Universidade Federal da Bahia

Universidade Estadual de Feira de Santana

DISSERTAÇÃO DE MESTRADO

Um Modelo de Apresentação e Navegação de Linked Data para o Usuário Final

André Luiz Carlomagno Rocha

Mestrado Multiinstitucional em Ciência da Computação – MMCC

Salvador - BA

2014

ANDRÉ LUIZ CARLOMAGNO ROCHA

Um Modelo de Apresentação e Navegação de Linked Data para o Usuário Final

Dissertação apresentada ao Mestrado Multiinstitucional em Ciência da Computação da Universidade Federal da Bahia e Universidade Estadual de Feira de Santana, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Cássio Vinicius Serafim Prazeres

Salvador

2014

Rocha, André Luiz Carlomagno.

Um Modelo de Apresentação e Navegação de Linked Data para o Usuário Final / André Luiz Carlomagno Rocha – Salvador, 2014.

131f. : il.

Não inclui anexos

Orientador: Prof. Dr. Cássio Vinicius Serafim Prazeres.

Dissertação (mestrado) – Universidade Federal da Bahia, Instituto de Matemática, Universidade Estadual de Feira de Santana, 2014.

1. Modelo de Apresentação e Navegação. 2. RDFa. 3. Linked Data. 4. Web de Dados. 5. Web Semântica. I. Prazeres, Cássio Vinicius Serafim. II. Universidade Federal da Bahia. Instituto de Matemática. III. Universidade Estadual de Feira de Santana. IV. Título.

CDD - 004.6
CDU - 004.55

ANDRÉ LUIZ CARLOMAGNO ROCHA

UM MODELO DE APRESENTAÇÃO E NAVEGAÇÃO DE LINKED DATA PARA O USUÁRIO FINAL

Esta dissertação foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Mestrado Multiinstitucional em Ciência da Computação da UFBA-UEFS.

Salvador, 31 de Março de 2014.

_____

Prof. Cássio Vinicius Serafim Prazeres (orientador), D.SC.
Universidade Federal da Bahia – UFBA

_____

Prof. Cesar Augusto Camillo Teixeira, D.SC.
Universidade Federal de São Carlos – UFSCar

_____

Prof. Frederico Araujo Durão, D.SC.
Universidade Federal da Bahia – UFBA

RESUMO

Linked Data permite a ligação entre dados de diferentes fontes para criar um único espaço global de dados. Dados publicados obedecendo aos princípios de Linked Data possuem significado explicitamente definido e podem ser tratados e processados por máquinas. No entanto, pessoas também podem se beneficiar diretamente da semântica explícita contida na estrutura dos dados. Este trabalho trata a carência de métodos, processos e modelos preocupados com a apresentação e navegação de dados estruturados obedecendo aos princípios de Linked Data. O foco da pesquisa é no usuário comum, aquele sem experiência com as técnicas que giram em torno da Web Semântica. Para tratar o problema em questão, o trabalho utiliza três linhas de investigação: as estratégias para apresentação e navegação de dados estruturados na Web Semântica; a modelagem de sistemas de hipertexto e a recuperação de dados estruturados embutidos em páginas Web. As contribuições desta pesquisa incluem: (i) um modelo de apresentação e navegação para Linked Data, focado no usuário comum, que pode ser implementado por sistemas interessados em prover tais recursos; (ii) um Serviço Web e uma biblioteca Javascript implementando o modelo, que podem ser utilizados pelo lado cliente de aplicações Web preocupadas em fornecer apresentação e navegação para Linked Data; e (iii) um protótipo desenvolvido para demonstrar a utilização do serviço e da biblioteca e, consequentemente, a viabilidade do modelo proposto, que fornece recursos de apresentação e navegação de dados estruturados embutidos em páginas Web.

Palavras Chaves: Linked Data, Modelo de Apresentação e Navegação, Usuário Comum, Web de Dados, Web Semântica, RDFa.

ABSTRACT

Linked Data enables connection between data from different sources to create a single global data space. Data published following the Linked Data principles have explicitly defined meaning and can be handled and processed by machines. However, people also can benefit directly from the explicit semantics contained in the data structure. This work addresses the lack of methods, processes and models concerned with the presentation and navigation of structured data obeying the Linked Data principles. The research is focus on common user, that without experience with the techniques that revolve around the Semantic Web. To deal with the problem in question, the work uses three lines of investigation: strategies for presentation and navigation of structured data on the Semantic Web; modeling of hypertext systems and retrieval of structured data embedded in Web pages. The contributions of this research include: (i) a model of presentation and navigation for Linked Data, focused on common user, which can be implemented by systems interested in providing such resources; (ii) a Web Service and a Javascript library implementing the model, which can be used by the client side of Web applications concerned to provide presentation and navigation for Linked Data; (iii) a prototype developed to demonstrate the use of the service and the library and hence the viability of the proposed model, which provides resources for presentation and navigation of structured data embedded in Web pages.

Keywords: Linked Data, Presentation and Navigation Model, Common User, Web of Data, Semantic Web, RDFa.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO DE TERMOS

Web tradicional	A Web que as pessoas comuns estão acostumadas a lidar. Aquela que se utiliza de páginas contendo formulários HTML para publicar, consumir e editar informações. A Web tradicional pode ser entendida como a Web de Documentos em um contexto de Linked Data.
Web 2.0	A Web que disponibiliza APIs proprietárias, por meio de Serviços Web de diferentes arquiteturas, para fornecer serviços variados. A Web 2.0 pode ser entendida como a Web tradicional + APIs. É possível também entender a Web 2.0 como a Web de Documentos, em um contexto de Linked Data.
Web de Documentos	A Web que não possui mecanismos para permitir que as máquinas (agentes de software) possam compreender e interpretar, de forma mais facilitada, as informações contidas nesse ambiente.
Web de Dados	A Web projetada obedecendo aos princípios de Linked Data. Também considerado um espaço global de dados contidos em fontes variadas e relacionados por meio de ligações semânticas.
Web Semântica	A Web que fornece mecanismos para permitir que as máquinas possam compreender e interpretar as informações contidas nesse ambiente.
Usuários	Qualquer entidade (indivíduo ou agente de software) capaz de fazer uso do modelo de apresentação e navegação proposto aqui, bem como das demais pacas que compõem o rol de soluções produzidas neste trabalho.

Usuário Final	Aquele usuário que interage diretamente com o protótipo desenvolvido neste trabalho.
Usuário Experiente	Aquele usuário que possui experiência com as técnicas e padrões que norteiam a Web Semântica e Web de Dados. Esse tipo de usuário é capaz de lidar mais facilmente com soluções voltadas para esses ambientes.
Usuário Comum	Aquele usuário que não possui nenhuma experiência com os métodos e técnicas que circundam a Web Semântica e Web de Dados. Esse tipo de usuário é completamente leigo em relação ao ambiente global de dados e seus aspectos.

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BD	*Banco de Dados*
gIBIS	*generalized Issue-Based Information System*
HAM	*Hypertext Abstract Machine*
IDE	*Integrated Development Environment*
IBIS	*Issue-Based Information System*
INDA	*Infraestrutura Nacional de Dados Abetos*
JAXB	*Java Architecture for XML Binding*
JAX-RS	*Java API for RESTful Services*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
OWL	*Ontology Web Language*
LDVM	*Linked Data Visualization Model*
LOGD	*Linked Open Government Data*
LOD	*Linking Open Data*
SGBD	*Sistema Gerenciador de Banco de Dados*
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework Schema*
REST	*REpresentational State Transfer*
RPC	*Remote Procedure Call*
RPI	*Rensselaer Polytechnic Institute*
SOAP	*Simple Object Access Protocol*
SPARQL	*Simple Protocol and RDF Query Language*
SQL	*Structured Query Language*
TWC	*Tetherless World Constellation*
URI	*Unique Resource Identifier*
URL	*Unique Resource Locator*
WI-FI	*Wireless Fidelity*
W3C	*World Wide Web Consortium*
WSDL	*Web Services Description Language*
XML	*eXtensible Markup Language*

Capítulo 1

Este capítulo apresenta alguns conceitos introdutórios referentes ao estudo proposto neste trabalho. O seu foco principal é descrever as motivações e problemáticas relacionadas ao tema, bem como os objetivos gerais e específicos que norteiam esta dissertação.

1 INTRODUÇÃO

A representação da informação na Web tem evoluído de um espaço global de documentos interligados, a Web de Documentos (Bizer et al., 2008), para um espaço global em que tanto documentos quanto dados podem estar conectados. Conduzindo essa evolução, está um conjunto de melhores práticas para publicar, consumir, conectar e reutilizar dados estruturados na Web conhecido como *Linked Data* (Berners-Lee, 2006). A adoção dessas melhores práticas está possibilitando o desenvolvimento de um ambiente global de dados, a Web de Dados (Kossmann, 2005). Esse ambiente permite a conexão entre fontes variadas e a disponibilização de dados sobre diversos domínios como pessoas, livros, música, ciências, entre outros (Bizer et al., 2009).

A Web de Dados começou a tomar forma depois da proposta da Web Semântica, primeiramente articulada por Berners-Lee et al. (2001). Desde então, a quantidade de dados na Web, representados por meio de técnicas e padrões semânticos, vem crescendo consideravelmente (Ziegler, 2011). Esses dados foram gerados, em sua maioria, pela extração de informações semiestruturadas de domínios como WikiPedia (Auer et al., 2007) e pela conversão automática de conhecimento originalmente residente em bases de dados relacionais (Luczak-Rösch e Heese, 2009). Além disso, uma crescente quantidade de dados tem sido embutida em documentos da Web 2.0 por meio de padrões de marcação, utilizando para isso os próprios atributos da linguagem HTML (Bizer et al., 2012). É o caso do projeto Yahoo! Searchmonkey[1] e do recente suporte do Google para marcações RDFa: *RDFa rich snippets*[2] (Tummarello et al., 2009).

[1] SearchMonkey: http://developer.yahoo.com/searchmonkey/

[2] Rich Snippets: http://googlewebmastercentral.blogspot.com/2009/05/ introducing-rich-snippets.html

Em contraste com a Web tradicional, que relaciona, por meio de *hyperlinks*, documentos completos como páginas Web, a Web Semântica relaciona dados, com base em um modelo de dados uniforme. Esse modelo consiste de declarações elementares na forma de "sujeito-predicado-objeto" expressados por meio do padrão RDF (*Resource Description Framework*) (Klyne e Carroll, 2004). Além disso, o modelo permite a utilização de vocabulários compartilhados definidos como ontologias RDFS (*RDF Schema*) (Brickley e Guha, 2004) e OWL (*Ontology Web Language*) (Bechhofer et al., 2004). As ligações entre as declarações individuais podem resultar em um único grafo, formando uma ampla nuvem de dados, que é frequentemente referenciada como a Web de Dados (Ziegler, 2011).

Na prática, *Linked Data* refere-se a dados publicados na Web de tal forma que possam ser lidos e entendidos por máquinas. O significado dos dados é explicitamente definido e os mesmos podem ser ligados a outros dados contidos em fontes externas. Entretanto, não apenas as máquinas podem tirar proveito da semântica explícita dos dados. Existem navegadores *Linked Data* que permitem a interação e navegação na Web de Dados por usuários finais (pessoas). Os usuários podem, dentre outras coisas, visitar fontes diversas e seguir as ligações para navegar entre dados relacionados. Além disso, motores de busca *Linked Data* podem explorar a Web de Dados, por meio das ligações entre as fontes. Esses motores fornecem recursos de consultas expressivos sobre dados agregados de forma semelhante a como uma base de dados tradicional é consultada (Bizer et al.,2009).

1.1 MOTIVAÇÃO

Apesar dos navegadores e buscadores *Linked Data* permitirem a interação direta com a Web de Dados, não é trivial, para um usuário sem experiência com a Web Semântica, explorar e usar esses dados de maneira satisfatória (Brunetti et al., 2012). Exibir dados estruturados de forma amigável é um problema encontrado por vários tipos de aplicações utilizando diferentes paradigmas de representação. Na maioria das vezes, tais dados são apresentados de maneira primitiva com pouco ou nenhum tratamento especializado para facilitar a navegação por parte do usuário final. Isso dificulta a utilização eficiente do conhecimento semântico disponibilizado por meio da estrutura dos dados, tornando-o desinteressante do ponto de vista da satisfação das necessidades de informações das pessoas comuns (Pietriga et al., 2007).

Segundo Berners-Lee et al. (2007), a primeira categoria de navegadores semânticos foi projetada para simplesmente apresentar conjuntos de dados RDF para leitura. Uma segunda categoria abordou mecanismos e perspectivas da exibição de *Linked Data* recuperados da Web de Dados. No entanto,

considerando as necessidades dos usuários não técnicos, mesmo diante dessa evolução, a apresentação e a navegação de/em dados estruturados continuam sendo um desafio. De acordo com Cheng et al. (2011), navegadores *Linked Data* como Tabulator (Berners-Lee et al., 2006) têm sido desenvolvidos seguindo a ideia de que a Web de *Linked Data* é uma grande base de dados, onde todo o esquema e dados estão disponíveis para uso. Entretanto, o fato é que, geralmente, os usuários estão cientes de apenas uma pequena porção desse ambiente, tal como uma entidade identificada por um URI (por exemplo, a entidade "João" do tipo "Pessoa" identificada pelo URI *http://abc.com.br/people/joao*). Ainda segundo Cheng et al. (2011), diante desse contexto, faltam ferramentas para apoiar o cidadão comum na tarefa de explorar a Web de *Linked Data* de maneira adequada.

Para Camarda et al. (2012), apesar das centenas de milhares de triplas RDF contidas na Web de Dados, é difícil encontrar ferramentas de apresentação e navegação fáceis de usar, que sejam realmente baseadas em padrões RDF e capazes de demonstrar a eficácia do modelo *Linked Data*. Davies et al. (2010) argumentam que: se a finalidade é a de que os usuários finais possam tirar proveito da Web de Dados da mesma forma que eles tiram proveito da Web de Documentos, então eles precisam de ferramentas, adequadas à sua condição de usuários não técnicos, que lhes permitam fazê-lo. Além disso, Garcia et al. (2011), comentam que o potencial da Web de Dados é enorme, no entanto, uma considerável quantidade de dados está disponível como dados brutos, acessível apenas por meio de terminais SPARQL (*SPARQL Endpoints*). Isso se traduz em uma barreira para usuários finais sem expertise com *Linked Data*. A forma de resolver essa questão é desenvolver métodos e processos para permitir a interação apropriada desse usuário com os dados disponíveis na nuvem *Linked Data*. Para Ziegler (2011), embora a Web Semântica, desde a sua idealização, com suas tecnologias fundamentais, tenha focado principalmente no processamento de informações Web por máquinas, é cada vez mais reconhecido que cidadãos comuns também podem se beneficiar mais diretamente de técnicas semânticas. E ainda, Latif et al. (2009), afirmam que: em seu estágio atual, a Web de *Linked Data* produz pouco benefício para os usuários finais, que não possuem conhecimento de ontologias, triplas e SPARQL.

Outro ponto importante, é que esforços estão sendo realizados com o propósito de desenvolver novas técnicas, bem como aprimorar técnicas preexistentes, de inserção de dados estruturados em documentos da Web tradicional, por meio de marcações embutidas nos atributos HTML das páginas. Essas pesquisas surgem como uma tendência no sentido de aproximar a Web de Dados da Web de Documentos (Mühleisen e Bizer, 2012). Segundo o Projeto Web Data Commons (Bizer et al., 2012) uma das técnicas mais utilizadas para embutir *Linked Data* em páginas Web é RDFa (Adida et al., 2012). De acordo com o relatório de agosto de 2012 do projeto, existem mais de 168 milhões de páginas Web contendo triplas RDFa, e uma média de 2 mil triplas por domínio. Nos últimos anos, tem se expressado uma tendência na convergência de algumas técnicas de inserção de dados estruturados em páginas HTML, tais como

microfomats, para RDFa, evidenciando a importância desse padrão. (Adida, 2008). Apesar dos esforços nas pesquisas e da crescente aceitação do padrão RDFa, o conhecimento semântico fornecido por essas técnicas ainda continua implícito nas páginas, disponíveis apenas para as máquinas, dificultando a utilização direta desses dados por usuários comuns.

Portanto, percebe-se que, embora humanos também se beneficiem do significado dos dados, eles devem ser munidos de meios adequados para obter tais benefícios. Dessa forma, esses usuários inexperientes podem satisfazer suas necessidades de informação de maneira mais orientada e eficaz. Eles podem explorar conceitos desconhecidos e novos relacionamentos; obter acesso personalizado e contextualizado a recursos e serviços relevantes; e, consequentemente, contribuir para o crescimento da Web de Dados, fazendo uso de conteúdo estruturado consistente (Ziegler, 2011). Isso cria uma necessidade por modelos, métodos e ferramentas que permitam aos usuários interagirem diretamente com os dados da Web de Dados e não somente por meio de uma aplicação Web que integra e fornece os dados em páginas da Web padrão.

1.2 OBJETIVOS

Este trabalho investigou a interação do usuário comum – aquele que não possui experiência com tecnologias semânticas – com a Web de Dados, no contexto da apresentação e navegação de conteúdo *Linked Data*, consistente com o padrão RDFa 1.1 (Adida et al., 2012).

O objetivo principal do trabalho é propor um modelo para tratar as questões relacionadas a esse contexto (navegação e apresentação) no âmbito da nuvem de dados do projeto LOD *(Linking Open Data)* (SweoIG TaskForces, 2013), mais precisamente da fonte de dados DBPedia (Auer et al., 2007). O modelo proposto é uma extensão do modelo Dexter (modelo de referência para sistemas hipertexto) (Halasz e Schwartz, 1990). Para demonstrar a proposta, foram produzidos três objetos de software (um Serviço Web REST, uma biblioteca Javascript/JQuery e uma extensão para o navegador Chrome) para apoiar o usuário comum na atividade de exploração e uso de dados estruturados *Linked Data* (padrão RDFa), levando em conta os problemas relacionados à navegação e apresentação na Web Semântica. Portanto, os objetivos específicos da pesquisa são:

1. Criação de um Serviço Web e uma biblioteca Javascript com a finalidade de implementar o modelo proposto (objetivo principal) e permitir sua utilização de forma livre e aberta. O propósito da

biblioteca é assumir o papel de interface de comunicação com o serviço do lado cliente das aplicações, abstraindo a camada de serviço.

2. Construção de uma ferramenta com base na biblioteca e, consequentemente, no serviço, para demonstrar a utilização e a viabilidade do modelo proposto. A ferramenta apoia o usuário inexperiente na tarefa de explorar *Linked Data* embutido em páginas HTML por meio de marcações RDFa.

1.3 METODOLOGIA

A metodologia utilizada durante a pesquisa obedeceu a seguinte sequência lógica de macroatividades: (1) revisão sistemática da literatura acerca de estudos abordando o problema da apresentação e navegação de *Linked Data*, com foco no usuário sem experiência com as técnicas da Web Semântica; (2) revisão da literatura acerca de modelos de referência de hipertexto e suas particularidades; (3) criação de um modelo de apresentação e navegação para *Linked Data* focado no usuário inexperiente, levando em conta os conhecimentos adquiridos nas revisões das macroatividades 1 e 2; (4) criação de um Serviço Web e uma biblioteca Javascript implementando o modelo proposto na macroatividade 3; e (5) construção de uma ferramenta de apoio ao usuário comum na tarefa de exploração de *Linked Data*, embutido em páginas HTML por meio de RDFa, utilizando as peças desenvolvidas na macroatividade 4 para demonstrar o modelo proposto na macroatividade 3.

Além destas macroatividades, atividades secundárias (microatividades) foram realizadas dentro destas atividades maiores: (1.1) estudos abordando interfaces de usuário frequentemente utilizadas na Web, como navegação facetada e apresentação por *tooltips*; (1.2) estudo da usabilidade dos tipos de interfaces investigadas na microatividade 1.1; (2.1) levantamento requisitos para a construção de um modelo de apresentação e navegação; (2.2) comparação entre os modelos de referência investigados na macroatividade 2, usando os requisitos aferidos na microatividade 2.1; (3.1) mapeamento prévio da Web de Dados para o modelo Dexter, com o objetivo de criar um modelo de apresentação e navegação estendendo a camada de *runtime* Dexter; (3.2) estudo da viabilidade da utilização de cada camada definida no modelo elaborado na macroatividades 3; (4.1) estudo do padrão REST e *framework* JQuery; e (5.1) estudo do desenvolvimento de extensões para navegadores Web.

1.4 CONTRIBUIÇÕES

Como resultados obtidos no trabalho foram desenvolvidos um modelo de apresentação e navegação para *Linked Data*, focado no usuário comum; um Serviço Web no padrão REST; uma biblioteca Javascript/JQuery e uma extensão para o navegador Google Chrome. O serviço fornece estruturas, baseadas no modelo proposto, para facilitar a criação de formas de apresentação e navegação amigáveis. A biblioteca Javascript/JQuery foi criada para acessar o serviço mencionado anteriormente. Ela é capaz de construir formas de apresentação e navegação dinâmicas, direcionadas para o usuário comum, por meio das estruturas fornecidas pelo serviço. Por último, foi desenvolvida uma extensão para o navegador Google Chrome com o objetivo de demonstrar a utilização da biblioteca, do serviço e, consequentemente, do modelo. A extensão é capaz de determinar quando existem marcações RDFa nas páginas Web visitadas pelo usuário por meio do navegador Chrome. O usuário pode interagir com a extensão, que fornece apresentação e navegação adequadas nos dados RDFa existentes na página, utilizando para isso os recursos providos pela biblioteca e pelo serviço.

1.5 AVALIAÇÃO

A avaliação foi realizada sobre o protótipo (extensão do Google Chrome) em combinação com o Serviço Web e a biblioteca Javascript, desenvolvidos como parte da proposta desta dissertação. A avaliação levou em conta o cálculo da complexidade dos algoritmos envolvidos nas operações e suboperações disponibilizadas pelo modelo de apresentação e navegação de *Linked Data*, objeto desta pesquisa. Além disso, testes de desempenho foram realizados com o objetivo de aferir os tempos de resposta acerca da utilização de tais operações. Embora o foco desse trabalho seja na interface com o usuário final, não houve a necessidade de testes com usuários durante a avaliação. Isso é reflexo da utilização de abordagens já amplamente aceitas e avaliadas na comunidade de pesquisa de usabilidade, como é o caso da apresentação por *tooltips* e da navegação por facetas (Fagan, 2010) (Oren et al., 2006) (Yee et al., 2003).

Capítulo 2

Neste capítulo, são abordados os principais tópicos e conceitos relacionados ao tema principal deste trabalho. Inicialmente, são apresentadas as motivações e benefícios para a publicação e disponibilização de dados abertos e, posteriormente, são levantados os princípios básicos que circundam a área de Linked Data.

2 LINKED DATA

A todo momento é preciso tomar decisões e, cada vez mais, indivíduos e organizações baseiam suas tomadas de decisão em dados. Dados sobre economia, comércio, pesquisa, governos, entre outros, podem ser encontrados e reutilizados de forma mais ou menos facilitada. Isso vai depender de certos fatores, tais como, propriedade, disponibilidade e meio de acesso (Ericson, 2010). A Web é hoje um importante meio compartilhado de acesso à informação. Na Web pode-se, por exemplo, encontrar a temperatura média prevista para um determinado dia, bem como, a taxa base do imposto sobre transações comerciais praticada na República das Honduras. O acesso compartilhado a essas informações é possível porque o centro de meteorologia local e a organização de comércio daquele país disponibilizaram esses dados de forma aberta. Isso cria condições para que qualquer um, pessoa ou organização, possa consultar e/ou reutilizar tal conhecimento.

Atualmente, indivíduos e organizações, incluindo grandes corporações e governos, vêm contribuindo ativamente para o aumento da oferta de dados abertos. Como exemplo, a varejista online *Amazon* possui uma *API* Web[3] *(Application Programming Interface)* para facilitar a disponibilização dos dados de seus produtos para terceiros. Outros exemplos são os governos do Reino Unido e dos Estados Unidos da América, que possuem programas de publicação e democratização do acesso a seus dados governamentais. No cenário nacional, a Lei 12.527[4], de 18.11.2011, permite que qualquer cidadão, sem necessidade de justificativa, solicite dados e informações a qualquer órgão ou entidade pública. Essa lei deu

[3] http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/

[4] http://www.planalto.gov.br/ccivil_03/_Ato2011-2014/2011/Lei/L12527.htm

origem a Infraestrutura Nacional de Dados Abertos – INDA[5], que tem o objetivo de atender as condições de disseminação e publicação de dados do governo brasileiro em formato estruturado.

Do outro lado, existe uma quantidade considerável de terceiros interessados em consumir essas informações e, criar novos negócios, construir novas formas de comércio online, acelerar o processo de pesquisa e agilizar o sistema democrático. No caso da *Amazon*, por exemplo, as informações de produtos disponibilizadas por meio de sua *API* geram uma crescente rede de negócios. Dessa forma, seus parceiros podem criar microempresas baseadas em transações direcionadas aos sítios da *Amazon* (Heath e Bizer, 2011). No contexto de dados governamentais, existem iniciativas a exemplo do projeto "Ligado nos Políticos". Esse projeto reúne informações coletadas de fontes abertas do governo brasileiro, como Senado Federal, Câmara dos Deputados, Ministérios, entre outros. O objetivo do projeto é disponibilizar esses dados em formato estruturado (Araújo e Souza, 2011).

O aparente sucesso obtido nas relações entre fornecedores e consumidores de dados dá origem a uma diversidade de novos negócios e projetos. Isso mostra uma demanda preexistente por acesso a dados, não reconhecida e não saciada até então. Indivíduos e organizações que escolhem disponibilizar e compartilhar seus dados vislumbram benefícios com o surgimento de tais relações. Diante disso, pode-se visualizar três questões: Qual a melhor forma de fornecer acesso aos dados de maneira que possam ser mais facilmente reutilizados? Como permitir a descoberta de dados relevantes diante da variedade de fontes disponíveis? E, como permitir que aplicativos integrem dados de fontes anteriormente desconhecidas (Heath e Bizer, 2011)? Para resolver estas questões, Berners-Lee et al. (2001) apresentaram a proposta inicial do que eles chamaram de Web Semântica.

Segundo Cunha et al. (2011), a Web Semântica é considerada uma extensão da Web tradicional, cujo objetivo é facilitar a interpretação e integração de dados. Boa parte do conteúdo existente na Web de hoje não seguem regras semânticas. A finalidade resume-se apenas a apresentação, dificultando imensamente a tarefa de extração de significado dessas informações (Costa e Yamate, 2009).

Portanto, a maior parte das informações disponibilizadas hoje na Web é organizada de forma a serem lidas e compreendidas por humanos e não por máquinas. Agentes de software precisam despender um considerável esforço computacional para entender e interpretar tais informações. Para facilitar essa tarefa é preciso adicionar semântica aos dados, tornando-os estruturados e possibilitando sua compreensão por máquinas (Cunha et al., 2011). Nesse contexto, semântica diz respeito à atribuição de significado explicito aos dados. Dessa forma, esses dados podem ser interligados com outros conjuntos de dados em outros domínios do conhecimento. Isso cria uma relação de significância entre os conteúdos de modo que sejam compreendidos tanto por humanos quanto por máquinas (Berners-Lee et al., 2011).

[5] http://www.governoeletronico.gov.br/acoes-e-projetos/Dados-Abertos/inda-infraestrutura-nacional-de-dados-abertos

Para evoluir de um espaço global, em que apenas informações não estruturadas associadas a documentos hipertexto estão disponíveis, para um espaço global, em que tanto informações não estruturadas como dados semânticos estão conectados, surgiu o conceito de *Linked Data*. O termo *Linked Data* diz respeito a um conjunto de melhores práticas para publicação, interconexão, reutilização e cosumo de dados estruturados na Web.

Na Seção 2.1, são apresentados os conceitos e princípios técnicos que envolvem *Linked Data* e seus fundamentos.

2.1 PRINCÍPIOS E FUNDAMENTOS DE LINKED DATA

A Web atual, conhecida como Web de Documentos (ou Web 2.0), está se fundindo a uma nova Web, a Web de Dados. Essa Web é constituída de dados interligados através de uma variedade de fontes, que abrangem os mais diversos domínios, dando origem a um único espaço global de dados. A Web de Dados é o primeiro passo para a consolidação da Web Semântica, vista como o próximo estágio de evolução da Web tradicional. Como parte do desenvolvimento da Web de Dados, os princípios de *Linked Data* são uma importante peça para a construção dessa nova rede mundial.

Linked Data permite a ligação entre dados de diferentes fontes para criar um único espaço global de dados. Dados publicados obedecendo a essas práticas possuem significado explicitamente definido e podem ser tratados e processados por máquinas (Magalhães et al., 2011). O conceito de *Linked Data* é baseado em quatro princípios fundamentais introduzidos pela primeira vez pelo pesquisador Tim Berners-Lee (Berners-Lee, 2006). São eles:

1. Usar URIs[6] como nomes para as coisas.
2. Usar URIs HTTP[7] para que as pessoas possam encontrar esses nomes.
3. Quando alguém procurar um URI, fornecer informação útil usando os padrões RDF[8] e SPARQL[9].
4. Incluir links para outros URIs, de modo que possam permitir a descoberta de mais coisas.

O objetivo principal da adoção das práticas de *Linked Data* é fornecer subsídios para o compartilhamento de dados estruturados em escala global, utilizando para isso a arquitetura existente da

[6] *Uniform Resource Identifier*

[7] *Hypertext Transfer Protocol*

[8] *Resource Description Framework*

[9] *SPARQL Protocol and RDF Query Language*

Web. Isso porque a Web combina características únicas como simplicidade, abertura e descentralização. Além disso, ela possui uma pilha de tecnologias consolidadas e bem definidas que refletem o seu rápido crescimento nos últimos vinte anos (Heath e Bizer, 2011).

Fazendo uma analogia entre a Web de Documentos e a Web de Dados, algumas semelhanças podem ser notadas. A Web de Documentos é baseada em um conjunto de padrões simples: URI, como mecanismo único de identificação global; HTTP, como mecanismo de acesso universal; e HTML, como formato de conteúdo amplamente utilizado. Da mesma forma, a Web de Dados possui padrões bem definidos: semelhante à Web de Documentos, a Web de Dados também utiliza URIs como mecanismo único de identificação global e HTTP como mecanismo de acesso universal. Além disso, a Web de Dados possui RDF, como modelo de dados comum e SPARQL, como uma linguagem padrão de consulta e acesso a dados.

Portanto, assim como é possível navegar na Web de Documentos por meio de navegadores HTML, é possível utilizar navegadores RDF para navegar na Web de Dados. Enquanto na Web de Documentos os elos (*links* HTML) conectam páginas (documentos), em diferentes servidores Web; na Web de Dados, os elos (*links* RDF) ligam dados, em diferentes fontes de dados (Magalhães et al., 2011). E ainda, assim como motores de busca usam indexadores Web para rastrear elos, através de servidores na Web de Documentos; na Web de Dados, buscadores utilizam consultas estruturadas SPARQL para seguir dados através das fontes de dados. A seguir, cada um desses padrões será tratado mais detalhadamente.

2.1.1 URI – Uniform Resource Identifiers

Antes de publicar dados na Web, itens de um determinado domínio de interesse, como, por exemplo, esportes ou música, devem ser corretamente identificados. Esses itens terão suas propriedades e relacionamentos mapeados em dados. Itens podem ser documentos Web, entidades do mundo real ou conceitos abstratos. Eles serão unicamente identificados por meio de URIs HTTP. Todo URI HTTP deve ser desreferenciável. Isso significa que o identificador deve ter a capacidade de ser encontrado por clientes HTTP. Por meio do protocolo HTTP, o identificador deve recuperar a descrição do recurso que é identificado por ele. Isso se aplica aos dois cenários: Web de Documentos, para identificar documentos HTML e Web de Dados, num contexto de *Linked Data*, para identificar objetos do mundo real e conceitos abstratos (Heath e Bizer, 2011). A Figura 1 mostra um esquema com a identificação de entidades do mundo real e seus relacionamentos.

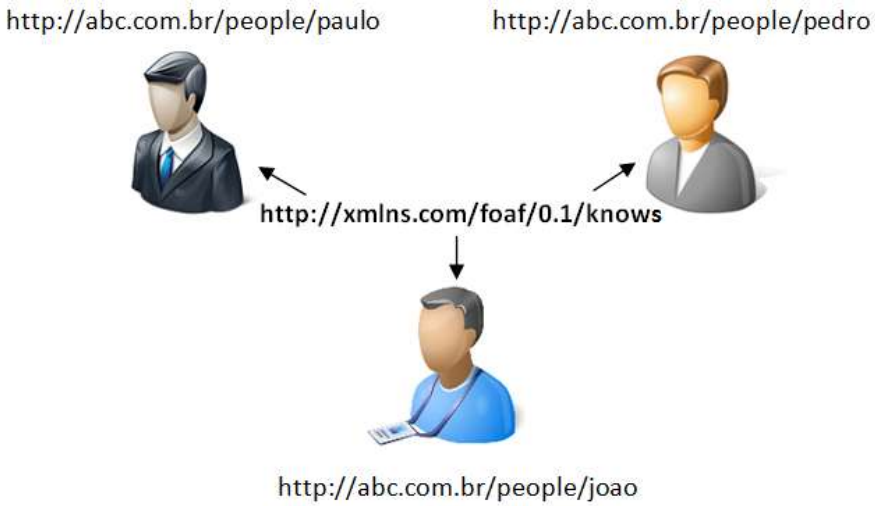

Figura 1 – URIs HTTP identificando pessoas e o relacionamento entre elas.

Na Figura 1, é possível observar que cada pessoa possui um URI distinto, e que eles estão alocados no mesmo domínio, *http://abc.com.br*. Pode-se imaginar que estas pessoas trabalham na mesma organização, a empresa *Abc*, e têm um relacionamento de conhecimento, definido pelo URI *http://xmlns.com/foaf/0.1/knows*.

É comum usar URIs distintos para identificar o mesmo recurso (item de interesse). Por exemplo, um URI pode se referir ao recurso em si (objeto do mundo real ou conceito abstrato), e outro, pode identificar a descrição do recurso, que, na maioria das vezes, encontra-se num documento HTML na Web. Quando um humano deseja recuperar informações desse recurso, a representação HTML é retornada como resposta. Por sua vez, quando um agente de software requisita o recurso, ele recebe sua representação no formato RDF. A distinção entre o URI que identifica o objeto do mundo real e o URI que identifica sua descrição na Web é crucial para o funcionamento da Web de Dados.

O mecanismo que permite a humanos e máquinas recuperarem a representação mais adequada do recurso, HTML ou RDF, é conhecido como "Negociação de Conteúdo HTTP". Essa técnica é baseada em duas estratégias recomendadas pelo W3C *Interest Group Note Cool URIs for the Semantic Web*: redirecionamento 303 e *hash URI* (Sauermann e Cyganiak, 2008). Quando se procura por uma entidade do mundo real na Web, não se pode receber como resposta a entidade em si, mas sim, uma descrição detalhada da mesma. A estratégia de redirecionamento 303 funciona enviando um novo URI como resposta para o cliente HTTP. Esse URI vai identificar a descrição da entidade.

Na Figura 1, existem três entidades do mundo real: Paulo, João e Pedro. Para recuperar a descrição de Paulo, uma requisição HTTP, por meio do método GET, é enviada com o URI *http://abc.com.br/people/paulo* para o servidor *http://abc.com.br*. No cabeçalho da requisição, o atributo *accept* é configurado com o tipo de representação que melhor se adéqua ao pedido. Se a requisição foi

enviada por um agente de software, o atributo *accept* conterá o valor *application/rdf+xml*, indicando que o cliente prefere receber o resultado em RDF/XML. Da mesma forma, se o cliente que está requisitando o recurso for um humano, o atributo *accept* conterá o valor *text/html*, informando sobre a preferência do resultado em documento HTML. A resposta ao pedido será um código *303 see other*, contendo um novo URI, que possui a descrição do recurso no formado desejado pelo cliente. Para máquinas, a resposta pode ser um novo URI como, por exemplo, *http://abc.com.br/people/paulo.rdf*, já para um cliente humano o URI pode ser *http://abc.com.br/people/paulo.html*. De posse do novo URI, o cliente efetua uma nova requisição HTTP GET e recupera a descrição do recurso no formato desejado.

Na estratégia *hash URI*, o URI do recurso possui uma parte especial chamada "fragmento identificador", que é adicionada depois de um símbolo *hash* (#). Quando o cliente HTTP requisita o recurso ao servidor, ele retira o fragmento identificador e envia na requisição apenas a parte do URI que está localizada antes do símbolo *hash*.

Considerando o URI *http://abc.com.br/people/paulo* (Figura 1), que contem informações sobre Paulo: endereço comercial, área de atuação, cargo ocupado, entre outros. Pretende-se recuperar os dados em RDF referente ao endereço comercial de Paulo, por meio do URI *http://abc.com.br/people/ paulo#endComercial*. Para isso o cliente envia o HTTP GET para o URI *http://abc.com.br/people/paulo*, sem o fragmento identificador (o fragmento permanece armazenado no *cache*). Em resposta, o servidor envia um arquivo RDF/XML contendo todos os dados a respeito de Paulo, inclusive o endereço comercial. De posse do arquivo, o cliente busca a chave *#endComercial* usando o fragmento armazenado para encontrar a tripla RDF referente ao endereço comercial de Paulo. Note que foi necessária apenas uma requisição HTTP GET para retornar o arquivo RDF com todos os dados. Portanto, o arquivo retorna os recursos *http://abc.com.br/people/paulo#endComercial*, *http://abc.com.br/people/paulo#cargoOcupado*, *http://abc. com.br/people/paulo#areaAtuacao*.

A diferença entre as estratégias é que, com *hash URI* são enviadas menos requisições HTTP GET, diminuindo a latência de rede. Por outro lado, essa estratégia sempre retorna todos os dados (triplas RDF) referentes a um recurso (*http://abc.com.br/people/paulo*), mesmo que o cliente precise de apenas uma informação. Se o recurso possuir muitos dados, consequentemente, isso retardará a recepção do arquivo, prejudicando consideravelmente o desempenho. Portanto, a estratégia *hash URI* é mais usada quando o montante de dados para um determinado recurso é relativamente pequeno. Como exemplo, *hash URIs* são usadas para identificar termos em vocabulários RDF, que possuem uma média de mil tripas (Heath e Bizer, 2011). Caso contrário, a melhor opção é usar redirecionamento 303, em que se pode definir um URI para referenciar cada dado ou recurso (*http://abc.com.br/people/paulo/cargoOcupado.rdf* ou *http://abc.com.br/ people/ paulo/endComercial.rdf*).

2.1.2 RDF – Resource Description Framework

RDF é o padrão utilizado para representar dados publicados na Web obedecendo aos princípios de *Linked Data*. Dados RDF podem ser serializados em diversos formatos. Os formatos de serialização RDF mais usados para publicação de *Linked Data* são: RDF/XML (Klyne e Carroll, 2004) e RDFa (Adida et al., 2012). O modelo de dados RDF representa a informação como nós e arestas nomeadas de um grafo direcionado. Ele é extensível e possui um alto grau de expressividade. O modelo é desenhado para representação integrada de informação que tem origem em múltiplas fontes de dados (Heath e Bizer, 2011).

No modelo RDF, os recursos são descritos como triplas. Cada tripla possui sujeito, predicado e objeto. O sujeito e o predicado de uma tripla sempre serão URIs. O objeto pode ser um literal, como um nome, número ou data – João da Silva, 2345, 12/03/2011 – ou um URI, no caso da tripla possuir uma ligação com outro recurso. O predicado expressa o relacionamento entre o sujeito e o objeto. O URI que representa o predicado é oriundo de vocabulários. Vocabulários são coleções de URIs que são usadas para representar informações sobre um determinado domínio.

Existem dois tipos principais de triplas RDF, triplas literais e *links* RDF. As triplas literais possuem um literal como objeto. Elas são usadas para descrever propriedades dos recursos, como endereço ou data de nascimento de um indivíduo. *Links* RDF conectam dois recursos. Um *link* RDF, diferente de uma tripla literal, possui um URI como obejeto. Portanto, *links* RDF são compostos de três URIs: o URI do sujeito identifica o recurso no contexto atual; o URI do objeto identifica o recurso relacionado ao sujeito; e o URI do predicado explicita o tipo de relação existente entre sujeito e objeto.

Links RDF podem ser internos ou externos. Uma forma de distinguir esses tipos de *links* é observar os URIs do sujeito e objeto. *Links* RDF internos possuem URIs do sujeito e do obejeto definidos no mesmo espaço de nomes. Isso significa que esses *links* conectam recursos em uma mesma fonte de dados. No caso de *links* RDF externos, os URIs do sujeito e do objeto não fazem parte do mesmo espaço de nomes, indicando que essa tripla RDF conecta recursos em diferentes fontes. A Figura 2 mostra exemplos de triplas RDF.

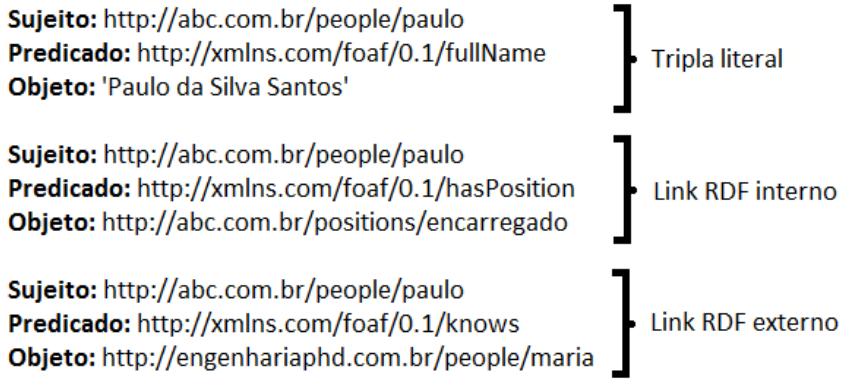

Figura 2 – Exemplo de triplas RDF.

O primeiro exemplo da Figura 2 mostra uma tripla literal relacionando o sujeito *http://abc.com.br/people/paulo* ao seu nome completo 'Paulo da Silva Santos', por meio do predicado *http://xmlns.com/foaf/0.1/fullName*. O nome completo é uma string (literal) contida no objeto da tripla. O segundo exemplo mostra um *link* RDF interno (mesmo espaço de nomes *http://abc.com.br/*) relacionando o recurso Paulo ao cargo que ele ocupa na empresa Abc, por meio do predicado *http://xmlns.com/foaf/0.1/hasPosition*. O cargo ocupado por Paulo também é definido como um recurso no objeto da tripla. Finalmente, o terceiro exemplo é um *link* RDF externo (espaço de nomes diferentes) que conecta o recurso Paulo com o recurso Maria, em outra fonte de dados, (*http://engenhariaphd.com.br/*), por meio de uma relação de conhecimento explicitada no predicado *http://xmlns.com/foaf/0.1/knows*.

Algumas considerações são necessárias para sedimentar a importância do modelo de dados RDF no contexto de *Linked Data*:

1. Pelo fato de utilizar URIs HTTP como identificadores únicos globais para referenciar recursos, o modelo de dados RDF pode ser usado em escala global para permitir que qualquer um referencie qualquer coisa.

2. Clientes HTTP podem desreferenciar URIs de um grafo RDF para recuperar informações sobre recursos. Assim, cada tripla RDF é parte de um conjunto global de dados e cada uma delas pode ser usada como ponto inicial de exploração desse espaço.

3. RDF permite a criação de *links* entre dados de diferentes fontes.

4. É possível combinar informações de diferentes fontes para unir dois conjuntos de triplas em um único grafo.

5. Com RDF é possível representar informações que são expressas em diferentes esquemas em um único grafo. Isso significa que é possível combinar termos de diferentes vocabulários para representar dados.

Combinado com linguagens de esquema e ontologia como RDF-Schema (Brickley e Guha, 2004) e OWL (Bechhofer et al., 2004), o modelo RDF permite o uso de mais, ou menos, estrutura tanto quanto desejado. Portanto, tanto dados estruturados quanto dados semiestruturados podem ser representados (Heath e Bizer, 2011).

2.1.3 SPARQL – Simple Protocol and RDF Query Language

SPARQL é uma linguagem de consulta padrão recomendada e padronizada pelo W3C DAWG – *RDF Data Access Working Group* – para recuperação e manipulação de informações contidas em grafos RDF. SPARQL pode ser usado para expressar consultas através de diversas fontes de dados. No entanto, SPARQL não é apenas uma linguagem de consulta declarativa, mas também um protocolo usado para enviar consultas e recuperar resultados através do protocolo HTTP (Clark et al., 2008).

SPARQL possui uma estrutura "*select-from-where*" semelhante ao SQL. *Select* especifica uma projeção sobre os dados como, por exemplo, a ordem e a quantidade de atributos e/ou instâncias que serão retornados como resultado. *From* declara as fontes de dados que serão consultadas. A cláusula *From* é opcional e, quando não especificada, assumi-se que a busca será realizada em um documento RDF/RDFS local. *Where* impõe restrições à consulta. Os registros retornados pela consulta deverão satisfazer as restrições impostas pela cláusula *Where* (Cunha et al., 2011).

Consultas SPARQL são compostas por três partes: (1) a parte de *pattern matching* possui recursos interessantes de correspondência de padrões em grafos. Alguns desses recursos são: união de padrões, nidificação, filtragem de valores de possíveis correspondências e a possibilidade de escolher a fonte para ser correspondida por um padrão; (2) a parte de *solution modifiers* permite modificar valores de variáveis aplicando os operadores clássicos como projeção, disjunção, ordem e limite. Isso pode ser feito depois que a saída de um padrão foi processada e computada (na forma de uma tabela de valores de variáveis); e (3) a parte de *output* de uma consulta, que pode ser de diferentes tipos: consultas *yes/no*, seleção de valores de variáveis que correspondem a padrões, construção de novos dados RDF a partir desses valores e descrição de recursos (Pérez et al., 2009).

O resultado de uma consulta SPARQL é um subgrafo oriundo da execução da consulta sobre o grafo completo que representa o modelo. A Figura 3 apresenta o exemplo de um grafo que representa a relação entre instancias de uma ontologia, cujo domínio é focado na descrição e formalização de escritores.

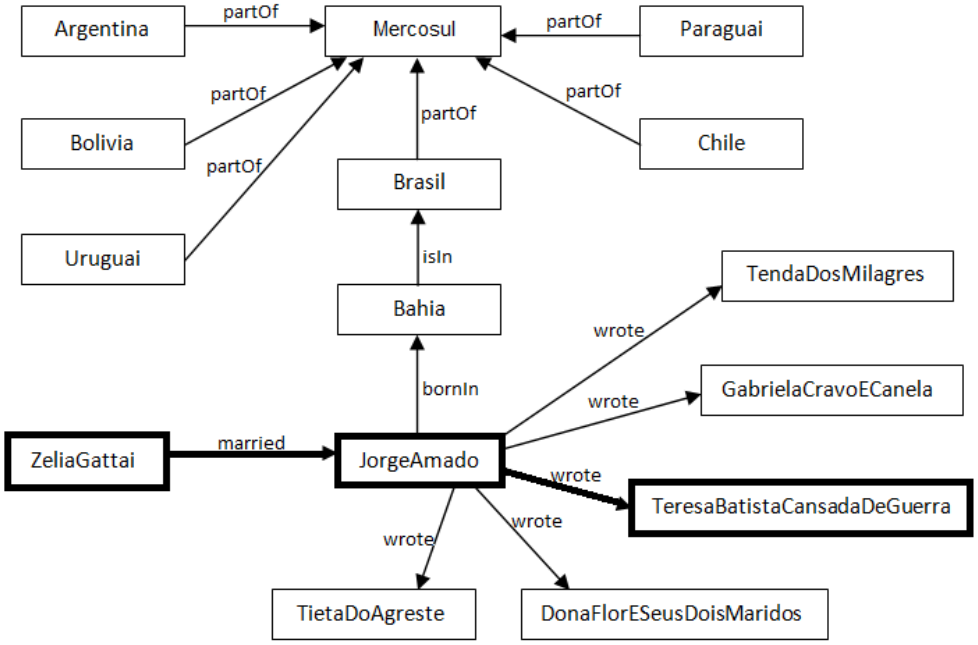

Figura 3 – Grafo representando o resultado de uma consulta SPARQL.

No subgrafo destacado, é possível notar o relacionamento entre a escritora Zélia Gattai e, o também escritor, Jorge Amado, que, por sua vez, está relacionado ao romance Teresa Batista Cansada de Guerra. Esse subgrafo pode ser visto como o resultado de uma consulta SPARQL que retorna: o escritor que concebeu o livro Teresa Batista Cansada de Guerra e é casado com a escritora Zélia Gattai (Cunha et al., 2011).

Fontes *Linked Data* geralmente fornecem um SPARQL *endpoint*, que é um Serviço Web com suporte ao protocolo SPARQL. O *endpoint* possui um URI específico para receber requisições HTTP que contenham consultas SPARQL e retornar os resultados para tais consultas. As respostas para as consultas SPARQL podem conter resultados em diferentes formatos. Consultas que usam os comandos *SELECT* e *ASK* tipicamente retornam resultados em formato XML, JSON ou texto plano. Para consultas realizadas através dos comandos *DESCRIBE* ou *CONSTRUCT*, os resultados normalmente usam os formatos RDF/XML, NTriples, Turtle ou N3 (Magalhães et al., 2011).

2.2 CONSIDERAÇÕES FINAIS

Linked Data revoluciona a forma como os dados são publicados e consumidos. Utilizando os mecanismos de acesso padronizados disponibilizados como *Linked Data*, é possível ter acesso a fontes de dados ilimitadas, em busca de um melhor aproveitamento do potencial da Web. O volume de dados disponibilizados seguindo os princípios de *Linked Data* cresce muito rapidamente, e cobre os mais variados domínios. No entanto, muitos desafios ainda precisam ser superados para que seja possível aproveitar todo o potencial que a Web de Dados oferece. Desse modo, interfaces mais interativas e de fácil uso fazem falta para aqueles usuários que não possuem expertise com esse ambiente, bem como outras questões que ainda precisam ser mais bem resolvidas. É o caso, por exemplo, do desempenho de consultas e da qualidade dos dados retornados.

Com relação ao problema da falta de interfaces mais adequadas ao usuário não técnico, a escassez de modelos – projetados com o objetivo de tornar a Web de Dados mais atraente e funcional, do ponto de vista desse tipo de usuário – é um fator preponderante. No Capítulo 3, será apresentado o modelo Dexter de referência de hipertexto. Esse modelo foi estendido para dar origem ao modelo de apresentação e navegação de *Linked Data* proposto nesta dissertação.

Capítulo 3

Este capítulo apresenta o modelo Dexter, abordando os principais aspectos e características de cada camada. A camada de armazenamento é descrita com maiores detalhes, uma vez que o modelo foi projetado para focar nas questões envolvendo o armazenamento da rede hipertexto, bem como, nas interfaces entre a camada de armazenamento e as demais camadas.

3 MODELO DEXTER DE REFERÊNCIA DE HIPERTEXTO

O modelo Dexter (Halasz e Schwartz, 1990) é uma tentativa de capturar, ambos, formal e informalmente, importantes abstrações encontradas em uma ampla variedade de sistemas hipertexto (hipermídia). O objetivo principal do modelo é fornecer princípios base para o desenvolvimento de padrões de troca e interoperabilidade entre tais sistemas. O modelo Dexter é dividido em três camadas: a camada de execução (*runtime layer*), a camada de armazenamento (*storage layer*) e a camada que trata o interior do componente (*within-component layer*), como pode ser visto na Figura 4.

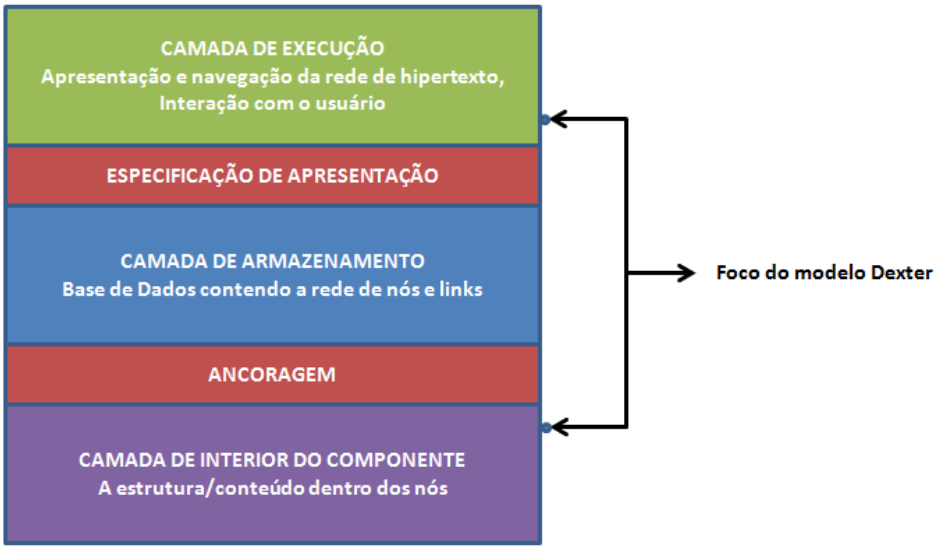

Figura 4 – Divisão em camadas do modelo Dexter.

O restante deste capítulo apresenta uma visão geral das camadas que compõem o modelo Dexter, abordando os aspectos mais importantes de cada uma delas. Como o foco do modelo é na camada de armazenamento, as descrições das camadas de execução e de interior do componente foram simplificadas neste texto. Na verdade, praticamente, não há descrição sobre a camada de interior do componente. Isso é uma característica do modelo, que espera que outros modelos de referência sejam desenhados (em combinação com o modelo Dexter) para cobrir as questões que envolvem essas camadas.

3.1 CAMADA DE ARMAZENAMENTO

A camada de armazenamento descreve a estrutura do sistema hipertexto como um conjunto finito de componentes, juntamente com duas funções: função de resolução e função de acesso. Essas funções são responsáveis por recuperar componentes, i.e., mapear uma especificação de componente em um componente propriamente dito.

A entidade fundamental e unidade básica de endereçamento da camada de armazenamento é o componente. Um componente pode ser um Átomo, um Link ou uma entidade composta (constituída de outros componentes). Átomos são componentes primitivos no modelo. A estrutura interna dos átomos é uma preocupação da camada de interior do componente. Links são entidades que representam relacionamentos entre outros componentes. Links são, basicamente, uma sequência de dois ou mais *endpoints*, cada um se referindo para um componente (ou parte de um componente) na rede hipertexto. Componentes compostos são constituídos de outros componentes. A hierarquia de componentes compostos, criada quando um componente composto contém outro componente, é restringida para ser um grafo acíclico. Ou seja, nenhum componente pode se autoconter nem direta nem indiretamente.

Todo componente possui um identificador globalmente único (UID). A função de acesso do sistema é responsável por "acessar" o componente dado seu UID, i.e., por mapear um UID para o componente endereçado por ele. UIDs fornecem um mecanismo garantido para endereçar qualquer componente na rede hipertexto. Endereçamento indireto também é suportado pela camada de armazenamento usando "especificação de componente" em conjunto com a função de resolução. Essa função é responsável por resolver uma especificação de componente em um UID (mapear uma especificação para um UID), que pode então ser usado pela função de acesso para recuperar o componente específico. Em particular, o próprio UID pode ser usado como um especificador de componente, caso em que a função de resolução se torna uma função identidade.

O modelo Dexter suporta *links span-to-span*, que são ligações entre partes de conteúdos de componentes diferentes. Por exemplo, pode ocorrer uma ligação entre a palavra “computação”, contida em um texto dentro do componente A, e a palavra “pesquisa”, contida em um texto dentro do componente B. Isso é provido por uma entidade de endereçamento indireto chamada Ancora. Uma ancora tem duas partes: um “id” e um “valor”. O valor da ancora especifica uma localização, região, item ou subestrutura dentro do componente. Esse valor é interpretado pela aplicação responsável por manipular o conteúdo/estrutura do componente. Ele é primitivo do ponto de vista da camada de armazenamento. O id da ancora identifica unicamente uma ancora dentro do escopo do seu componente. Uma ancora pode ser unicamente identificada no universo do sistema por meio do par [UID do componente, id da ancora].

O id da ancora pode ser combinado com o mecanismo de especificação do componente para fornecer uma maneira consistente para especificar *endpoints* de um link. Isso é realizado por uma entidade chamada Especificador, que consiste de uma especificação de componente, um id de ancora e dois campos adicionais: direção e especificação de apresentação. Um especificador define um ponto dentro de um componente (por meio da especificação do componente e do id da ancora) que pode ser usando como *endpoint* de um link. O campo direção informa se o *endpoint* especificado é a fonte do link, o destino, ambos (fonte e destino) ou nenhum (nem fonte nem destino). Isso é representado por valores de direção: FROM, TO, BIDIRECT e NONE. A especificação de apresentação é um valor primitivo que faz parte da interface entre a camada de armazenamento e a camada de execução.

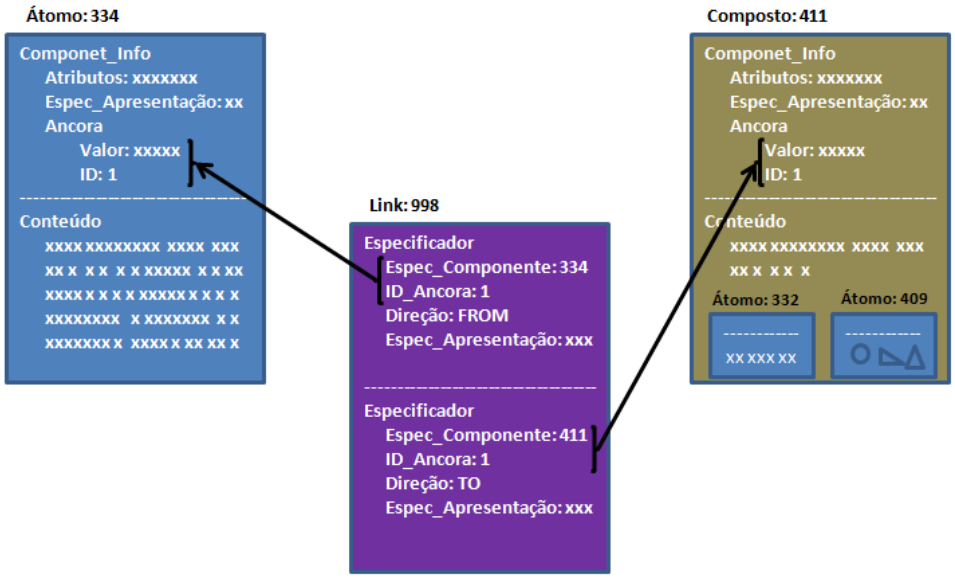

Figura 5 – Camada de armazenamento incluindo especificadores, links e ancoras.

A Figura 5 mostra a organização geral da camada de armazenamento incluindo especificadores, links e ancoras. A figura contém cinco componentes – três átomos, um composto (que é constituído de dois

átomos mais algum texto) e um link, que representa a ligação da ancora dentro do átomo 334 para a ancora no composto 441 (ligação span-to-span).

Como dito anteriormente, componentes podem ser átomos, links ou compostos de outros componentes. No modelo, isso é chamado de "componente base". Por outro lado, componentes são entidades complexas, que contêm um componente base juntamente com informações associadas ao componente (meta informação do componente). A informação do componente descreve as propriedades do componente sem se preocupar com seu conteúdo. Especificamente, a informação do componente contém uma sequência de ancoras, a especificação de apresentação (que contém informações para a camada de execução sobre como o componente deve ser apresentado para o usuário) e um conjunto de atributos (par chave/valor). Esses atributos podem ser usados para anexar qualquer propriedade arbitrária para o componente. Por exemplo, o tipo de componente pode ser adicionado a cada componente por um atributo "tipo".

Em adicional, a camada de armazenamento define um pequeno conjunto de operações que podem ser usadas para acessar e/ou modificar a rede hipertexto. Essas operações são definidas de tal maneira para que a consistência da rede seja mantida. Por exemplo, o fato de que a hierarquia de componentes compostos permaneça acíclica. As operações incluem adição de componente (átomo, link ou composto) para a rede hipertexto; remoção de componente e modificação de conteúdo ou informações auxiliares do componente (e.g. ancoras ou atributos). Existem também operações para recuperar um componente dado seu UID ou qualquer especificador que possa ser convertido para um UID. Finalmente, existe uma operação necessária para determinar a interconectividade da estrutura da rede hipertexto. Essa operação, *linkToAnchor*, retorna o conjunto de links que se refere para uma ancora, quando dado a ancora e o componente que a contém.

3.2 CAMADA DE EXECUÇÃO

O conceito fundamental da camada de execução é a "instanciação de um componente". Uma instanciação é uma apresentação do componente para o usuário. Operacionalmente, uma instância deve ser vista como um tipo de *cache* de execução para o componente. Na instanciação, uma cópia do componente é armazenada em *cache*. O usuário visualiza e/ou edita essa cópia e as alterações são escritas de volta na camada de armazenamento. Podem existir mais de uma instância, simultaneamente, para um dado componente. Cada instancia é identificada por um identificador de instanciação único (IID).

A qualquer momento, o usuário da rede hipertexto pode visualizar e/ou editar várias instâncias de componentes. A camada de execução inclui uma entidade chamada de *sessão*, que serve para manter o controle, momento a momento, do mapeamento entre componentes e suas instâncias. Quando o usuário deseja acessar a rede hipertexto, uma sessão é aberta. O usuário pode então criar instâncias de componentes (uma ação conhecida como *presenting*). Ele pode editar essas instâncias modificando o componente baseado nas edições acumuladas das mesmas (uma ação conhecida como *realizing*). Finalmente, o usuário pode destruir a instância (uma ação conhecida como *unpresenting*). A sessão é fechada quando o usuário termina sua interação com a rede hipertexto.

O núcleo da camada de execução é a função de instanciação (*instantiator function*). A entrada para essa função consiste de um UID e da especificação de apresentação do componente. A função de instanciação retorna uma instância do componente como parte da sessão. A especificação de apresentação contém informações específicas de como o componente sendo instanciado será apresentado pelo sistema durante essa instanciação. Note que no próprio componente existe uma especificação de apresentação da camada de armazenamento do modelo. Essa especificação de apresentação destina-se a conter informações sobre a noção de como o próprio componente deve ser apresentado. É responsabilidade da função de instanciação julgar (por seleção, combinação ou outro método) entre a especificação de apresentação passada como entrada para ela e a especificação de apresentação contida no componente sendo instanciado. O modelo, em sua concepção, não faz esse julgamento explicitamente.

A função de instanciação possui uma função inversa chamada de função *realizer* que recebe uma instância e retorna um novo componente refletindo o atual estado da instância (i.e. incluindo edições recentes). Esse é o mecanismo básico de reescrita por meio do *cache*, depois de uma instância ter sido alterada por edições. O componente produzido pela função *realizer* é usado como argumento na camada de armazenamento para substituir o componente correspondente da rede hipertexto, que foi modificado pelo usuário no nível da camada de execução.

3.3 CAMADA DE INTERIOR DO COMPONENTE

A camada de interior do componente é particularmente preocupada com o conteúdo e estrutura dentro dos componentes da rede hipertexto. Essa camada não é elaborada no modelo Dexter propositadamente. A grande variedade de possíveis tipos de conteúdo/estrutura que podem ser incluídos nos componentes do sistema é deixada em aberto. Texto, imagens, vídeos, execução de programas, simulações e

muitos outros tipos de dados têm sido usados como componentes em uma diversidade de sistemas hipertexto. Não seria interessante tentar criar um modelo genérico para cobrir todos esses tipos de dados. Dessa forma, o modelo Dexter considera a estrutura dentro do componente fora do modelo hipertexto. Assume-se que outros modelos de referência, projetados especificamente para modelar a estrutura de aplicações particulares, documentos ou tipos de dados, serão utilizados em conjunto com o modelo Dexter para capturar a totalidade da rede hipertexto, incluindo o conteúdo e a estrutura dentro dos componentes do sistema.

3.4 CONSIDERAÇÕES FINAIS

Por meio do modelo Dexter é possível projetar sistemas hipertexto variados. O modelo é capaz de cobrir abstrações importantes encontradas em uma grande variedade desses sistemas. A distribuição em três camadas do modelo consegue descrever componentes, o armazenamento de tais componentes por meio de uma rede de nós e *links* e a apresentação e navegação desses componentes para o usuário da rede. A camada de apresentação e navegação do modelo Dexter (*runtime layer*) foi parcialmente elaborada de forma proposital. O objetivo é que tal camada seja definida e/ou estendida por outros modelos com a finalidade de tratar, especialmente, a questão da apresentação e navegação da rede hipertexto definida pelo modelo Dexter. A mesma coisa acontece com a camada de interior do componente (*within-component layer*). Pelo fato de poder lidar com uma grande variedade de tipos de conteúdos, essa camada é definida de forma parcial propositadamente. Uma tentativa de modelagem genérica cobrindo todos esses tipos não é interessante do ponto de vista do modelo. Portanto, o modelo Dexter permite que essa camada também seja estendida ou elaborada por outros modelos, preocupados com o problema da modelagem de conteúdo dos componentes da rede.

O Capítulo 5, que descreve o objeto principal desta pesquisa – a proposta de um modelo de apresentação e navegação para *Linked Data* – apresenta mais detalhes do modelo Dexter, explicando como as estruturas definidas por esse modelo foram utilizadas na concepção do modelo proposto neste trabalho. Embora o modelo desenvolvido aqui trate, principalmente, das questões envolvendo a apresentação e navegação dos nós e *links* da rede hipertexto (recursos e relacionamentos da Web de Dados), muitas questões abrangendo a camada de armazenamento do modelo Dexter também são abordadas no Capítulo 5.

Capítulo 4

Este capítulo apresenta a metodologia adotada para o desenvolvimento deste trabalho, abordando as técnicas utilizadas em função das etapas executadas. O objetivo principal é descrever a concepção das técnicas empregadas e correlacioná-las com as ações tomadas durante o desenvolvimento do trabalho, possibilitando que outras pessoas possam seguir o mesmo caminho para atingir resultados semelhantes ou estender o que foi realizado na presente pesquisa.

4 METODOLOGIA

A organização das atividades realizadas durante o trabalho obedeceu a seguinte sequência lógica: (1) revisão sistemática da literatura acerca de estudos abordando o problema da apresentação e navegação de *Linked Data*, com foco no usuário sem experiência com as técnicas da Web Semântica; (2) revisão da literatura acerca de modelos de referência de hipertexto e suas particularidades; (3) criação de um modelo de apresentação e navegação de *Linked Data* focado no usuário inexperiente, levando em conta os conhecimentos adquiridos nos itens 1 e 2; (4) criação de um serviço Web REST e uma biblioteca Javascript implementando o modelo proposto no item 3; e (5) construção de uma ferramenta de apoio ao usuário comum na tarefa de exploração de *Linked Data*, embutido em páginas HTML por meio de RDFa, utilizando as peças desenvolvidas no item 4 para demonstrar o modelo proposto no item 3.

Além das atividades principais descritas acima (macroatividades), atividades secundárias (microatividades) foram realizadas dentro dessas atividades maiores: (1.1) estudos abordando interfaces de usuário frequentemente utilizadas na Web, como navegação facetada e apresentação por *tooltips*; (1.2) estudo da usabilidade dos tipos de interfaces investigadas no subitem 1.1; (2.1) levantamento dos requisitos para a construção de um modelo de apresentação e navegação; (2.2) comparação entre os modelos de referência investigados no item 2 das macroatividades, usando os requisitos aferidos no subitem 2.1; (3.1) mapeamento prévio da Web de Dados para a rede Dexter, com o objetivo de criar um modelo de apresentação e navegação estendendo a camada de *runtime* Dexter; (3.2) estudo da viabilidade da utilização

de cada camada definida no modelo elaborado no item 3 das macroatividades; (4.1) estudo do padrão REST e framework JQuery; e (5.1) estudo do desenvolvimento de extensões para navegadores Web.

Além das atividades diretamente ligadas ao desenvolvimento da proposta (macro e microatividades), atividades acadêmicas de escrita de artigos foram realizadas durante o período da pesquisa. Cinco artigos acadêmicos foram escritos durante o curso do trabalho. Dois artigos foram resultado de trabalhos oriundos das disciplinas cursadas no período inicial do Mestrado (um aceito no SBCUP2013 e outro recusado no SBSI2012). Outros dois artigos foram resultado direto da pesquisa (ambos aceitos no WebMedia2013). O último artigo foi escrito e submetido ao *Journal of Information Science and Engineering* – JISE e aguarda resultado. O trabalho completo, correspondendo a dissertação do Mestrado, será submetido ao *Journal of Web Semantics* – Elsevier.

A seguir uma descrição mais detalhada das macro e microatividades realizadas no decorrer da pesquisa, levando em conta os trabalhos que foram analisados e avaliados durante todo o processo.

4.1 REVISÃO SISTEMÁTICA (APRESENTAÇÃO E NAVEGAÇÃO DE LINKED DATA)

A primeira fase da pesquisa (macroatividade 1) foi dedicada ao aprendizado da tarefa de revisão sistemática, que é um meio para identificar, avaliar e interpretar toda a pesquisa relevante disponível para uma questão de pesquisa específica, área temática, ou fenômeno de interesse (Kitchenham, 2004).

As razões principais para a realização de uma revisão sistemática no contexto desta pesquisa são: (i) resumir as evidências das limitações que o modelo *Linked Data* apresenta com relação à apresentação e navegação de conteúdo semântico, do ponto de vista dos usuários não técnicos; (ii) identificar eventuais lacunas na pesquisa atual dentro dessa esfera, a fim de sugerir áreas para uma investigação mais aprofundada; e (iii) reunir subsídios para a construção de um modelo para tratar as questões que envolvem apresentação e navegação de conteúdo estruturado para usuários comuns na Web de Dados (nuvem de dados do projeto LOD, fonte de dados DBPedia).

Entre os trabalhos analisados na revisão, os que foram considerados mais promissores do ponto de vista das necessidades da pesquisa foram: (Brunetti et al., 2012), (Dadzie e Rowe, 2011), (Latif et al., 2009) e (Ziegler, 2011) na área de interfaces semânticas de usuário; (Camarda et al., 2012), (Cheng et al., 2011), (Fionda et al., 2012), (Hastrup et al., 2008), (Huynh eu al., 2007) e (Nowack, 2009) na área de apresentação e navegação; (Ding et al., 2011), (Garcia et al., 2011) e (Graves, 2010) na área de publicação; e

(Davies et al., 2010), (Luczak-Rösch e Heese, 2009), (Mendes et al., 2011) e (Shakya et al., 2009) na área de autoria.

Dentro dessa atividade (macroatividade 1) foram realizadas as seguintes microatividades:

1.1. *Interfaces de usuário (facetas e tooltips):* Dentro da atividade de revisão sistemática, estudos direcionados para interfaces de usuário frequentemente usadas na Web foram realizados. O objetivo foi identificar tipos de interfaces que poderiam se adaptar com facilidade às necessidades da apresentação e navegação de dados estruturados no contexto da Web Semântica. Os trabalhos mais proeminentes nesse contexto foram: (Clarkson et al., 2009), (Hearst, 2008), (Hudson et al., 2004), (Prazeres et al., 2006) e (Tarasov et al., 2010).

1.2. *Usabilidade de facetas e tooltips:* Depois de investigar e identificar as interfaces que melhor se adaptam a apresentação e navegação dentro do ambiente da Web de Dados, a usabilidade desses tipos de interfaces foi avaliada segundo: (Fagan, 2010), (Oren et al., 2006) e (Yee et al., 2003).

4.2 REVISÃO DA LITERATURA (MODELOS DE REFERÊNCIA DE HIPERTEXTO)

A segunda fase da pesquisa (macroatividade 2) consistiu na revisão da literatura acerca dos modelos de referência em hipertexto mais conhecidos e amplamente aceitos nesse domínio. O objetivo dessa etapa foi identificar um modelo (ou mais de um) que pudesse ser utilizado no contexto da Web de Dados para dar suporte a apresentação e navegação de dados estruturados, mantendo a consistência com os princípios de *Linked Data*.

O modelo escolhido foi usado como referência na construção do modelo de apresentação e navegação, objeto desta pesquisa. A possibilidade de extensão de um modelo preexistente foi fortemente considerada na avaliação dos modelos de hipertexto analisados aqui. Por conta disso, os trabalhos mais importantes desta fase foram: (Campbell e Goodman, 1988), (Conklin e Begeman, 1989), (Furuta e Stotts, 1989), (Halasz e Schwartz, 1990) e (Leiva, 2003).

Dentro dessa atividade (macroatividade 2) foram realizadas as seguintes microatividades:

2.1. *Requisitos para o modelo de apresentação e navegação:* A fim de avaliar mais precisamente e facilmente os modelos de hipertexto, foram definidos alguns requisitos para a construção do modelo de apresentação e navegação. O objetivo aqui foi identificar os pontos mais importantes para facilitar e acelerar a criação do modelo de apresentação e navegação. Pontos como “camada de armazenamento bem

definida", "componentes da rede hipertexto bem definidos" e "flexibilidade para mudanças e adaptações", foram determinados durante essa etapa.

2.2. *Comparação entre os modelos:* Logo após a definição dos principais requisitos para a construção do modelo de apresentação e navegação, os modelos avaliados no item 2 das macroatividades foram confrontados levando-se em conta tais necessidades. O objetivo dessa atividade foi determinar o modelo (ou os modelos) que melhor atendesse às necessidades definidas no subitem 2.1 das microatividades. Feito isso, o modelo que mais se aproximou das características desejadas foi escolhido para ser o modelo de referência na construção do modelo de apresentação e navegação proposto aqui. O modelo Dexter foi escolhido por receber a maior pontuação na comparação com outros modelos de referência.

4.3 CRIAÇÃO DO MODELO DE APRESENTAÇÃO E NAVEGAÇÃO

Na terceira fase do trabalho (macroatividade 3) o modelo escolhido no subitem 2.2 das microatividades (modelo Dexter) foi analisado minuciosamente para determinar a melhor forma de usá-lo para contribuir na construção do modelo de apresentação e navegação para *Linked Data*. Existia a possibilidade de criar um modelo derivado do modelo Dexter, ou seja, com base no modelo Dexter, criar um modelo completo de apresentação e navegação. Por outro lado, como a Web de Dados obedece aos princípios do modelo RDF, a melhor opção foi criar um modelo específico de apresentação e navegação, estendendo o modelo Dexter.

Portanto, foi definido nessa fase que o modelo de apresentação e navegação seria construído como uma extensão do modelo Dexter, mais precisamente, uma extensão da camada de execução do modelo Dexter. Essa tarefa foi facilitada pela elaboração parcial de uma camada de apresentação e navegação (camada de *runtime*), voltada para o usuário final, preexistente no modelo Dexter.

O modelo de apresentação e navegação foi construído seguindo-se uma estrutura sequencial em camadas, onde cada camada representa uma fase específica do processo. O processo se inicia com a aquisição de dados estruturados *Linked Data* e finaliza com a apresentação adequada desses dados para o usuário final, com suporte a navegação.

Dentro dessa atividade (macroatividade 3) foram realizadas as seguintes microatividades:

3.1. *Mapeamento da Web de Dados para a rede Dexter:* Antes de criar o modelo de apresentação e navegação, foi necessário mapear a Web de Dados para uma rede Dexter, seguindo a referência do modelo Dexter. Durante o mapeamento algumas adaptações foram necessárias. Por exemplo: o

componente composto, definido no modelo Dexter, não foi usado na tradução entre os modelo RDF e Dexter. O objetivo do mapeamento é criar uma rede Dexter, a partir do modelo RDF, consistente com os princípios de *Linked Data*.

3.2. *Estudo da utilização das camadas definidas no modelo:* Para deixar o modelo de apresentação e navegação "limpo" e bem definido, cada camada criada na sequência de atividades foi avaliada considerando sua viabilidade dentro do processo geral. Existem camadas que executam acessos externos a Web de Dados e Web de Documentos (Web 2.0). Esses acessos podem gerar atrasos consideráveis e, consequentemente, tornar o processo inviável do ponto de vista do tempo de resposta. Portanto, a utilização de cada camada foi avaliada para determinar sua eficácia dentro da integralidade do método.

4.4 CRIAÇÃO DO SERVIÇO WEB E DA BIBLIOTECA JAVASCRIPT

Na quarta fase do trabalho (macroatividade 4) foram construídos um Serviço Web, seguindo os padrões REST, e uma biblioteca Javascript com base no framework JQuery. O propósito dessa fase foi implementar o modelo de apresentação e navegação definido no item 3 (macroatividade 3). Foi decidido que o serviço implementaria as quatro primeiras camadas do modelo. A quinta e última camada (camada de navegação e apresentação) foi implementada pela biblioteca Javascript, para dar suporte ao lado cliente das aplicações.

O serviço foi projetado para permitir o acesso genérico de aplicações. No entanto, a biblioteca foi desenvolvida para oferecer uma interface simples e amigável de acesso ao serviço, provendo suporte eficiente a extração de *Linked Data* embutido em páginas HTML por meio do padrão RDFa.

Dentro dessa atividade (macroatividade 4) foram realizadas as seguintes microatividades:

4.1. *Estudo do padrão REST e do framework JQuery:* Com o objetivo de implementar o Serviço Web e a biblioteca Javascript, estudos do padrão REST e do framework JQuery foram realizados nesta etapa da pesquisa. Vários trabalhos foram analisados nesta fase. Os trabalhos que foram mais usados no desenvolvimento do serviço e da biblioteca foram: (Fielding, 2000), (Pautasso et al., 2008), (Peng et al., 2009), (Richardson e Ruby, 2007) e (Silva, 2010).

4.5 CONSTRUÇÃO DO PROTÓTIPO COMO UMA EXTENSÃO PARA NAVEGADOR

Na última fase do trabalho (macroatividade 5) foi desenvolvida uma ferramenta usando a biblioteca Javascript e o Serviço Web REST construídos na macroatividade 4. A ferramenta foi concebida como uma extensão para o navegador Web Google Chrome. Foi decidido que a extensão deveria ser capaz de determinar a existência de marcações RDFa em páginas Web visitadas pelo usuário.

A extensão foi desenvolvida para, a partir da interação do usuário, fornecer acesso aos dados RDFa, implícitos nas páginas Web, por meio das apresentações com suporte a navegação oferecidas pela biblioteca em combinação com o serviço.

O objetivo da ferramenta é demonstrar a utilização da implementação do modelo por meio do Serviço Web e da biblioteca Javascript e, consequentemente, a viabilidade do modelo de apresentação e navegação definido na macroatividade 3.

Dentro desta atividade (macroatividade 5) foram realizadas as seguintes microatividades:

5.1. *Estudo do desenvolvimento de extensões para navegadores Web:* Para construir o protótipo em forma de extensão para navegadores, foi necessário realizar uma série de estudos em desenvolvimento desse ambiente. Foi decidido que a extensão seria criada especificamente para o navegador Google Chrome. Isso porque esse navegador disponibiliza uma ampla documentação acerca do desenvolvimento de extensões. Os trabalhos mais usados na tarefa de desenvolvimento dessa etapa foram: (Barth et al., 2010), (Carlini et al., 2012) e (Liu et al., 2012).

4.6 CONSIDERAÇÕES FINAIS

A metodologia para a realização da pesquisa seguiu uma sequência de atividades predeterminada. No decorrer do processo surgiu a necessidade da inclusão de novas fases, bem como, da retirada ou realocação de fases pré-existentes. Por exemplo, ao término da fase 1 (revisão sistemática acerca da apresentação e navegação de *Linked Data*), concluiu-se que seria necessário um estudo aprofundado a respeito de modelos de referência de hipertexto (fase 2), para levantar os modelos que poderiam ser utilizados como referência na elaboração de um modelo de apresentação e navegação para *Linked Data*.

As etapas auxiliares (microatividades) foram importantes para sedimentar o estudo e o trabalho das principais etapas (macroatividades). Na maioria das vezes, as etapas auxiliares foram definidas durante o processo realizado em uma etapa principal. Por exemplo, durante a fase de criação do Serviço Web e da biblioteca Javascript, foi necessário realizar um estudo paralelo minucioso acerca do padrão REST e do framework JQuery.

Capítulo 5

Este capítulo descreve o modelo de apresentação e navegação para Linked Data, focado no usuário comum, desenvolvido neste trabalho. O capítulo está organizado da seguinte forma: Seção 5.1, Camada de Filtragem Analítica; Seção 5.2, Camada de Mapeamento; Seção 5.3, Camada de Descoberta e Aconselhamento; Seção 5.4, Camada de Preparação de Interface; e Seção 5.5, Camada de Apresentação e Navegação. Além das cinco primeiras seções descrevendo cada camada da modelagem, a Seção 5.6 apresenta as operações fornecidas pelo modelo de apresentação e navegação, e a Seção 5.7, as considerações finais.

5 MODELO DE APRESENTAÇÃO E NAVEGAÇÃO PARA LINKED DATA

O modelo proposto neste trabalho é definido como uma extensão do modelo Dexter (Capítulo 3), mais precisamente como uma extensão da camada de execução desse modelo de referência de hipertexto. Conforme Seção 3.2, a camada de execução do modelo Dexter trata da apresentação e navegação dos componentes da rede hipertexto. Embora a formalização do modelo Dexter apresente uma descrição parcial da camada de execução, essa descrição é bastante superficial. O objetivo é deixar o desenvolvimento dessa camada a cargo de outros modelos, preocupados em tratar as questões envolvendo a apresentação e navegação do ponto de vista do usuário final.

A Figura 6 mostra um corte do modelo Dexter, contendo a camada de execução na região superior da figura (o modelo completo pode ser visto na Figura 4 do Capítulo 3). Na região inferior, a Figura 6 exibe parte da camada de armazenamento Dexter e a interface entre essa camada e a camada de execução. No interior da camada de execução está o modelo de apresentação e navegação para *Linked Data* proposto nesta dissertação, contendo cinco camadas que serão detalhadas nas Seções de 5.1 a 5.5 deste capítulo. A construção do modelo de apresentação e navegação no interior da camada de execução Dexter identifica o foco do modelo, embora algum trabalho tenha sido realizado na camada de armazenamento (Seção 5.2). Ou seja, o modelo descrito aqui estende a camada de execução do modelo Dexter para fornecer diretrizes com o

objetivo de apoiar a construção de sistemas focados na apresentação e navegação de *Linked Data* para o usuário final.

Figura 6 – Modelo proposto estendendo a camada de execução Dexter.

A escolha do modelo Dexter, como base para a elaboração da proposta, foi fundamentada principalmente no fato de que esse modelo cobre uma ampla variedade de sistemas hipertexto, como, por exemplo, a Web de Documentos. Outra questão a ser destacada, é que o modelo Dexter pode ser considerado um modelo flexível do ponto de vista da extensibilidade (Halasz e Schwartz, 1990). Em outras palavras, o modelo impõe poucos obstáculos para mudanças, aperfeiçoamentos e derivações em sua estrutura. Além disso, diferente de outros modelos de referência, (Engelbart, 1962) (Nelson, 1967), o modelo Dexter possui suporte nativo a uma camada de apresentação e navegação, focada no usuário final. Levando em conta que: (1) a proposta deste trabalho é fundamentada na Web de Dados; (2) esse ambiente pode ser considerado um sistema hipertexto; (3) adequações precisam ser realizadas no modelo de referência para manter a consistência com o modelo *Linked Data*; e (4) a reutilização e extensão de camadas está consistente com a estratégia de boas práticas de modelagem (Gomaa, 2000), o modelo Dexter se mostrou uma excelente opção.

Assim como o modelo Dexter, outros modelos de referência foram considerados durante a pesquisa e comparados entre si. Entre eles, os mais proeminentes foram:

1. **HAM – Uma Máquina Abstrata de Hipertexto de Propósito Geral:** O modelo HAM (Campbell e Goodman, 1988) (Leiva, 2003) é baseado na separação entre interação (*front-end*) e armazenamento (*back-end*). O modelo distingue três camadas em um sistema hipertexto: banco de dados, HAM (ou máquina abstrata de hipertexto) e apresentação. Na camada de banco de dados existe um servidor que armazena e compartilha as informações. A camada HAM contém a estrutura organizacional do hipertexto, usando cinco componentes organizados hierarquicamente: grafo, contexto, nó, ligação e atributo. A camada de apresentação especifica como as informações são exibidas, sem detalhar a interação com o usuário. Embora as ligações sejam armazenadas no nível HAM, a posição de uma âncora em um nó depende da estrutura de armazenamento do nó, portanto, a ancoragem da ligação não pode ser tratada nesse nível.

2. **glBIS - Hipertexto Baseado em um Modelo Retórico:** gIBIS (*generalized Issue-Based Information System*) (Conklin e Begeman, 1989) é uma implementação baseada no IBIS (Conklin e Begeman, 1989). IBIS é um modelo retórico que permite relacionar questões, argumentos e posições, por meio de um conjunto predefinido de componentes e ligações semânticas. Quando questões, argumentos e posições são representados por componentes do hipertexto e relacionamentos semânticos são representados por *links*, tem-se um hipertexto generalizado adequado a uma ampla variedade de projetos e processos. gIBIS fornece uma estrutura semiformal que pode facilitar a compreensão de problemas, melhorar o foco de reuniões de projeto e proporcionar um rastreamento histórico de deliberações.

3. **Trellis - Um Modelo Incorporando Navegação Semântica:** Em termos de navegação, a semântica do modelo Trellis (Furuta e Stotts, 1989) permite expressar ambos: os estados possíveis em uma sessão de leitura da rede hipertexto; e as condições necessárias para a transferência de um estado para outro. Essa capacidade torna o modelo Trellis extremamente atraente em ambientes educacionais. Isso garante que os leitores tenham acesso aos componentes críticos do hipertexto na sequência apropriada, facilitando a adaptação ao comportamento do usuário e as preferências de exibição durante a navegação. Com Trellis, é possível especificar as transições que ocorrerão depois de um estado de inatividade do sistema para um estado específico de valores pré-estabelecidos de tempo. Por exemplo, sob o pressuposto de que o tempo ocioso representa confusão, os projetistas podem especificar que um botão de ajuda aparece após 15 segundos e uma janela com instruções aparece depois de um adicional de mais 15 segundos.

A Tabela 1 mostra uma comparação entre esses modelos, baseada nos requisitos aferidos durante a fase de estudos (Capítulo 4). Ou seja, os itens usados para confrontar tais modelos são aqueles levantados durante os estudos e considerados de grande importância para a concepção do modelo de apresentação e navegação proposto neste trabalho. Para cada item, uma pontuação é atribuída (de 0 a 5), segundo a aderência do modelo ao item avaliado. É importante notar que o modelo Dexter obtém nota

máxima em todos os itens, com exceção do item “Camada de Apresentação e Navegação bem definida”, em que recebeu nota 3. Isso porque, assim como nos demais modelos, propositadamente essa camada não é elaborada completamente, esperando que outros modelos específicos realizem essa tarefa.

Tabela 1 – Comparação entre modelos de hipertexto

Modelos / Requisitos	HAM	gIBIS	Trellis	Dexter
Camada de Armazenamento bem definida	5	4	2	5
Camada de Apresentação e Navegação bem definida	3	2	2	3
Componentes da rede hipertexto bem definidos	5	2	2	5
Flexibilidade para mudanças e adaptações	2	3	3	5
Documentação e formalização bem definidas	3	3	2	5

Legenda:

0 - Não Possui
1 - Muito Fraco
2 - Fraco
3 - Médio
4 - Forte
5 - Muito Forte

É interessante observar que o modelo HAM foi o que mais se aproximou do modelo Dexter durante a avaliação. As três primeiras linhas da Tabela 1 mostram uma igualdade entre os modelos HAM e Dexter. As deficiências nos requisitos de Flexibilidade para mudanças e adaptações e Documentação e formalização bem definidas (duas últimas linhas da Tabela 1), prejudicaram a avaliação do modelo HAM, enquanto que, para o modelo Dexter, esses requisitos foram avaliados como muito fortes, ou seja, obteve pontuação máxima.

Tal como ilustrado na Figura 7, o modelo proposto neste trabalho possui cinco camadas: Camada de Filtragem Analítica (Figura 7, Camada 1), Camada de Mapeamento (Figura 7, Camada 2), Camada de Descoberta e Aconselhamento (Figura 7, Camada 3), Camada de Preparação de Interface (Figura 7, Camada 4) e Camada de Apresentação e Navegação (Figura 7, Camada 5). Essas camadas foram projetadas levando-se em conta um fluxo sequencial de atividades, de forma que a saída (resultado/produto) de uma determinada camada seja aproveitada como entrada da camada seguinte na sequência. Assim, a camada de filtragem analítica recebe um grafo RDF (ou um conjunto de triplas RDF) como entrada, e gera dados refinados como saída. Por sua vez, a camada de mapeamento recebe esses dados refinados como entrada, e gera o modelo Dexter correspondente como saída, e assim por diante. Cada camada será descrita detalhadamente nas próximas seções.

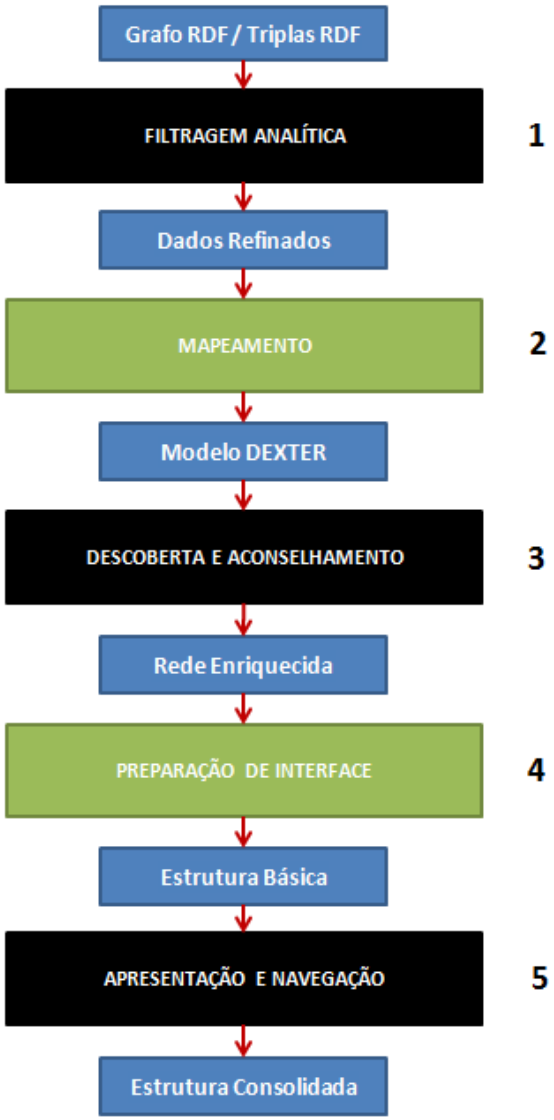

Figura 7 – Fluxo sequencial do modelo proposto mostrando entradas e saídas.

O modelo sequencial de atividades permite que componentes sejam adicionados, removidos, ou realocados de maneira simples, reduzindo o esforço de remodelagem. Por exemplo, conforme pode ser visto na Figura 8, uma camada adicional de análise de perfil do usuário pode ser inserida na sequência entre a camada de mapeamento e a camada de descoberta e aconselhamento, para determinar informações úteis de perfil. Essa tarefa seria realizada depois do mapeamento entre modelos e antes da descoberta e aconselhamento de novos dados. A entrada da nova camada seria o modelo Dexter (saída da camada de mapeamento) (Figura 8, linha 2) e a saída, uma rede básica contendo indicações de perfil, que seria a entrada da camada de descoberta e aconselhamento (Figura 8, linha 4). A camada de descoberta e aconselhamento poderia usar as indicações de perfil para descobrir dados relevantes e aconselhar navegação voltada para os interesses do usuário. Da mesma forma, a camada de filtragem analítica pode ser removida para simplificar o

modelo (em detrimento da precisão e desempenho), deixando a camada de mapeamento receber o grafo RDF diretamente como entrada, para gerar o modelo Dexter na saída. A seguir, a definição de cada uma das camadas exibidas na Figura 7.

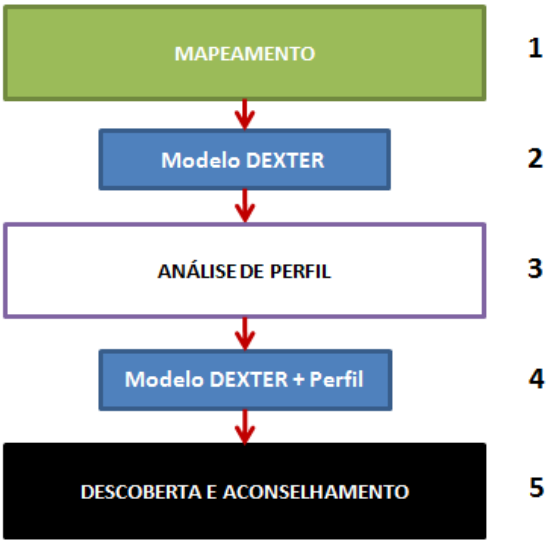

Figura 8 – Adição ilustrativa de uma nova camada no modelo.

5.1 CAMADA DE FILTRAGEM ANALÍTICA

A camada de filtragem analítica (Figura 7, Camada 1) recebe como entrada uma descrição RDF, que pode ser representada por um grafo ou um conjunto de triplas. As descrições RDF de recursos *Linked Data* podem conter centenas e, algumas vezes, até centenas de milhares de triplas. Dentro desse vasto conjunto de dados é comum encontrar, por exemplo, subconjuntos de metadados. Esses metadados são inseridos na descrição RDF, por meio de triplas (metatriplas), pela fonte de dados que disponibiliza os dados estruturados sobre o recurso. Na maioria das vezes, as metatriplas servem apenas para controle interno da fonte de dados, e não possuem nenhuma informação relevante do ponto de vista do usuário final. A Figura 9 exibe um exemplo dessa prática sendo utilizada pela fonte de dados DBPedia[10]. Na secção exibida na figura, extraída da descrição de um determinado recurso, existem 35 triplas. Dentre essas triplas, 16 são metatriplas, ou seja, aproximadamente 45,7% dos dados desta secção são metadados.

[10] http://dbpedia.org/About

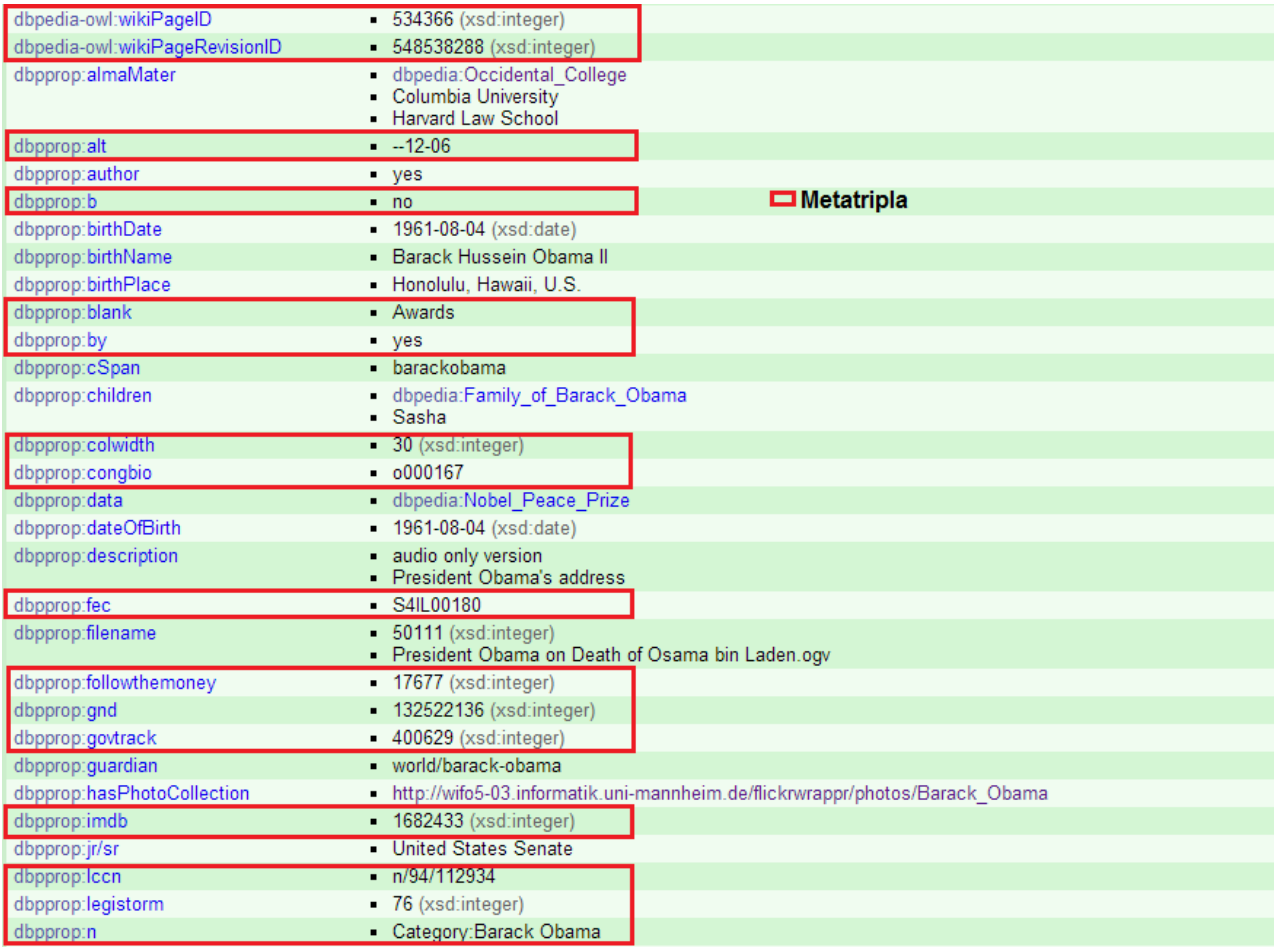

Figura 9 – Descrição DBPedia contendo metatriplas.

Além da considerável quantidade de metainformação inserida diretamente na descrição RDF dos recursos *Linked Data*, muitas fontes de dados ainda inserem informações redundantes na tentativa de fornecer recursos mais próximos dos interesses do usuário. Por exemplo, o suporte a multi-idioma faz com que as descrições RDF apresentem a mesma informação, repetidas vezes, em diferentes idiomas (várias triplas contendo o mesmo dado em idiomas diversos). A Figura 10 mostra uma parte das triplas definindo comentários, em vários idiomas, sobre um determinado recurso *Linked Data* da fonte DBPedia (DBPedia fornece suporte a 119 idiomas). O recurso de multi-idioma é interessante e válido porque permite atender melhor as necessidades dos usuários, apresentando informações em um idioma familiar. Por outro lado, para uma aplicação que pretende processar computacionalmente as triplas de uma descrição RDF, a redundância pode tornar o processo mais lento e ineficiente.

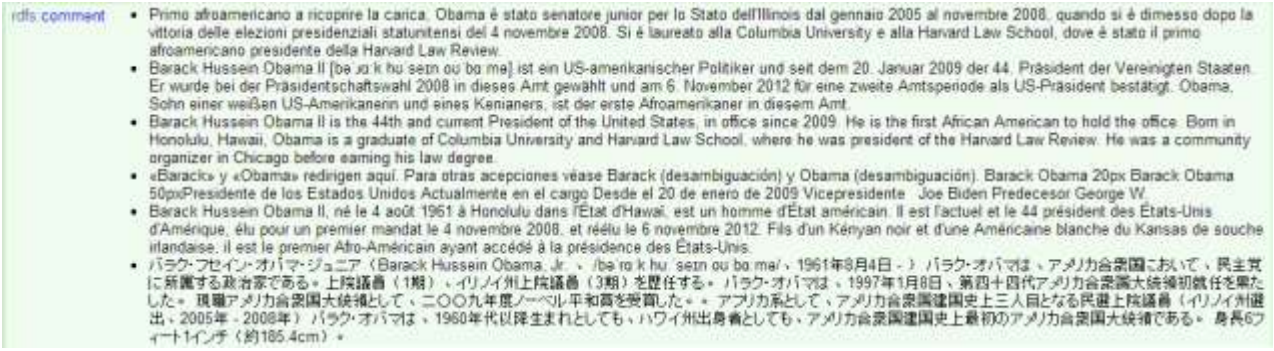

Figura 10 – Descrição DBPedia contendo dados multi-idioma.

Portanto, o objetivo da camada de filtragem analítica é varrer as descrições RDF para determinar metatriplas e triplas redundantes e em seguida removê-las, deixando apenas as triplas contendo informações consideradas úteis, do ponto de vista do usuário final. A filtragem analítica reduz consideravelmente a quantidade de triplas a serem tratadas nas camadas posteriores do modelo. Apesar do processo de filtragem analítica consumir tempo e recursos computacionais, o resultado final se reflete em um melhor desempenho no processo geral, tal como apresentado no Capítulo 8 (Avaliação).

Além de melhorar o desempenho diminuindo a quantidade de triplas a serem tratadas, a filtragem analítica contribui para a apresentação e navegação mais adequadas dos dados, levando em consideração as necessidades dos usuários comuns. Não é interessante exibir dados, aparentemente, sem sentido (metadados) ou dados repetidos (redundantes) para esse tipo de usuário. Logo, após a fase de filtragem analítica as demais camadas do modelo não precisam se preocupar com a tarefa de "limpeza" das informações. Ou seja, as triplas que permanecerem após o processo de filtragem, serão aquelas a serem consideradas na apresentação e navegação no final do processo.

A execução da tarefa de filtragem sobre as triplas de uma descrição RDF não é um processo trivial. Determinar computacionalmente o que é metatripla ou tripla redundante exige conhecimento antecipado dos vocabulários utilizados pelas diversas fontes de dados. Por exemplo, para reconhecer a tripla *<Barack_Obama> <wikiPageRevisionID> <548538288>* como metatripla, é necessário conhecer o vocabulário que possui o termo *wikiPageRevisionID* definido como meta informação. Algumas fontes de dados já estão tomando providências para facilitar esse reconhecimento. No caso da fonte DBPedia, a informação de idioma já vem marcada no próprio dado. A DBPedia insere no final da string (literal) uma *tag* indicando o idioma da informação (<string/literal>@<código do idioma>). Assim, considerando uma tripla com o objeto do tipo literal no idioma inglês, o final da string conterá a *tag @en*, sinalizando que essa string está definida na língua inglesa. Isso facilita o processamento da informação, agilizando a tarefa de reconhecimento de idioma e, consequentemente, da eliminação de redundâncias. Algumas metatriplas já

estão sendo marcadas pelo DBPedia usando a mesma técnica. No entanto, essa atualização ainda está no início e muitas triplas de metadados continuam sendo dificilmente identificáveis.

Outra maneira de determinar metatriplas é recuperar automaticamente o esquema do vocabulário que está sendo usado pela fonte de dados (RDFS ou OWL). O esquema define os atributos e propriedades de cada um dos termos do vocabulário. Dessa forma, é possível identificar os termos que apontam para metadados. Apesar de mais precisa, essa é uma abordagem custosa e que pode resultar em falhas, pois está sujeita aos problemas relacionados à transferência de dados por meio da rede. Portanto, a recuperação de ontologias para cada tripla em uma descrição RDF (considerando que uma descrição RDF pode conter termos de diversos vocabulários relacionando sujeitos a objetos) é uma tarefa que pode levar a inviabilidade do processo, em termos de confiabilidade e tempo de resposta. Uma abordagem interessante seria a criação de um vocabulário único descrevendo o domínio de metadados. Dessa forma, todas as fontes de dados usariam o mesmo vocabulário para definir seus metadados. Isso facilitaria o reconhecimento de metatriplas de forma mais precisa e eficiente. No Serviço Web, desenvolvido como parte desta proposta (Capítulo 6), o problema da identificação, tanto de metatriplas quanto de triplas redundantes, é resolvido pela técnica de leitura de *tags* inseridas no final do dado. No entanto, no caso das metatriplas, esse processo ainda apresenta limitações, como explicado anteriormente.

5.2 CAMADA DE MAPEAMENTO

Como visto na Seção 5.1, a camada de filtragem analítica (Figura 7, Camada 1) recebe, como entrada, dados estruturados no modelo RDF (padrão *Linked Data*). Esses dados podem estar representados por meio de um grafo ou, simplesmente, por meio de um conjunto de triplas RDF. Depois de realizar o processamento dos dados, baseado na análise e filtragem, a camada de filtragem analítica produz um modelo refinado como saída. O modelo refinado, apesar de manipulado, ainda obedece aos padrões RDF. Para obter os benefícios oferecidos pelo modelo Dexter (Capítulos 3), particularmente, pela camada de execução Dexter, é necessário mapear a descrição RDF refinada, resultante do trabalho da camada de filtragem analítica, para o modelo Dexter.

Portanto, o objetivo da camada de mapeamento (Figura 7, Camada 2) é traduzir os dados estruturados no modelo RDF, para dados obedecendo às características do modelo Dexter de referência de hipertexto. Depois do processo de mapeamento realizado, o resultado é uma rede hipertexto (adaptada às necessidades da proposta), modelada nos padrões Dexter, que será utilizada pelas camadas seguintes na

sequência de atividades. A rede Dexter é importante, principalmente, no nível da camada de preparação de interface e camada de apresentação e navegação, onde as características do modelo, e de sua camada de execução, serão exploradas para criar estruturas especializadas, com o objetivo de facilitar a tarefa de apresentação e navegação focadas no usuário comum.

A Web de Dados, assim como a Web de Documentos, pode ser considerada um sistema hipertexto (Conklin, 1987) e, portanto, mapeada para um modelo de referência. No entanto, por conta de algumas particularidades desse espaço de dados, algumas adequações e adaptações devem ser realizadas para que o mapeamento corresponda à realidade do ambiente *Linked Data*. A técnica de mapeamento praticada neste trabalho procurou satisfazer as necessidades da proposta da forma mais simples possível. Isso ajudou a criar uma correspondência "limpa" entre os modelos RDF e Dexter, deixando de lado as características mais complexas que não são de interesse explícito do trabalho. A Figura 11 mostra: um grafo RDF contendo dois recursos (R1 e R2), cinco literais (R1{L1, L2, L3} e R2{L1, L2}) e o relacionamento semântico entre os recursos, definido por P1 (Parte 1 da Figura 11); e o mapeamento desse grafo RDF para o modelo Dexter (Parte 2 da Figura 11).

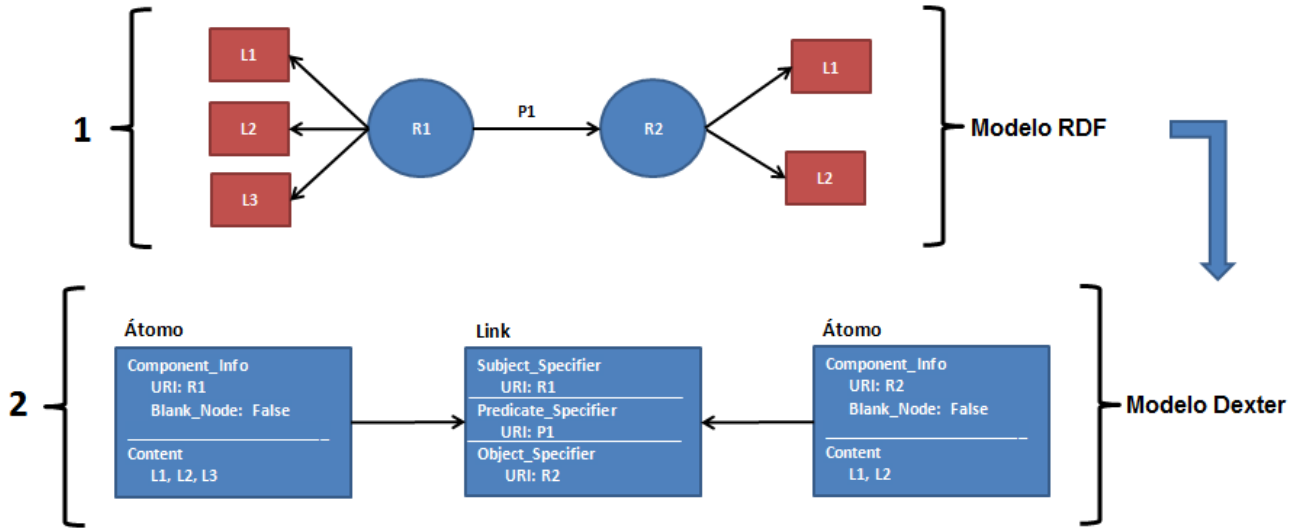

Figura 11 – Mapeamento básico entre os modelos RDF e Dexter.

O modelo Dexter define três tipos de componentes para modelar uma rede hipertexto: Átomo, Link e Composto. O componente composto contém, em seu interior, outros átomos, além de outros tipos de conteúdo (texto, figura, entre outros). O mapeamento projetado neste trabalho não prevê a utilização do componente composto do modelo Dexter. Isso é consequência do fato de que na Web de Dados um recurso não pode conter outro. De acordo com os princípios *Linked Data* (Berners-Lee, 2006), recursos se relacionam com outros recursos. Ou seja, um recurso é uma entidade única na Web de Dados, e não há hierarquia entre essas entidades, portanto, uma entidade não pode conter outra entidade em seu interior.

Além disso, recursos podem estar ligados a informações literais (texto, número, figura, url, entre outros) diretamente relacionadas a eles.

Nesse contexto, um recurso *Linked Data* corresponde a um componente Átomo do modelo Dexter. Na Parte 1 da Figura 11, os recursos R1 e R2 foram mapeados para átomos Dexter, que podem ser vistos na Parte 2 da Figura 11. O componente átomo, definido neste trabalho, é constituído de duas regiões: um *Component_Info* e um *Content*, assim como no modelo Dexter original (Parte 2 da Figura 11). O *component_info* contém informações do átomo (metadados), como, por exemplo, o URI do recurso que o átomo representa, e se esse recurso tem características de *Blank Node* na descrição RDF em tratamento. Essa região pode conter diversos tipos de informações, de acordo com as necessidades do sistema. A região *content* contém todos os literais ligados diretamente ao recurso mapeado. Ou seja, o conteúdo de todos os literais, relacionados ao recurso, é armazenado no interior do átomo.

O relacionamento entre os recursos é resolvido com a utilização do componente Link do modelo Dexter. Na Parte 1 da Figura 11, é possível observar que o relacionamento entre os recursos R1 e R2, representado pelo predicado P1, foi mapeado para um link Dexter, que pode ser visto na Parte 2 da Figura 11. No link Dexter original, os especificadores usam os campos especificação de componente, id da âncora e direção, para determinar a ligação entre dois átomos ou compostos (Figura 5 da Seção 3). No modelo desenvolvido neste trabalho, esse componente foi modificado para disponibilizar três especificadores (diferente do link Dexter original, com apenas dois especificadores). Cada especificador corresponde a uma parte da tripla RDF: *Subject_Specifier*, *Predicate_Specifier* e *Object_Specifier*. Os especificadores contêm os URIs que correspondem ao sujeito, predicado e objeto, respectivamente. Além dessa informação, os especificadores podem conter dados de apresentação, como definido na camada de execução Dexter. O campo URI desses especificadores (Parte 2 da Figura 11) faz o papel do campo especificador de componente. O direcionamento da ligação sempre se dará do *Subject_Specifier* para o *Object_Specifier*, dispensando o uso do campo direção. A semântica do relacionamento é provida pelo *Predicate_Specifier*, que possui apenas um campo, o tipo de relacionamento. Esse campo é usado para definir o tipo de ligação entre os recursos.

Dessa forma, é possível perceber que o modelo Dexter passou por algumas modificações em sua estrutura para manter a consistência com o modelo *Linked Data* (RDF) e, consequentemente, permitir a elaboração mais adequada do modelo de apresentação e navegação proposto nesta dissertação. Essas mudanças se deram, em sua maioria, na definição dos componentes da rede de nós e links, correspondendo, portanto, à camada de armazenamento Dexter. A Figura 12 mostra recursos reais da Web de Dados, e seus relacionamentos, mapeados para a adaptação do modelo Dexter projetada neste trabalho.

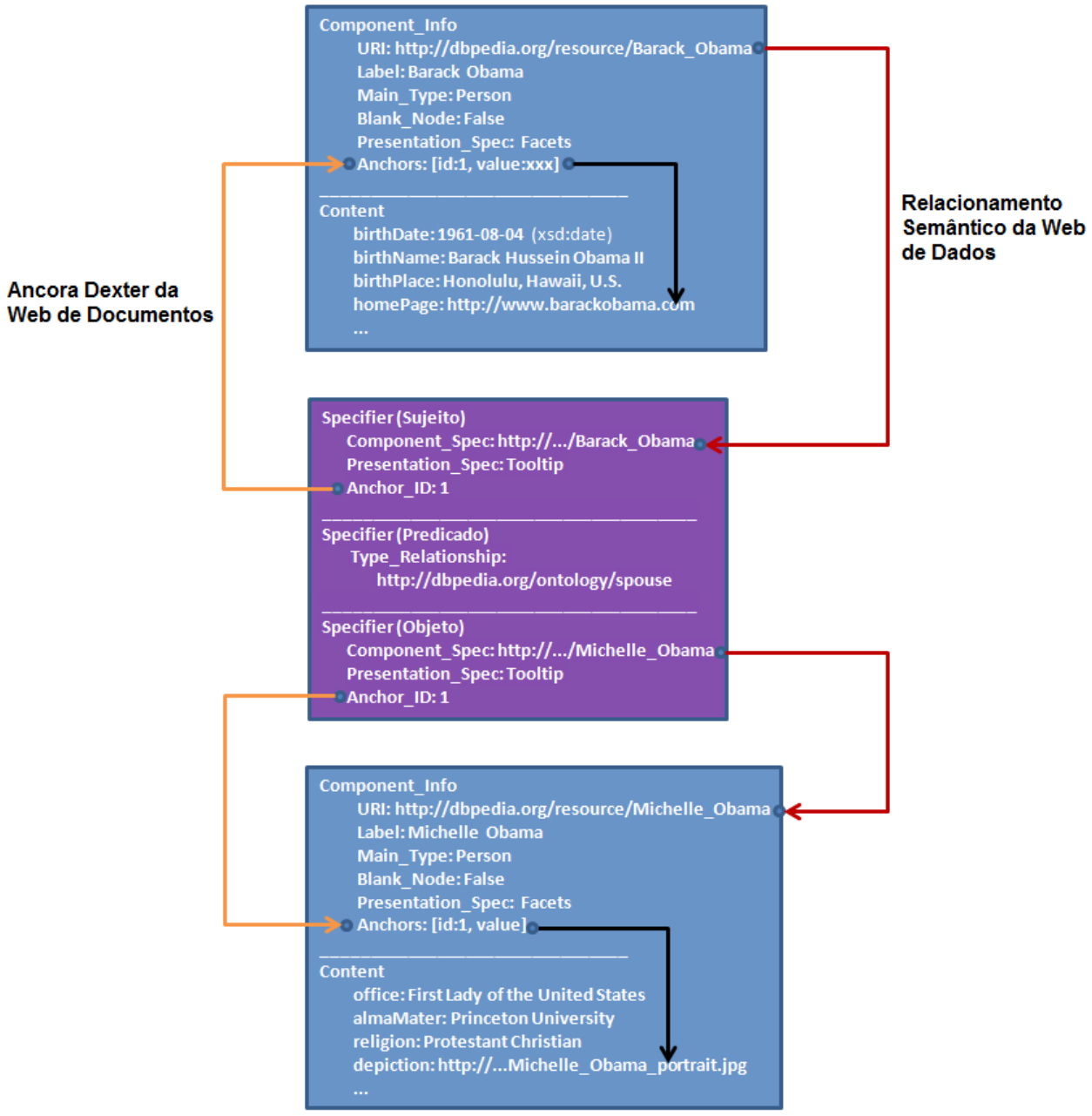

Figura 12 – Mapeamento de recursos reais Linked Data.

Na Figura 12, os recursos *http://dbpedia.org/resource/Barack_Obama* e *http://dbpedia.org/resource/Michelle_Obama* (bem como o relacionamento entre eles), disponibilizados pela fonte de dados DBPedia, foram mapeados para o modelo Dexter, obedecendo às necessidades da proposta. Cada recurso foi transformado em um átomo Dexter. Na região *componente_info*, os átomos possuem: *URI*, *Label*, *Main_Type*, *Blank_node*, *Presentation_Spec* e *Anchors*. O URI é usado como identificador único do átomo. Todo átomo presente na rede Dexter deve ser capaz de ser encontrado por meio de seu URI. O *label* é usado pela camada de apresentação e navegação com o objetivo de identificar o átomo de forma mais

amigável para o usuário final. *Main_type* contém o tipo principal do recurso, por exemplo, pessoa, lugar, organização, entre outros. Os recursos *Linked Data* podem possuir vários tipos em sua definição. O tipo mais relevante é armazenado em *main_type*. Essa informação é usada pela camada de descoberta e aconselhamento para obter novos conhecimentos, com base no tipo dos recursos. Um *blank node* é um recurso que não possui identificação no grafo RDF. Esses nós não identificados são gerados automaticamente por sistemas que manipulam as descrições RDF (sistemas de interpretação de dados RDFa, por exemplo). É importante identificar se um recurso é um *blank node* para tratá-lo de forma adequada. O atributo *blank_node* tem a função de identificar se o átomo representa um recurso não identificado. *Presentation_Spec* é usado também pela Camada de Apresentação e Navegação para definir o tipo mais adequado de apresentação que será usado para exibir o átomo. No exemplo da Figura 12, a forma de apresentação definida nesse atributo é faceta. O atributo *Anchors* é usado para manter a consistência do modelo com a Web de Documentos (Web 2.0), oferecendo suporte ao tipo de ancoragem usado na Web tradicional.

Na região *content*, os átomos armazenam todos os literais ligados aos recursos. É importante frisar que os átomos não armazenam outros recursos, apenas literais. Por exemplo, na Figura 12, o átomo com *label* "Barack Obama" possui como conteúdo a data, lugar e nome de nascença, além da *homepage*. O átomo "Michelle Obama" contém informações de ocupação, universidade, religião e imagem.

O componente link (parte central da Figura 12) faz a ligação entre os dois átomos, por meio dos especificadores de sujeito, predicado e objeto. Os especificadores de sujeito e objeto possuem como atributos: *Component_Spec*, *Presentation_Spec* e *Anchor_ID*. *Componet_Spec* identifica o átomo (sujeito ou objeto) da ligação. *Presentation_Spec* define o tipo de apresentação da ligação. Nesse caso, foi definida uma apresentação por *tooltip*. *Anchor_ID* identifica a ancora definida no *component_info* do átomo. O especificador predicado define o tipo de relacionamento semântico existente entre os átomos. Esse especificador contém apenas um atributo: *Type_Relationship*, que contém o URI do predicado da tripla RDF. É possível ainda observar, no exemplo, que o modelo de mapeamento oferece suporte tanto aos relacionamentos semânticos oriundos da Web de Dados, quanto aos relacionamentos tradicionais da Web de Documentos por meio das ancoras.

O mapeamento da descrição RDF para o modelo Dexter dá origem a uma rede Dexter, contendo todos os recursos *Linked Data* convertidos em átomos Dexter. Essa rede é interconectada por meio de links Dexter, que representam as ligações semânticas do modelo RDF. A rede Dexter é então passada como entrada para a camada seguinte na sequência de atividades (camada de descoberta e aconselhamento).

5.3 CAMADA DE DESCOBERTA E ACONSELHAMENTO

Depois do trabalho realizado pela camada de mapeamento (Figura 7, Camada 2), a rede Dexter está pronta para ser usada pelas camadas remanescentes. Na maioria das vezes, a rede produzida pelo processo de mapeamento possui uma considerável quantidade de nós e conexões. Para facilitar o trabalho da camada de descoberta e aconselhamento (Figura 7, Camada 3), bem como, do restante das camadas do modelo, é necessário eleger o átomo mais relevante da rede. O objetivo dessa eleição é criar um alvo para o qual as camadas possam direcionar seus esforços, diminuindo o custo computacional e acelerando o processo. Por exemplo, o processo de descoberta e aconselhamento envolve acessos externos a diferentes fontes de dados e domínios. Portanto, esse procedimento é altamente dependente das condições da rede. Nesse cenário, é possível imaginar uma tentativa da camada de descoberta e aconselhamento objetivando obter novos conhecimentos para cada átomo existente na rede recentemente criada. A quantidade de átomos pode chegar a centenas e até milhares. O processo facilmente se tornaria inviável do ponto de vista do desempenho nas respostas. Por esse motivo, o trabalho realizado pelas camadas, a partir desse ponto, é centrado em um único átomo (aquele considerado de maior relevância).

Para determinar qual o átomo mais relevante da rede é possível usar um princípio básico da teoria dos grafos: o grau do nó. Portanto, o átomo considerado mais relevante pode ser aquele que possui o maior número de conexões, tanto de entrada como de saída. Ou seja, o nó que possui o maior grau. Considerar a quantidade de conexões dos átomos como critério de relevância é consistente com o conceito da Web de Dados. Um recurso que possui muitos relacionamentos é mais rico em informações e conhecimento. A partir desse recurso é mais simples descobrir novas referências para outros recursos, até então desconhecidos. Um problema que pode ocorrer durante a eleição do átomo mais relevante é a existência de mais de um átomo com o mesmo número de conexões (nós com o mesmo grau). Uma maneira de resolver essa questão é deixar o usuário decidir qual recurso ele deseja explorar. Por outro lado, pode-se simplesmente escolher aleatoriamente um dos átomos. No Serviço Web, desenvolvido como parte desta proposta (descrito no Capítulo 6), esse problema é resolvido por escolha aleatória.

A partir do momento que o átomo mais relevante passa a ser conhecido, a camada de descoberta e aconselhamento pode direcionar melhor seus esforços. Uma das abordagens usadas pela camada de descoberta e aconselhamento para obter novos conhecimentos é a utilização de consultas SPARQL. Basicamente, uma consulta ao URI do recurso, representado pelo átomo mais relevante, é executada na fonte de dados que disponibiliza tal entidade. O resultado dessa consulta básica é uma descrição RDF contendo todos os recursos e literais (disponibilizados pela fonte de dados) diretamente relacionados ao recurso

consultado. A partir daí, outras consultas mais complexas podem ser formuladas. Por exemplo, é possível seguir ligações definidas como *sameAs* para obter mais dados sobre o recurso consultado em outras fontes de dados. É possível também, descobrir novos recursos com base nos tipos definidos para o recurso consultado. Ou seja, considerando que o recurso consultado seja do tipo "político brasileiro" (entre outros tipos definidos), é possível recuperar recursos do mesmo tipo, mesmo que esses recursos não estejam diretamente ligados ao recurso consultado. Essa técnica auxilia o aconselhamento da navegação considerando o tipo de conhecimento que está sendo explorado pelo usuário.

Outra abordagem utilizada pela camada de descoberta e aconselhamento é a consulta a Web 2.0 por meio das APIs disponibilizadas pelos diversos domínios existentes nesse ambiente. Usar elementos contidos na Web 2.0 para enriquecer o conjunto de informações acerca de recursos *Linked Data* é uma tendência que vem se firmando cada vez mais na comunidade da Web Semântica. Afinal de contas, a Web de Dados e a Web de Documentos não são disjuntas, pelo contrário, a ideia por trás da Web de Dados é que ela seja uma extensão da Web tradicional (Berners-Lee et al., 2001). Portanto, são cada vez mais reconhecidos os processos e métodos de convergência no sentido de aproximar as entidades da Web de Dados das entidades da Web de Documentos. Além disso, os domínios da Web 2.0, estão integrando suas bases a fontes de dados *Linked Data*, provendo suporte a consultas semânticas e fortalecendo ainda mais a convivência entre esses ambientes. É o caso do Flickr[11], Amazon[12], Yahoo[13], entre outros. Do ponto de vista do usuário final, a descoberta de elementos da Web 2.0, relacionados aos recursos *Linked Data* explorados por esses indivíduos, traz novas possibilidades, uma vez que eles podem ter acesso a vídeos, fotos, notícias e muito mais. A Figura 13 mostra o passo a passo do processo de descoberta e aconselhamento.

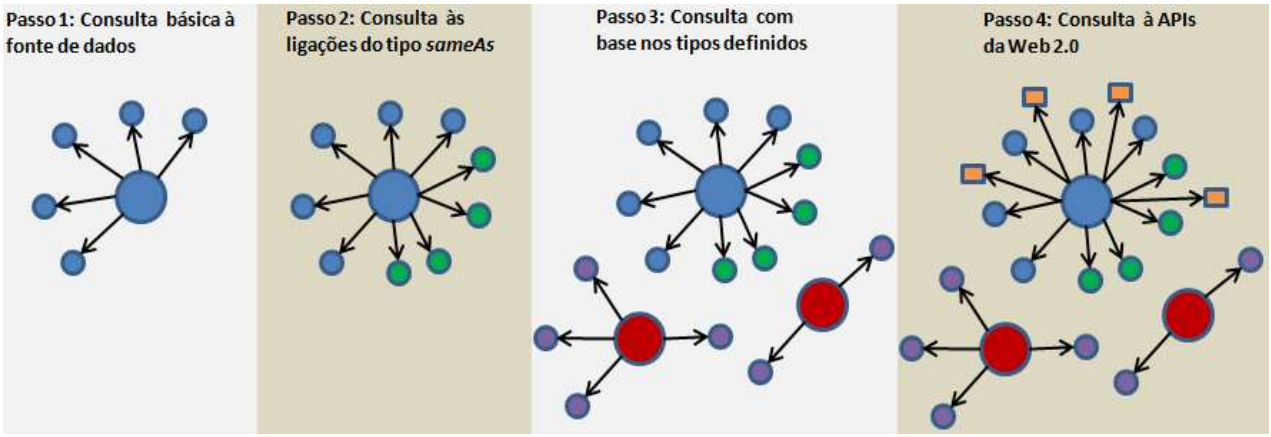

Figura 13 – Passo a passo da descoberta e aconselhamento.

[11] http://wifo5-03.informatik.uni-mannheim.de/flickrwrappr/

[12] https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html

[13] http://developer.yahoo.com/everything.html

No passo 1, da Figura 13, uma consulta SPARQL básica é executada contra a fonte de dados que detém o recurso consultado, representado pelo átomo mais relevante da rede Dexter, por meio do URI. O resultado dessa consulta é a descrição RDF do recurso. Nessa descrição é possível identificar, entre outras coisas, ligações de tipo *sameAs* (tipo de relacionamento usado para sinalizar que o recurso possui outras descrições que podem residir em fontes diversas ou na própria fonte de dados sendo consultada). Assim, no passo 2, consultas são executadas com base nas ligações *sameAs*. O resultado são novas descrições RDF do recurso consultado contendo relacionamentos para outros recursos e literais, até então desconhecidos. Esse processo acrescenta novas triplas ao grafo RDF (novos átomos e links na rede Dexter). Em seguida, no passo 3, são realizadas consultas levando-se em consideração os relacionamentos que definem os tipos do recurso consultado (ligações de tipo *rdfs:type*). O resultado das consultas por tipo trazem novos recursos semelhantes ao recurso consultado. Esses recursos podem não estar necessariamente ligados ao recurso consultado. Na etapa final, passo 4, uma consulta às APIs da Web 2.0 é executada para obter informações e elementos relacionados ao recurso consultado, como vídeos, fotos, notícias, entre outros.

É importante observar que o resultado das consultas são descrições RDF. Ou seja, os dados estão modelados no padrão RDF. No entanto, o mapeamento do modelo RDF para o modelo Dexter já foi realizado na camada anterior (camada de mapeamento). Isso levanta a questão da localização mais adequada para a camada de descoberta e aconselhamento. Por exemplo, a camada de descoberta e aconselhamento poderia estar alocada antes da camada de mapeamento. Dessa forma, os dados coletados durante as consultas seriam adicionados ao grafo ou triplas RDF e, só então, na camada de mapeamento, transformados em átomos e links Dexter. Por outro lado, com a camada de descoberta e aconselhamento alocada depois da camada de mapeamento (abordagem utilizada neste trabalho) é possível se beneficiar de todo o potencial do modelo Dexter. Por exemplo, a manipulação de um único átomo Dexter, do ponto de vista do esforço de desenvolvimento, é mais simples que a manipulação de um grande conjunto de triplas. Outra questão importante é que a rede Dexter, gerada no processo de mapeamento, é menor (em quantidade de nós e arestas) que o grafo RDF correspondente. Isso porque os átomos Dexter contêm todos os literais referentes aos recursos que eles representam. Essa característica torna a manipulação da rede Dexter mais “leve”, aumentando o desempenho do processo geral.

5.4 CAMADA DE PREPARAÇÃO DE INTERFACE

Depois do processo de descoberta e aconselhamento finalizado, a rede Dexter resultante, enriquecida com novos conhecimentos e dados relevantes, está pronta para ser utilizada pelas próximas camadas na sequência de atividades: camada de preparação de interface e camada de apresentação e navegação. O foco dessas camadas é na interface com o usuário comum. Entretanto, utilizar a estrutura da rede Dexter puramente, para prover apresentação e navegação, adequadas às necessidades desse tipo de usuário, é uma tarefa complexa. Para facilitar a realização desse trabalho, a camada de preparação de interface aproveita os elementos desenvolvidos na construção da rede Dexter (átomos, links e especificadores) para criar estruturas adequadas ao projeto de interfaces, focadas no usuário comum.

Portanto, o objetivo da camada de preparação de interface (Figura 7, Camada 4) é confeccionar elementos que possam ser usados para facilitar a construção de interfaces, especialmente preocupadas com a apresentação e navegação de *Linked Data*, focada no usuário inexperiente. A camada de preparação de interface fornece construções prontas para serem utilizadas por outras camadas de mais alto nível (por exemplo, a camada de apresentação e navegação) ou diretamente por sistemas que desejam implementar interfaces com base nessas estruturas (sem utilizar a camada de apresentação e navegação fornecida pelo modelo proposto). As estruturas criadas na camada de preparação de interface podem referenciar o átomo mais relevante (construções automatizadas) ou um determinado átomo, definido por uma ação (um clique, por exemplo). A Figura 14, mostra duas das estruturas criadas na camada de preparação de interface: um Box e uma List.

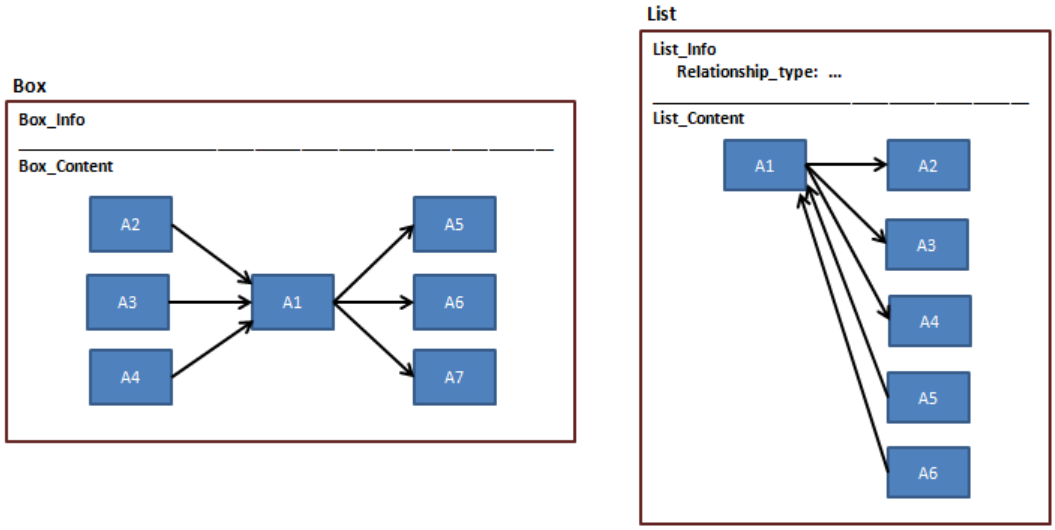

Figura 14 – Estruturas Box e List da camada de preparação de interface.

Conforme pode ser visto na Figura 14, toda estrutura contem duas regiões: *info* e *content* (*Box_Info*, *List_Info*, *Box_Content* e *List_Content*). A região *info* contém metadados (atributos e propriedades) sobre a estrutura, e a região *content* é onde o conteúdo da estrutura é armazenado. A estrutura Box contém um átomo referencial (que pode ser o átomo mais relevante da rede Dexter ou qualquer átomo da rede, definido por outro método) e todos os átomos que estão diretamente relacionados ao átomo referencial. Os relacionamentos podem ser de entrada (nesse caso, o átomo referencial está no contexto de objeto da tripla RDF) ou de saída (átomo referencial no contexto de sujeito da tripla RDF). Na Figura 14, o Box contém o átomo referencial A1 e os átomos diretamente ligados a ele: A2, A3, A4 (relacionamento de entrada), A5, A6 e A7 (relacionamento de saída). A estrutura Box pode ser utilizada para criar interfaces de diversos tipos, sem a preocupação de percorrer o grafo RDF para determinar recursos relacionados. Por exemplo, um Box pode ser usado para criar um *tooltip* dinâmico no evento *mouseover* sobre um recurso. O recurso em destaque seria o átomo referencial e o *tooltip* apresentaria informações relacionadas a esse recurso por meio dos átomos diretamente ligados ao referencial.

A estrutura List contém, assim como o Box, um átomo referencial e átomos diretamente ligados ao referencial. A diferença é que essas ligações são baseadas em apenas um tipo de relacionamento. Uma List pode ser considerada um Box filtrado por um tipo de relacionamento. O atributo *Relationship_type*, na região *info* da List, define o tipo de relacionamento anotado na List. Por exemplo, considerando que a List exibida na Figura 14 está anotada com o relacionamento do tipo "conhece". Isso significa que o átomo referencial A1 conhece os átomos A2, A3 e A4 (relacionamento de saída) e, da mesma forma, os átomos A5 e A6 conhecem o referencial A1 (relacionamento de entrada). Diferente do Box, que considera todos os tipos de relacionamento, a List é baseada apenas no relacionamento anotado em *Relationship_type*. É possível disponibilizar variantes da List considerando a direção dos relacionamentos. Assim, uma inList é uma List que possui, além do referencial, apenas os átomos que estão ligados a ele por meio de um determinado relacionamento de entrada. Da mesma forma, uma outList é uma lista contendo átomos ligados ao referencial por meio de um relacionamento de saída. Lists podem ser utilizadas para fornecer, por exemplo, navegação facetada. O filtro usado para definir as facetas pode ser o tipo de relacionamento anotado para cada List. A Figura 15 mostra mais duas estruturas disponibilizadas pela camada de preparação de interface.

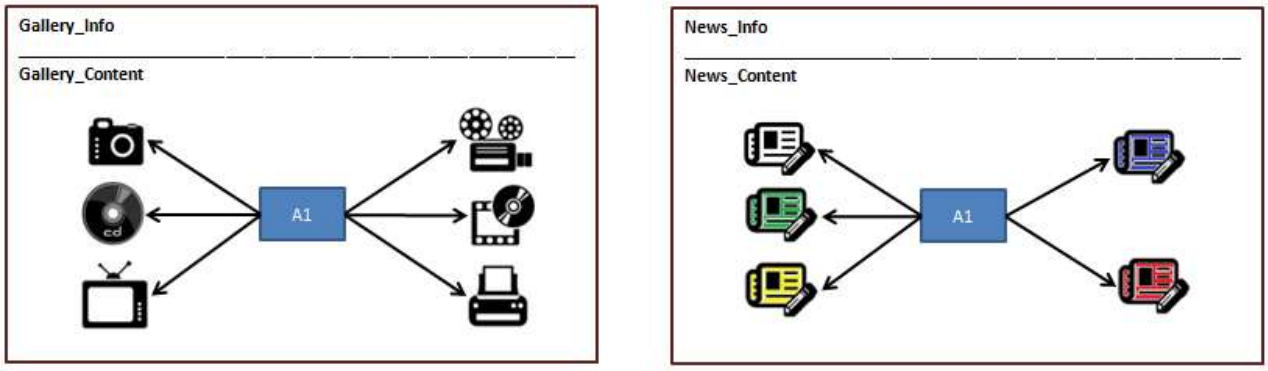

Figura 15 – Estruturas Gallery e News construídas a partir da Web 2.0.

As estruturas Gallery e News são construídas a partir dos resultados obtidos pelas consultas realizadas às APIs da Web 2.0, por meio da camada de descoberta e aconselhamento. Essas estruturas possuem um átomo referencial, assim como as estruturas descritas anteriormente, e recursos de mídia e notícias diretamente ligados aos seus respectivos referenciais. Na estrutura Gallery, o átomo referencial está ligado a diversos conjuntos de mídias (fotos, vídeos, áudio, TV, entre outros), oriundos de domínios como Flickr, Youtube e Yahoo. Na estrutura News, o referencial possui ligações com artigos de notícias disponibilizados por diversas agências de notícias espalhadas pelo mundo, como New York Times, BBC, CNN, entre outras. Os relacionamentos entre o átomo referencial e os conjuntos de mídia e notícias são determinados pela integração dos domínios com a Web de Dados. Ou seja, cada um desses domínios possui uma base RDF que disponibiliza recursos *Linked Data* relacionados a seus objetos de mídia e notícias. Essas estruturas são importantes para facilitar o intercâmbio de conhecimento durante a interação do usuário com ambos os espaços de informação, Web de Dados e Web de Documentos. A camada de preparação de interface ainda fornece outro tipo de estrutura, a estrutura General Presentation, que disponibiliza uma apresentação geral a partir de outras estruturas (Box, List, inList, outList, Gallery e News). A ideia da General Presentation é disponibilizar, em um única estrutura, o total atendimento às necessidades de apresentação e navegação do usuário final. Outras estruturas podem ser criadas e definidas na camada de preparação de interface para prover tipos variados de apresentação e navegação, aperfeiçoando a interação com o usuário final.

5.5 CAMADA DE APRESENTAÇÃO E NAVEGAÇÃO

A camada de preparação de interface (Figura 7, Camada 4) cria estruturas especializadas para facilitar a construção de interfaces voltadas para o usuário comum. Por sua vez, a camada de apresentação e navegação (Figura 7, Camada 5) utiliza essas estruturas para, efetivamente, apresentar e prover navegação nos dados contidos na rede Dexter, produzida durante o processo. A camada de apresentação e navegação é a camada de mais alto nível do modelo, implementando a interface entre o usuário e a rede Dexter.

A camada de apresentação e navegação não é rígida. Por possuir as características e propriedades de uma camada de interface, ela pode ser modificada para dar origem a interfaces mais robustas e adequadas às necessidades dos sistemas. Todas as alterações nas interfaces entre o usuário e a rede Dexter devem ser implementadas nessa camada. Considerando uma arquitetura cliente-servidor, por exemplo, a camada de apresentação e navegação está intrinsecamente relacionada ao lado cliente. Ou seja, em uma implementação efetiva do modelo proposto neste trabalho, as camadas de mais baixo nível (Filtragem Analítica, Mapeamento, Descoberta e Aconselhamento e Preparação de Interface) estariam alocadas no lado servidor, enquanto a camada de apresentação e navegação estaria alocada no lado cliente do sistema. Essa é a arquitetura adotada na implementação do modelo descrita no Capítulo 6, onde as quatro primeiras camadas do modelo proposto são implementadas em um Serviço Web (Seção 6.1) e a camada de apresentação e navegação é implementada em uma biblioteca no lado cliente (Seção 6.2).

As informações utilizadas pela camada de apresentação e navegação, para gerar as interfaces com o usuário, são obtidas dos atributos contidos nas regiões *info* das estruturas criadas pela camada de preparação de interface. Como descrito na Seção 5.4, essas regiões possuem metadados acerca de cada tipo de estrutura. Entre esses metadados, está o atributo *Presentation_Spec*, que define o tipo de apresentação a ser usado pela camada de apresentação e navegação. É importante lembrar que cada componente da rede Dexter (átomos e links) também possuem o atributo *Presentation_Spec* em suas regiões *info*. O *Presentation_Spec* de uma estrutura é definido com base no *Presentation_Spec* do átomo referencial, dos links entre o referencial e os demais átomos e dos átomos relacionados. Cabe ao sistema que implementa o modelo determinar as diretrizes para essa definição. Por exemplo, o *Presentation_Spec* de uma estrutura pode ser definido pelo *Presentation_Spec* do átomo referencial, uma vez que a estrutura é uma representação desse átomo e seus relacionamentos. Outra abordagem interessante é a utilização de pesos em conjunto com o *Presentation_Spec*. Ou seja, para cada átomo e link Dexter, um peso seria atribuído na região *info* (um par chave-valor), relativo ao *Presentation_Spec* ({Presentation_Spec: faceta, peso: 3}). Esses pesos seriam computados para cada tipo de *Presentation_Spec*, e o tipo definido na estrutura seria aquele com maior

resultado no somatório. A Figura 16 mostra um exemplo da construção de interfaces por meio da camada de apresentação e navegação com base nas estruturas criadas na camada de preparação de interface.

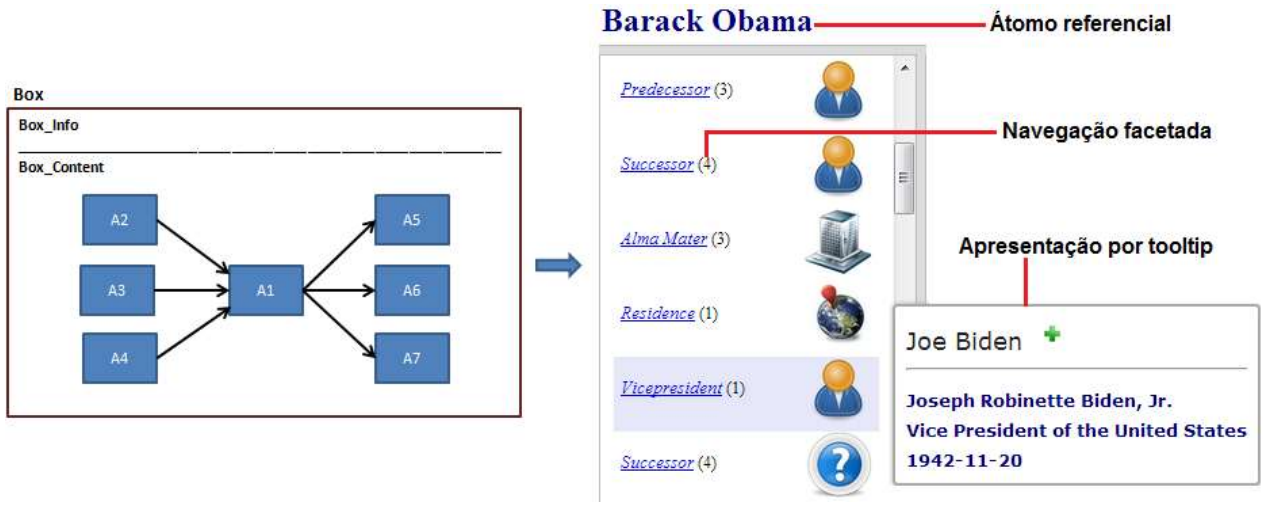

Figura 16 – Exemplo de interface construída a partir de uma estrutura Box.

A Figura 16 mostra o Box do átomo que representa o recurso *http://dbpedia.org/resource/Barack_Obama*, disponível na fonte DBPedia, e a interface correspondente que pode ser gerada a partir dessa estrutura. O *label* do átomo referencial é apresentado no topo de uma lista de recursos, relacionados ao átomo. A lista foi construída com suporte a navegação facetada, filtrada pelos tipos de relacionamento. Os recursos relacionados são apresentados por meio de *tooltips* dinâmicos. O suporte a navegação facetada da lista e a apresentação por *tooltips* dos recursos relacionados foram definidos pelo *Presentation_Spec* da estrutura e dos átomos. A Figura 17 mostra outro tipo de interface que pode ser gerada a partir de uma estrutura List.

Figura 17 – Exemplo de interface construída a partir de uma estrutura List.

A List exibida na Figura 17 possui como referencial o átomo que representa o recurso *http://dbpedia.org/resource/Barack_Obama*, assim como o Box da Figura 16. Além disso, a List está anotada, em seu atributo *Relationship_Type*, com o tipo de relacionamento *Successor*. A interface correspondente, gerada a partir dessa List, apresenta uma lista de recursos ligados ao referencial por meio de relacionamentos de saída e entrada. É possível identificar algumas das triplas que definem esses relacionamentos: <*Barack Obama*> <*Successor*> <*Roland Burris*> (saída) e <*George W. Bush*> <*Successor*> <*Barack Obama*> (entrada). Nesse caso, o tipo de apresentação definida no *Presentation_Spec* da estrutura é uma lista simples. A Figura 18 mostra a interface que pode ser gerada a partir de uma estrutura Gallery.

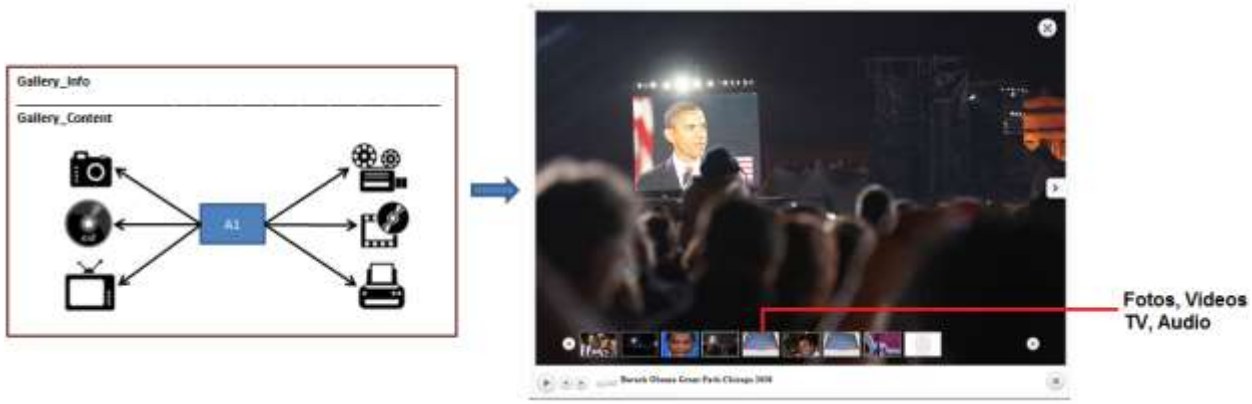

Figura 18 – Exemplo de interface construída a partir de uma estrutura Gallery.

A Gallery da Figura 18 também possui "Barack Obama" como átomo referencial. A interface criada a partir dessa estrutura é um tipo de galeria de mídias, onde o usuário pode navegar e visualizar elementos relacionados ao referencial como: fotos, vídeos, TV, entre outros. É comum que estruturas Gallery sejam definidas com esse tipo de apresentação no seu *Presentation_Spec*, pelas características apresentadas por esses elementos de mídia.

A estrutura News, semelhante a Gallery, pode ser usada pala camada de apresentação e navegação para construir interfaces onde o usuário pode navegar entre listas animadas (por exemplo, *accordions*[14]) para visualizar notícias relacionadas ao referencial. A estrutura General Presentation pode usar todas essas subestruturas e seus correspondentes tipos de interface para gerar um *frame* contendo uma apresentação completa com suporte a navegação e interação do usuário final.

Os exemplos de tipos de interfaces, construídos a partir das estruturas criadas pela camada de preparação de interface, mostrados acima, foram projetados usando a biblioteca cliente desenvolvida para se comunicar com o Serviço Web (Seções 6.1 e 6.2), por meio do protótipo, descrito na Seção 6.3.

[14] Accordion: http://jqueryui.com/accordion/
Apresenta painéis de conteúdo dobráveis para apresentação de informações em uma quantidade limitada de espaço.

5.6 OPERAÇÕES

Baseado nas cinco camadas descritas anteriormente, o modelo proposto fornece algumas operações elementares, que definem ações a serem executadas durante a sequência de atividades envolvendo a apresentação e navegação. Essas operações podem ser de mais alto nível, resultando no propósito geral de uma camada, ou de mais baixo nível, gerando resultados que podem ser utilizados pelas operações de alto nível. Por exemplo, a camada de filtragem analítica disponibiliza a operação de alto nível *Filter RDF Description* (descrita abaixo). O objetivo dessa operação é filtrar os dados RDF recebidos como entrada e gerar dados "limpos" na saída. Como dito anteriormente, ações de mais baixo nível são mapeadas em suboperações para dar suporte às operações de alto nível. Por exemplo, as suboperações *Remove Metatriples* e *Remove Redundant Triples* auxiliam na execução da operação de alto nível *Filter RDF Description*. A seguir, a descrição das operações fornecidas pelo modelo proposto neste trabalho.

1. ***Filter RDF Description*:** Recebe como parâmetro de entrada uma descrição RDF, que pode ser representada por um grafo ou um conjunto de triplas. O objetivo geral da operação é analisar cada tripla da descrição, determinando a existência de metadados e/ou redundâncias (Seção 5.1). Em seguida, caso existam, remover as metatriplas e triplas redundantes da descrição. O retorno da operação é uma descrição RDF filtrada, livre de metadados e informações redundantes e, portanto, mais "leve" (contendo uma quantidade menor de triplas). Além disso, a descrição RDF resultante possui apenas triplas úteis do ponto de vista da apresentação e navegação. Esta operação pode ser utilizada no início do processo, antes de qualquer outra operação e, durante o processo, para filtrar novos dados adquiridos por outras operações (é o caso da operação *Discover Knowledge* que descobre novos recursos e os acrescenta à rede Dexter). A operação utiliza as suboperações *isMetatriple(triple)* e *isRedundantTriple(triple)*. Que por sua vez, utilizam as suboperações *removeMetatriple(triple)* e *removeRedundantTriple(triple)*.

2. ***Map RDF to Dexter*:** Recebe como parâmetro de entrada uma descrição RDF, que pode ser representada por um grafo ou um conjunto de triplas. O objetivo geral da operação é mapear os dados do modelo RDF para o modelo Dexter. Ou seja, transformar os recursos *Linked Data* contidos na descrição RDF em Átomos e relacioná-los por meio de Links, formando uma rede hipertexto Dexter. O retorno da operação é a rede Dexter, bem formada, resultante do método de mapeamento (Seção 5.2), pronta para ser utilizada por outras operações no restante do processo. Esta operação pode ser utilizada depois da operação *Filter RDF Description* para mapear a descrição RDF resultante da filtragem e, depois de operações de aquisição de dados RDF (por exemplo, *Discover Knowledge*), para mapear os novos dados obtidos. A

operação utiliza as suboperações *isResource(tripleObject)*, *isLiteral(tripleObject)*, *insertLiteral(tripleObject, atom)*, *createAtom()*, *createLink()*, *conectAtoms(atom1, link, atom2)* e *netContains(atom)*.

3. ***Discover Knowledge*:** Recebe como parâmetro de entrada uma rede Dexter, representada pelos elementos (átomos e links) definidos na Seção 5.2. O objetivo geral da operação é, primeiramente, determinar o átomo mais relevante da rede (Seção 5.3). Em seguida, recuperar novos conhecimentos, por meio de acessos a Web de Dados (SPARQL) e Web 2.0 (APIs), com base no átomo mais relevante. Os novos dados obtidos no processo podem ser tratados pelas operações *Filter RDF Description* e *Map RDF to Dexter* para, respectivamente, filtrar e converter os dados para o modelo Dexter e, em seguida, adicioná-los à rede. O retorno da operação é uma rede enriquecida pela descoberta de novos recursos, literais e relacionamentos ligados ao átomo mais relevante, que será usada para criar interfaces de usuário mais requintadas e expressivas. Esta operação pode ser usada logo após a operação *Map RDF to Dexter*, para executar o método de descoberta aproveitando a rede Dexter recentemente criada. A operação utiliza as suboperações *getRepresentAtom(dexterNet)*, *dereferenceAtom(atomURI)*, *getSameAs(triples)*, *followSameAs(URIs)*, *getType(triples)*, *getResourcesByType(type)*, *getResources2.0(URIs, APIs)* e *insertTriples(triple)*.

4. ***Prepare Interface*:** Recebe como parâmetro de entrada uma rede Dexter, representada pelos elementos (átomos e links) definidos na Seção 5.2. O objetivo geral da operação é criar estruturas apropriadas para facilitar a construção de interfaces focadas nas necessidades de apresentação e navegação dos usuários comuns. O retorno da operação é um conjunto de estruturas especializadas contendo dados organizados que podem ser usados para criar interfaces de usuário de maneira simples e fácil (Seção 5.4). Esta operação pode ser usada depois da operação *Discover Knowledge*, aproveitando a rede enriquecida pela descoberta de novos conhecimentos acerca do átomo mais relevante. A operação utiliza as suboperações *createBox(atom)*, *createList(atom)*, *createInList(atom)*, *createOutList(atom)*, *createGallery(atom)*, *createNews(atom)*, *getAtom(URI, dexterNet)* e *createGeneral Presentation(dexterNet)*.

5. ***Presents Network*:** Recebe como parâmetro de entrada um conjunto de estruturas especializadas apropriadas para facilitar a construção de interfaces de usuário. O objetivo geral da operação é criar interfaces adequadas para prover apresentação e navegação dos dados *Linked Data*, contidos nas estruturas. O resultado da operação é a exibição, com suporte a navegação, dos dados semânticos para o usuário final (Seção 5.5). Esta operação pode ser usada no lado cliente das aplicações, depois da utilização da operação *Prepare Interface*, com o objetivo de fazer uso das estruturas disponibilizadas. A operação utiliza as suboperações *presentsBasicInfo(box)*, *presentsBox(box)*, *presentsList(list)*, *presentsGallery(gallery)*, *presentsNews(news)*, *presentsAtom(atom)* e *presentsGeneral(generalPresentation)*.

Algumas suboperações utilizam outras suboperações para realizar a tarefa sob sua responsabilidade. É o caso, por exemplo, da suboperação *isMetatriple(triple)*, que utiliza a suboperação

removeMetatriples(triples) e da suboperação *createAtom()*, que utiliza *isResource(tripleObject)*, *isLiteral(tripleObject)* e *insertLiteral(tripleObject, atom)*. Abaixo uma descrição de algumas suboperações disponibilizadas pelo modelo de apresentação e navegação proposto aqui.

1. **isMetatriple(triple) e isRedundantTriple(triple)**: Identifica se uma determinada tripla é (ou não) uma meta tripla ou uma tripla redundante. Ou seja, se a informação contida em seu objeto é meta informação ou já está presente na descrição RDF. Na implementação do modelo (Capítulo 6), a tarefa de identificação usa a técnica de leitura de *tag*.
2. **removeMetatriple(triple) e removeRedundantTriple(triple)**: Ambas as suboperações têm o objetivo de remover triplas indesejadas contidas na descrição RDF do contexto.
3. **isResource(tripleObject) e isLiteral(tripleObject)**: Ambas as suboperações têm o objetivo de identificar o tipo de objeto de uma determinada tripla, recurso ou literal. Na implementação do modelo (Capítulo 6), a identificação é feita pelo URI (se o conteúdo é um URI, então é recurso).
4. **insertLiteral(tripleObject)**: Insere o conteúdo do objeto de uma tripla literal na região *content* de um átomo Dexter.
5. **createAtom()**: Cria um átomo Dexter a partir de um recurso *Linked Data*. Para criar o átomo, todos os vizinhos do recurso são avaliados. Literais são inseridos no átomo. Outros recursos são transformados em outros átomos.
6. **createLink()**: Cria um link Dexter a partir de um relacionamento entre dois recursos *Linked Data*. O *Predicate_Specifier* é carregado com o tipo de relacionamento definido pelo predicado da tripla.
7. **conectAtoms(atom1, link, atom2)**: Cria uma ligação entre dois átomos Dexter por meio de um link Dexter. O *Subject_Specifier* e o *Object_Specifier* do link são carregados com os URIs dos átomos.
8. **netContains(atom)**: Verifica a existência de um átomo Dexter na rede. Em uma rede Dexter cada átomo é único.

O restante das suboperações são autoexplicativas e já foram abordadas na descrição das camadas de descoberta e aconselhamento, preparação de interface e apresentação e navegação. Por exemplo: *getRepresentAtom(dexterNet)*, determina o átomo mais relevante da rede; *getSameAs(triples)*, recupera as triplas com relacionamento do tipo *SameAs* (Seção 5.3); *createBox(atom)*, cria um Box a partir de um átomo; *createGallery(atom)*, cria uma galeria a partir de um átomo (Seção 5.4); *presentsBox(box)*, apresenta um Box para o usuário final; *presentsGallery(gallery)*, apresenta uma galeria para o usuário final (Seção 5.5).

A Figura 19 mostra o diagrama de sequência do processo geral com destaque para as operações, bem como suas suboperações, disponibilizadas pelo modelo proposto.

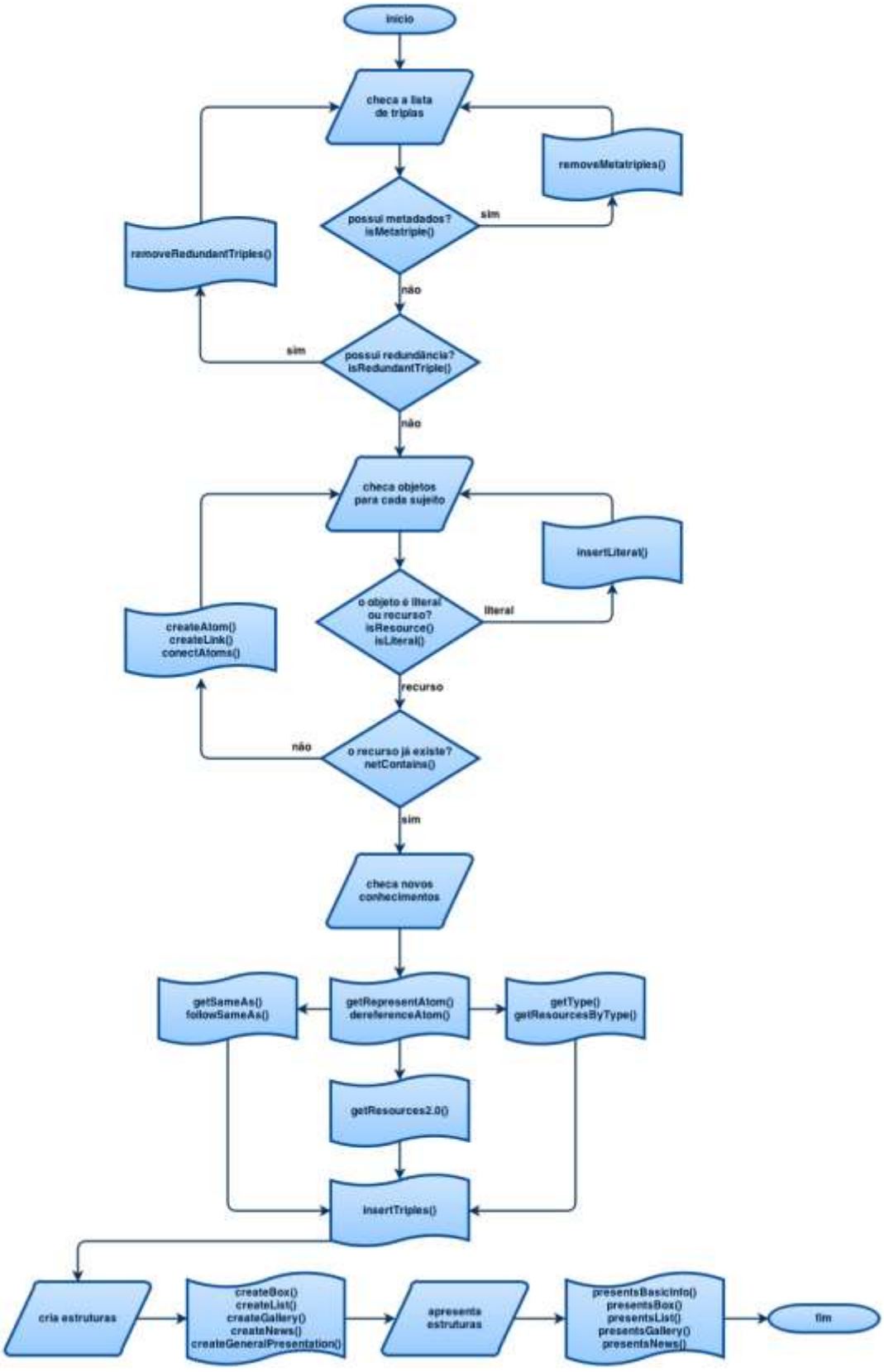

Figura 19 – Diagrama de sequência destacando suboperações.

O diagrama da Figura 19, mostra a seguinte sequência de ações: primeiro, a lista de triplas (descrição RDF) é checada. Se existem metatriplas, elas são removidas, caso contrário, verifica-se a existência de triplas redundantes. Em caso positivo de redundância, elas são removidas da mesma forma que as metatriplas. Em seguida, para cada sujeito das triplas, os objetos, ligados a ele, são checados. Se o objeto é um literal, então o literal é inserido no átomo que representa o sujeito da tripla (recurso). Caso contrário, significa que o objeto é um recurso. Nesse caso, verifica-se se o átomo que representa o recurso já está presente na rede. Caso o átomo não esteja presente, ele é criado e conectado à rede por meio de um link. Caso contrário, a conexão entre os átomos é criada e o processo segue em frente. A partir daí, novos conhecimentos são checados e as novas triplas adquiridas são convertidas em átomos e links que serão inseridos na rede. Em seguida, as estruturas (Box, List, Gallery, entre outras) são construídas e, por fim, exibidas com suporte à apresentação e navegação.

5.7 CONSIDERAÇÕES FINAIS

O modelo de apresentação e navegação de *Linked Data*, proposto nesta dissertação, foi concebido como uma extensão da camada de execução do modelo Dexter de referência de hipertexto. Ou seja, o trabalho desenvolvido aqui aproveita as peças e conceitos definidos na camada de execução Dexter para elaborar um modelo preocupado, especificamente, com o tratamento das questões envolvendo a apresentação e navegação de *Linked Data* para o usuário comum. O modelo proposto ainda se beneficia das estruturas definidas na camada de armazenamento Dexter, que descreve a rede de nós e *links* utilizada no hipertexto.

A escolha do modelo Dexter foi baseada em um conjunto de pontos determinados durante a fase de análise dos modelos de referência de hipertexto. Esses pontos descrevem as principais necessidades requeridas para a construção do modelo de apresentação e navegação de *Linked Data*, objeto desta dissertação. Os modelos avaliados foram confrontados segundo uma lista requisitos. O modelo Dexter obteve a maior pontuação, considerando tais necessidades.

O modelo de apresentação e navegação de *Linked Data*, proposto aqui, foi projetado com o propósito de ser um modelo genérico. O objetivo é que, a partir da referência deste trabalho, seja possível implementar o modelo de maneira simples e facilitada. Além disso, a proposta foi elaborada com a preocupação de permitir que o modelo possa ser adaptado e aperfeiçoado de forma prática e flexível.

Capítulo 6

Este capítulo apresenta a implementação do modelo proposto, realizada por meio do desenvolvimento de um Serviço Web, que disponibiliza as operações suportadas pelo mesmo. Para utilizar o serviço de forma mais fácil e eficiente, foi criada uma biblioteca (script), como parte da implementação, para ser utilizada no lado cliente das aplicações. Com o objetivo de demonstrar a utilização destas peças e, consequentemente, a viabilidade da proposta, foi desenvolvido um protótipo – como uma extensão de um navegador Web – aderido ao modelo proposto, utilizando unicamente o Serviço Web por meio da biblioteca cliente. O capítulo está organizado da seguinte forma: na Seção 6.1, a descrição do Serviço Web; na Seção 6.2, a apresentação da biblioteca de comunicação com o serviço; na Seção 6.3, o detalhamento do protótipo de demonstração; e na Seção 6.4, as considerações finais.

6 IMPLEMENTAÇÃO DO MODELO

Como dito anteriormente, a proposta deste trabalho consiste na definição de um modelo de apresentação e navegação de/em dados estruturados obedecendo aos princípios *Linked Data*, centrado no usuário comum. A implementação do modelo proposto foi projetada em duas etapas. A primeira etapa correspondeu ao desenvolvimento de um Serviço Web, implementando as quatro primeiras camadas do modelo de apresentação e navegação de *Linked Data*. Na segunda etapa, foi desenvolvida uma biblioteca Javascript como interface de comunicação com o serviço. Além de dar suporte a comunicação com o Serviço Web, a biblioteca Javascript, criada como peça desta proposta, implementa a última camada do modelo desenvolvido aqui, a camada de apresentação e navegação.

A demonstração dessas peças (Serviço Web e biblioteca Javascript) foi efetivada pela utilização de um protótipo, desenvolvido como uma extensão para navegador Web. O protótipo foi projetado para detectar e extrair *Linked Data* implícito em páginas Web por meio de marcações RDFa, acessadas por intermédio do navegador. Com o desenvolvimento do protótipo, foi possível avaliar a viabilidade do modelo de apresentação e navegação de *Linked Data* através da utilização do serviço e da biblioteca.

6.1 SERVIÇO WEB

Como parte da proposta deste trabalho, foi desenvolvido um serviço Web seguindo os princípios REST (*REpresentational State Transfer*) (Fielding, 2012). O serviço foi implementado usando a linguagem de programação Java[15] e os *frameworks* Jersey[16] e Jaxb[17]. Jersey é a implementação de referência para a especificação Java JAX-RS. Ela contém basicamente um servidor e um cliente REST. O cliente pode ser usado para prover uma biblioteca de comunicação com o servidor. Jaxb (*Java Architecture for XML Binding*) é um padrão que define como objetos Java são convertidos de/para XML. Como Jaxb é definido por meio de especificação, é possível usar diferentes implementações para esse padrão.

O serviço desenvolvido neste trabalho, de modo geral, consiste de três camadas principais, como é possível observar na Figura 20: (i) a Camada de Requisição, que é uma interface REST, ou seja, uma classe Java contendo anotações Jersey e Jaxb para definir o tipo de requisição (POST, PUT, DELETE, entre outros), a URL de acesso aos recursos, o caminho de acesso, o tipo de resposta (XML, JASON, TEXT, entre outros) e outros; (ii) a Camada de Comunicação, que faz a comunicação entre a camada de requisição e a camada de processamento, por meio de uma interface contendo métodos específicos; e (iii) a Camada de Processamento, que executa o processo, correspondente às camadas do modelo proposto, para responder aos pedidos da camada de requisição. As camadas de comunicação e processamento juntas formam o núcleo do sistema, que também pode ser denominado de API. A Figura 20 mostra o esquema com as três principais camadas do serviço.

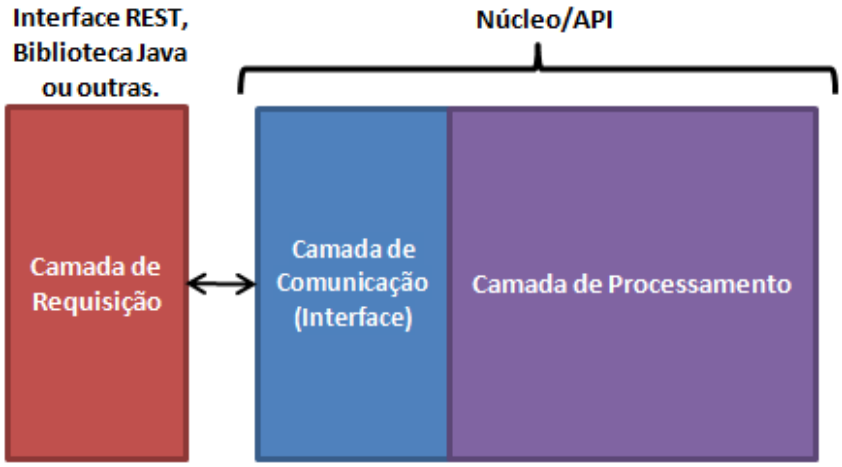

Figura 20 – Arquitetura básica do serviço Web.

[15] http://www.oracle.com/technetwork/java/index.html

[16] https://jersey.java.net/

[17] https://jaxb.java.net/

É importante resaltar que por meio da camada de requisição a API pode ser usada de formas variadas. Neste trabalho, por exemplo, foi usada uma interface REST para criar um Serviço Web RESTful. É possível, no entanto, modificar a implementação da camada de requisição para criar um Serviço Web SOAP ou uma biblioteca Java, mantendo o núcleo do sistema consistente.

A API implementa as quatro camadas iniciais do modelo proposto – Camada de Filtragem Analítica, Camada de Mapeamento, Camada de Descoberta e Aconselhamento e Camada de Preparação de Interface. A quinta camada, Camada de Apresentação e Navegação, é implementada por meio da biblioteca cliente, descrita na Seção 6.2. A Figura 21 mostra uma parte da árvore de pacotes Java usados para organizar a implementação com base nas camadas definidas pelo modelo.

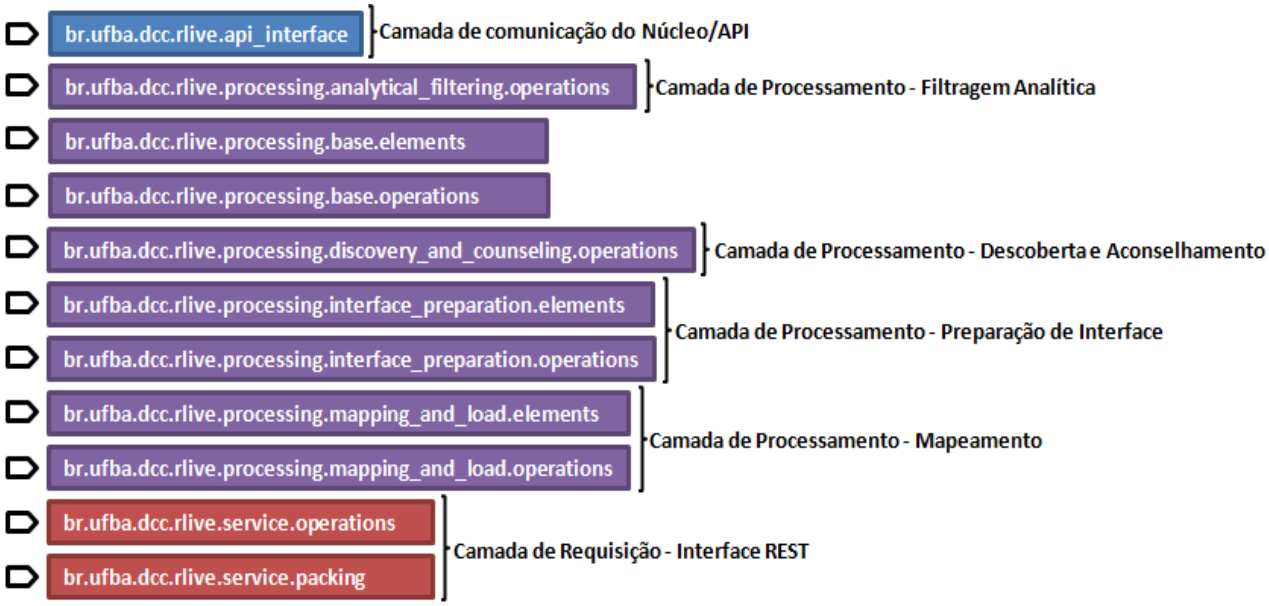

Figura 21 – Implementação das camadas do modelo por meio do serviço Web.

Tal como ilustrado na Figura 21, algumas camadas foram divididas em duas partes, elementos e operações, para organizar melhor o código. Por exemplo, a camada de mapeamento possui alguns elementos como átomos, links, especificadores, entre outros, e operações que são realizadas sobre esses elementos, como *createAtom()*, *createLink()*, *getAtom()*, *netContains()*, *connectsAtoms()*, entre outras.

A API fornece uma interface (camada de comunicação) para disponibilizar os métodos específicos que correspondem às operações e suboperações definidas pelo modelo (Seção 5.6). Além disso, a API também oferece, por meio da interface, métodos variados para apoiar as necessidades do desenvolvimento de apresentações com suporte à navegação, voltadas para o usuário comum. A interface está contida no pacote *api_interface*, mostrado na primeira linha da Figura 21. Na Figura 22 é possível observar um trecho de código contendo alguns dos métodos disponibilizados pela interface da API.

```
21 public RLTriples filterRDFDescription(RLTriples rlTriples) throws Exception{
22     rlTriples = Removal.removeMetatriples(rlTriples);
23     rlTriples = Removal.removeRedundantTriples(rlTriples);
24     return rlTriples;
25 }
26
27 public RLNetwork mapRDF2Dexter(RLTriples rlTriples) throws Exception{
28     return Mounting.buildRLNet(rlTriples);
29 }
30
31 public boolean isResource(String recUri, String predUri){
32     if(isWellFormedURI(recUri)){
33         return BasicOperation.isLDResourceURI(recUri, predUri);
34     }
35     return false;
36 }
37
38 public PNBox createBox(RLAtom atom, RLNetwork rlNet) throws Exception{
39     return Making.makeBox(atom, rlNet);
40 }
41
42 public PNOutList createOutList(PNBox box, String pred) throws Exception{
43     return Making.makeOutList(box, pred);
44 }
45
46 public String getAtomLabel(String uri) throws Exception{
47     if(isWellFormedURI(uri)){
48         return Labelling.defineAtomLabel(uri);
49     }
50     return null;
51 }
52
53 public String getLinkLabel(String uri) throws Exception{
54     if(isWellFormedURI(uri)){
55         return Labelling.defineLinkLabel(uri);
56     }
57     return null;
58 }
```

Figura 22 – Métodos disponibilizados pela camada de interface da API.

Nas linhas 21 e 27, da Figura 22, pode-se observar os métodos *filterRDFDescription()* e *mapRDF2Dexter()*, respectivamente, que correspondem as duas primeiras operações elementares definidas pelo modelo e descritas na Seção 5.6. Nas linhas 31, 38 e 42, da Figura 22, é possível ver alguns métodos correspondentes às suboperações que podem ser usadas pelas operações principais: *isResource()*, *createBox()* e *createOutList()*, também referenciadas na Seção 5.6. Por fim, nas linhas 46 e 53 são mostrados alguns dos métodos variados que podem ser utilizados para suprir necessidades da construção de interfaces de apresentação e navegação: *getAtomLabel()* e *getLinkLabel()*. Esses métodos, em especial, retornam um *label*, referente a um determinado átomo ou link, que pode ser usado na apresentação de tal elemento para o usuário final.

Como explicado anteriormente, a interface da API pode ser vista como uma camada de comunicação com o processamento. Portanto, é possível construir sistemas diversos a partir da utilização adequada dessa camada. Por exemplo, no contexto deste trabalho, utilizou-se a camada de comunicação para criar um Serviço Web, por meio de uma interface REST, alocada na camada de requisição (Figura 20). A interface REST utiliza os métodos disponibilizados pela camada de comunicação que, por sua vez, se

comunica com a camada de processamento para responder as requisições desta interface. Logo, é possível criar, por exemplo, uma biblioteca Java (.jar) para prover, localmente (independente da utilização de Serviços Web), suporte ao modelo proposto neste trabalho. Nesse caso, o sistema se comunica diretamente com a camada de comunicação da API, por meio de uma interface própria alocada na camada de requisição. Isso cria uma versatilidade do ponto de vista do desenvolvimento. Ou seja, mantendo o núcleo do sistema consistente, é possível dar suporte ao modelo proposto por meio de arquiteturas variadas como Serviço Web RESTful, Serviço Web SOAP/WSDL, bibliotecas Java, entre outros. A Figura 23 mostra, como exemplo, um trecho de código contendo o método *getBox()*, definido na interface REST construída neste trabalho. Esta interface reside na camada de requisição da arquitetura e utiliza a camada de comunicação (interface da API), conforme Figura 20.

```
171 @POST
172 @Path("getBox")
173 @Consumes({ MediaType.APPLICATION_JSON })
174 @Produces({ MediaType.APPLICATION_JSON })
175 public PacOutBox getBoxPOST(JAXBElement<PacInTNetAtom> piTnetAtom) {
176     PacInTNetAtom mix = piTnetAtom.getValue();
177     PacOutBox poBox = new PacOutBox();
178 
179     try{
180         PresentNavi pn = new PresentNavi();
181         RLNetwork rlNet = pn.convertTNet2RLNet(mix.gettNet());
182         RLAtom atom = pn.getAtom(mix.getAtom().getAtomUID(), rlNet);
183 
184         if(atom == null){
185             return null;
186         }
187 
188         PNBox box = pn.createBox(atom, rlNet);
189         poBox.setBox(box); poBox.setResponseOk(true); poBox.setInfo("Response ok");
190     }
191     catch(Exception e){
192         e.printStackTrace();
193         poBox.setBox(null); poBox.setResponseOk(false); poBox.setInfo(e.toString());
194     }
195     return poBox;
196 }
```

Figura 23 – Camada de requisição – Método *getBox()* da interface REST.

Na linha 171, uma anotação define o tipo de requisição que pode ser realizada para acessar este método, neste caso POST. Na linha 172, o caminho de acesso ao método é definido por outra anotação (<raiz>/getBox). A linha 173 indica que o método consome dados em formato JSON. Da mesma forma, a linha 174 indica que o método produz dados também no formato JSON. Na linha 175, é possível observar que o método retorna um pacote de saída contendo uma estrutura Box (Seção 5.4) e recebe como parâmetro um pacote de entrada contendo uma rede e um átomo Dexter. Na linha 180, a interface da API (camada de comunicação) é instanciada. Na linha 188, o método *createBox()* da interface da API é invocado. Na linha 189, o pacote de saída contendo o Box, e outras informações, é criado. E, finalmente, na linha 195, o pacote é retornado como resposta ao cliente.

6.2 BIBLIOTECA

De acordo com o que foi visto na Seção 6.1, o Serviço Web, desenvolvido como parte da proposta, implementa todas as camadas do modelo proposto, com exceção da camada de apresentação e navegação. Para implementar a camada de apresentação e navegação foi desenvolvida uma biblioteca Javascript que funciona como um *plugin* do *framework* JQuery, para atuar no lado cliente das aplicações. O objetivo principal da biblioteca é prover a comunicação com o Serviço Web. Essa comunicação consiste de requisições do tipo GET, PUT ou POST, executadas sobre a interface REST do serviço, usando as URLs que representam os métodos definidos nessa interface. As respostas às requisições, na maioria das vezes, consistem em estruturas especializadas construídas pela camada de preparação de interface, idealizada no modelo proposto e implementada pelo Serviço Web. No entanto, para executar a maioria das requisições é comum enviar dados para subsidiar o processamento desses pedidos. Por exemplo, para dar início ao processo e solicitar uma estrutura General Presentation (Seção 5.4), é necessário enviar a descrição RDF (grafo ou triplas) na requisição.

Portanto, a biblioteca foi desenvolvida, de forma especializada, para extrair triplas RDF de páginas HTML contendo dados estruturados *Linked Data*, embutidos por meio de marcações RDFa. Ou seja, a biblioteca Javascript concebida nesse trabalho é capaz de extrair todos os conjuntos de triplas RDF embutidas em qualquer página Web usando o padrão RDFa 1.1 (Adida et al., 2012), e, posteriormente, enviá-las nas requisições ao serviço. Para dar suporte a extração de triplas RDFa, a biblioteca utiliza a API Greenturtle[18], que é uma implementação de RDFa API (Rixham et al., 2012). Greenturtle estende o DOM HTML[19] para incluir a RDFa API. Ele também disponibiliza um processador RDFa 1.1 para processar quaisquer documentos auxiliares na coleta de triplas. A Figura 24 apresenta a arquitetura básica da biblioteca Javascript desenvolvida como parte da proposta.

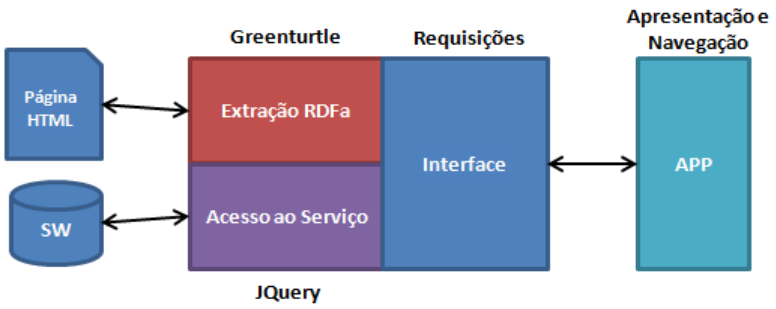

Figura 24 – Arquitetura básica da biblioteca.

[18] https://code.google.com/p/green-turtle/

[19] Document Object Model HTML

Quando a aplicação cliente, que deseja dar suporte a apresentação e navegação de *Linked Data*, adiciona a biblioteca Javascript em seu diretório, a extração de triplas RDFa pode ser iniciada, automaticamente, dependendo das configurações (Greenturtle, Figura 24), no momento do carregamento das páginas Web exibidas para o usuário. Após a extração (Figura 28), um objeto Javascript contendo o conjunto de triplas fica disponível no sistema, para uso pela biblioteca. A partir daí, a aplicação cliente pode realizar solicitações ao Serviço Web por meio da interface da biblioteca. A Camada de Interface (Requisições, Figura 24) usa a Camada de Acesso ao Serviço para executar as requisições (JQuery) (Figura 24). A Figura 25 mostra um fragmento de código contendo uma das funções de requisição contidas na camada de acesso ao serviço.

```
26 function RLApost(RLAmethod, RLAdata, RLApostCallback){
27     $.ajax({
28         type: 'POST',
29         url: RLAserviceBaseURL + RLAmethod,
30         contentType: 'application/json; charset=UTF-8', // o tipo de dado enviado
31         dataType: "json", // o tipo de dado recebido
32         crossDomain: true,
33         data: RLAdata,
34         success: function(data, textStatus, jqXHR){
35                     console.log("RDFa Live " + RLAmethod+" "+textStatus);
36                     RLApostCallback(data, textStatus);
37                 },
38         error: function(jqXHR, textStatus, errorThrown){
39                     console.log("RDFa Live " + RLAmethod+" "+textStatus+" "+errorThrown);
40                     RLApostCallback(null, textStatus);
41                 }
42     });
43 }
```

Figura 25 – Camada de acesso ao serviço da biblioteca Javascript.

A função *RLApost()*, na Figura 25, recebe como parâmetros: o nome do método a ser executado no Serviço Web; o dado a ser enviado ao método; e uma função de *callback* (linha 26 da Figura 25). A requisição é do tipo POST (linha 28 da Figura 25), pois é necessário o envio de informações encapsuladas. Na linha 29 da Figura 25, é definido a URL de acesso, composta pela URL base mais o nome do método. Tanto o conteúdo (tipo de dado) enviado, quanto o conteúdo recebido estão no formato JSON (linhas 30 e 31 da Figura 25). Caso a requisição tenha sucesso, o dado recebido como resposta é retornado na função de *callback* (linhas 34 a 37 da Figura 25). Caso a requisição falhe, a função de *callback* retorna com nulo (linhas 38 a 41 da Figura 25). A Figura 26 mostra mais um fragmento de código exibindo algumas funções da camada de interface executando a função *RLApost()* da camada de acesso ao serviço, ilustrada na Figura 25.

```
21 function getBox(tNet, atom, cbFunction){
22     RLApost("getBox", JSON.stringify({tNet:tNet, atom:atom}),
23         function(data, textStatus){
24             cbFunction(data);
25         });
26 }
27
28 function getOutList(box, pred, cbFunction){
29     RLApost("getOutList", JSON.stringify({box:box, pred:pred}),
30         function(data, textStatus){
31             cbFunction(data);
32         });
33 }
34
35 function getGallery(atom, cbFunction){
36     RLApost("getGallery", JSON.stringify(atom),
37         function(data, textStatus){
38             cbFunction(data);
39         });
40 }
```

Figura 26 – Camada de interface da biblioteca Javascript.

As funções *getBox()* (linha 21 da Figura 26), *getOutList()* (linha 28 da Figura 26) e *getGallery()* (linha 35 da Figura 26) executam a função de acesso ao serviço *RLApost()* passando como parâmetro o nome do método a ser executado no serviço; o dado requerido pelo método; e a função de *callback*, como descrito anteriormente. No caso da função *getBox()*, a chamada a função *RLApost()* (linha 22 da Figura 26) passa como parâmetros "*getBox*", como nome do método a ser executado no serviço; um pacote de entrada contendo uma rede e um átomo Dexter; e uma função de *callback*. O dado retornado como resposta, nesse caso um pacote de saída contendo um Box, entre outros dados, é repassado para a função de *callback* enviada como parâmetro da função *getBox()*, *cbFunction* (linha 24 da Figura 26). As aplicações cliente têm acesso somente às funções da camada de interface da biblioteca. A Figura 27 mostra um trecho de código com a utilização de uma das funções da camada de interface da biblioteca por uma aplicação cliente.

```
38 getBox(tNet, atom,
39     function(data){
40         if(data != null){
41             if(data.responseOk === "true"){
42                 createRelatedResources("listResources", data.box);
43             }
44             else{
45                 console.log(data.info);
46                 info = "An error occurred: <br>"+data.info+"<br> Try again later... ";
47                 dialogMsg("dialog-message", info, "Error Message", null);
48             }
49         }
50         else{
51             info = "An error occurred during communication. Try again later... ";
52             dialogMsg("dialog-message", info, "Error Message", null);
53         }
54 });
```

Figura 27 – Utilização da camada de interface pelos clientes.

Na linha 38 da Figura 27, a aplicação cliente faz uma chamada a função *getBox()* na camada de interface, passando como parâmetros uma rede Dexter; um átomo Dexter; e uma função de *callback*. A função *getBox()*, por sua vez, executa a função *RLApost()* na camada de acesso ao serviço. A resposta do serviço é repassada, sucessivamente, até chegar à função de *callback* parametrizada pela aplicação cliente na chamada a *getBox()* (linha 39 da Figura 27). Dentro da função de *callback* o cliente verifica se o dado recebido é nulo (linha 40). Caso seja nulo, exibe uma mensagem de erro (linha 52 da Figura 27). Caso contrário, verifica se o dado contém a resposta "Ok", ou seja, se o dado é consistente. Caso não seja consistente, exibe uma mensagem de erro (linha 47 da Figura 27). Caso contrário, o cliente faz uma chamada a uma função de criação de interface, disponibilizada pela camada de interface da biblioteca, *createRelatedResources()*, passando o *id* de um elemento HTML, no qual a apresentação deve ser montada (por exemplo: * *); e o Box retornado pelo serviço. A função *createRelatedResources()* é utilizada pela operação *Presents Network*, disponibilizada pela camada de apresentação e navegação do modelo proposto. *createRelatedResources()* é usada por *presentsBox(box)*, suboperação de *Presents Network*. O resultado dessa chamada é a exibição de uma lista de recursos relacionados ao átomo referencial contido no Box, que pode ser vista na Figura 16 da Seção 5.5. Além da suboperação *presentsBox(box)*, a biblioteca da suporte as suboperações *presentsBasicInfo(box)*; *presentsList(list)*; *presentsGallery(gallery)*; *presentsNews(news)*; *presentsGeneral(generalPresentation)*; entre outras, definidas no modelo de apresentação e navegação de *Linked Data*.

```
5  RDFa.attach(document);
6  document.addEventListener('rdfa.loaded', function() {if(r2d2==0)startRLE(); r2d2++;}, false);
7
8  //variaveis globais
9  var listaSujeitosURI; // uma lista contendo todos as URIs de sujeitos da pagina
10 var listaSujeitosComp; // uma lista contendo todos os sujeito da pagina em formato de objeto javascript
11 var listaPredicados = [[]]; // lista com as URIs de todos os perdicados da pagina
12 var listaObjetos = [[]]; // lista com todos os objetos da pagina em formato de objeto javascript
13 var listaArrayTriplas = []; // uma lista com todas as triplas da pagina em formato de array de strings
14 var rlTriplesList = []; // uma lista com todas as triplas da pagina em formato de objeto javascript
15
16
17 //usa a API do green turtle para iniciar a identificacao e recuperacao de triplas
18 function startRLE(){
19     listaSujeitosURI = document.data.getSubjects();
20     if(listaSujeitosURI.length > 0){
21         console.log("RDFa Live on fire");
22         listaSujeitosComp = new Array(listaSujeitosURI.length);
23         listaPredicados = new Array();
24         listaObjetos = new Array();
25         listaArrayTriplas = new Array();
26         rlTriplesList = new Array();
27         getTriples();
28     }
29     else{
30         console.log("RDFa Live found no triples");
31     }
32 }
```

Figura 28 – Processo de extração de triplas RDFa.

A Figura 28 mostra um fragmento de código contendo uma parte do processo de extração de triplas RDFa contidas em páginas HTML. Nas linhas 5 e 6 da Figura 28, a API Greenturtle aguarda o carregamento total do DOM HTML das páginas, para, só então, executar a função *startRLE()* (linha 18 da Figura 28). Nas linhas de 9 a 14 da Figura 28, são definidas variáveis globais que armazenam as informações necessárias para executar o processo de extração. A linha 14 da Figura 28 contém a variável que vai guardar o conjunto de triplas das páginas HTML em formato de objeto Javascript, *rlTriplesList*. Esta variável vai ser usada pela camada de interface da biblioteca para realizar requisições ao serviço Web. Nas linhas 18 a 32 da Figura 28 é definida a função *startRLE()*, que usa chamadas a API Greenturtle para colher as triplas contidas nas páginas. Nas linhas 19 e 20 da Figura 28 é realizada uma verificação para determinar a existência de marcações RDFa na página. Caso não existam tais marcações, o processo termina. Na linha 27 da Figura 28 a função *getTriples()* é invocada dentro de *startRLE()*. Nesse ponto a variável *rlTriplesList* recebe as triplas da página Web sendo carregada para o usuário.

O suporte a extração de triplas RDFa foi desenvolvido para atender as necessidades do protótipo (descrito na Seção 6.3). No entanto, é possível fazer uso da biblioteca sem explorar o recurso de extração. Portanto, aplicações cliente que desejam se beneficiar das interfaces criadas dinamicamente pela biblioteca, não são obrigadas a executar essa tarefa apenas por meio de *Linked Data* embutido em páginas Web visitadas por seus usuários. A biblioteca pode receber *Linked Data* (grafo RDF ou conjunto de triplas RDF) oriundos de qualquer fonte e enviá-los ao serviço para requisitar estruturas especializadas na apresentação e navegação de tais dados. Ou seja, a ativação, ou não, do modelo de extração de RDFa, contido na arquitetura da biblioteca, fica a cargo das aplicações clientes.

6.3 PROTÓTIPO

O protótipo foi desenvolvido com o objetivo de demonstrar a utilização tanto do Serviço Web, quanto da biblioteca Javascript (descritos nas Seções 6.1 e 6.2), e, consequentemente, a viabilidade do modelo de apresentação e navegação de *Linked Data* proposto neste trabalho. O protótipo foi concebido como uma extensão para o navegador Web Google Chrome[20]. Esse navegador foi escolhido por apresentar uma vasta documentação acerca do desenvolvimento de extensões. Outro navegador que apresenta uma boa documentação sobre o assunto é o navegador Mozilla Firefox[21]. É possível desenvolver uma extensão para

[20] http://developer.chrome.com/extensions/extension.html

[21] https://developer.mozilla.org/en-US/Add-ons

esse navegador Web de maneira semelhante à forma como a extensão para o navegador Chrome foi desenvolvida. A Figura 29 mostra a extensão, batizada de "RDFa Live Extension", listada na página de gerenciamento de extensões do navegador Chrome.

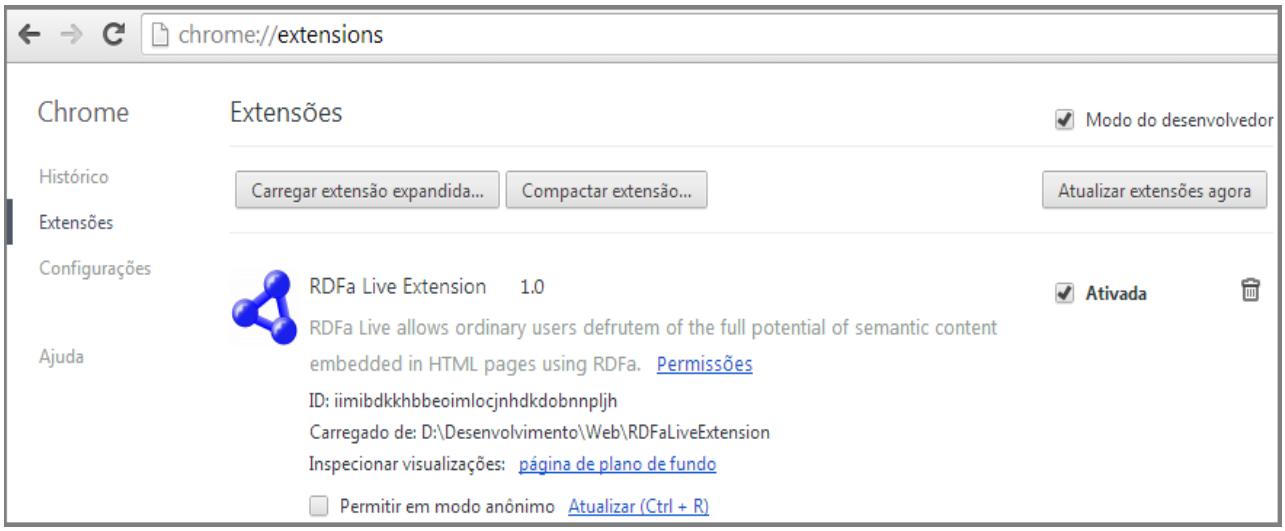

Figura 29 – Extensão carregada e ativada no navegador Chrome.

Uma extensão pode ser adicionada ao navegador Chrome de duas formas: localmente e online. Para carregar uma extensão em modo online, é necessário que a mesma esteja disponível na loja de aplicativos do Google Chrome (Google Chrome Web Store[22]). Dessa forma, basta efetuar o download da extensão, e ela será instalada automaticamente. Como a extensão RDFa Live não está disponível na loja online do navegador Chrome, é preciso adicioná-la de forma local. Para isso, basta clicar no botão "Carregar Extensão Expandida" na página de gerenciamento *chrome://extensions* (Figura 29) e selecionar a extensão a partir da árvore de diretórios local. Com a extensão carregada e ativada no navegador, é possível detectar marcações RDFa contidas em páginas da Web, visitadas pelo usuário, por meio da camada de extração RDFa da biblioteca Javascript (Seção 6.2). Ou seja, a extensão é capaz de identificar qualquer página Web contendo marcações RDFa. Além disso, ela pode sinalizar essa característica especial, para o usuário que navega na página, por meio de um ícone posicionado no canto direito da barra de endereços do navegador, como pode ser visto na Figura 30.

[22] https://chrome.google.com/webstore/category/apps?hl=pt-BR

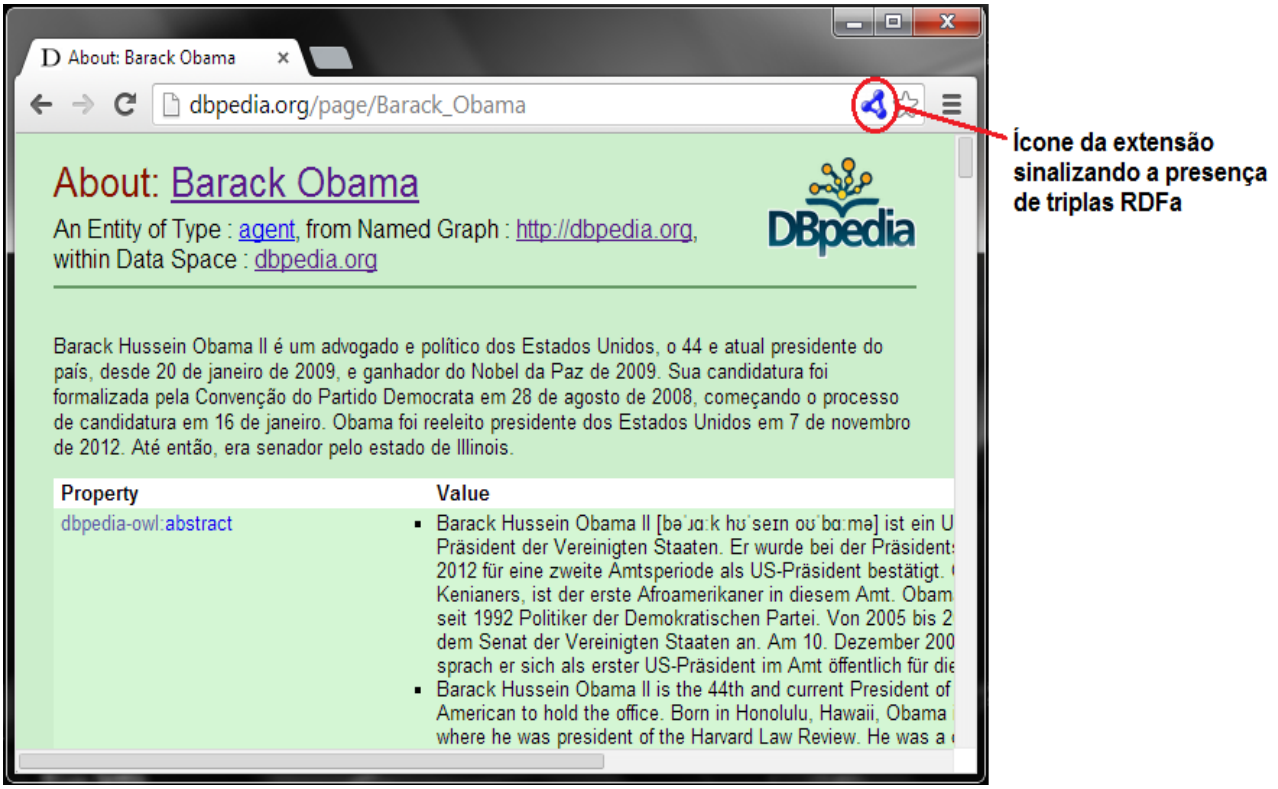

Figura 30 – Ícone sinalizando a existência de triplas RDFa na página Web.

A Figura 30 mostra a descrição de um recurso *Linked Data* (*http://dbpedia.org/resource/Barack_Obama*) alocado na fonte de dados DBPedia. A descrição está em formato HTML, formato adequado para usuários humanos. No entanto, é possível observar, pela presença do ícone da extensão RDFa Live, que existem marcações RDFa na página Web em questão. Ou seja, existem triplas RDF embutidas no código HTML da página, por meio do padrão RDFa, contendo dados relacionados ao recurso apresentado. Diante disso, o usuário pode clicar sobre o ícone para obter acesso a essas informações semânticas, implícitas na página, de forma adequada a sua condição de usuário não técnico. Clicando sobre o ícone, uma nova aba é criada na barra de abas do navegador. A nova aba gera uma apresentação, com suporte a navegação, baseada nos dados obtidos na extração das triplas RDFa da página visitada.

A Figura 31 mostra a aba gerada pela extensão, após o clique no ícone, destacando cada construção (numeradas de 1 a 11) criada pela biblioteca Javascript, por meio da interação com o Serviço Web. É importante observar que a apresentação gerada pelo protótipo está focada no recurso "Barack Obama". Isso ocorre porque durante o processamento dos dados existe uma eleição para determinar o recurso (átomo Dexter) mais relevante da descrição RDF recebida como entrada (Seção 5.3). Outro fato que chama a atenção, é que a apresentação é criada levando-se em conta o tipo do recurso a ser explorado. No caso exemplificado pela Figura 31, o recurso é do tipo "Pessoa". Portanto, as construções criadas pela extensão seguirão o padrão de apresentação para esse tipo de entidade. Caso o recurso fosse do tipo "Lugar",

por exemplo, a apresentação seguiria esse padrão para criar construções contendo informações adequadas a esse tipo de entidade.

Figura 31 – Apresentação e navegação de *Linked Data* fornecida pela extensão.

A construção 1, da Figura 31, mostra informações básicas acerca do recurso explorado (nesse caso, o recurso Barack Obama), contendo o *label*, data de nascimento, nome completo e ocupação do recurso. Essa construção é proveniente de uma requisição a um átomo lançada contra o Serviço Web pela biblioteca Javascript. As informações estão contidas no interior do átomo, como triplas literais. É possível também obter essas informações por meio de uma requisição a uma estrutura Box, configurado com o recurso explorado como seu átomo referencial. A construção 2, consiste de *links* (URLs) relacionadas ao recurso explorado. Essas informações também podem ser obtidas por meio de requisições tanto a um átomo, quanto a um Box, pois esses dados são, invariavelmente, do tipo literal, e, portanto, contidos no interior dos átomos.

A construção 4 exibe uma galeria de mídias, que pode conter fotos, vídeos, *streaming* de TV, áudio, entre outros. Para criar essa construção é necessária uma requisição a uma estrutura Gallery. A estrutura Gallery é confeccionada pela camada de preparação de interface a partir de consultas a APIs da Web 2.0, realizadas pela camada de descoberta e aconselhamento. Clicando sobre qualquer imagem diminuta da galeria, uma janela central se abre para exibir as mídias mais adequadamente em tamanho apropriado. O

usuário pode navegar entre as mídias sequencialmente (uma após a outra), seguindo a lista de mídias, ou diretamente (indo para a mídia desejada), clicando sobre uma determinada mídia. A Figura 32 mostra a galeria criada pelo protótipo.

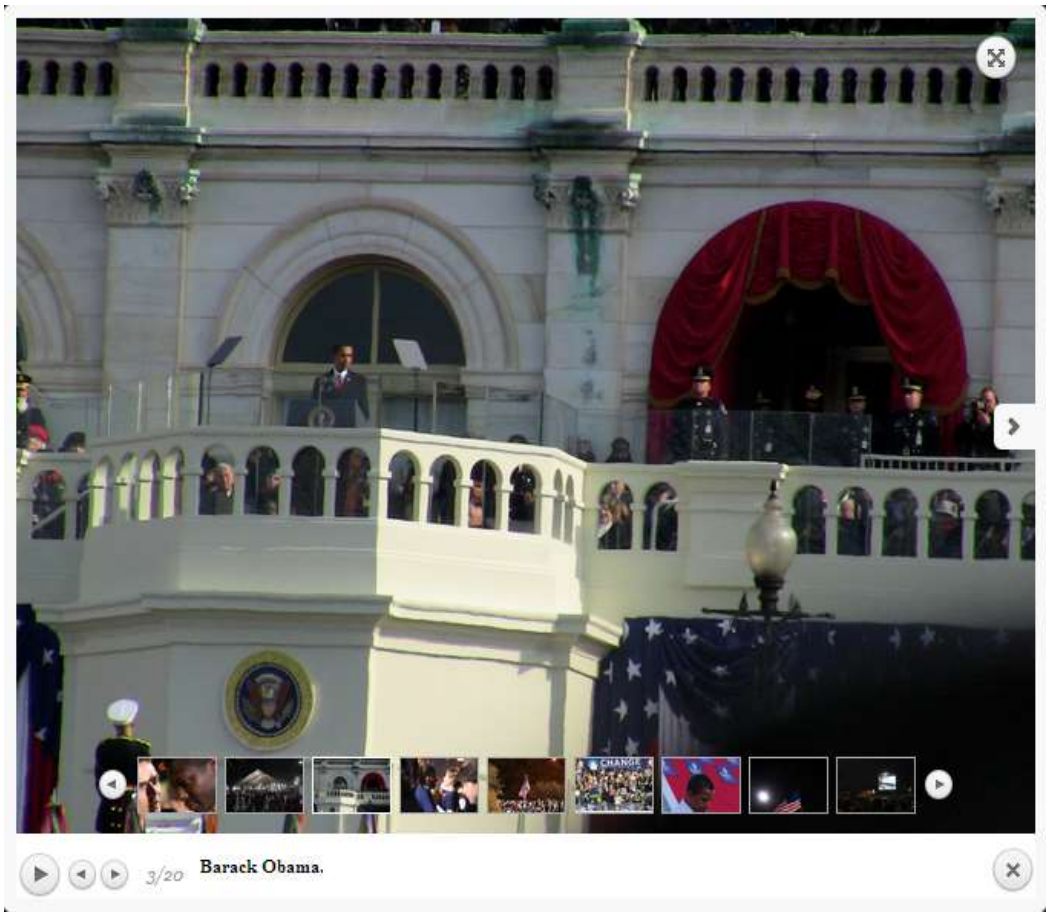

Figura 32 – Construção da galeria de mídia por meio da estrutura Gallery.

A construção 6 exibe um texto contendo um resumo sobre o recurso explorado. Além do texto, uma figura relacionada ao recurso é mostrada na parte esquerda superior da construção. Assim como as construções 1 e 2, a construção 6 pode ser obtida a partir de requisições tanto a um átomo, quanto a um Box, pois o texto do resumo e a URL da figura são informações contidas no interior dos átomos do tipo pessoa. A construção 7 é uma lista de recursos relacionados ao recurso explorado. A lista tem suporte a navegação facetada. O filtro usado nas facetas é o tipo de relacionamento existente entre o recurso explorado e os demais recursos. Esta construção só pode ser criada a partir de uma estrutura Box, pois apenas o Box contém todos os recursos ligados diretamente ao recurso explorado (átomo referencial). A lista de recursos também prove suporte a apresentação por *tooltips*. Passando o ponteiro do mouse sobre os recursos, um *tooltip*, contendo informações relacionadas, é exibido. A Figura 33 mostra um *tooltip* exibindo informações acerca do recurso "Joe Biden" (vice-presidente dos EUA), contido na construção 7.

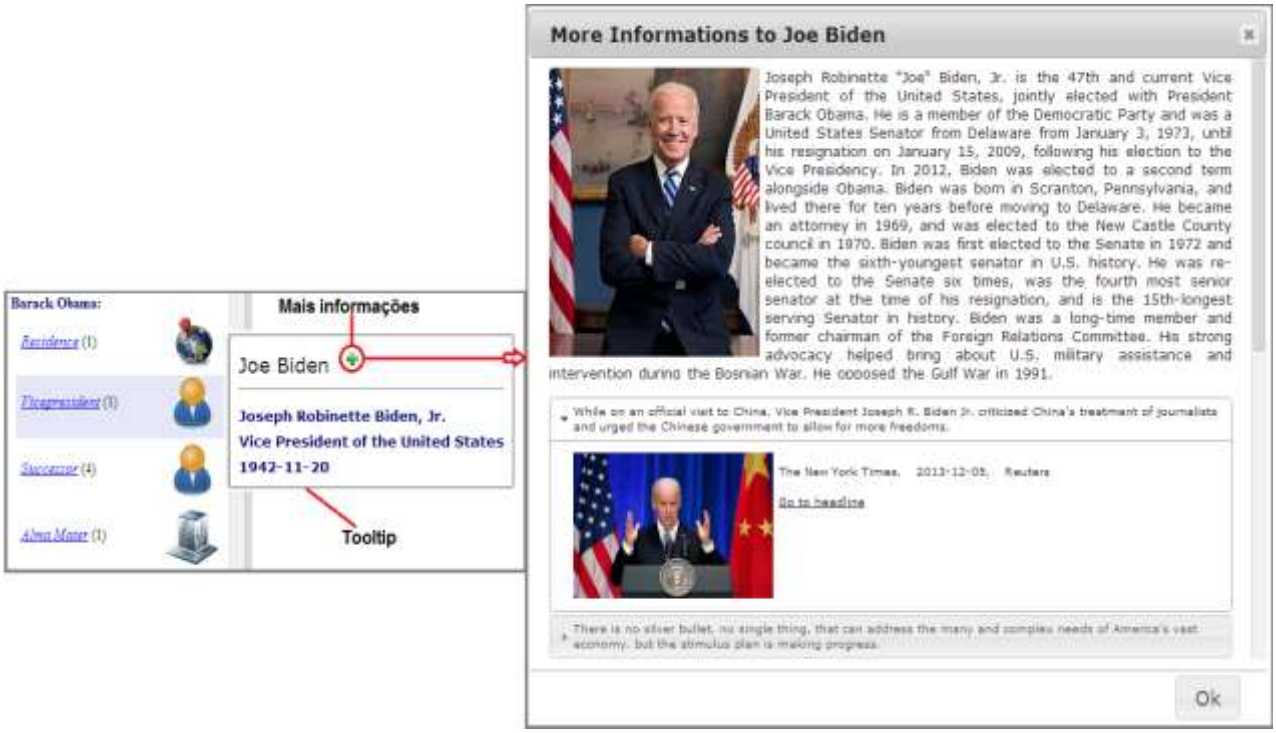

Figura 33 – Apresentação por *tooltips* na lista de recursos relacionados.

O *tooltip* apresenta informações básicas sobre os recursos. Para obter mais informações relevantes o usuário pode clicar sobre o sinal de "mais" ao lado do *label* do recurso. Essa ação faz com que o protótipo exiba uma janela contendo mais informações acerca do recurso pesquisado. Na Figura 33, o clique no sinal "mais" no *tooltip* exibe a janela de mais informações para o recurso "Joe Biden". A construção 8 apresenta notícias acerca do recurso explorado por meio de uma exibição por *accordion*. Essa construção é proveniente de uma estrutura News, obtida por meio do acesso a APIs da Web 2.0 de fontes como New York Times, BBC, CNN, entre outros. A construção 9 é um combinado para exibir listas de recursos relacionados ao recurso explorado. Para usar o mecanismo, o usuário pode arrastar qualquer tipo de relacionamento, contido na construção 7, e soltá-lo no campo *input* da construção 9. Por exemplo, para o recurso "Barack Obama" (ilustrado na Figura 31), o usuário pode arrastar o tipo relacionamento "Alma Master" da construção 7 e soltá-lo no campo de entrada da construção 9. Essa ação vai fazer com que a extensão exiba uma janela contendo os recursos ligados a Barack Obama por meio do relacionamento Alma Master, como pode ser visto na Figura 34.

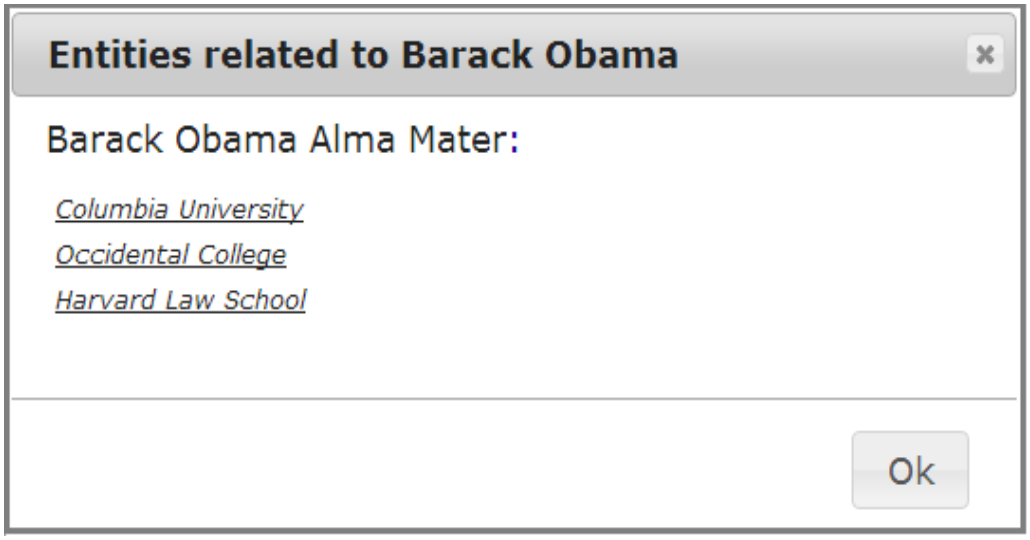

Figura 34 – Exibição de recursos por tipo de relacionamento.

A construção 9 pode ser obtida por requisições às estruturas de lista (List, outList e inList). Isso porque essas estruturas armazenam os recursos relacionados ao recurso explorado (átomo referencial) levando em conta um determinado tipo de relacionamento. A construção 10 apresenta uma lista de recursos semelhantes ao recurso explorado. Os recursos contidos nessa lista não são, necessariamente, ligados ao recurso explorado. Ou seja, tais recursos podem não ter nenhuma relação com o recurso explorado. A semelhança é determinada pelo tipo dos recursos. Por exemplo, o recurso "Barack Obama" possui vários tipos, dentre eles, "American Nobel Laureates". Assim, é possível listar recursos que também sejam do tipo American Nobel Laureates, como Bruce Beutler, Craig Mello e Frank Wilczek. Esta construção pode ser obtida por meio de uma estrutura inList. No entanto, existe uma particularidade: nesse caso o átomo referencial não é aquele que representa o recurso explorado e sim o tipo desejado para os recursos. Ou seja, no exemplo dado acima, o átomo referencial da inList é American Nobel Laureates e o tipo de relacionamento usado como filtro da lista é Type. Logo, é possível identificar as triplas: <Barack Obama><Type><American Nobel Laureates>; <Bruce Beutler><Type><American Nobel Laureates>; <Craig Mello><Type><American Nobel Laureates> e <Frank Wilczek><Type><American Nobel Laureates>.

As construções 3, 5 e 11, podem ser clicadas pelos usuários para apresentar mais informações acerca das construções 2, 4 e 10, respectivamente, por meio de janelas interativas. Por exemplo, clicando na construção 3, uma janela é exibida para apresentar mais *links* relacionados, além daqueles contidos na construção 2. Clicando na construção 5, uma nova galeria de mídias é exibida, contendo mais mídias relacionadas, além daquelas exibidas na construção 4. E, da mesma forma, clicando na construção 11, novos recursos são listados para novos tipos, além daqueles mostrados na construção 10.

6.4 CONSIDERAÇÕES FINAIS

Para demonstrar que o modelo de apresentação e navegação proposto no Capítulo 5 pode ser implementado, um Serviço Web e uma biblioteca Javascript foram desenvolvidos com essa finalidade. Cada uma destas peças implementa parte do modelo. O Serviço Web implementa as 4 primeiras camadas (Camada de Filtragem Analítica, Camada de Mapeamento, Camada de Descoberta e Aconselhamento e Camada de Preparação de Interface). Por sua vez, a biblioteca implementa a última camada, Camada de Apresentação e Navegação.

Por meio do Serviço Web é possível ter acesso a estruturas especializadas, próprias para serem utilizadas na construção de interfaces de usuário centradas na apresentação e navegação de *Linked Data*. Portanto, pode-se fazer uso do serviço diretamente, sem a necessidade de usar a biblioteca Javascript para realizar essa tarefa. A partir das estruturas fornecidas pelo serviço, é possível construir formas variadas de apresentação com suporte à navegação, focadas no usuário comum.

A biblioteca Javascript foi desenvolvida para demonstrar a utilização do serviço. Por meio da biblioteca é possível se comunicar com o serviço de maneira simples. A biblioteca é capaz de extrair *Linked Data* embutido em páginas Web por meio de marcações RDFa. Esses dados podem ser enviados para o serviço, que retorna as estruturas especializadas correspondentes. De posse de tais estruturas, a biblioteca cria interfaces de apresentação com suporte à navegação.

Para demonstrar a utilização destas peças e, consequentemente, a viabilidade do modelo proposto, foi desenvolvido um protótipo usando unicamente a biblioteca Javascript, que por sua vez, utiliza o Serviço Web (ambos implementando o modelo). O protótipo permite que o usuário tenha acesso a *Linked Data* (apresentação e navegação) implícito em páginas da Web por meio de uma extensão do navegador Google Chrome.

Capítulo 7

Este capítulo apresenta uma revisão literária dos principais trabalhos que norteiam as áreas de apresentação e navegação, publicação e autoria de Linked Data, entre outras. Os resultados do levantamento bibliográfico realizado são sumarizados, ao mesmo tempo em que uma avaliação parcial das soluções apresentadas é discutida.

7 TRABALHOS RELACIONADOS

Os trabalhos apresentados neste capítulo fazem parte de um conjunto mais amplo de estudos, levantados no decurso da fase de revisão bibliográfica e durante o decorrer de toda a pesquisa. O subconjunto apresentado aqui corresponde às pesquisas que se relacionam, mais diretamente, com o problema levantado neste estudo. Embora, muitos trabalhos envolvendo publicação, consumo e autoria de dados estruturados *Linked Data*, com foco no usuário comum, tenham sido descritos neste capítulo, o foco do estudo crítico é na parte da apresentação e navegação, por se tratar da área de atuação da pesquisa. Portanto, os trabalhos correspondentes às outras áreas serão sucintamente abordados.

7.1 APRESENTAÇÃO E NAVEGAÇÃO

O projeto LodLive (Camarda et al., 2012) fornece uma demonstração do uso dos padrões que acompanham os princípios de *Linked Data* (RDF, SPARQL) para navegar em recursos RDF. A ferramenta tem o objetivo de divulgar e disseminar os princípios de *Linked Data* com uma interface simples e amigável, além de técnicas reutilizáveis. LodLive foi pensado e projetado para: (1) navegar em recursos RDF usando uma visualização em grafo dinâmico; (2) interligar recursos armazenados em diferentes *endpoints*, descobrindo relacionamentos inesperados; (3) navegar em relações inversas, mesmo entre *endpoints*

diferentes; (4) divulgar e disseminar os princípios e padrões de *Linked Data* para usuários sem experiência com as técnicas semânticas da Web de Dados; (5) oferecer uma ferramenta pronta para uso (*out of the box*) de fácil manipulação, que qualquer um seja capaz de explorar; (6) coletar e mostrar imagens relacionadas em recursos navegados; e (7) mostrar recursos navegados em um mapa.

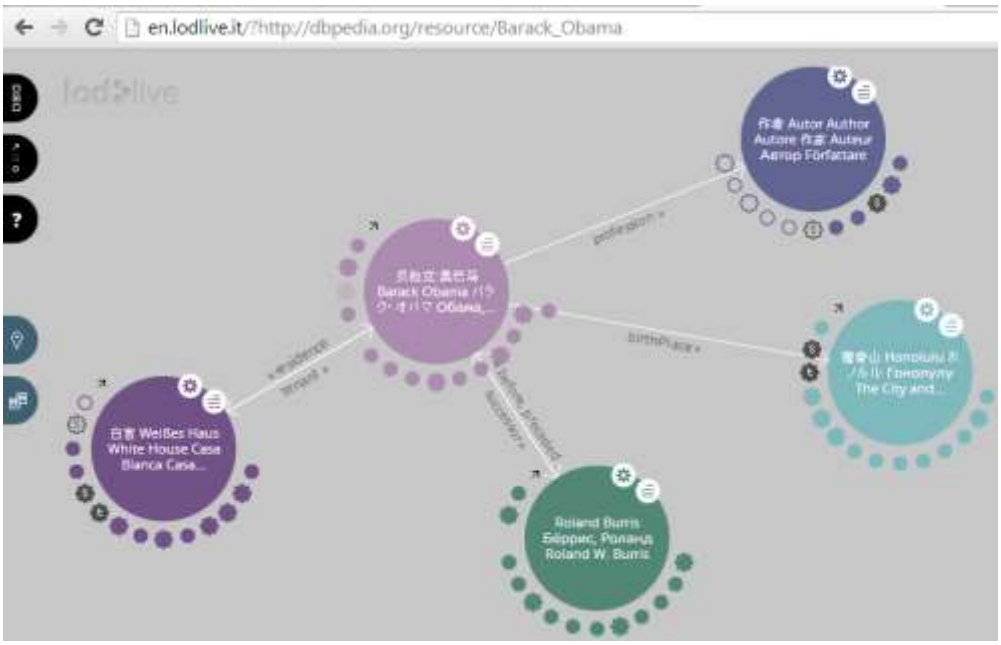

Figura 35 – Apresentação e navegação em grafo de LodLive.

A Figura 35 mostra a apresentação em grafo exibida para o usuário de LodLive por meio de uma consulta ao URI do recurso "Barack Obama" na fonte de dados DBPedia. Um dos pontos negativos da ferramenta é que o usuário, necessariamente, precisa selecionar a fonte de dado que ele deseja explorar. A combinação de resultados de fontes de dados diferentes é pouco utilizada nas respostas. Por exemplo, a ferramenta poderia apresentar o grafo resultante da descoberta de informações sobre o recurso "Barack Obama" existentes em diversas fontes de dados como DBPedia, FreeBase, Flickr, entre outras, invés disso se atem às informações contidas na fonte selecionada. Do ponto de vista de usuários experientes e acostumados às técnicas semânticas, a apresentação em grafo fornecida pela ferramenta é bastante rica. No entanto, mesmo para *experts* o acesso às informações dos recursos se torna confuso e desestimulante. LodLive definitivamente não é uma ferramenta para ser utilizada por usuários comuns. A Interface não é intuitiva e não deixa claro, para tais usuários, como eles podem aproveitar o conhecimento obtido da Web de Dados. LodLive pode ser combinado com a proposta apresentada nesta pesquisa para oferecer novas formas de apresentação e navegação fazendo uso da biblioteca Javascript e do Serviço Web desenvolvidos aqui. Assim, essa ferramenta poderia disponibilizar, por exemplo, suporte a facetas e *tooltips* dinâmicos, mais adequados ao usuário comum.

NautiLOD (Fionda et al., 2012) é uma linguagem declarativa para enfrentar o desafio da navegação ao longo dos nós da Web de Dados, utilizando a semântica armazenada em cada fonte de dados. Baseada em recursos de navegação, ela é projetada para especificar fragmentos da Web de Dados e ações a serem executadas com base nesses dados. NautiLOD foi implementada de forma centralizada, para mostrar seu poder e performance. A aplicação foi desenvolvida sobre a ferramenta *swget*, que explora protocolos da Web atual e funciona na nuvem LOD. A versão distribuída de *swget* foi explorada em uma implementação de prova de conceito para mostrar a sua viabilidade, potencialidades e desafios.

NautiLOD não é uma ferramenta para usuários leigos. Mesmo porque, ela não possui interface convencional, direcionada para esse tipo de usuário. NautiLOD é uma linguagem criada especificamente para o desenvolvedor que deseja dar suporte a navegação na Web de Dados para suas aplicações, sem precisar entender ou utilizar recursos avançados de SPARQL. O principal ponto negativo de NautiLOD é sua implementação na plataforma *swget*. A utilização de *swget*, mesmo para desenvolvedores experientes, é bastante pesada e complicada, sem falar que é necessário conhecer uma grande variedade de comandos de terminal para, efetivamente, fazer a ferramenta funcionar. Por meio de NautiLOD, a recuperação de informações é bastante surpreendente, uma vez que a ferramenta consegue atravessar o espaço de dados para recuperar conhecimento, até então, não explorado e executar ações baseadas nesses dados. Por outro lado, para certos recursos (principalmente da fonte DBPedia), a *engine* de NautiLOD se comportou como um simples desreferenciamento de URI (i.e. como um HTTP GET da URI do recurso), se comparando a ferramentas comuns de navegação. Outro problema é o tempo (bastante longo) necessário para recuperação das informações. Por esses motivos, foi decidido não apostar em NautiLOD como um componente do Serviço Web desenvolvido neste trabalho, pois uma das principais preocupações desta pesquisa é com a viabilidade de tempo na entrega das estruturas de apresentação e navegação fornecidas pelo serviço.

MyView (Cheng et al., 2011) é um navegador *Linked Data* que permite aos usuários consultarem *Linked Data* por navegação, de uma coleção de entidades para outra. Com essa ferramenta, os usuários podem reutilizar suas consultas passadas de várias maneiras, incluindo: (i) categorização de *links* favoritos com diferentes visualizações; (ii) montagem de *links* complexos combinados com outros já existentes; e (iii) revisitação de consultas passadas via histórico e mecanismos de *bookmark*. Como um sistema inteligente, MyView avalia consultas em forma de programação lógica, com suporte a raciocínio. Sua implementação apresenta várias estratégias para lidar com o ambiente distribuído, aberto e de grande escala que é a Web. A ferramenta também gera respostas explicáveis que carregam informações de sua procedência. Finalmente, MyView interage com os usuários para resolver correferências de entidades, para descobrir mais fontes e reduzir a redundância.

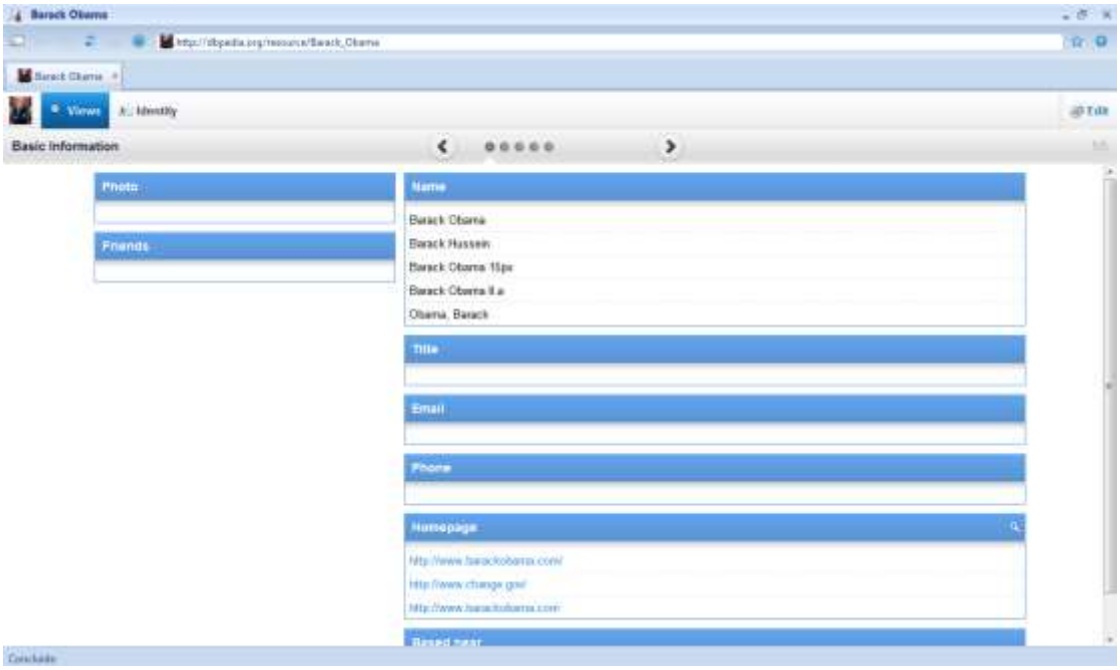

Figura 36 – Apresentação e navegação de MyView.

A forma de apresentação e navegação de MyView (Figura 36) se assemelha bastante com o tipo de apresentação e navegação suportados nas primeiras categorias de browsers *Linked Data* como Tabulator[23] e Marbles[24]. Apesar da promessa de diferentes visualizações, a ferramenta parece possuir apenas um único tipo desse recurso (visualização em tabela). O principal ponto negativo é o tempo de carregamento necessário para apresentar os dados no formato definido pela ferramenta. É praticamente inviável, para o usuário final, esperar mais de cinco minutos para visualizar alguma informação útil. Em termos de tempo de carregamento, a solução apresentada neste trabalho (extensão do navegador Google Chrome) é bastante superior a MyView, exibindo toda a gama de dados para o usuário em tempo hábil. No que se refere ao conteúdo recuperado, a extensão desenvolvida aqui produz melhores resultados por exibir dados relacionados oriundos da Web 2.0 (mídia, *news*, entre outros), em adicional aos dados provenientes da Web de Dados. Na parte de apresentação e navegação MyView se mostrou pobre e confuso. A extensão apresentada como uma das soluções desta pesquisa organiza melhor as informações além de prover recursos de navegação mais "limpos", adequados à condição de inexperiência dos usuários comuns.

Piggy Bank (Huynh et al., 2007), idealizado para permitir o acesso mais flexível à informação estruturada, habilita usuários a fazer uso de conteúdo da Web Semântica dentro do próprio conteúdo da Web tradicional. Piggy Bank torna possível que usuários naveguem na Web Semântica da mesma forma que eles navegam na Web atual. Quando o conteúdo da Web Semântica não está disponível a ferramenta pode

[23] Tabulator: http://www.w3.org/2005/ajar/tab

[24] Marbles: http://mes.github.io/marbles/

invocar *screenscrapers* para reestruturar a informação dentro de páginas Web em formato compatível com a Web Semântica. Por meio do uso de tecnologias da Web Semântica, Piggy Bank fornece, diretamente, benefícios imediatos para usuários no uso da Web. Assim, a existência de poucos sítios da Web contendo informação semântica ou alguns *scrapers* já pode beneficiar os usuários. Dessa forma, Piggy Bank oferece um aprimoramento incremental para os usuários sem requerer a adoção de conhecimento de técnicas da Web Semântica.

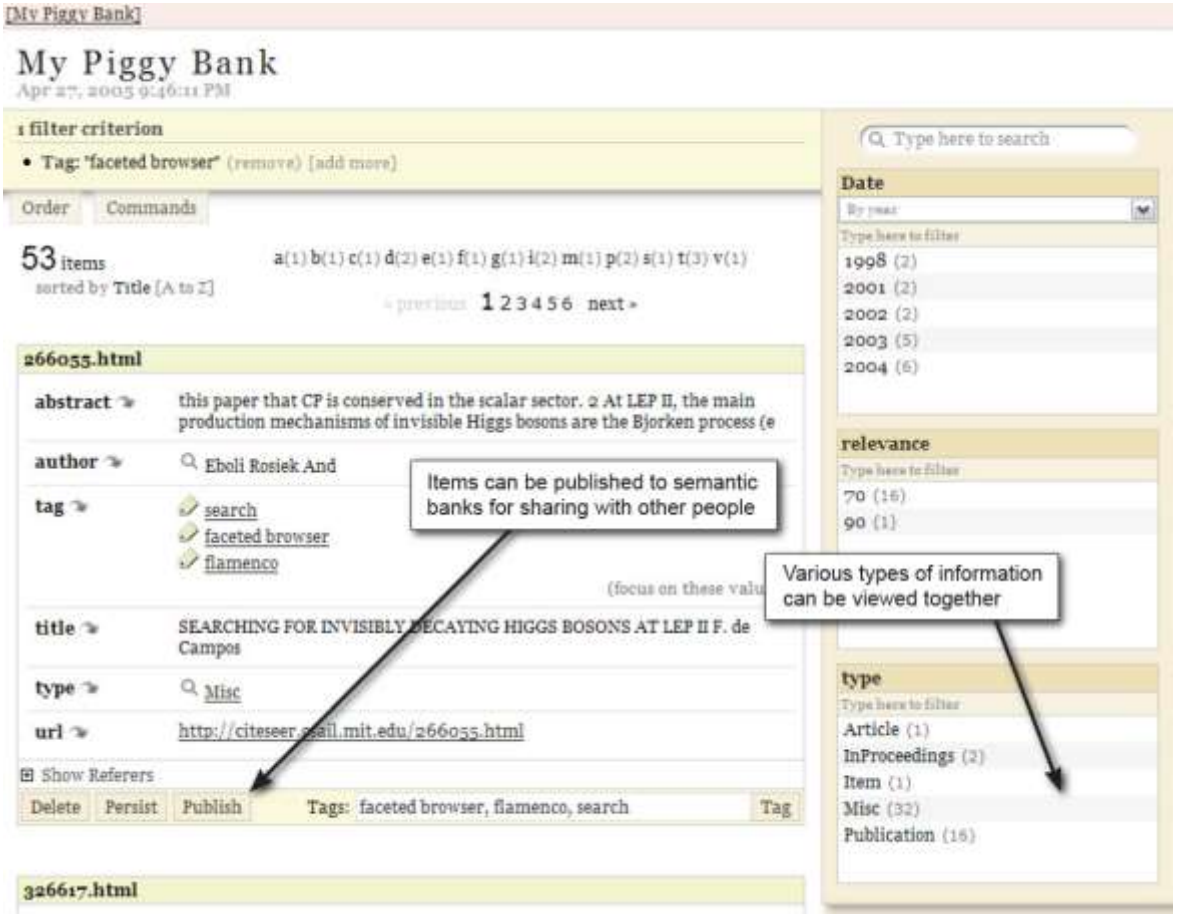

Figura 37 – Apresentação e navegação de Piggy Bank.

Um recurso bastante interessante fornecido por Piggy Bank é o suporte a conversão de dados de HTML para RDF, usando uma abordagem de *screenscraping*. O usuário pode visualizar qualquer página Web como dados estruturados. É uma função útil para autoria de *Linked Data*. No entanto, a disponibilidade de *screenscrapers* é limitada para alguns poucos *web sites* como o portal de *scraper* da ACM (*ACM Portal Scraper*[25]) e o *scraper* de fotos do Flickr (*Flickr Photo Scraper*[26]). *Screenscrapers* adicionais necessitam serem desenvolvidos por outros *web sites* em Javascript ou por meio de uma ferramenta associada (Solvent),

[25] ACM Portal Scraper: http://simile.mit.edu/wiki/ACM_Portal_Scraper

[26] Flickr Photo Scraper: http://simile.mit.edu/wiki/FlickrPhotoScraper

baseada na estrutura subjacente do site alvo. Em termos de apresentação do conteúdo, Piggy Bank não foge ao padrão tabular e não acrescenta novidades a já conhecida representação desorganizada de dados em tabela. Mais uma vez, a extensão Chrome desenvolvida neste trabalho consegue se sobressair na parte da organização das informações apresentadas ao usuário comum, diferente de Piggy Bank.

Paggr (Nowack, 2009) é um sistema baseado no padrão de agregação e apresentação de porções de informações da Web por meio de *widgets* baseados em HTML. Enquanto os *widgets* Web de hoje são, principalmente, baseados em chamadas a APIs proprietárias, invés disso Paggr usa operações SPARQL. *Widgets* Paggr (*Sparqlets*) podem ser organizados em *dashboards* e uma ferramenta de desenvolvimento baseada em navegador é fornecida para permitir que desenvolvedores possam colaborativamente criar *widgets*. Paggr tenta oferecer um compromisso entre a criação de uma solução personalizada e o uso de ferramentas padronizadas. Cada *widget* pode construir outro componente para acelerar a criação de soluções individuais. O editor de *widgets* baseado em Web permite a adição de novas opções em tempo real.

Figura 38 – Apresentação e navegação de Paggr.

Paggr fornece uma apresentação em forma de *widgets* bastante criativa. Cada *widget* pode ser customizado para exibir determinado tipo de dado. O ponto negativo de Paggr é na criação e edição dos *widgets*. Embora uma ferramenta Web esteja disponível para ajudar na realização dessa tarefa, um usuário comum certamente encontrará bastante dificuldade no uso de tal ferramenta. É necessário ter conhecimento aprofundado de técnicas semânticas e criação de consultas SPARQL para obter bons resultados no conteúdo

exibido nos *widgets*. Essa é uma tarefa árdua até para um usuário experiente. Outra questão é que a navegação fica restrita aos *widgets* configurados. Caso haja a necessidade de navegar por outros tipos de dados, é necessário criar ou configurar outros *widgets*. Uma combinação entre Paggr e a solução apresentada aqui pode aprimorar e facilitar a construção dos *widgets* por parte dos usuários comuns, uma vez que a biblioteca e o serviço desenvolvidos neste trabalho fornecem estruturas amigáveis, baseadas no modelo proposto, direcionadas para tal tipo de usuário.

Fenfire (Hastrup et al., 2008) é um navegador RDF livre e de código aberto, que emprega uma visão de grafo focada em uma experiência de navegação interativa. Isso deixa Fenfire separado de existentes navegadores *Linked Data* baseados em tabelas. A interface de usuário usa uma representação convencional de grafo para apresentar as informações do modelo RDF. Para tornar a visualização escalável em número de nós e para focar em apenas uma "coisa de cada vez", somente um nó central e seus vizinhos são apresentados concorrentemente. A sessão de navegação é iniciada por meio de um URI recuperado de um documento ou uma descrição RDF. Esse URI vai ser o foco inicial do grafo. A partir daí é possível navegar interagindo com os nós vizinhos.

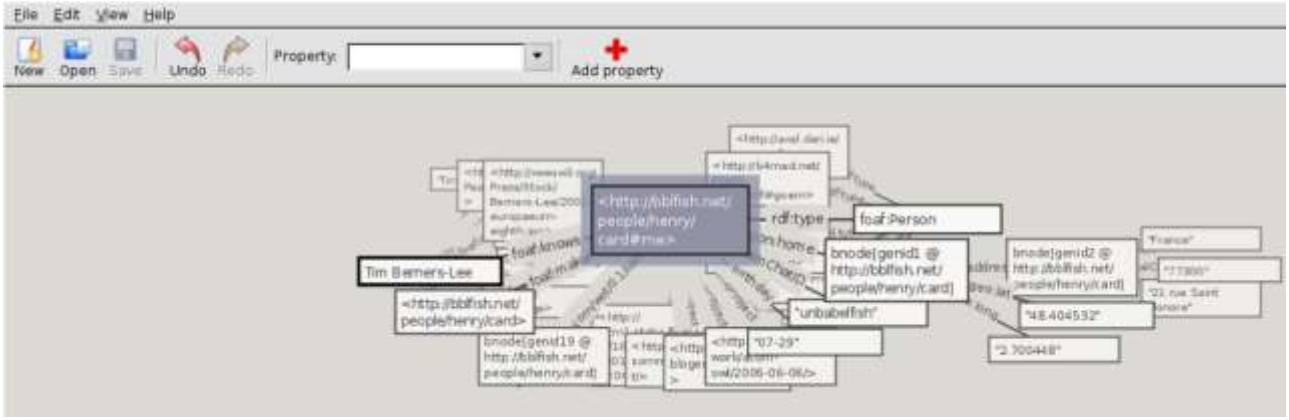

Figura 39 – Apresentação e navegação de Fenfire.

A apresentação das informações semânticas por Fenfire está longe de ser direcionada para usuários sem experiência. A primeira ferramenta mostrada nessa sessão, LodLive, possui uma apresentação em grafo muito mais detalhada e mais próxima do que seria uma apresentação em grafo voltada para o usuário final. A navegação nos nós do grafo, apresentado por Fenfire, também deixa muito a desejar, uma vez que para cada clique em um nó, a visão do grafo é centrada no nó clicado. É difícil saber qual nó deu origem a atual visualização exibida na tela. A visualização dos relacionamentos é confusa e quase impossível de determinar. Fenfire poderia combinar sua apresentação em grafo com a apresentação por *tooltips* e navegação por facetas fornecidas pela solução desenvolvida nesta pesquisa. Isso facilitaria a vida de usuários

não experientes, que poderiam ter acesso a informações mais organizadas nos *tooltips* e navegar mais facilmente no grafo por meio das facetas.

7.2 PUBLICAÇÃO

Segundo García et al. (2011), o potencial da Web de Dados é enorme. No entanto, existem barreiras impedindo que usuários finais, sem experiência com *Linked Data*, possam realmente contribuir para o crescimento desse espaço de dados. Uma das principais barreiras encontradas por esses usuários, é que tais dados estão, em sua maioria, disponíveis como *dumps*[27] de dados brutos ou SPARQL *endpoints*. Quando se lida com *dumps* de dados, é realmente complicado determinar: (i) que dados tem-se em mãos; (ii) a que eles se referem; e (iii) que tipo de termos são usados. Para realizar essa tarefa é necessário possuir experiência com ferramentas da Web Semântica. Para SPARQL *endpoints*, a quantidade de trabalho necessária para compreender fatores internos do conjunto de dados pode ser reduzida. Entretanto, um bom conhecimento de SPARQL é necessário a fim de gerar e entender um conjunto de *queries* que permitem determinar: (i) o tamanho do conjunto de dados; (ii) quais são os principais tipos de recursos; e (iii) como eles se inter-relacionam.

Considerando esses problemas, o trabalho de García et al. (2011) propõe Rhizomer, que consiste em uma ferramenta de publicação de dados estruturados. A ferramenta fornece um conjunto de componentes que facilita a conscientização do usuário, diante dos dados que se tem em mãos. Rhizomer gera automaticamente menus de navegação, levando em conta as ontologias usadas pelo conjunto de dados. Além disso, a ferramenta também gera facetas com base em como as propriedades são instanciadas para cada uma das classes no conjunto de dados. Isso capacita o usuário a estar sempre ciente dos principais tipos de recursos no conjunto de dados, bem como, suas principais propriedades e valores. Esses componentes são complementados com serviços especializados de interação, que podem ser implantados de forma dinâmica e associados a recursos, usando serviços semânticos.

Graves (2010) defende que a informação referente à segurança pública é importante para sociedades organizadas por várias razões: (i) Cidadãos podem usar registros de segurança pública para identificar riscos em suas comunidades. Isso pode influenciar suas decisões de curto prazo (se visita ou não determinada localidade) ou longo prazo (se adquire ou não um imóvel em determinada vizinhança); (ii) o

[27] *Dumps* de dados contêm registros da estrutura de tabela e/ou dados de uma base de dados e está, geralmente, sob a forma de uma lista de instruções SQL. *Dumps* de dados são frequentemente publicados por software livre e projetos de conteúdo livre, para permitir a reutilização da base de dados [http://en.wikipedia.org/wiki/Database_dump].

Estado pode usar registros de segurança pública e emergências para tomar decisões que têm impacto na comunidade (como ajustes de orçamento para as forças policiais); e (iii) informações sobre as atividades relacionadas com a segurança podem melhorar a transparência de, por exemplo, organizações de aplicação da lei, melhorando suas relações com as comunidades.

Diante disso, foi desenvolvida uma aplicação que permite aos cidadãos encontrar dados verídicos sobre eventos de segurança pública (por exemplo, assaltos, incêndios, entre outros), a partir de fontes confiáveis. De acordo com os autores, a ferramenta é capaz de responder a perguntas como: "*é seguro andar na rua x entre 04:00pm e 06:00pm*?", "*posso estacionar o carro nessa vizinhança sem o risco de ser roubado*?" ou "*o quão segura é a área em torno da escola que meus filhos frequentam*?". Para tanto, registros de agencias de segurança pública foram capturados e transformados em dados estruturados (no padrão RDF). Posteriormente, as informações desejadas foram extraídas, com base em ontologias, e os dados foram publicados na Web de Dados seguindo os princípios de *Linked Data*.

Para Ding et al. (2011), nos últimos anos, observou-se um crescimento constante na publicação de dados abertos de governos (OGD – *open government data*), emergindo como um canal de comunicação vital entre os governos e seus cidadãos. Um considerável número de portais Web (e.g. Data.gov e Data.gov.uk) têm sido utilizados para disponibilizar conjuntos de dados online de OGD. Para os governos, os custos do fornecimento de dados são reduzidos quando lançado através destes portais. No entanto, para os usuários dos dados, isso pode causar problemas de interoperabilidade, escalabilidade e usabilidade. Para acelerar o uso de dados de governos por parte dos cidadãos e desenvolvedores, é necessária uma infraestrutura eficaz com poder computacional suficiente para processar grandes volumes de dados OGD.

O TWC – *Tetherless World Constellation* do RPI – *Rensselaer Polytechnic Institute*, desenvolveu o portal TWC LOGD, baseado em tecnologias da Web de Dados, para apoiar a disseminação de dados abertos ligados de governos (LOGD – *Linked Open Government Data*). O portal é uma infraestrutura aberta de apoio à produção e consumo de LOGD. Além disso, TWC LOGD educa e atende a crescente comunidade internacional de desenvolvedores, proprietários e usuários de dados abertos de governos. A fim de apoiar os usuários na produção e consumo de dados em diferentes níveis de granularidade, TWC LOGD define um modelo de organização de dados construído em torno do detalhamento estrutural de conjuntos de dados LOGD. Este modelo é usado para projetar URIs *Linked Data*. O conjunto de dados do portal Data.gov foi usado para demonstrar o modelo.

7.3 AUTORIA

Shakya et al. (2009) pontuam os seguintes problemas relacionados ao modelo *Linked Data*: (i) não está claro como pessoas comuns, sem qualquer experiência com a Web Semântica, podem publicar e compartilhar dados estruturados diretamente; (ii) o fato de que podem haver várias perspectivas sobre o mesmo conceito, diferentes aspectos ou contextos a serem considerados, é frequentemente ignorado; e (iii) o estado da arte não possui estruturas (ontologias) suficientes que possam representar e organizar a vasta gama de conceitos necessários para as comunidades de usuários da Web.

Considerando essas questões, o trabalho propõe uma abordagem para permitir que pessoas comuns compartilhem dados estruturados diretamente, por meio de uma plataforma social de fácil uso. Usuários podem definir seus próprios conceitos (múltiplas conceitualizações são permitidas). Esses conceitos são consolidados usando técnicas semiautomáticas de alinhamento de esquema. Além disso, conceitos são agrupados por similaridade. Como resultado da consolidação e agrupamento de conceitos, ontologias informais emergem gradualmente. Para validar a proposta, uma ferramenta chamada StYLiD foi desenvolvida. Essa ferramenta funciona como uma plataforma para motivar pessoas a compartilharem qualquer tipo de coisas. A ferramenta pode também ser dirigida a portais verticais para comunidades de usuários específicas, com dados integrados de múltiplas fontes.

Segundo Berners-Lee et al. (2007), a primeira categoria de navegadores semânticos foi projetada para apresentar conjuntos de dados em formato RDF, apenas para leitura. Uma segunda categoria abordou mecanismos e problemas de exibição em torno de dados recuperados da nuvem de dados, obedecendo aos princípios de *Linked Data*. Contudo, o desafio de, uma vez os dados recuperados, como editá-los, compartilhá-los, anotá-los ou comentá-los, tem sido deixado sem solução. Isso acontece, em grande parte, porque existe uma série de problemas para determinar como apoiar a autoria de informações na Web de Dados. Para atacar alguns desses desafios, o trabalho propõe uma nova versão do navegador Tabulator.

O projeto para a extensão do navegador Tabulator é focado no lado da escrita da Web de Dados. A abordagem é a de permitir modificações e edições de informações naturalmente dentro da interface do navegador. Essas alterações são retransmitidas para o servidor tripla por tripla tentando manter o mínimo de fragilidade possível. Desafios que permanecem inclui a propagação de alterações, realizadas de forma colaborativa, de volta para a interface dos colaboradores, no caso da utilização de um sistema de edição compartilhada. Para apoiar a tarefa de escrita em recursos semânticos da Web de Dados, o trabalho tem

contribuído com várias tecnologias. Uma dessas contribuições é um protocolo de comunicação, baseado em atualização HTTP/SPARQL, entre um editor e os recursos editáveis, mantidos em fontes de dados abertas.

Davies et al. (2010) levantam algumas questões sobre a habilidade de novatos (usuários não técnicos) para gerar dados estruturados consistentes com os princípios *Linked Data*, requeridos pela Web de Dados. Segundo os autores, representação formal do conhecimento é um processo não trivial e propenso a erros, para a maioria das pessoas sem experiência com a Web Semântica. Essa é uma atividade muito diferente de escrever em linguagem natural, que é a maneira que a maioria dos leigos têm contribuído para a Web até então. Além disso, os autores apontam que, autoria de *Linked Data* exige: (1) um esquema de nomes, inabalavelmente, consistente; (2) um nível inédito de exatidão; (3) fluência com um novo conjunto de conceitos; e (4) uma adesão a um conjunto de regras rígidas e aparentemente arbitrárias (para o leigo), que vão contra a forma como a maioria das pessoas pensa, muito menos conversa.

O trabalho de Davies et al. (2010) também relata um experimento controlado, no qual novatos tentam usar um protótipo de interface *Linked Data*, tanto para encontrar, quanto para produzir conhecimento comum do dia a dia. A ferramenta apresenta uma aparência amigável, manifestando a natureza essencial do modelo baseado em grafo da Web de Dados, enquanto protege o usuário da complexidade de sintaxe. O experimento permitiu o estudo aprofundado do comportamento do usuário, com relação ao problema cognitivo da quebra do conhecimento em uma estrutura baseada em triplas, requerido pelo modelo RDF. O trabalho esclarece alguns dos principais aspectos da dificuldade encontrada pelos novatos na formulação do conhecimento estruturado. Várias abordagens de projetos específicos para o ambiente de autoria *Linked Data* são sugeridas, deixando claro os benefícios que podem ser adquiridos e as questões cruciais que devem ser tratadas.

7.4 OUTROS

Ziegler (2011) aborda questões relacionadas com a falta de trabalhos voltados para a construção de ambientes adequados ao uso da Web Semântica, por parte dos usuários iniciantes (não técnicos). Para o autor, os objetivos da Web Semântica foram – e ainda são – principalmente direcionados em tornar a informação contida na Web legível por máquinas. Em muitos casos, no entanto, o último destinatário dos dados é um usuário humano. É cada vez mais reconhecido que esses usuários também podem se beneficiar mais diretamente de técnicas semânticas. Usuários comuns podem ser apoiados com meios adequados para interagir com a crescente Web de Dados. Suas necessidades de informação podem ser satisfeitas de forma

mais orientada e eficaz. Assim, eles podem explorar conceitos ou relações desconhecidas, e obter acesso personalizado e contextualizado a recursos e serviços relevantes. Isso cria a necessidade por métodos e ferramentas que permitam a esses usuários interagirem com os dados da Web Semântica diretamente, e não, somente, através de uma aplicação Web que integra e fornece tais dados em páginas Web padrão.

Para Dadzie and Rowe (2011), a captação e consumo de *Linked Data*, atualmente, está restrito, quase que totalmente, à comunidade da Web Semântica. Embora a utilidade de *Linked Data* para usuários inexperientes seja evidente, a falta de conhecimento técnico e compreensão da pilha de tecnologias semânticas, limitam tais usuários em sua capacidade de interpretar e fazer uso da Web de Dados. Os autores defendem que a solução-chave para superar este obstáculo é permitir a visualização de *Linked Data* de uma forma coerente e legível. Consequentemente, isso permitirá que o público não técnico possa obter uma boa compreensão da estrutura semântica e, portanto, implicitamente compor consultas, identificar ligações entre recursos e intuitivamente descobrir novas peças de informação. O trabalho descreve os requisitos essenciais que a visualização de *Linked Data* deve satisfazer, a fim de diminuir as barreiras técnicas e tornar a Web de Dados acessível para todos.

Segundo Brunetti et al. (2012), a quantidade de dados semânticos disponíveis na Web tem crescido consideravelmente, especialmente devido a iniciativas como o projeto LOD. Esses dados possuem um enorme potencial, contudo, em muitos casos, é complicado e difícil para usuários não técnicos visualizarem, explorarem e usarem esses dados. A aplicação de técnicas de visualização da informação na Web Semântica pode ajudar esses usuários a explorar grandes quantidades de dados e interagir mais facilmente com eles. As visualizações são úteis para a obtenção de uma visão geral dos conjuntos de dados, seus principais tipos, propriedades e suas relações. O modelo unificado de dados RDF, prevalente na Web de Dados, permite associar dados para visualização de uma forma imprevisível e dinâmica. Considerando essas questões, o trabalho propõe um modelo de visualização *Linked Data* (LDVM - *Linked Data Visualization Model*), que permite a conexão de diferentes conjuntos de dados com diferentes visualizações de uma forma dinâmica.

Latif et al. (2009) esclarecem que um dos fundamento mais importantes da Web Semântica é *Linked Data*. O projeto LOD (nuvem *Linked Data*) oferece a oportunidade de explorar e combinar conjuntos de dados em escala global – algo que nunca foi possível antes. No entanto, em seu estágio atual, a nuvem *Linked Data* produz pouco benefício para os usuários finais, que não possuem conhecimento de ontologias, triplas e SPARQL. O trabalho apresenta uma técnica inteligente para localizar URIs desejados a partir da Web de Dados. Palavras-chave de busca, fornecidas pelos usuários, são utilizadas de forma inteligente para localizar o URI pretendido. A técnica proposta foi aplicada em uma interface simples de usuário final para acesso a nuvem LOD. A avaliação do sistema mostrou que a técnica reduziu a carga cognitiva do usuário na busca de informações relevantes.

7.5 CONSIDERAÇÕES FINAIS

A maioria dos trabalhos apresentados neste capítulo foram avaliados por meio da utilização direta das ferramentas resultantes de cada pesquisa. Com base nessa utilização, tais soluções foram analisadas segundo sua facilidade de uso, interface direcionada para o usuário inexperiente, descoberta de conhecimento relevante, tempo de resposta, entre outros.

Com relação à área de apresentação e navegação, a grande maioria se mostrou deficiente em algum desses pontos. Isso deixa claro que as soluções abordando a apresentação e navegação de *Linked Data* ainda precisam resolver muitas questões importantes para se tornarem viáveis.

Em comparação com alguns desses trabalhos, o protótipo desenvolvido como parte da proposta desta dissertação se mostrou bastante promissor, uma vez que conseguiu se sobressair nos quesitos apontados acima. É importante notar que os trabalhos analisados neste capítulo podem se beneficiar com a utilização do modelo de apresentação e navegação de *Linked Data* proposto aqui. A aplicação do modelo nessas soluções pode: (i) melhorar a facilidade de uso e a interface de usuário, com o advento da camada de preparação de interface; (ii) aperfeiçoar a descoberta de conhecimento relevante, a partir da camada de descoberta e aconselhamento; e (iii) diminuir o tempo de resposta, por meio das camadas de filtragem analítica e mapeamento.

Capítulo

8

Este capítulo apresenta a avaliação do modelo, objeto desta proposta, realizada por meio do protótipo, e a partir do uso da biblioteca Javascript e do Serviço Web, construídos como parte deste trabalho. A avaliação foi fundamentada em termos de medidas de complexidade dos algoritmos que compõem a arquitetura das operações e suboperações fornecidas pelo modelo. Além disso, experimentos foram realizados com o propósito de aferir os tempos obtidos nas respostas a partir das requisições a essas operações.

8 AVALIAÇÃO

Embora a proposta desta dissertação seja fundamentada em um modelo de apresentação e navegação focado no usuário final, a pesquisa executada aqui não realizou estudos de usuários objetivando avaliar as interfaces criadas pelo protótipo (por meio da biblioteca Javascript) para prover os recursos de apresentação e navegação para *Linked Data*. A explicação para essa decisão está no fato de que: (1º) o modelo apresentado neste trabalho é independente de interface. Ou seja, o modelo pode ser implementado em sua totalidade para gerar interfaces variadas, definidas pelo sistema que o implementa, usando, ou não, a camada de apresentação e navegação; (2º) os tipos de interfaces de usuário, criadas pela biblioteca Javascript, utilizadas como exemplo para demonstrar a viabilidade da proposta, são baseadas em interfaces bem definidas e amplamente aceitas no ambiente Web e, portanto, previamente avaliadas em trabalhos relevantes da comunidade de pesquisa desse domínio, como é o caso de (Clarkson et al., 2009), (Fagan, 2010), (Hearst, 2008), (Hudson et al., 2004), (Oren et al., 2006), (Prazeres et al., 2006), (Tarasov et al., 2010) e (Yee et al., 2003).

Portanto, a avaliação desta pesquisa foi executada com base em medidas de complexidade e de tempo, calculadas sobre as operações fornecidas pelo modelo de apresentação e navegação (Seção 5.6), objeto desta dissertação. Essa abordagem permitiu avaliar separadamente a viabilidade das camadas definidas no modelo, uma vez que cada camada realiza uma série de processos não triviais, que podem se tornar custosos do ponto de vista do desempenho.

O ambiente onde os experimentos foram realizados possui as seguintes características: Processador Intel(R) Core(TM) 2 Duo, CPU T7250 2.00GHz; Cache L2 2MB; Memória RAM 4,00 GB; Sistema Operacional de 64 Bits Windows 7 Utimate; Rede WiFi 802.11 com banda de 10Mbps; e Linguagem de Programação Java versão SE 7.

Os experimentos tiveram o objetivo de avaliar o limite de desempenho das operações, chegando ao ponto de inserir 62.500 triplas RDFa em uma página HTML. No entanto, de acordo com o Projeto Web Data Commons (Bizer et al., 2012), atualmente, existem cerca de 168 milhões de páginas Web contendo marcações RDFa, e a média de triplas RDFa contidas nessas páginas é de apenas 6,4 triplas RDFa por página. Dessa forma, nos experimentos executados, e apresentados a seguir, fica claro que o desempenho é considerado bom até 2.500 triplas – superior às 6,4 triplas RDFa por página relatado por Bizer et al. (2012).

8.1 OPERAÇÃO FILTER RDF DESCRIPTION

O objetivo da operação *Filter RDF Description* é verificar cada tripla da descrição RDF, recebida como parâmetro de entrada, para determinar a presença de metatriplas e triplas redundantes (Seção 5.6). Caso uma metatripla ou tripla redundante seja identificada, ela é removida da descrição. As suboperações de identificação e remoção de triplas possuem instruções executadas diretamente por operações nativas da linguagem de programação utilizada no desenvolvimento. Esse trabalho possui um custo computacional baixo, se comparado ao trabalho de verificação, e, portanto, considerado desprezível nesta avaliação, como pode ser visto na Tabela 2.

Tabela 2 – Trabalho realizado pelas suboperações de Filter RDF Description

Operação	Trabalho
Verificação	n
isMetatriple(triple)	1
isRedundantTriple(triple)	1
removeMetatriple(triple)	1
removeRedundantTriple(triple)	1
Trabalho Total	**n + 1 + 1 + 1 + 1**
Complexidade	**O(n)**

Assim sendo, a complexidade do trabalho de verificação é O(n), uma vez que todas as triplas devem ser, sempre, analisadas. Ou seja, uma análise sempre será executada em cada tripla do conjunto de triplas recebido como entrada da operação (entrada de tamanho n). A Tabela 3 mostra um experimento onde o tempo de processamento da operação *Filter RDF Description* é avaliado segundo o tamanho crescente da entrada de dados (conjunto de triplas RDF).

Tabela 3 – Tempos de processamento da operação Filter RDF Description

Complexidade O(n)	
Triplas	**Tempo (ms)**
100	18
500	95
2500	493
12500	2634
62500	13680

Na Tabela 3 é possível observar que o desempenho da operação *Filter RDF Description* até 2.500 triplas é aceitável, consumindo apenas 493 milissegundos.

Outra questão importante é que durante a pesquisa, um levantamento empírico, por amostragem, foi realizado como forma de adquirir mais conhecimento sobre o conteúdo das descrições RDF de recursos contidos na Web de Dados. O levantamento coletou 25 descrições RDF de recursos contidos na fonte de dados DBPedia. Posteriormente, uma análise foi realizada nas descrições coletadas para determinar metatriplas e triplas redundantes. O resultado obtido mostrou que as descrições RDF analisadas continham, aproximadamente, 35% de meta dados e informações redundantes, em média. Portanto, tais descrições apresentam apenas uma média de 65% de informação útil, do ponto de vista do usuário final. Considera-se aqui informação útil, triplas que carregam dados, relacionados ao recurso *Linked Data*, que possam ser reutilizados por outras fontes de dados, aplicações ou diretamente pelo usuário final, sem repetições e redundâncias.

Logo, apesar da operação *Filter RDF Description* consumir tempo e recursos computacionais, o resultado pode se refletir em um melhor desempenho no processo geral. Tratar 35% a menos num universo de milhares de triplas RDF tem impacto positivo animador, como pode ser visto na Tabela 4.

Tabela 4 – Processo geral com e sem a operação Filter RDF Description

Total de Triplas	Triplas Removidas	Tempo Total sem Operação	Tempo Total com Operação
100	31	353	319
500	154	621	503
2500	893	4478	3154
12500	4037	21258	10175
62500	20934	387100	80226

A Tabela 4 mostra uma comparação do processo geral de tratamento de descrições RDF até se tornarem uma estrutura consolidada de apresentação e navegação voltada para o usuário comum. O processamento foi realizado pela implementação do modelo proposto, por meio do Serviço Web e da biblioteca Javascript descritos no Capítulo 6. No primeiro passo, um conjunto de dados contendo descrições RDF com diferentes quantidades de triplas foi processado, sem usar a operação *Filter RDF Description*. Os resultados podem ser vistos na terceira coluna da Tabela 4 (com tempo total em milissegundos). Depois disso, o mesmo conjunto de dados foi processado usando a operação *Filter RDF Description* na primeira fase do procedimento. É possível observar, na última coluna da Tabela 4, que o tempo de processamento dos dados é menor quando a operação *Filter RDF Description* está ativa. Quanto maior a quantidade de triplas envolvidas no processo, a diferença de desempenho (baseado no tempo) é mais perceptível.

8.2 OPERAÇÃO MAP RDF TO DEXTER

O objetivo da operação *Map RDF to Dexter* é realizar o mapeamento da descrição RDF, recebida como parâmetro de entrada, para uma rede Dexter (Seção 5.6). Cada recurso RDF é transformado em um átomo Dexter, e os relacionamentos entre os recursos, transformados em links Dexter. A suboperação de montagem dos átomos precisa analisar todos os objetos (objeto da tripla) ligados a cada recurso para determinar se tais objetos são literais ou se tratam de outros recursos (análise dos vizinhos do nó). Caso sejam literais, podem ser adicionados ao átomo que representa o recurso do contexto. Caso contrário, deve-se analisar se o recurso (objeto do recurso do contexto) já está presente na rede. Se não está presente, um novo átomo deve ser criado para representar tal recurso.

As suboperações de identificação de objetos (literais ou recursos), de adição de literais ao átomo e da construção de links são consideradas desprezíveis por usarem operações nativas da linguagem de programação usada no desenvolvimento, como pode ser visto na Tabela 5.

Tabela 5 – Trabalho realizado pelas suboperações de Map RDF to Dexter

Operação	Trabalho
createAtom()	n*k
netContains(atom)	m (m=n)
createLink()	1
conectAtoms(atom1, link, atom2)	1
isResource(tripleObject)	1
isLiteral(tripleObject)	1
insertLiteral(tripleObject, atom)	1
Trabalho Total	**nkm + 1 + 1 + 1 + 1 + 1**
Complexidade	**O(n^3)**

Como é possível observar na Tabela 5, a operação *Map RDF to Dexter* possui complexidade O(n^3), uma vez que a suboperação *createAtom()* realiza um trabalho com custo *n*k*, onde *n* é a quantidade de recursos *Linked Data* da descrição RDF, e *k* é o número de conexões de saída de cada recurso *n*. No pior caso, *k* pode ser maior que *n* (*k*>*n*). Isso porque cada recurso pode estar ligado a outros recursos por vários tipos de relacionamento diferentes. Por exemplo, um dado recurso A pode estar ligado a um dado recurso B por um relacionamento do tipo *isFather* (A *isFather* B) e um relacionamento do tipo *isBoss* (A *isBoss* B), simultaneamente. A suboperação *netContains(atom)* verifica a presença de um recurso na rede Dexter em fase de construção, onde *m* é a quantidade de átomos na rede. O valor de *m*, no pior caso, pode ser igual ao valor de *n* (*m*=*n*), em um cenário onde a rede Dexter estaria completa, com todos os recursos RDF mapeados em átomos, e verificações para um determinado recurso, mapeado para um átomo contido na última posição da rede, fossem necessárias. A Tabela 6 mostra os resultados dos testes de desempenho realizados para aferir a viabilidade da operação *Map RDF to Dexter*.

Tabela 6 – Tempos de processamento da operação Map RDF to Dexter

Complexidade O(n)			
Triplas	**Nós RDF**	**Nós Dexter**	**Tempo (ms)**
100	87	22	110
500	407	288	203
2500	2356	1738	1120
12500	12128	10157	4936
62500	61805	49328	38580

A Tabela 6 apresenta, para quantidades variadas de triplas, o número de nós RDF e o número de nós Dexter resultante do mapeamento por meio da operação *Map RDF to Dexter*. Na última coluna da tabela é possível observar o custo de tempo para realizar a tradução do modelo RDF para o modelo Dexter.

É perceptível que, assim como a operação *Filter RDF Description*, a operação *Map RDF to Dexter* diminui o tamanho da rede a ser tratada pelas operações restantes na sequência de atividades. Dessa forma, a operação *Map RDF to Dexter* contribui para um melhor desempenho em termos de tempo de resposta, considerando todo o processo. Nesta Seção, foram mostrados os testes realizados "com e sem" a operação *Filter RDF Description* para demonstrar que o processo geral se beneficia da utilização de tal operação. No caso da operação descrita nesta seção (*Map RDF to Dexter*), não é possível testar o comportamento do processo geral sem utilizá-la, pois todo o processo gira em torno da tradução entre modelos efetuada por *Map RDF to Dexter*. Para realizar um experimento nesses moldes, seria necessário reimplementar as últimas três camadas do modelo, tanto no Serviço Web quanto na biblioteca Javascript. Um trabalho que exigiria bastante esforço de desenvolvimento.

8.3 OPERAÇÃO DISCOVER KNOWLEDGE

O objetivo da operação *Discover Knowledge* é, primeiramente, determinar o átomo mais relevante da rede Dexter, recebida como parâmetro de entrada, e, em seguida, recuperar novos conhecimentos a partir de buscas realizadas na Web de Dados e na Web 2.0. Essas buscas são procedidas por acessos externos executados sobre a Internet. Logo, o tempo entre as requisições e suas, respectivas, respostas não serão considerados nesta avaliação. A suboperação de determinação do átomo mais relevante precisa analisar cada átomo da rede Dexter em termos do grau do nó. Ou seja, para cada átomo da rede, verifica-se a quantidade de vizinhos diretamente ligados ao átomo. No final do processo, o átomo com o maior grau será eleito como átomo mais relevante. Feito isso, em dado momento, é necessário recuperar os recursos ligados diretamente ao átomo mais relevante pelos relacionamentos do tipo *sameAs* e *rdfs:type*. Para realizar essa tarefa é necessário percorrer todos os links do átomo mais relevante.

As suboperações que utilizam acessos externos, bem como, as que utilizam operações nativas da linguagem de programação usada no desenvolvimento, serão consideradas desprezíveis, como pode ser visto na Tabela 7.

Tabela 7 – Trabalho realizado pelas suboperações de Discover Knowledge

Operação	Trabalho
getRepresentAtom(dexterNet)	n
getSameAs(triples)	k
getType(triples)	k
dereferenceAtom(atomURI)	1
followSameAs(URIs)	1
getResourcesByType(type)	1
getResources2.0(URIs, APIs)	1
Trabalho Total	**n + k + k + 1 + 1 + 1 + 1**
Complexidade	**O(n)**

De acordo com a Tabela 7, é possível notar que a operação *Discover Knowledge* possui complexidade O(n), uma vez que a suboperação *getRepresentAtom(dexterNet)* analisa cada átomo da rede perfazendo um custo de *n*. A rede Dexter oferece a facilidade de descoberta do grau do átomo a partir da invocação de um único método nativo. Por esse motivo, não há a necessidade de verificação de cada ligação de entrada e de saída, simplificando o trabalho. As suboperações *getSameAs(triples)* e *getType(triples)* possuem custo *k*. Isso porque, para recuperar os recursos ligados diretamente ao átomo mais relevante por esses tipos de relacionamentos, é necessário analisar todos os links de saída desse átomo. No pior caso, grafo fortemente conectado, *k=(n-1)*. Considera-se aqui um grafo fortemente conectado, aquele em que cada nó está ligado a todos os outros nós por, no máximo, uma aresta. Por exemplo, se em um grafo fortemente conectado apenas um recurso se relacionar com outro por mais que um único tipo de relacionamento, então *k=n*. O que não afetaria a complexidade final dessa operação.

Não faz sentido executar testes de desempenho em termos de tempo de resposta para a operação *Discover Knowledge*. Como dito anteriormente, os resultados desta operação são diretamente dependentes de acessos externos a fontes da Web de Dados e APIs da Web 2.0. Por esse motivo os tempos obtidos nos testes podem variar consideravelmente a depender das condições da rede.

8.4 OPERAÇÃO PREPARE INTERFACE

O objetivo da operação *Prepare Interface* é criar estruturas especializadas para facilitar a construção de interfaces focadas nas necessidades de apresentação e navegação dos usuários sem experiência

com o ambiente *Linked Data*. O desempenho geral desta operação é dado pela performance na criação da estrutura de maior complexidade, definida na camada de preparação de interface do modelo de apresentação e navegação, apresentada na Seção 5.4. Uma vez que cada estrutura possui uma complexidade de criação diferente, os tempos de resposta na construção de tais estruturas serão diferentes.

A estrutura de maior complexidade definida no modelo proposto é a estrutura General Presentation, que utiliza as estruturas de menor complexidade – como Box, List, Gallery, entre outras – para gerar um *frame* único contendo vários tipos de interfaces apropriadas para satisfazer as necessidades de apresentação e navegação do usuário comum. Logo, a complexidade da estrutura General Presentation corresponde a soma das complexidades de criação das estruturas menores que a compõem. A Tabela 8 mostra a complexidade do trabalho de criação das estruturas mais utilizadas pelo Serviço Web descrito na Seção 6.1.

Tabela 8 – Trabalho realizado pelas suboperações de Prepare Interface

Operação	Trabalho
createBox(atom)	n
createList(atom)	m
createInList(atom)	k
createOutList(atom)	k
createGallery(atom)	n
createNews(atom)	n
createGeneral Presentation(net)	n + m + k + K + n + n
Trabalho Total	**n + m + k + K + n + n**
Complexidade	**O(n)**

Portanto, considerando os dados contidos na Tabela 8, a operação *Prepare Interface* possui complexidade O(n), uma vez que a complexidade de *createGeneralPresentation(net)* é a soma das complexidades de todas as suboperações menores. A suboperação *createBox(atom)* tem custo *n*. Neste caso, *n* refere-se a quantidade de vizinhos do átomo recebido como entrada. Ou seja, para criar um Box é necessário recuperar todos os átomos ligados diretamente ao átomo referencial (recebido como parâmetro de entrada da operação) (Seção 5.4). O custo da suboperação *createList(atom)* é *m*. Aqui, *m* corresponde a quantidade de vizinhos ligados ao átomo recebido como entrada (átomo referencial) pelo tipo de relacionamento definido para a List. Considerando sempre o mesmo átomo recebido como entrada das operações, *m* será um valor menor que *n*, pois em uma List os vizinhos são filtrados pelo tipo de relacionamento. Da mesma forma, a suboperação *createInList(atom)* e *createOutList(atom)* possuem custo *k*. Logicamente, o valor de *k* será menor que *m*, pois em uma List a quantidade de vizinhos corresponde a relacionamentos de entrada e saída. No entanto, em uma inList ou outList os relacionamentos só podem ser

de entrada (*in*) ou de saída (*out*), respectivamente. As suboperações *createGallery(atom)* e *createNews(atom)* possuem custo *n*, onde *n* representa a quantidade de recursos de mídia relacionados ao átomo recebido como entrada. A Tabela 9 mostra os testes de desempenho na aplicação da operação *Prepare Interface* na criação de uma General Presentation (suboperação de maior complexidade) e uma inList (uma das suboperações de menor complexidade).

Tabela 9 – Tempos de processamento da operação Prepare Interface

General Presentation	
Complexidade O(n)	
Triplas	Tempo (ms)
100	203
500	297
2500	1982
12500	3645
62500	28522

inList	
Complexidade O(n)	
Triplas	Tempo (ms)
100	11
500	23
2500	188
12500	783
62500	8255

É possível observar na Tabela 9 os tempos de resposta na construção de dois tipos de estruturas suportadas pelo Serviço Web (Seção 6.1). Para criar uma General Presentation a suboperação *createGeneralPresentation(net)* utiliza as suboperações de criação de estruturas de menor complexidade. A soma dos custos dessas operações menores produz o custo da suboperação *createGeneralPresentation(net)*, que é considerado o custo total da operação *Prepare Interface*. A Tabela 9 também apresenta o custo da suboperação *createInList(atom)*, uma das suboperações usadas pela operação *createGeneralPresentation(net)*.

8.5 OPERAÇÃO PRESENTS NETWORK

O objetivo da operação *Presents Network* é criar interfaces para prover apresentação e navegação dos dados *Linked Data*. Essas interfaces (construções) são elaboradas a partir das estruturas obtidas pela invocação da operação *Prepare Interface*, descrita na Seção 8.4.

A montagem dessas interfaces envolve a criação dinâmica de código HTML no lado cliente das aplicações. No caso do protótipo construído para demonstrar a viabilidade do modelo apresentado nesta dissertação, as interfaces são construídas por meio da biblioteca Javascript desenvolvida como parte da proposta e descrita na Seção 6.2. O custo de construção de tais interfaces é reduzido se comparado as outras

operações apresentadas neste capítulo. Isso porque a criação dinâmica de código HTML é realizada pela linguagem Javascript em combinação com o framework JQuery, o que torna o processo bastante ágil. Por esse motivo, e pelo fato de que a avaliação dos tempos de resposta a partir da criação dinâmica de código HTML em páginas da Web não possui precisão comprovada, foi decidido não medir o tempo de resposta na construção das interfaces suportadas pela operação *Presents Network*.

Com relação ao calculo da complexidade, as interfaces são criadas com base nas estruturas fornecidas pela operação *Prepare Interface*, como dito anteriormente. Portanto, a complexidade de criação de um tipo de interface é igual ou menor que a complexidade de criação da estrutura correspondente, usada para dar origem a tal interface. Por exemplo, para construir uma estrutura Box é necessário obter todos os átomos ligados diretamente ao átomo referencial. Ou seja, custo *n*, onde *n* é a quantidade de vizinhos do átomo referencial. A partir da estrutura Box é possível construir uma interface semelhante a uma lista de recursos com suporte a navegação facetada e apresentação por *tooltips* (Seção 5.5). Para construir a lista, é necessário percorrer todos os átomos relacionados ao átomo referencial do Box usado como base. Ou seja, custo *n*, igual ao custo da construção do Box. A geração da lista em formato de facetas e a disponibilização de *tooltips* dinâmicos é um processo realizado pelo framework JQuery (Silva, 2010).

8.6 CONSIDERAÇÕES FINAIS

De acordo com as avaliações realizadas sobre as operações disponibilizadas pelo modelo de apresentação e navegação (Capítulo 5) e implementadas pelo Serviço Web e pela biblioteca Javascript (Capítulo 6), o custo de execução é linear (O(n)). Ou seja, uma pequena quantidade de processamento é realizada para cada elemento da entrada de dados dessas operações.

Os testes de desempenho em termos de tempo de resposta foram realizados sem considerar os acessos externos a Web de Dados e Web 2.0, efetuados pela operação *Discover Knowledge* correspondente a camada de descoberta e aconselhamento. Esses acessos podem ser considerados, de certa forma, o gargalo de processos que realizam consultas a Web de Dados e Web de Documentos. Por exemplo, as buscas realizadas sobre fontes da Web de Dados ainda levam um tempo considerável se comparadas as buscas efetivadas na Web tradicional. Esse cenário tende a melhorar com o desenvolvimento de novas técnicas de recuperação de *Linked Data* e à medida que esses princípios forem aceitos em definitivo pelas pessoas e instituições.

Capítulo

9

Este capítulo apresenta as conclusões desta dissertação de mestrado, abordando as suas contribuições, limitações e problemas encontrados durante o trabalho. Além disso, são descritos os trabalhos futuros e as próximas atividades que podem ser realizadas com o objetivo de aperfeiçoar e desenvolver o modelo de apresentação e navegação para Linked Data proposto aqui, bem como as peças produzidas nesta pesquisa.

9 CONCLUSÕES

Este trabalho tratou a carência de métodos e processos relacionados à apresentação e navegação de *Linked Data*, com foco no usuário sem experiência. O objetivo principal do trabalho foi a construção de um modelo de apresentação e navegação de *Linked Data*, projetado para permitir que pessoas comuns possam explorar dados estruturados, oriundos da Web de Dados, de maneira adequada a sua condição de usuário não técnico. O trabalho foi realizado seguindo uma sequência lógica de atividades definidas no início da pesquisa que envolveu: (i) estudos da área de interfaces utilizadas na Web Semântica, com foco na apresentação e navegação; (ii) estudos de modelos de referência de hipertexto; (iii) estudos da utilização da estratégia de dados estruturados embutidos em páginas Web; (iv) projeto e confecção do modelo de apresentação e navegação para *Linked Data*; (v) implementação do modelo proposto; e (vi) utilização e avaliação da implementação por meio de uma ferramenta direcionada para o usuário comum.

O restante deste capítulo aborda, na Seção 9.1, os problemas encontrados durante a pesquisa e a maneira de tratá-los mais adequadamente; na Seção 9.2, as contribuições; na Seção 9.3, algumas limitações das soluções propostas aqui; e na Seção 9.4, uma discussão dos trabalhos futuros que podem ser realizados para aperfeiçoar a proposta desta pesquisa.

9.1 PROBLEMAS

Durante o desenvolvimento do trabalho, alguns problemas característicos do ambiente *Linked Data* foram encontrados e tratados. Entre esses problemas estão o Problema do *Blank Node* e o Problema da Ancoragem. Tais problemas foram tratados em artigo acadêmico publicado no WebMedia 2013 (produto desta dissertação) (Carlomagno et al., 2013).

O problema do *blank node* se refere ao tratamento dos nós não identificados do grafo RDF (um *blank node* é um nó que não possui URI). Esses nós são produzidos automaticamente por sistemas que manipulam as descrições RDF. Por exemplo, em um cenário de apresentação e navegação, para lidar com dados RDF embutidos em páginas Web por meio do padrão RDFa, é necessário tratar os *blank nodes*, pois não é interessante apresentar um recurso não identificado para o usuário comum.

O problema da ancoragem, também tratado no artigo (Carlomagno et al., 2013), diz respeito a como determinar o melhor elemento HTML de uma página Web para ancorar informações relacionadas a um dado recurso *Linked Data*. Em páginas Web contendo marcações RDFa, recursos *Linked Data* e suas informações relacionadas podem estar espalhados pelo código HTML e, portanto, podem ser exibidos em diferentes regiões da página. Logo, o melhor elemento para ancorar um conjunto de informações relacionadas a um recurso contido na página, é aquele que guarda a informação mais representativa de tal recurso. Por exemplo, se o tipo do recurso é "Pessoa", então a informação mais representativa pode ser o nome da pessoa, ou, na falta do nome, o email. Portanto, o elemento HTML da página que armazena essa informação deve ser escolhido, e o conjunto de informações acerca do recurso, ancorado a esse elemento. Assim, é possível usar tipos de interfaces para apresentar as informações do recurso de forma organizada como, por exemplo, *tooltips*.

Outros problemas encontrados durante o decorrer dos trabalhos, além daqueles descritos acima, foram os problemas da Identificação do Objeto e da Rotulagem de Recursos e Relacionamentos. Esses dois problemas são provocados pelo mesmo fator: a lentidão nas respostas diante das requisições externas efetuadas contra as fontes da Web de Dados.

O problema da identificação do objeto está relacionado à necessidade de determinar se o objeto de uma tripla RDF é um recurso ou um literal. O padrão RDF fornece recursos para que essa informação possa constar na própria descrição RDF. Nesse caso, o problema é facilmente resolvido, pois determinar o tipo de objeto da tripla é uma operação simples e rápida. No entanto, a maioria das descrições RDF contidas na Web de Dados não possuem as informações de tipificação de objetos. Portanto, para determinar se um objeto é recurso ou literal, é necessário, em último caso, realizar um acesso ao URI definido no objeto

(requisição do tipo GET HTTP) e analisar a resposta. Caso a resposta esteja no formato RDF, isso significa que o objeto é do tipo recurso. Algumas estratégias podem ser usadas para evitar o acesso externo no ato da identificação. Por exemplo, é possível verificar se o conteúdo do objeto é um URI bem formado. Caso não seja, não há possibilidade do objeto ser um recurso.

O problema da rotulagem de recursos e relacionamentos se refere ao fato da necessidade de rotular os recursos e relacionamentos no momento da apresentação dessas entidades para o usuário final. Exibir tais elementos, pura e simplesmente, por meio de URIs não é a forma mais adequada de apresentação para essa ocasião. A grande maioria das descrições RDF de recursos *Linked Data* possuem uma ou mais triplas definindo rótulos. O problema é que, de posse apenas do URI, é necessário realizar uma requisição externa para obter o rotulo do recurso (que nesse contexto pode ser também um relacionamento). No contexto da apresentação e navegação de *Linked Data*, é comum lidar com grafos RDF contendo centenas e até milhares de triplas associando diferentes recursos por meio de diferentes relacionamentos. Portanto, realizar uma requisição externa para cada recurso e relacionamento presente no grafo RDF que deve ser apresentado, é uma tarefa que pode se tornar inviável, do ponto de vista da espera pelas respostas. Uma maneira de evitar os acessos externos é definir rótulos automaticamente com base no URI da entidade. Ou seja, a partir do URI é possível dar origem a um rótulo. Por exemplo, baseado no URI *http://dbpedia.org/resource/Barack_Obama* é possível criar o rótulo "*Barack Obama*".

9.2 CONTRIBUIÇÕES

A principal contribuição deixada por este trabalho é o modelo de apresentação e navegação de *Linked Data* focado no usuário comum. O modelo foi concebido como uma extensão da camada de execução do modelo Dexter. O modelo proposto define atividades, propriedades e operações que podem ser usadas para padronizar a construção de sistemas no domínio da apresentação e navegação de *Linked Data*. O modelo foi projetado de forma a permitir que novas contribuições possam ser realizadas em termos de aperfeiçoamentos e sofisticações da proposta. O modelo é genérico e independente de interfaces. Ou seja, o modelo proposto nesta dissertação pode ser usado para gerar interfaces diversas no suporte a apresentação e a navegação de dados ligados.

Como contribuições secundárias, o trabalho deixa um Serviço Web desenvolvido no padrão REST, que implementa as quatro primeiras camadas do modelo proposto. Por meio do serviço é possível recuperar estruturas especializadas que podem ser usadas na construção de interfaces de apresentação e

navegação. Essas estruturas são desenhadas para facilitar a criação de tais interfaces, aprimorando a interação do usuário comum, com o objetivo de satisfazer suas necessidades de informação.

Além do Serviço Web, este trabalho desenvolveu uma biblioteca Javascript, com base no *framework* JQuery, implementando a última camada do modelo proposto (camada de apresentação e navegação). A biblioteca oferece uma interface de acesso ao serviço, além de fornecer métodos prontos para gerar interfaces de apresentação e navegação com base nas estruturas disponibilizadas pelo mesmo. A biblioteca também pode extrair *Linked Data* implícito em páginas Web por meio de marcações RDFa. As interfaces fornecidas pela biblioteca são baseadas em listas, facetas e *tooltips*, amplamente aceitas pela comunidade de desenvolvimento Web.

Para demonstrar a utilização do Serviço Web, bem como, da biblioteca Javascript, foi projetada uma ferramenta direcionada para o usuário final. O protótipo desenvolvido nesta pesquisa é capaz de extrair triplas RDF embutidas nas páginas Web visitadas pelos usuários, provendo apresentação e navegação adequadas nesses dados. O protótipo foi construído como uma extensão para o navegador Web Chrome. A extensão pode sinalizar a existência de marcações RDFa nas páginas acessadas pelo navegador por meio de um ícone. Essa tarefa é realizada pela biblioteca Javascript. A partir da interação do usuário, a extensão exibe os dados extraídos com suporte a apresentação e navegação (tarefa realizada pelo Serviço Web com o advento da biblioteca). O objetivo da ferramenta é demonstrar a viabilidade do modelo proposto.

9.3 LIMITAÇÕES

Analisando a proposta desta dissertação, é possível identificar algumas limitações presentes no modelo de apresentação e navegação de *Linked Data* idealizado aqui. Essas limitações servem de motivação para a realização de esforços futuros com o objetivo de aperfeiçoar o trabalho realizado.

Na camada de filtragem analítica, a identificação de metatriplas e triplas redundantes ainda é dependente das estratégias oferecidas pelas fontes de dados (Seção 5.1). Caso a fonte consultada não forneça estratégias para acelerar o processo de identificação, é necessário recuperar o vocabulário (RDFS ou OWL) contendo o termo pesquisado. A recuperação de vocabulários para muitos termos torna o processo extremamente lento por conta das questões envolvendo a transmissão de dados via rede.

Na camada de mapeamento, a identificação do tipo de objeto contido nas triplas RDF é dependente da presença dessa informação na descrição RDF dos recursos. Embora esse problema tenha sido tratado na Seção 9.1, em certos casos é impossível determinar o tipo de objeto sem realizar uma requisição

ao URI contido no mesmo. Isso aumenta o tempo de resposta na tradução entre modelos e, consequentemente, do processo geral. Além disso, a solução sugerida para o problema da rotulagem (também tratado na Seção 9.1) é dependente do formato do URI. Caso as fontes de dados disponibilizem URIs em formatos diversos, a geração de rótulos a partir desses URIs pode se tornar inconsistente.

O processo realizado pela camada de descoberta e aconselhamento é dependente de acessos externos a diversas fontes da Web de Dados e APIs da Web 2.0. É nessa camada que reside o gargalo do processo geral. Os acessos externos necessários para cumprir o papel da camada de descoberta e aconselhamento podem retardar as respostas e tornar a aplicação que utiliza o modelo ineficiente.

Outra limitação é que o Serviço Web, descrito na Seção 6.1, foi desenvolvido para trabalhar dentro do escopo da nuvem de dados do projeto LOD, mais precisamente, da fonte DBPedia. Portanto, as buscas realizadas pelo serviço estão limitadas a essa fonte de dados. Além disso, a biblioteca Javascript descrita na Seção 6.2, opera com interfaces baseadas em apenas três tipos de entidades da Web de Dados, Pessoa; Organização e Lugar. Entidades diferentes são consideradas desconhecidas e a geração de interfaces, nesse caso, pode se tornar inconsistente.

9.4 TRABALHOS FUTUROS

Baseado nas limitações pontuadas na seção anterior, alguns trabalhos futuros podem ser sugeridos como segue. Proceder a investigação de novas estratégias com o objetivo de definir abordagens mais adequadas para: (1) o processo de análise de triplas, visando identificar metatriplas e triplas redundantes de forma mais simples e prática; (2) o processo de análise de objetos das triplas, visando identificar o tipo de objeto (recurso ou literal) de maneira mais eficiente; (3) o processo de rotulagem de recursos e relacionamentos, visando recuperar rótulos predefinidos ou criar novos rótulos de modo mais apropriado; e (4) a tarefa de recuperação de dados provenientes das fontes da Web de Dados e das APIs da Web 2.0, com o propósito de reduzir o gargalo do processo geral diante dos acessos externos realizados durante o trabalho da camada de descoberta e aconselhamento.

Além disso, é importante expandir as soluções concebidas durante este trabalho para oferecer recursos mais atraentes, do ponto de vista do atendimento das necessidades do usuário comum: (a) expansão do Serviço Web para lidar com as entidades contidas em fontes de dados variadas, sem a necessidade de conhecer previamente os padrões utilizados em seus vocabulários para definição de recursos e conceitos; (b) expansão da biblioteca Javascript para compreender qualquer tipo de entidade contida da Web de Dados de

forma dinâmica e, dessa maneira, fornecer recursos de apresentação e navegação mais apropriados levando em conta essa informação; e (c) investigar novas abordagens de obtenção de dados estruturados, além da extração de triplas RDFa das páginas visitadas pelos usuários, com o propósito de aumentar a utilidade da biblioteca ampliando a maneira de adquirir e explorar *Linked Data*.

Com relação ao modelo, é interessante pensar em uma camada preocupada com a identificação do perfil do usuário. O objetivo dessa nova camada estaria relacionado com a determinação e tratamento do perfil do usuário comum, bem como, de suas preferências. As informações adquiridas na camada de perfil podem ser usadas, por exemplo, durante o trabalho realizado na camada de descoberta e aconselhamento para aprimorar as consultas externas com base nas preferências de tal indivíduo. Com o advento das informações de perfil e preferências, os resultados da descoberta estariam mais próximos da satisfação das necessidades das pessoas e o aconselhamento poderia ser mais bem direcionado nesse sentido.

Outro trabalho relevante é a expansão da camada de descoberta e aconselhamento por meio de uma máquina de inferências. Por intermédio de inferências é possível descobrir relacionamentos antes desconhecidos, novos recursos e informações relevantes acerca de entidades da Web de Dados.

Realizar uma reavaliação dos requisitos necessários para a construção de um modelo mais genérico para *Linked Data* pode se tornar um trabalho bastante promissor. Nesse caso, devem ser considerados requisitos para tratar mais amplamente as questões que envolvem esse ambiente, como autoria, reutilização, publicação, apresentação, navegação, entre outros. A partir desse trabalho é possível verificar se a utilização do modelo Dexter ainda se mostra uma solução viável, ou se a proposta de um modelo de referência para aplicações *Linked Data*, independente de outros modelos, seria uma solução mais adequada. Nesse contexto, uma tarefa interessante seria a de avaliar se o modelo Dexter impôs alguma limitação à criação do modelo de apresentação e navegação de *Linked Data* e, em caso positivo, verificar se tais limitações podem ser anuladas ou reduzidas com a proposta de um modelo independente.

REFERÊNCIAS

ADAMCZYK, P.; SMITH, P.; JOHNSON, R.; HAFIZ, M. **REST and Web Services: In Theory and in Practice.** Chapter 2. New York: Springer, ISBN: 9781441983039, 1441983031. Editors: Erik Wilde and Cesare Pautasso, 2011.

ADIDA, B. **hGRDDL: Bridging microformats and RDFa.** Web Semantics: Science, Services and Agents on the World Wide Web, v. 6, n. 1, p. 54-60, 2008.

ADIDA, B.; HERMAN, I.; SPORNY, M.; BIRBECK, M. **RDFa Primer**, Disponível em http://www.w3.org/TR/xhtml-rdfa-primer/ (último acesso em 28 de abril, 2013), 2012.

ARAÚJO, R.; SOUZA, J. **Aumentando a Transparência do Governo por Meio da Transformação de Dados Governamentais Abertos em Dados Ligados.** Revista Eletrônica de Sistemas de Informação. Edição Temática sobre Governo Eletrônico. ISSN 1677-3071, 2011.

AUER, S.; BIZER, C.; KOBILAROV, G.; LEHMANN, J.; IVES, Z. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the* 6th Int'l Semantic Web Conference, Busan, Korea, 2007.

BARTH, A.; FELT, A.; SAXENA, P.; BOODMAN, A. **Protecting Browsers from Extension Vulnerabilities.** In NDSS, 2010.

BECHHOFER, S.; VAN HARMELEN, F.; HENDLER, J.; HORROCKS, I.; MCGUINNESS, D.; PATEL-SCHNEIDER, P.; STEIN, L. **OWL Web Ontology Language Reference.** W3C Recommendation 10 February 2004. Series editor: Brian McBride and Guus Schreiber, 2004.

Berners-Lee, T. **What the Semantic Web Can Represent.** Disponível em http://www.w3.org/DesignIssues/RDFnot.html/ (último acesso em 15 de agosto, 2013), (1998).

BERNERS-LEE, T.; CHEN, Y.; CHILTON, L.; CONNOLLY, D.; DHANARAJ, R.; HOLLENBACH, J.; LERER, A; SHEETS, D. **Tabulator: Exploring and Analyzing Linked Data on the Semantic Web.** In: 3rd International Semantic Web User Interaction Workshop, 2006.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. **The Semantic Web. Scientific American.** 284(5):34-44, Mai 2001. Disponível em http://dx.doi.org/10.1038/ (último acesso em 15 de agosto, 2013) scientificamerican0501-34DOI, 2001.

BERNERS-LEE, T. **Design Issues: Linked Data.** Disponível em http://www.w3.org/DesignIssues/LinkedData.html/ (último acesso em 15 de agosto, 2013), 2006.

BERNERS-LEE, T.; HOLLENBACH, J.; LU, K.; PRESBREY, J.; PRU D'OMMEAUX, E.; SCHRAEFEL, M.C. **Tabulator Redux: Writing Into the Semantic Web.** Disponível em http://eprints.soton.ac.uk/264773/1/tabulatorWritingTechRep.pdf (último acesso em 03 de outubro, 2013). 2007.

BEVAN, N.; MACLEOD, M. **Usability measurement in context.** Behaviour and Information Technology, 13, 132-145, (1994).

BIZER, C.; CYGANIAK, R.; HEATH, T. **How to publish linked data on the Web.** Disponível em http:Ilwww4.wiwiss.fu-berlin.de/bizerlpublLinkedDataTutorial/ (último acesso em 03 de outubro, 2013), 2007.

BIZER, C.; HEATH, T.; IDEHEN, K; BERNERS-LEE, T. **Linked data on the web (LDOW2008).** In Proceedings of the 17th international conference on World Wide Web(WWW '08). ACM, New York, NY, USA, 1265-1266, 2008.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. **Linked Data – The Story So Far.** Int. J. Semantic Web Inf. Syst., 5(3):1-22, 2009.

BIZER, C.; MÜHLEISEN, H.; HARTH, A.; STADTMÜLLER, S.; MEUSEL, R.; SCHUHMACHER, M.; VÖLKER, J.; ECKERT, K. **Web Data Commons Extraction Report.** Disponível em http://webdatacommons.org/2012- 08/stats/stats.html (último acesso em 03 de outubro, 2013), 2012.

BRICKLEY, D.; GUHA, R. **RDF Vocabulary Description Language 1.0: RDF Schema.** W3C Recommendation 10 February 2004. Series editor: Brian McBride, 2004.

BRUNETTI, J.; AUER, S.; GARCÍA, R. **The Linked Data Visualization Model.** In Proceedings of the International Semantic Web Conference (Posters & Demos), 2012.

CAMARDA, D.; MAZZINI, S.; ANTONUCCIO A. **LodLive, exploring the web of data.** In Proceedings of the 8th International Conference on Semantic Systems (I-SEMANTICS '12), Harald Sack and Tassilo Pellegrini (Eds.), 2012.

CAMPBELL, B.; GOODMAN, J. **HAM: a general purpose hypertext abstract machine.** ACM 31, 7 (July 1988), 856-861. DOI=10.1145/48511.48515 http://doi.acm.org/10.1145/48511.48515, 1988.

CARLINI, N.; FELT, A.; WAGNER, D. **An evaluation of the google chrome extension security architecture.** In Proceedings of the 21st USENIX Conference on Security, 2012.

CARLOMAGNO, A.; FILHO, T.; CERQUEIRA, M.; PRAZERES, C. **jRDFa: browsing and visualization of linked data on the web.** In Proceedings of the 19th Brazilian symposium on Multimedia and the web (WebMedia '13). ACM, New York, NY, USA, 141-148, 2013.

CHENG, G.; WU, H.; GONG, S.; ZHANG, H.; QU, Y. **Browsing Linked Data with MyView.** In Proceedings of The 10th International Semantic Web Conference (ISWC2011), 2011.

CLARK, K.; FEIGENBAUM, L.; TORRES, E. **SPARQL protocol for RDF.** Disponível em http://www.w3.org/TR/rdf-sparql-protocol/ (último acesso em 15 de agosto, 2013), 2008.

CLARKSON, E.; NAVATHE, S.; FOLEY, J. **Generalized Formal Models for Faceted User Interfaces.** In Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries (JCDL '09). ACM, New York, NY, USA, 125-134, 2009.

CONKLIN, J. **Hypertext: An Introduction and Survey.** Computer 20, 9 (September 1987), 17-41. DOI=10.1109/MC.1987.1663693 http://dx.doi.org/10.1109/ MC.1987.1663693, 1987.

CONKLIN, J.; BEGEMAN, M. **gIBIS: A tool for all reasons.** In Proceedings of the Journal of the American Society for Information Science, 40, 200-213, 1989.

COSTA A.; YAMATE F. **Semantic Lattes: Uma Ferramenta de Consulta Baseada em Ontologia.** Trabalho de graduação em Engenharia da Computação – Escola Politécnica. IME/USP, 2009.

CUNHA, D.; LÓSCIO, B.; SOUZA, D. **Linked Data: da Web de Documentos para a Web de Dados.** In Escola Regional de Computação Ceará, Maranhão e Piauí (V Ercemapi), 2011.

DADZIE, A.; ROWE, M. **Approaches to visualising Linked Data: A survey.** In Proceedings of Semantic Web Journal, Volume 2, DOI - 10.3233/SW-2011-0037, 2011.

DAVIES, S.; HATFIELD, J.; DONAHER, C.; ZEITZ, J. **User Interface Design Considerations for Linked Data Authoring Environments.** In Proceedings of LDOW, 2010.

DING, L.; LEBO, T.; ERICKSON, J.; DIFRANZO, D.; WILLIAMS, G.; LI, X.; MICHAELIS, J. **Publishing - TWC LOGD: A portal for linked open government data ecosystems.** Web Semantics: Science, Services and Agents on the World Wide Web, 9(3), 325–333. doi:10.1016/j.websem.2011.06.002, 2011.

ENGELBART, D. **Augmenting human intellect: A conceptual framework.** Tech. Rep. AFOSR-3223, Contract AF 49(638)-1024, Stanford Research Institute Technical Report, Palo Alto, CA, Oct, 1962.

ERICSON, J. **Net expectations - what a web data service economy implies for business.** Information Management Magazine, Jan/Feb, 2010.

FAGAN, J. **Usability Studies of Faceted Browsing: A Literature Review.** In Proceedings of Information Technology and Libraries, 2010.

FIELDING, R. **Architectural styles and the design of network-based software architectures.** Ph.D. thesis, University of California, Irvine, 2000.

FIONDA, V.; GUTIERREZ, C.; PIRRÓ, G. **Navigation - Semantic Navigation on the Web of Data : Specification of Routes, Web Fragments and Actions.** In Proceedings of the 21st international conference on World Wide Web (WWW '12), 2012.

FURUTA, R.; STOTTS, P. **Programmable browsing semantics in Trellis.** In Proceedings of the second annual ACM conference on Hypertext (HYPERTEXT '89). ACM, New York, NY, USA, 27-42. DOI=10.1145/74224.74227 http://doi.acm.org/10.1145/74224.74227, 1989.

GARCÍA, R.; BRUNETTI, J.; LÓPEZ-MUZÁS, A.; GIMENO, J.; GIL, R. **Publishing and interacting with linked data.** In Proceedings of the International Conference on Web Intelligence, Mining and Semantics (WIMS '11), 2011.

GOMAA, H. **Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures.** Cambridge University Press. 978-0-521-76414-8, 2000.

GRAVES, A. **Integrating and publishing public safety data using semantic technologies.** In Proceedings of the 11th Annual International Digital Government Research Conference on Public Administration Online: Challenges and Opportunities (dg.o '10). Digital Government Society of North America 193-199, 2010.

HALASZ, F. AND SCHWARTZ, M. **The Dexter Hypertext Reference Model.** In Proceedings of the NIST Hypertext Standardization Workshop, Gaithersburg, MD, 1990.

HARTH, A. **VisiNav: A System for Visual Search and Navigation on Web Data.** J. Web Semant. 8(4), 348–354, 2010.

HASTRUP, T.; CYGANIAK, R.; BOJARS, U. **Browsing Linked Data with Fenfire.** In: Linked Data on the Web (LDOW 2008) Workshop, in Conjunction with WWW 2008 Conference, 2008.

HEARST, A. **Uis for faceted navigation: Recent advances and remaining open problems.** In HCIR 2008: Proceedings of the Second Workshop on Human-Computer Interaction and Information Retrieval (pp. 13-17), 2008.

HEATH, T.; BIZER, C. **Linked Data Evolving the Web into a Global Data Space.** Morgan & Claypool, 1st edition, 2011.

HUDSON, M.; HUYNH, T.; LOWRY, K.; MAGUIRE III, J.; PATTERSON, E.; RADER, M.; TIKUNOVA, M. **U.S. Patent No. 6,828,988.** Washington, DC: U.S. Patent and Trademark Office, 2004.

HUYNH, D.; MAZZOCCHI, S.; KARGER, D. **Piggy Bank: Experience the Semantic Web Inside Your Web Browser.** Web Semant. 5, 1 (March 2007), 16-27, 2007.

KITCHENHAM, B. **Procedures for Performing Systematic Reviews.** NICTA Technical Report 0400011T.1, 2004.

KLYNE, G.; CARROLL, J. **Resource Description Framework (RDF): Concepts and Abstract Syntax.** W3C Recommendation 10 February 2004. Series editor: Brian McBride, 2004.

KOSSMANN, D. **A web of data: new architectures for new technology?.** In Proceedings of the 7th annual ACM international workshop on Web information and data management (WIDM '05). ACM, New York, NY, USA, 1-1, 2005.

LATIF, A.; AFZAL, M.; HOEFLER, P.; SAEED, A.; TOCHTERMANN, K. **Turning Keywords into URIs : Simplified User Interfaces for Exploring Linked Data.** In*Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human* (ICIS '09), 2009.

LEIVA, W. **Um modelo de hipertexto para apoio ao ensino mediado pela Web.** Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional, 2003.

LIU, L.; ZHANG, X.; YAN, G.; CHEN, S. **Chrome extensions: Threat analysis and countermeasures.** In Network and Distributed System Security Symposium (NDSS), 2012.

LUCZAK-RÖSCH, M.; HEESE, R. **Linked Data Authoring for Non-Experts.** In proceeding of: Workshop Linked Data on the Web (LDOW2009), At Madrid, Spain, Volume: Proceedings of the WWW09, Workshop Linked Data on the Web (LDOW2009), 2009.

MAGALHÃES, R.; MACEDO, J.; VIDAL, V. **Linked Data: Construindo um Espaço de Dados Global na Web.** In Simpósio Brasileiro de Sistemas Multimídia e Web (XVII WebMedia), 2011.

MENDES, P., JAKOB, M., GARCÍA-SILVA, A. AND BIZER, C. **DBpedia spotlight: shedding light on the web of documents.** In Proceedings of the 7th International Conference on Semantic Systems (I-Semantics '11), Chiara Ghidini, Axel-Cyrille Ngonga Ngomo, Stefanie Lindstaedt, and Tassilo Pellegrini (Eds.), 2011.

MITCHELL, T.; BETTERIDGE, J.; CARLSON, A.; HRUSCHKA, E.; WANG, R. **Populating the Semantic Web by Macro-reading Internet Text.** In Proceedings of the 8th International Semantic Web Conference (ISWC '09), Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan (Eds.). Springer-Verlag, Berlin, Heidelberg, 998-1002, 2009.

MÜHLEISEN, H.; BIZER, B. **Web Data Commons - Extracting Structured Data from Two Large Web Corpora.** In Proceedings of the 4th Linked Data on the Web Workshop LDOW2012, 2012.

NELSON, T. **Getting it out of our system In Information Retrieval: A Critical Review.** G. Schechter, Ed. Thompson Books, Washington, D.C., pp. 191– 210, 1967.

NIELSEN, J. **User Testing: interface usability and website design.** Disponível em http://www.nngroup.com/topic/user-testing/all/ (último acesso em 22 de novembro, 2013), 2010.

NIELSEN, J. **Usability inspection methods.** In Conference Companion on Human Factors in Computing Systems (CHI '94), Catherine Plaisant (Ed.). ACM, New York, NY, USA, 413-414, 1994.

NOWACK, B. **Paggr: Linked Data widgets and dashboards.** Semsol, Bielefelder Str. 5, 40468 Düsseldorf, Germany, 2009.

OREN, E.; DELBRU, R.; DECKER, S. **Extending Faceted Navigation for RDF Data.** In Proceedings of The Semantic Web - ISWC 2006. Lecture Notes in Computer Science Volume 4273, 2006, pp 559-572, 2006.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. **RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision.** In Proceedings of the 17th International World Wide Web Conference (WWW2008), Bejing, China, 2008.

PENG, Y.; MA, S.; LEE, J. **REST2SOAP: a Framework to Integrate SOAP Services and RESTful Services.** In Proceedings of Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference on, 2009.

PÉREZ, J.; ARENAS, M.; GUTIERREZ, C. **Semantics and complexity of SPARQL.** The Semantic Web - ISWC 2006. Lecture Notes in Computer Science, 2006, Volume 4273/2006, 30-43, DOI: 10.1007/11926078_3, 2006.

PIETRIGA, E.; BIZER, C.; KARGER, D.; LEE, R. **Fresnel: A Browser-Independent Presentation Vocabulary for RDF.** In Proceedings of the Semantic Web - ISWC 2006 Lecture Notes in Computer Science Volume 4273, 2006, pp 158-171, 2007.

PRAZERES, C.; LUCRÉDIO, D.; FORTES, R.; TEIXEIRA, C. **Uma Proposta de Navegação na Web Utilizando Facetas.** Passo Fundo: II Escola Regional de Banco de Dados. Disponível em http://www.upf.br/erbd/download/15641.pdf (último acesso em 28 de dezembro, 2013), 2006.

RICHARDSON, L.; RUBY, S. **RESTful Web Services: Web services for the real world.** O'Reilly Media, Inc. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2007.

RIXHAM, N.; BIRBECK, M.; HERMAN, I. **An API for extracting structured data from Web documents.** W3C Working Group Note 05 July 2012. Disponível em http://www.w3.org/TR/rdfa-api/ (último acesso em 15 de dezembro, 2013), 2012.

SAUERMANN, L.; CYGANIAK, R. **Cool uris for the semantic web - w3c interest group note.** Disponível em http://www.w3.org/TR/cooluris/ (ultimo acesso em 22 de novembro, 2013), 2008. 11, 13, 14, 15, 43, 46, 2008.

SHAKYA, A.; TAKEDA, H.; WUWONGSE, V. **Community-Driven Linked Data Authoring and Production of Consolidated Linked Data.** In Proceedings of International Journal on Semantic Web and Information Systems, 5(3), 2348, July-September 2009 23, 2009.

SILVA, M. **jQuery - A bíblia do programador JavaScript.** 2. ed.. Editora Novatec, ISBN: 978-85-7522-178-5, Editor: Rubens Prates, Editoração eletrônica: Carolina Kuwabata, 2010.

SWEOIG / TASKFORCES / COMMUNITYPROJECTS / LINKINGOPENDATA. **Linking Open Data W3C SWEO Community Project.** Disponível em http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData/ (último acesso em 13 de setembro, 2013).

TARASOV, D.; AKBEROVA, N.; IZOTOVA, E.; ALISHEVA, D.; ASTAFIEV, M.; FREITAS JR, R. A. **Optimal tooltip trajectories in a hydrogen abstraction tool recharge reaction sequence for positionally controlled diamond mechanosynthesis.** J. Comput. Theor. Nanosci, 7, 325-353, 2010.

TASKFORCES / COMMUNITYPROJECTS / LINKINGOPENDATA / DATASETS / STATISTICS. Disponível em http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics/ (ultimo acesso em 25 de agosto, 2013).

TRAN, T.; WANG, H.; RUDOLPH, S.; CIMIANO, P. **Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data.** In: 2009 IEEE International Conference on Data Engineering, pp. 405–416. IEEE Computer Society, Washington, DC, 2009.

THIRAN, P.; HAINAUT, J.; HOUBEN, G. **Database Wrappers Development: Towards Automatic Generation.** In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering* (CSMR '05). IEEE Computer Society, Washington, DC, USA, 207-216, 2005.

TUMMARELLO, G.; CYGANIAK, R.; CATASTA, M.; DANIELCZYK, S.; DELBRU, R.; DECKER, S. **Sig.ma: Live views on the Web of Data.** In Proceedings of Semantic Web Challenge 2009. User Interaction in Semantic Web research. Volume 8, Issue 4, November 2010, Pages 355–364, 2009.

YEE, K.; SWEARINGEN, K.; LI, K.; HEARST, M. **Faceted Metadata for Image Search and Browsing.** In Proceedings of CHI, 2003.

ZIEGLER, J. **Semantic web meets UI: context-adaptive interaction with semantic data.** In Proceedings of the 29th Annual European Conference on Cognitive Ergonomics (ECCE '11). ACM, New York, NY, USA, 15-16, 2011.