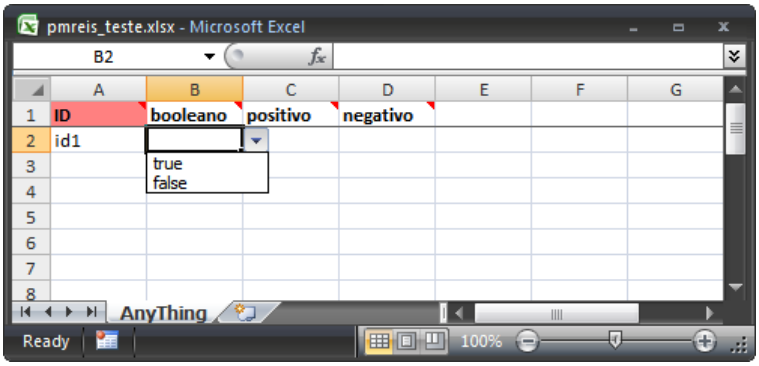

Figura 25 – Lista de validação para valores booleanos

Durante o processo de *upload*, ocorre a última operação de validação, consistindo na conversão explicita de literais para tipos internos *.NET*, evitando incoerências de tipos durante carregamento de dados para o *triplestore*. As incoerências de tipo, após o *upload*, têm como consequência causar excepções internas no repositório, sendo preferível uma excepção aplicacional no cliente durante o processo de *upload*, do que corromper o *triplestore* e invalidar o correcto funcionamento do mesmo.

A Listagem 9 representa um caso de incoerência de tipos, tendo o valor literal *onze* definido para uma propriedade que deverá receber valores numéricos do tipo inteiro.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:mus="http://musica.org/exemplo">

  <rdf:Description rdf:about="http://albuns/thebluenotebooks">
    <mus:compostoPor rdf:resource="http://musicos/richter/max" />
    <mus:temas rdf:datatype="http://www.w3.org/2001/XMLSchema#int">onze</mus:temas>
  </rdf:Description>

</rdf:RDF>
```

Listagem 9 - Exemplo de incoerências de tipo em RDF

Após o carregamento da Listagem 9 para o *triplestore* é possível realizar consultas que expõem falhas internas por inconsistência de tipo, sendo representado em primeiro lugar o resultado expectável de uma consulta ao repositório. Assim, num pedido de descrição do recurso *Max Richter* com a construção `describe <http://musicos/richter/max>`, o resultado obtido é ilustrado na Figura 26.

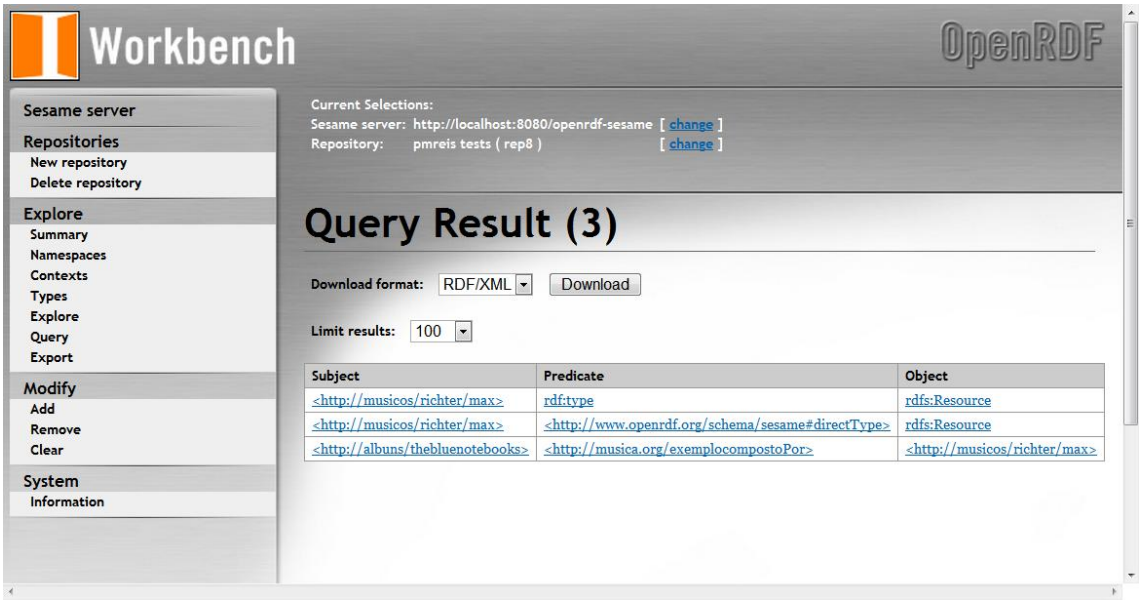

Figura 26 – Resultado normal da directiva para descrição de recursos

Caso seja utilizada a directiva `describe <http://albuns/thebluenotebooks>` o resultado, apresentado na Figura 27, é uma excepção no *triplestore*.

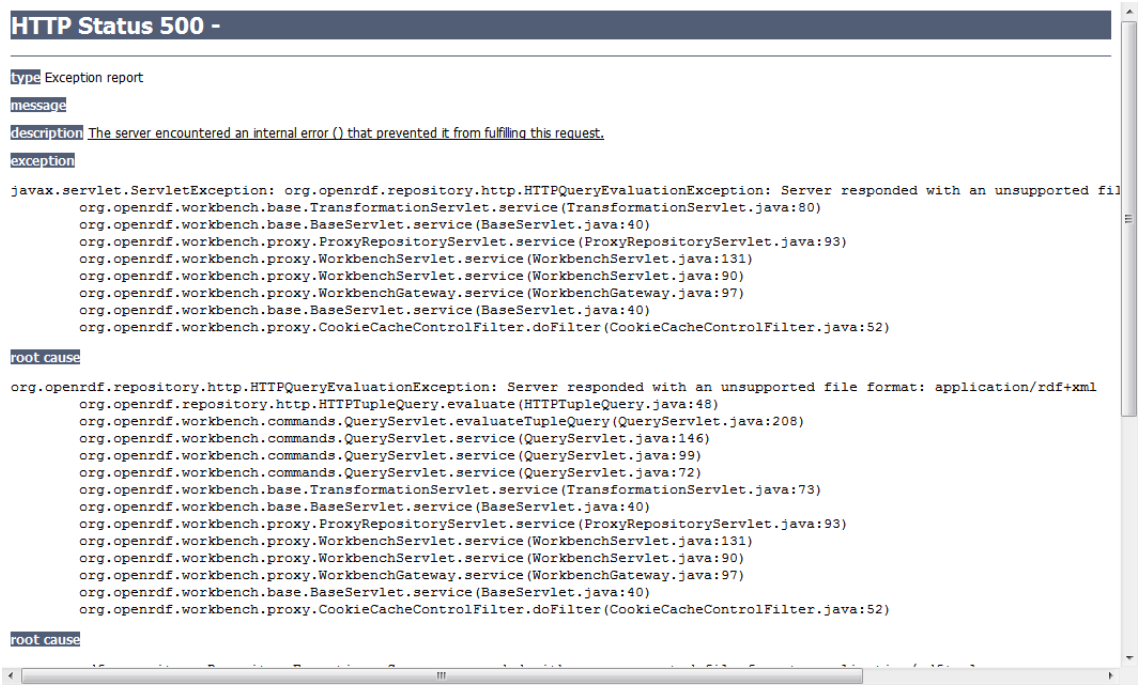

Figura 27 – Excepção no *triplestore* por incoerência de tipos

3.2.6 Configuração da aplicação

A aplicação permite a configuração de qual o repositório alvo para carregamento de dados. A configuração é suportada por um ficheiro `app.config` que se encontra em formato XML. A Listagem 10 apresenta a estrutura do ficheiro de configuração.

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>
  <appSettings>
    <add key="hostUrl" value="http://localhost:8080/openrdf-sesame/repositories/rep5" />
    <add key="recPerPage" value="10" />
  </appSettings>
</configuration>
```

Listagem 10 – Ficheiro de configuração

Os parâmetros envolvidos na configuração da aplicação são apenas dois: o URL do repositório e a quantidade de registos por página no formulário de validação.

Para evitar a alteração manual deste ficheiro, por parte do utilizador, a aplicação disponibiliza um formulário de configuração. Este formulário manipula o ficheiro, persistindo as configurações definidas pelo utilizador e, garante a adopção imediata dos parâmetros pela aplicação, ou seja, se durante o processo de introdução de dados nas folhas de cálculo, o utilizador pretender alterar a quantidade de registos por página, então basta aceder ao formulário de configuração e alterar o respectivo valor, confirmando de seguida a alteração. A Figura 28, apresentada de seguida, ilustra o formulário de configuração.

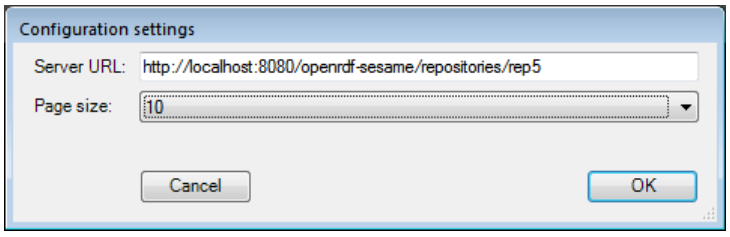

Figura 28 – Formulário de configuração

3.2.7 Modelo de instalação

Uma vez que são raros os postos de trabalho que têm simultaneamente as versões 2003 e 2007 do Excel, não é prudente realizar um instalável da solução que incorpore os PIAs para ambas as versões referidas. Durante o processo de instalação, é verificada a correspondência do *Office* para os PIAs a instalar, caso o instalável tenha ambas as versões dos PIAs e o posto de trabalho disponha apenas de uma versão do *Office*, a instalação será abortada. Como tal, é necessária a produção de dois instaláveis, um para cada versão do *Excel*. Este sistema prevê o suporte e manutenção, desta solução, quando aplicada às versões 2003 e 2007 do *Excel*.

O modelo de instalação das soluções VSTO exige a indicação explícita, por parte do utilizador, da confiança que este deposita no fornecedor da solução. Esta medida de segurança, designada de *Code Access Security* (CAS) [35], obriga à utilização da técnica de instalação *ClickOnce* [36] para que findando o processo de instalação o utilizador possa de seguida utilizar a solução. Este modelo recorre assim à utilização de certificados digitais X.509[22], que possibilita a utilização da cadeia de confiança inerente ao certificado, dispensando a confirmação explícita de confiança por parte do utilizador, quando o utilizador reconhece como sendo de confiança, uma das entidades pertencentes à cadeia relacionada com o certificado utilizado no instalável.

[22] http://www.itu.int/rec/T-REC-X.509/en

4 Contribuições adicionais e avaliação

Este capítulo expõe, na sua secção de contribuições adicionais, um conjunto de contributos organizacionais que têm impacto na infra-estrutura de suporte e metodologias de trabalho. Adicionalmente, na secção de avaliação da solução, são detalhados os principais aspectos que constituem vantagens e limitações da solução. Alguns dos aspectos referidos incidem sobre as tecnologias que constituem a base de trabalho deste projecto, sendo realizadas comparações com outras tecnologias utilizadas no âmbito da Engenharia Informática e de Computadores.

4.1 Contribuições adicionais

4.1.1 Infra-estrutura de suporte

Implementações baseadas em VSTO requerem um ambiente de desenvolvimento específico para cada versão do *Office*. Por exemplo, para produzir uma solução que estenda o *Excel* versão 2003 é necessária a instalação exclusiva da versão 2003 do *Office*. Isto tem implicações organizacionais e de planeamento infra-estrutural, pois caso uma instituição ou empresa pretenda desenvolver em simultâneo para múltiplas versões do *Office*, terá que ter postos de trabalho exclusivos para cada versão [37].

Soluções baseadas em VSTO, como é o caso da solução proposta, são consideradas *fat client*, ou seja, soluções que realizam uma parte importante do processamento no sistema, tendo algum nível de autonomia do servidor. Desta forma, este tipo de soluções requer obrigatoriamente a instalação de um conjunto de pré-requisitos no posto cliente. Isto é contrário ao que acontece nas soluções *Web*, que dependem apenas da existência de um *browser* que suporte as normas W3C. Assim, é também importante a verificação de conformidade do instalável com postos de trabalho que tenham apenas o *Excel* instalado, dispondo do mínimo de bibliotecas de suporte. Este tipo de verificação não pode ser realizado no ambiente de desenvolvimento, uma vez que este dispõe de todos os requisitos necessários à execução da solução.

Com base nas considerações acima descritas, foi definido como requisito de suporte ao desenvolvimento e testes deste projecto, um conjunto de quatro ambientes diferentes de trabalho. Um dos ambientes, dispondo do *Visual Studio* 2008 e do *Excel* 2007, permite o desenvolvimento da solução para a versão 2007 do *Excel*. Outro ambiente de desenvolvimento é requerido, desta feita, recorrendo ao *Visual Studio* 2008 e ao *Excel* 2003. Por fim são necessários mais dois sistemas para validação do instalável, em cada versão do *Excel*

respectivamente. Estes sistemas dispõem apenas do sistema operativo e de uma das versões do *Excel*.

Dispor de uma infra-estrutura deste tipo, é nos dias de hoje possível, sem incorrer em custos de aquisição de computadores, utilizando plataformas de virtualização. Desta forma, os ambientes de desenvolvimento e testes foram disponibilizados através da utilização do sistema de virtualização *Proxmox Virtual Environment* [38].

A Figura 29 ilustra a infra-estrutura utilizada para a realização deste projecto.

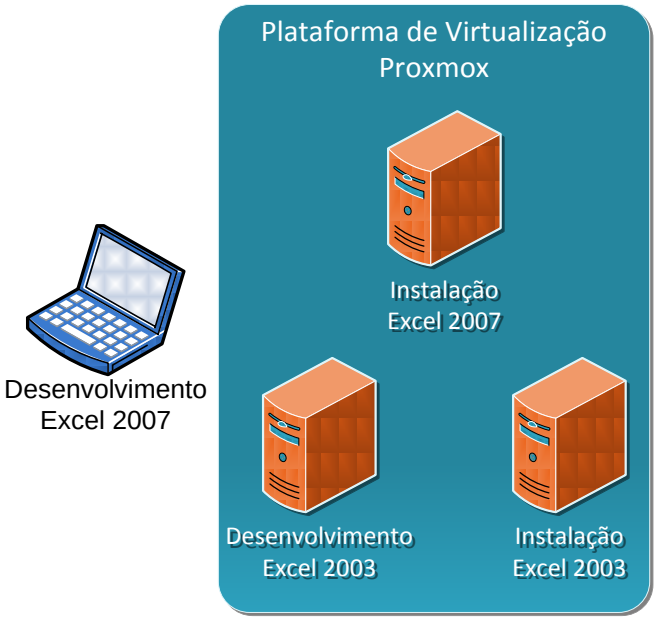

Figura 29 – Infra-estrutura de suporte ao desenvolvimento e testes

4.1.2 Metodologias

A forte dependência do *Excel* torna difícil a decomposição da solução, em partes reutilizáveis apenas por referência em outros projectos. Soluções VSTO estão dependentes do produto *Office*, não pelo modelo de objectos do produto, que são interfaces, mas sim pelas classes de suporte criadas automaticamente pelo *template* de projecto VSTO. Uma dessas classes é a `Globals` que permite o acesso à instância do *Excel* e seus controlos gráficos, nomeadamente o *Ribbon*, os menus e barras de ferramentas. Havendo necessidade de criar formulários como interface gráfica com o utilizador, estes estão dependentes da solução VSTO, porque têm que manipular classes funcionais que estão directamente dependentes do *Excel* e não são facilmente convertidas numa interface genérica.

A solução para o reaproveitamento passa pela organização do código fonte. Tendo a solução para *Excel* 2003 que obrigatoriamente implementar programaticamente a sua estrutura de menus, mantendo referências raiz para cada opção de menu criada. Assim, a utilização do código produzido é realizada através da cópia directa das classes e formulários do *SDLinkAddIn*, incluindo o ponto de entrada na solução: o ficheiro `ThisAddIn.cs`.

Os projectos *Visual Studio* referentes às bibliotecas *SDLinkLib* e *SDLinkUserControls* não necessitam de ser incorporados na solução *Visual Studio*, bastando apenas copiar os respectivos *assemblies* e definir as respectivas referências no projecto VSTO.

A Figura 31 apresenta o ambiente de desenvolvimento para a versão 2003 do *Excel*, já o com o código migrado e a solução em execução.

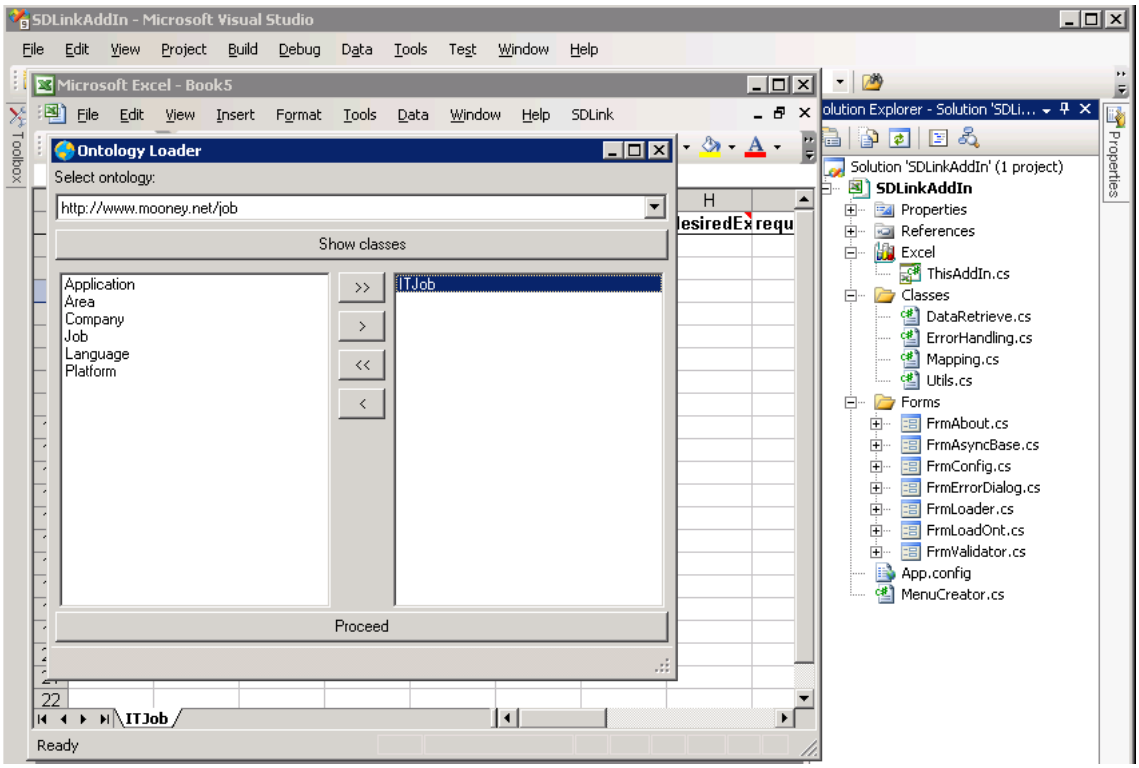

Figura 30 – Solução implementada na versão 2003 do *Excel*

Note-se que existe apenas um projecto no *Visual Studio*, o projecto base do VSTO. Isto pelo facto de serem aproveitadas as bibliotecas *SDLinkLib* e *SDLinkUserControls*, tornando desnecessária a migração de mais código.

4.2 Avaliação da solução

4.2.1 Vantagens da *Web* Semântica

O recurso às tecnologias e normalizações que constituem a *Web* Semântica tem várias vantagens na criação de sistemas de informação. As vantagens mais relevantes são apresentadas de seguida.

Heterogeneidade dos dados: num sistema de informação semântico e consequentemente na *Web* Semântica todos os dados são representados em RDF. Isto permite que vários repositórios de triplos possam ser consultados em simultâneo sem o problema da representação dos dados existentes em cada repositório, como acontece com as representações em bases de dados relacionais.

Expressividade relacional: muitas ontologias representam conceitos que têm múltiplas relações entre si. Num modelo relacional é possível que os diagramas ilustrem as diferentes relações através do uso de um verbo, no entanto, este verbo não é persistido fisicamente na base de dados, tornando difícil a consulta dos dados que pertençam a apenas uma das relações. Por exemplo, considerando uma ontologia com a classe `Pessoa`, esta classe pode ser relacionada consigo mesmo através de predicados tais como: `temAscendente`, `temDescendente`, `temIrmão`, entre outros. Assim, torna-se mais natural realizar interrogações em SPARQL do tipo: quais os irmãos da Alice? Numa base de dados relacional, para realizar esta consulta, é necessária a utilização de uma junção[23], referindo explicitamente a chave primária da tabela `Pessoa` e a chave estrangeira que refere a relação de irmão.

Inferência: a utilização do OWL para descrever ontologias permite a definição de conceitos através de condições necessárias e suficientes. Em sistemas onde o estado dos dados varie ao longo do tempo, as conclusões sobre as instâncias da ontologia variam dinamicamente de acordo com as alterações introduzidas. Este tipo de inferência não é possível num sistema de informação relacional, uma vez que não é possível definir conceitos com base em condições necessárias e suficientes.

Adaptabilidade: as tecnologias da *Web* Semântica permitem que um sistema de informação semântico acompanhe, sem grandes penalizações, as evoluções do conhecimento humano. Isto é suportado pela facilidade em criar alterações em ontologias através da simples manipulação de afirmações. Este facto evita processos de migração de dados, como acontece nos sistemas de bases de dados relacionais. Adicionalmente, recorrendo a nós anónimos, podem ser

[23] Termo em inglês é *inner-join*.

realizadas afirmações nas quais o sujeito ou o objecto é ainda desconhecido, podendo no futuro serem definidos esses nós de acordo com a descoberta de novo conhecimento.

Integração do conhecimento: um dos principais problemas relacionado com a evolução do conhecimento humano é a segregação. Esta segregação é ainda constatável na actualidade, uma vez que existem múltiplas disciplinas de estudo. As tecnologias subjacentes à *Web* Semântica permitem a obtenção de respostas a questões que atravessem mais que um domínio do conhecimento. Isto não seria possível sem *Linked Data*[24] e ontologias representadas numa linguagem que seja comum a todos os sistemas de informação semânticos, como é o caso do OWL. Adicionalmente, como as ontologias são grafos, é possível a integração de várias ontologias através da junção dos respectivos grafos. Numa base de dados relacional, a junção de dois esquemas relacionais que tenham colisão de conceitos é bastante problemática. As tecnologias da *Web* Semântica resolvem este problema através do uso dos *namespaces*, permitindo que a integração de ontologias seja realizada de forma simples, bastando apenas a indicação, através de afirmações, de quais os conceitos das ontologias que são considerados equivalentes.

Colaboração em comunidade: embora este aspecto não esteja directamente relacionado com a *Web* Semântica, um dos principais motivadores para a colaboração de comunidades em diversos projectos tem sido a existência da *Web*. O trabalho destas comunidades é geralmente disponibilizado ao abrigo de licenças como a General Public Licence (GLP) [39]. Isto permite o aparecimento de ferramentas sem custos para o utilizador final e cujo código fonte se encontra disponível. A disponibilização do código fonte permite que ferramentas existentes sejam estendidas e melhoradas, sendo estas alterações disponibilizadas de novo ao público em geral. Existem várias extensões para *Protégé* e *Sesame*, que demonstram valiosos contributos por comunidades externas a estes projectos.

4.2.2 Vantagens da Implementação

Esta implementação oferece um conjunto de vantagens para futuros projectos que recorram à *.NET Framework* ou ao *Excel*. Algumas destas vantagens são apresentadas de seguida.

Processo de engenharia reversa: não é usual existirem projectos que utilizem SPARQL para a obtenção de uma ou mais ontologias armazenadas num repositório de triplos. À data de publicação deste relatório, não foi encontrado outro projecto no universo de soluções para a *Web* Semântica que o fizesse. Este processo poderá ser relevante para projectos onde é

[24] Dados no formato RDF que se encontrem disponíveis na *Web*.

necessária a criação de dados de acordo com ontologias armazenadas em *triplestores*. Algumas das interrogações utilizadas para a obtenção da ontologia são complexas, pelo que, projectos que as reutilizem poderão ter como resultado directo a diminuição do tempo dispendido no desenvolvimento. Este processo abre caminho para a criação dinâmica de interfaces com o utilizador a partir de repositórios de triplos remotos. A solução implementada utiliza parte deste benefício na criação e formatação de folhas de cálculo.

Independência da versão do *Excel*: apesar de parte do código desenvolvido ser dependente do *Excel*, existem garantias de independência da versão do *Excel* ao recorrer à *.NET Framework* em detrimento de outras tecnologias de extensibilidade. Desta forma, através da utilização de um modelo de objectos baseado em interfaces, o código produzido neste projecto pode ser reutilizado em futuras versões do *Excel*.

Robustez da solução: a implementação foi verificada através de um conjunto de testes unitários, testes manuais e testes de carga. Uma vez que a solução recorre ao *Excel* como interface gráfica, são fundamentais os testes manuais para validar grande parte da implementação. Dos testes de carga realizados, constatou-se que a solução suporta ontologias com duas dezenas de milhar de classes. O *Excel* é o principal factor condicionante, devido à latência existente nas operações de manipulação de livros com a quantidade de folhas de cálculo já referida. Assim, o código produzido poderá tirar partido de eventuais melhorias na manipulação de livros com grandes quantidades de folhas de cálculo em futuras versões do *Excel*.

4.2.3 Limitações da *Web* Semântica

A expressividade adicional imposta pelo paradigma semântico, em particular na construção de ontologias, pode tornar-se um obstáculo quando os modelos a representar têm maioritariamente relações de composição. Nestes casos o mesmo predicado ou propriedade não pode ser reutilizado para relacionar todos os conceitos para os quais a relação de composição é aplicável, isto porque em OWL a utilização de vários domínios *(domains)* e intervalos *(ranges)* em propriedades têm semântica de intersecção. Assim, é difícil criar ontologias que não sejam taxonómicas e que tenham várias relações de composição. Para tal é necessário utilizar todos os sinónimos conhecidos para descrever relações do tipo: `A compostoPor B`.

Para a consulta a uma determinada ontologia, geralmente realizadas através de SPARQL, o utilizador mesmo que conheça bem o domínio de conhecimento, pode não conhecer os predicados que foram utilizados na criação da ontologia. Por exemplo, considerando uma ontologia que contenha a classe `Escola` e esta classe tenha a propriedade `Nome`, é espectável

que o predicado utilizado para ligar a `Escola` e `Nome` seja: `temNome`. No entanto, pode ter sido utilizado outro predicado, por exemplo, `nomeDoEstabelecimento`. Isto torna mais difícil a criação de interrogações SPARQL do que interrogações SQL, quando o objectivo é apenas consultar propriedades de conceitos. Na Figura 31 é ilustrada esta diferença, sendo `ns` o hipotético *namespace* da ontologia.

SQL	SPARQL

`select Nome from Escola`	`select ?Nome where {`
	`  ?Escola a ns:Escola ;`
	`  ns:nomeDoEstabelecimento ?Nome`
	`}`

Figura 31 – Consulta de propriedades em SQL e SPARQL

O exemplo da consulta SPARQL, apresentado na Figura 31, é adaptado de um dos exemplos de consultas [40] aos dados governamentais disponibilizados no Reino Unido [41].

Existem alguns casos de modelação que podem originar conclusões ou resultados absurdos na concepção de ontologias, noutros casos ainda, por excesso de flexibilidade, são criadas dificuldades na concepção de interfaces com o utilizador que validem os dados de acordo com ontologias. Deveria ser realizado um trabalho exaustivo para a identificação destes casos, trazendo benefícios para a comunidade que desenvolve trabalho na área da *Web* Semântica.

São de seguida apresentados alguns casos problemáticos, identificados ao longo da realização deste projecto.

Propriedades transitivas e funcionais: uma propriedade não deve ser definida como transitiva e funcional ou funcional inversa[25]. A combinação destas características é permitida na *Web* Semântica, mas não faz sentido.

Figura 32 – Propriedade transitiva em OWL

Na Figura 32, está representada a propriedade transitiva `temAscendente`, através da qual é possível concluir que Carlos é ascendente de Alice, mesmo não estando estes últimos directamente relacionados. No entanto, ao ser adicionada a esta propriedade a característica de

[25] Termo original é *inverse functional*.

funcional, leva a que o motor de inferência conclua que Bruno e Carlos são a mesma pessoa, o que não é verdadeiro. Casos como este, podem surgir porque as ferramentas de modelação não advertem ou impedem esta combinação.

Definição de propriedades sem *domain* ou com *domain owl:Thing*: por omissão uma propriedade sem *domain* é aplicada a todas as classes da ontologia, seguindo a regra que todas as classes definidas na ontologia são subclasses da classe `owl:Thing`. No entanto, constata-se que alguns destes axiomas originam resultados despropositados, nomeadamente na ontologia dos vinhos disponibilizada pela própria W3C a propriedade `locatedIn` é definida para todas as classes da ontologia através da atribuição da mesma à classe `owl:Thing`, como exemplificado na Listagem 11.

```
<owl:ObjectProperty rdf:ID="locatedIn">
        <rdf:type rdf:resource="&owl;TransitiveProperty" />
        <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
        <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>
```

Listagem 11 – Excerto da ontologia dos vinhos da W3C

Esta propriedade é assim aplicada a classes tais como `wineTaste`, o que não faz sentido pois o sabor do vinho não está localizado numa região vinícola. A liberdade de poder associar uma propriedade ao universo de coisas existentes numa ontologia, é pouco razoável principalmente se considerarmos que o futuro da *Web* Semântica pretende que várias ontologias sejam partilhadas e usadas em simultâneo. Então, permitir a definição de uma propriedade como universal, é permitir que alguém determine propriedades aplicáveis a todos os domínios do conhecimento, podendo originar inconsistências.

Definição de propriedades sem *range*: propriedades sem range impedem a criação adequada de interfaces, com o utilizador. Não é possível determinar que tipo de dados devem ser introduzidos para determinadas propriedades, embora a propriedade tenha uma semântica que torne evidente ao utilizador qual o tipo de dados a introduzir. Na Listagem 12 é apresentado um excerto de uma ontologia produzida pela *Lehigh University*.

```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:label>is age</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
</owl:DatatypeProperty>
```

Listagem 12 – Exemplo de propriedade sem *range*

Embora seja evidente para um humano que a propriedade idade deve ser um inteiro positivo, numa interface gráfica a falta desta informação de forma explícita pode dar origem a erros na introdução dos dados, tendo consequências sérias posteriormente. Por exemplo, se existirem valores para idades introduzidos por extenso em vez de numericamente, não será possível determinar indicadores tais como média de idades de uma população. A W3C já está atenta a este problema, que foi abordado num *working draft* do OWL 2 [42], resultado do contributo da comunidade de utilizadores do *Protégé*.

Garantia de cardinalidade: como não existe garantia de nomes únicos na *Web*, através da *No Unique Names Assumption*, não é possível que restrições de cardinalidade sejam correctamente asseguradas. Sempre que um recurso for identificado por dois ou mais URIs distintos, qualquer restrição de cardinalidade será incoerente, pois o mesmo recurso pode aparecer múltiplas vezes na relação e impedir a participação de outros recursos na relação. Este cenário é possível quando um recurso está disponível em vários locais distintos na *Web*, não havendo controlo sobre a diversidade de URIs que podem identificar o mesmo. Este problema irá certamente ocorrer à escala global, devido ao âmbito de utilização da *Web* Semântica.

Aplicação de condição necessária e suficiente: algumas classes podem ser definidas através das condições necessárias e suficientes permitindo inferência em ambos os sentidos da relação. Caso seja definida uma condição necessária e suficiente através da primitiva `owl:allValuesFrom` a implicação não irá funcionar num dos sentidos pelo facto desta última não constituir uma restrição existencial.

```
<owl:Class rdf:ID="Vegetariano">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#come" />
      <owl:allValuesFrom rdf:resource="#Vegetais" />
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>

<Alice>
  <rdf:type rdf:resource="#Vegetariano"
</Alice>
```

Listagem 13 – Caso excepcional em condição necessária e suficiente

Do exemplo apresentado na Listagem 13, a definição directa de Alice como vegetariana não permitirá inferir que Alice come vegetais, mesmo tendo sido utilizada uma construção que é considerada necessária e suficiente.

As ferramentas de modelação no contexto da Engenharia do Conhecimento são fundamentais e devem facilitar a consulta visual dos modelos em concepção. Constatou-se que é ainda difícil a consulta de ontologias em ferramentas como o *Protégé* por não suportar a visualização de propriedades das classes, nem as relações entre classes para além da relação taxonómica. Isto é particularmente útil para validar que dados são aceites como propriedades das classes. A Figura 33 ilustra como é representada, por omissão, uma ontologia no *Protégé*.

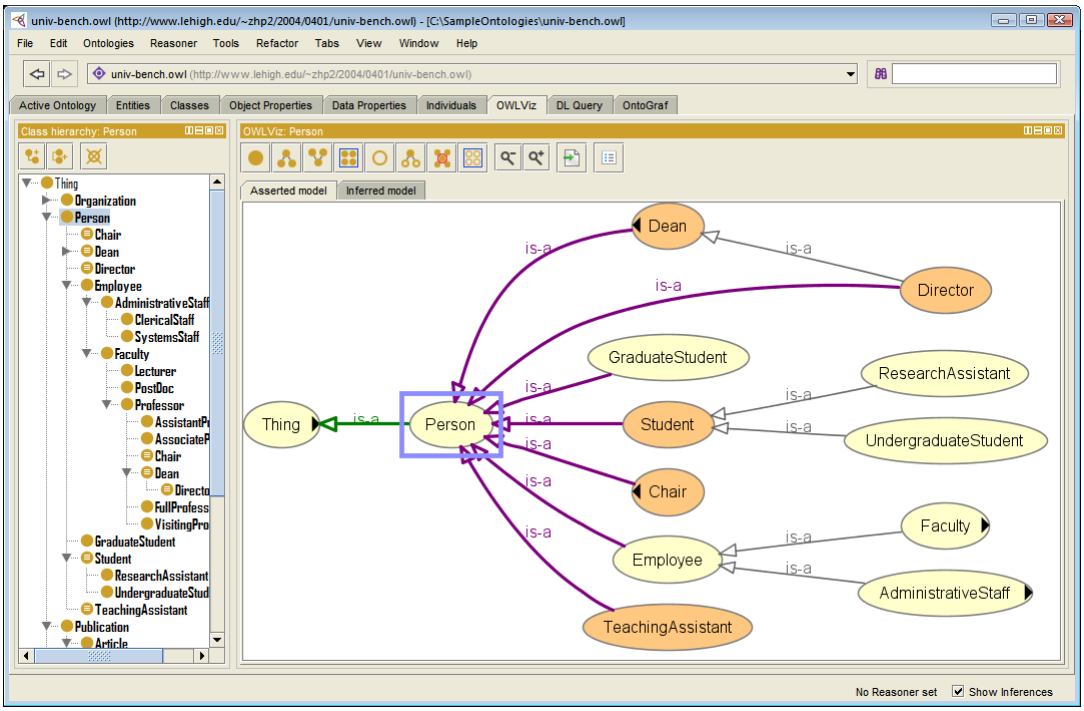

Figura 33 – Visualização de ontologia em *Protégé* através do *OWLViz*

Alguns *plug-ins* existentes para o *Protégé* podem abordar algumas das suas limitações de representação gráfica da ontologia, como a consulta de relações entre conceitos nas ontologias que não sejam apenas taxonómicas. No entanto, ainda não possibilita a consulta de propriedades literais. De um modo geral, ferramentas como o *Protégé* ainda não oferecem o mesmo nível de usabilidade na criação e consulta de ontologias como ferramentas de modelação existentes para outras linguagens, como o UML. A Figura 34 ilustra um contributo recente para a visualização de ontologias no *Protége*.

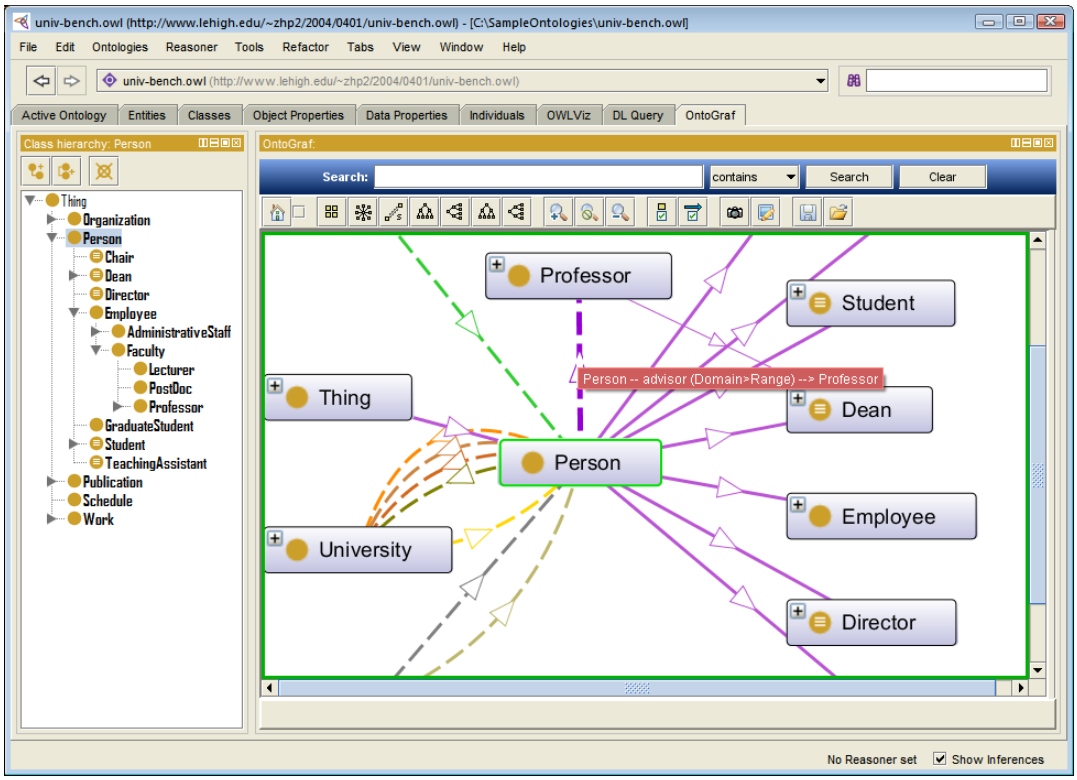

Figura 34 - Visualização de ontologia em *Protégé* através do *OntoGraf*

4.2.4 Limitações do *Excel*

4.2.4.1 Especificações técnicas

A quantidade de dados ou quantidade de propriedades suportadas na solução depende principalmente da versão do *Excel*. A Figura 35 detalha as características na especificação do *Excel* que têm impacto neste projecto.

	***Excel* 2003**	***Excel* 2007**
Colunas por folha	256	16.384
Linhas por folha	65.536	1.048.576
Dimensão da célula	32.767	32.767

Figura 35 – Especificações principais do *Excel* por versão

As colunas por folha determinam a quantidade máxima de propriedades em cada classe da ontologia. A quantidade de linhas por folha determina a quantidade máxima de instâncias por classe que o utilizador pode introduzir. A dimensão da célula determina a quantidade máxima de caracteres possíveis nas propriedades das classes. Considerando propriedades que refiram

outros recursos e sem restrição de cardinalidade, a quantidade máxima de instâncias referidas na propriedade será o resultado da expressão ilustrada na Figura 36.

$$\frac{Dimensão\ da\ célula}{tamanho(identificador)}$$

Figura 36 – Expressão da quantidade máxima de referências por célula

Na Figura 36, a variável $identificador$ representa o URI do recurso. Assim, considerando que o identificador de instâncias de uma classe tem 4 caracteres de tamanho, uma propriedade que refira outra classe e que não tenha restrições de cardinalidade poderá referir no máximo 8191 instâncias.

4.2.4.2 Recolha de referências managed

Na versão 2007 do *Excel* foi introduzido um novo conceito de barra de ferramentas designada de *Ribbon*. Isto facilita o desenvolvimento, pois é possível utilizar em tempo de desenho, um editor visual deste novo conceito. Ao usar este editor é garantida a existência de uma referência raiz *managed* para o *Ribbon* em tempo de execução e consequentemente para todos os controlos contidos no mesmo. Isto inviabiliza a recolha pelo *Garbage Collector* (GC) das referências para os controlos.

Um editor equivalente não existe na versão 2003, pelo que toda a criação de opções de menu, barras de ferramentas e comandos nelas contidos terão que ser criados programaticamente. Se não forem tomadas as medidas adequadas haverá recolha das referências *managed* para os objectos nativos do *Excel* e consequentemente estes não terão qualquer tratamento de eventos associado. Para que os controlos continuem a ter representação *managed* cabe ao programador criar referências raiz para os mesmos. Como exemplo, caso existam opções de menu implementadas na versão 2003 do *Excel* que não disponham de referências raiz em ambiente *managed*, o utilizador verá as opções no menu aplicacional durante todo o tempo de vida da aplicação. No entanto, a utilização destas opções não irá produzir qualquer resultado.

4.2.4.3 Validação

Os valores existentes nas células de uma folha de cálculo podem ser formatados, mas o formato destes não determina o seu tipo. Mais precisamente, não é por atribuir o formato numérico a uma determinada célula, que passa a existir garantia que os valores da célula são

do tipo numérico. Por conseguinte, é possível a existência de valores alfanuméricos em células cujo âmbito de utilização seja para valores numéricos e se encontrem formatadas para valores numéricos.

Uma outra possibilidade de validação de valores encontra-se no facto do *Excel* permitir listas de validação. No entanto, verificou-se que estas listas de validação suportam no máximo 256 caracteres, sendo a lista uma *string* de valores separada por ponto e vírgula. A falta de suporte para listas como mais de 256 caracteres só é evidente quando se grava um livro *Excel* onde haja pelo menos uma célula com uma lista nas condições já referidas. Quando se volta a abrir o livro *Excel* é lançada uma excepção de alerta ao utilizador, referindo que os valores da lista de validação não puderam ser carregados.

Adicionalmente, não há forma de utilizar funcionalidades internas ao *Excel* para evitar que o utilizador deixe uma célula vazia, impedindo a validação imediata de uma restrição existencial. Desta forma, a implementação tem que assegurar a validação de restrições existenciais num único evento global de utilização, ocorrendo imediatamente antes do *upload* dos dados para o *triplestore*.

O *Excel* não é a ferramenta ideal para validação de tipos de dados, permitindo apenas validações elementares de valores. Para validações mais elaboradas, cabe ao programador desenvolver rotinas de validação. As validações de tipos são as mais penosas devido à multiplicidade de tipos existentes e ao facto de alguns dos tipos em XSD não terem correspondência directa nos tipos da *.NET Framework*. Sabendo que os valores existentes nas células são todos do tipo *string*, torna-se mais difícil esta validação devido à existência de notações que embora válidas, não sejam passíveis de conversão directa, como acontece com a notação científica[26].

4.2.4.4 Desempenho

O modelo de objectos do *Excel* não permite a execução em paralelo de tarefas cíclicas, como acontece durante a criação das folhas de cálculo de um novo livro. Embora o modelo de objectos vise a reutilização de código nas soluções VSTO para versões futuras do *Excel*, através do uso de interfaces, a representação directa em interface gráfica de alguns objectos, impede a utilização de algoritmos de execução paralela. O ideal seria o modelo de objectos permitir a criação em memória de folhas de cálculo e posterior representação visual das mesmas, potenciando a criação em paralelo.

[26] Por exemplo: 1,23E+08

5 Conclusões e trabalho futuro

5.1 Conclusões

Este projecto demonstra que é possível adoptar o *Excel* enquanto interface com o utilizador, num sistema de informação semântico. A implementação realizada serve de prova de conceito para outros projectos, no âmbito da *Web* Semântica, que visem a criação de interfaces dinamicamente com base em ontologias.

A solução implementada será adoptada por um conjunto de instituições de investigação, nacionais e internacionais, nas áreas das Ciências da Vida.

Tanto quanto é do conhecimento do autor, este projecto é único no universo de implementações para a *Web* Semântica, devido ao facto de construir parte da interface gráfica a partir de ontologias, recorrendo a um *endpoint* SPARQL. Para além disto, a implementação permite a introdução de dados de acordo com ontologias, o que não é típico nas aplicações já conhecidas para a *Web* Semântica. O ciclo de vida dos dados num sistema de informação começa necessariamente pela criação dos mesmos. Havendo ontologias criadas para representar determinado domínio de conhecimento, então faz sentido tirar partido dessas ontologias para a criação de dados nesse domínio. Este aspecto não é abordado na maioria da bibliografia e referências utilizadas para a realização deste projecto. A maioria dos projectos desenvolvidos para a *Web* Semântica dispõe de interfaces para a consulta de dados. Assim, os exemplos citados na bibliográfica deixam sempre uma dúvida subjacente: como é que os dados surgem nos sistemas de informação semânticos em primeiro lugar? Parece evidente, após o estudo de algumas referências [43], que a maioria dos sistemas de informação que dão lugar ao aparecimento de *Linked Data* e consequentemente vão dando forma à *Web* Semântica, são sistemas de informação relacionais cujos dados são exportados para RDF e disponibilizados na *Web*. Este aspecto torna mais relevante ainda, a criação de interfaces para o carregamento de dados de acordo com ontologias, pois permitirá que instituições recém-criadas reduzam os custos iniciais na implementação dos seus sistemas de informação semânticos, quando estas não possuam dados como património institucional.

Este projecto permitiu que o autor tomasse conhecimento de um paradigma que embora não seja recente, está relacionado com sistemas de informação, sendo esta uma temática que o autor considera relevante. É importante ter a noção que o conhecimento será certamente o património mais valioso da humanidade. Sem uma sistematização do conhecimento, não haverá progresso na descoberta de soluções para os problemas da nossa sociedade. Esta sistematização passa necessariamente por redução da dispersão, segregação e desorganização do conhecimento, sendo esta uma responsabilidade de todos e não apenas dos informáticos. Este projecto permitiu consolidar ainda mais estas convicções no autor, demonstrando como a

participação directa na área das ciências da informação e computação são fundamentais para uma melhor e mais fundada abordagem aos problemas da representação do conhecimento.

5.2 Desafios e dificuldades

Para a realização deste projecto foi necessário o estudo e análise das tecnologias relacionadas com a *Web* Semântica, tendo estas actividades representado a fase mais longa do projecto. As recomendações W3C não são a fonte mais indicada para iniciar o estudo a problemas relacionados com semântica e representação de conhecimento, sendo os seus conteúdos ambíguos em alguns exemplos. Isto tornou mais difícil a tarefa de compreender a aplicabilidade das primitivas da linguagem OWL, pelo que foi necessária a identificação de bibliografia de suporte. As disciplinas de sistemas de informação, existentes no programa curricular da licenciatura, permitem a criação de analogias entre o paradigma semântico e relacional, no entanto, foi considerável o esforço necessário para a aprendizagem das tecnologias e linguagens da *Web* Semântica usadas neste projecto.

A utilização do *Excel* como interface gráfica também teve impacto no ritmo de desenvolvimento da solução. Foi necessária a aprendizagem do modelo de objectos do *Excel* e lidar com um conjunto de métodos cuja sintaxe de utilização não era simples. Os actuais ambientes de desenvolvimento integrado, como é o caso do *Visual Studio*, facilitam o processo de desenvolvimento através de mecanismos como o *IntelliSense*. Como o modelo de objectos de aplicações *Office* recorre com frequência a métodos como parâmetros opcionais, o *IntelliSense* não tem muita utilidade, pois os métodos aceitam maioritariamente argumentos do tipo `Object`, sendo necessária a consulta de conteúdos especializados[27] para progredir no trabalho.

Outro factor condicionante, foi a escassez de ferramentas concebidas com a *.NET Framework* para a *Web* Semântica. As ferramentas analisadas[28] revelaram ser inadequadas para este projecto. Desta forma, não foi possível a adopção de ferramentas que abreviassem toda a análise necessária para a definição das regras de mapeamento.

5.3 Trabalho futuro

Em Outubro de 2009 a W3C publicou a recomendação para o OWL 2.0, tendo sido disponibilizada a primeira versão do *Protégé*, com suporte total para esta nova recomendação, apenas em meados de Junho de 2010. Como trabalho futuro deverá ser considerada uma

[27] Como é o caso da *Excel 2007 Developer Reference*.

[28] ROWLEX (http://rowlex.nc3a.nato.int/) e SemWeb.NET (http://razor.occams.info/code/semweb/).

análise ao OWL 2.0 e eventual implementação de suporte às novas primitivas da linguagem, sendo recomendada a espera por bibliografia de suporte.

Sabendo que o *Excel* é uma ferramenta propensa à produção de gráficos, é possível projectar um módulo de descarga de dados do *triplestore* a partir das consultas mais ocorrentes, permitindo que os utilizadores criem os seus gráficos de análise. Isto terá a vantagem dos utilizadores recorrerem aos dados do *triplestore* para produzir gráficos, em vez dos dados existentes nas folhas de cálculo usadas para carregamento. Outra vantagem reside no facto dos utilizadores dispensarem a aprendizagem de uma nova ferramenta para a produção dos seus gráficos.

É também possível a implementação de uma interface *Web*, similar a uma folha de cálculo, que permita o carregamento de dados para o *triplestore*. Caso seja utilizada uma ferramenta como o *Google Web Toolkit* [44], será também possível a produção de gráficos sobre os dados do *triplestore*, permitindo que o utilizador usufrua de uma interface rica e disponibilizando acesso aos dados a partir de qualquer computador com acesso à *Web*.

Referências

[1] World Wide Web Consortium (W3C). Semantic Web - W3C. [Online]. http://www.w3.org/standards/semanticweb/

[2] TED. (2009) Tim Berners-Lee on the next Web. [Online]. http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html

[3] World Wide Web Consortium (W3C). OWL Web Ontology Language Reference. [Online]. http://www.w3.org/TR/owl-ref/

[4] Microsoft. Excel. [Online]. http://office.microsoft.com/en-us/excel/default.aspx

[5] Microsoft. .NET Framework. [Online]. http://www.microsoft.com/NET/

[6] Microsoft. Office Development with Visual Studio. [Online]. http://msdn.microsoft.com/en-us/vsto/default.aspx

[7] Microsoft. Primary Interop Assemblies (PIAs). [Online]. http://msdn.microsoft.com/en-us/library/aa302338.aspx

[8] Unicode Consortium. Unicode Standard. [Online]. http://www.unicode.org/standard/standard.html

[9] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.1. [Online]. http://www.w3.org/TR/2006/REC-xml11-20060816/

[10] World Wide Web Consortium (W3C). RDF - Semantic Web Standards. [Online]. http://www.w3.org/RDF/

[11] Oracle. Java. [Online]. http://www.oracle.com/technetwork/java/index.html

[12] R. T. Fielding. (2000) Representational State Transfer (REST). [Online]. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[13] World Wide Web Consortium (W3C). Notation3 (N3) A readable RDF syntax. [Online]. http://www.w3.org/DesignIssues/Notation3.html

[14] World Wide Web Consortium (W3C). Turtle - Terse RDF Triple Language. [Online]. http://www.w3.org/TeamSubmission/turtle/

[15] World Wide Web Consortium (W3C). RDF/XML Syntax Specification. [Online]. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

[16] Object Management Group (OMG). Unified Modeling Language (UML). [Online]. http://www.uml.org/

[17] Microsoft. The C# Language. [Online]. http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx

[18] Stanford Center for Biomedical Informatics Research. The Protégé Ontology Editor and Knowledge Acquisition System. [Online]. http://protege.stanford.edu/

[19] World Wide Web Consortium (W3C). SPARQL Query Language for RDF. [Online]. http://www.w3.org/TR/rdf-sparql-query/

[20] SYSTAP, LLC. Bigdata. [Online]. http://www.systap.com/bigdata.htm

[21] D. Tsarkov. factplusplus. [Online]. http://code.google.com/p/factplusplus/

[22] Clark & Parsia. Pellet: The Open Source OWL Reasoner. [Online]. http://clarkparsia.com/pellet

[23] Ontotext. OWLIM - OWL Semantic Repository. [Online]. http://www.ontotext.com/owlim/index.html

[24] JSON.org. JSON. [Online]. http://www.json.org/

[25] Microsoft. COM: Component Object Model Technologies. [Online]. http://www.microsoft.com/com/default.mspx

[26] Microsoft. Office Online. [Online]. http://office.microsoft.com/en-us/default.aspx

[27] Microsoft. Dictionary Class. [Online]. http://msdn.microsoft.com/en-us/library/xfhwa508.aspx

[28] openRDF.org. openRDF.org. [Online]. http://www.openrdf.org/

[29] World Wide Web Consortium (W3C). HTTP - Hypertext Transfer Protocol Overview. [Online]. http://www.w3.org/Protocols/

[30] World Wide Web Consortium (W3C). SPARQL Query Results XML Format. [Online]. http://www.w3.org/TR/rdf-sparql-XMLres/

[31] World Wide Web Consortium (W3C). Wine Ontology. [Online]. http://www.w3.org/TR/owl-guide/wine.rdf

[32] World Wide Web Consortium (W3C). Serializing SPARQL Query Results in JSON. [Online]. http://www.w3.org/TR/rdf-sparql-json-res/

[33] Microsoft. Single-Threaded Apartments (COM). [Online]. http://msdn.microsoft.com/en-us/library/ms680112%28VS.85%29.aspx

[34] Microsoft. Delegate Class. [Online]. http://msdn.microsoft.com/en-us/library/system.delegate.aspx

[35] Microsoft. Code Access Security. [Online]. http://msdn.microsoft.com/en-us/library/930b76w0%28VS.71%29.aspx

[36] Microsoft. ClickOnce Deployment. [Online]. http://msdn.microsoft.com/en-us/library/t71a733d%28VS.80%29.aspx

[37] Microsoft. Configuring a Computer to Develop Office Solutions. [Online]. http://msdn.microsoft.com/en-us/library/bb398242.aspx

[38] Proxmox. Proxmox VE. [Online]. http://pve.proxmox.com/wiki/Main_Page

[39] GNU's Not Unix. The GNU General Public Licence. [Online]. http://www.gnu.org/licenses/gpl.html

[40] n² community blog. n². [Online]. http://blogs.talis.com/n2/archives/818

[41] Her Majesty's Government. Linked Data. [Online]. http://data.gov.uk/sparql

[42] World Wide Web Consortium (W3C). OWL 2 Web Ontology Language: New Features and Rationale. [Online]. http://www.w3.org/TR/2008/WD-owl2-new-features-20081202/#Use_Case_.2312_.E2.80.93_Prot.C3.A9g.C3.A9_report_on_the_experiences_of_OWL_users_.5BTool.5D

[43] E. Hyvönen, M. Salminen, M. Junnila, and S. Kettula. A Content Creation Process for the Semantic Web. [Online]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.2621&rep=rep1&type=pdf

[44] Google. Google Web Toolkit. [Online]. http://code.google.com/webtoolkit/

[45] T. Segaran, C. Evans, and J. Taylor, *Programming the Semantic Web*. O'Reilly Media, 2009.

[46] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, 2007.

[47] M. C. Daconta, L. J. Obrst, and K. T. Smith, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. Wiley Publishing, Inc., 2003.

[48] J. Hebeler, M. Fisher, R. Blace, and A. Perez-Lopez, *Semantic Web Programming*. Wiley Publishing, Inc., 2009.