5
Conclusão

De acordo com tudo que foi pesquisado, estudado, analisado e desenvolvido nessa dissertação, esta área é dedicada exposição de toda a experiência adquirida e as contribuições geradas. Serão apresentados comentários sobre trabalhos similares feitos anteriormente e que de certa forma serviram de inspiração e base para esta dissertação, também idéias e possíveis melhorias sobre o que foi realizado.

5.1
Trabalhos relacionados

Nesta seção estão os trabalhos relacionados à dissertação. Trabalhos que, de alguma forma, têm uma relação conceitual ou prática.

Primeiramente apresentamos um modelo conceitual alternativo proposto por Parunak (PARUNAK, 1991) para hipermídia baseada em conjuntos, e em seguida exemplos de trabalhos já implementados que podem ser divididos em três categorias: trabalhos relacionados que induzem conjuntos a partir de um modelo, trabalhos que induzem facetas a partir de um RDF e trabalhos relacionados puramente à navegação facetada.

5.1.1
Hipermídia Baseada em Conjuntos

Hipermídia geralmente é descrita por nós de informação que possuem links entre si, sugerindo o modelo conceitual de grafo. Uma definição mais abrangente para hipermídia seria um sistema de nós de informação através do qual pessoas podem se mover não linearmente. Este tipo de definição permite tipos de implementação alternativos.

Modelos alternativos são necessários para exibição de uma tarefa particular onde a navegação entre os nós através de links explícitos é menos efetiva do que um modelo de interseção de conjuntos de nós.

A tecnologia hipermídia permite que os usuários se movam não linearmente entre informações relacionadas, dependendo do seu nível de interesse e necessidade. Tradicionalmente, a hipermídia é implementada de acordo com um modelo conceitual baseado em grafos. Isso significa que o usuário interpreta as informações como se estivessem em nós de um grafo e se move de um nó para outro através das arestas do grafo. Este modelo conceitual é apropriado para tarefas onde um nó explica, amplifica ou elucida outro. Por exemplo: uma palavra leva para sua definição; uma citação leva para o documento citado.

Para outros tipos de tarefas, um modelo conceitual distinto é mais apropriado, um modelo baseado na teoria de conjuntos. Este modelo facilita a manipulação de coleções de nós similares que estão associados a um ou mais conjuntos. Os usuários transitam de um nó para outro dentro de um mesmo conjunto, e de um conjunto para outro pelos nós que fazem parte do conjunto interseção destes conjuntos. Eles não interpretam os nós como estando ligados diretamente uns aos outros, mas em termos de a quais conjuntos eles pertencem. A implementação de um modelo desse tipo sobre um modelo hipermídia baseado em grafo é complexa demais, uma vez que grafos correspondentes necessitariam de arestas não direcionadas com aridade maior ou igual a dois.

Essa abordagem deve ser contrastada com dois outros usos de conjuntos em hipermídia:

- A teoria de grafos, por sua natureza, pode ser baseada na teoria de conjuntos, e diversas formalizações da hipermídia convencional baseada em grafos de fato utilizam formalismos de conjuntos (GARG, 1988 e HALASZ, 1990). Além disso, sistemas convencionais sempre consideram usuários se movendo de um nó de um grafo para outro, e não de um conjunto para outro através de um nó compartilhado por ambos ou de um nó para outro através de um conjunto em comum.

- Mesmo a nível conceitual, conjuntos podem ser empregados em diferentes problemas. Técnicas de agregação requerem a representação de conjuntos de nós como um nó de mais alto nível que pode ser tratado como uma entidade simples, e expandido quando necessário. Elementos dentro de um conjunto de agregação podem não ser distinguidos entre si

em seu tipo ou podem preencher *slots* específicos na agregação. Este uso de conjuntos para modelar agregações suporta um diferente requerimento cognitivo e resultados em um diferente conjunto de requerimentos para a interface.

5.1.1.1. Domínios apropriados

É interessante exemplificar a utilização de diferentes domínios de problemas que se encaixam nessa nova abordagem do modelo de hipermídia.

Um biólogo retornando de uma viagem de exploração encontrou 3 mil diferentes espécies de fungos, com anotações extensas onde cada uma descreve as dimensões físicas de colônia como um todo, a elevação onde foi encontrada, espécie de planta ou pedra ou solo onde a colônia estava crescendo, tipo de insetos que foram encontrados andando por cima de colônia, entre outras características. Então coloca todos os papéis sobre uma mesa e tenta agrupar as espécies similares e classificá-las.

Um lingüista estudando uma língua recentemente descoberta. Ele encontrou milhares de palavras em textos, exemplos de frases, informações como estruturas internas da língua, entre outros. Então ele ordena cada exemplo encontrado em diferentes pilhas, explorando como eles podem ser classificados baseados em funcionalidades como palavras de função compartilhada, assuntos de problemas, tipo de literatura, etc.

Um pesquisador organizando o uso de tecnologias avançadas na manutenção de serviços de consultoria, onde ele visita pequenas fábricas e realiza auditorias para otimizar o uso destas tecnologias pra melhorar a produção. Agora o pesquisador quer encontrar padrões no corpo de informações para concluir se há algum tipo de lição geral que pode ser aplicada. Ele precisa de um jeito de comparar, contrastar, agrupar estudos de caso.

O biólogo, o lingüista e o pesquisador estão realizando variações de tarefas de raciocínio taxonômico. Estas tarefas têm os seguintes tipos de características:

- Os objetos que estão sendo manipulados são muito similares entre si, o que torna a tarefa de organizá-los muito complexa. Os objetos dos

exemplos acima são fungos, frases ou estudos de caso. Em contraste, os nós de informações em uma rede hipermídia baseada em grafos são tipicamente diferentes tipos ou níveis de especificação. Por exemplo: um *paper* contendo termos técnicos e definições destes termos.

- Raciocínio taxonômico desenvolve descrições de cada item em um conjunto de dimensões, mas estas dimensões não estão completamente definidas quando alguma inicia o processo de raciocínio. Por exemplo: o biólogo pode perceber somente depois de um tempo que a presença ou a falta de uma subdivisão é uma relevante característica de discernimento do fungo em estudo.

- Características que necessitam ser atendidas para facilitar esses tipos de tarefas são essencialmente operações de conjuntos, como a ordenação de objetos em conjuntos baseados em suas características; procurar correlações dentre diferentes características; encontrar os diferentes conjuntos aos quais um item simples pertence; e gerar novos conjuntos a partir de conjuntos já existentes. Considerando uma espécie, o analista não pensa em mover para outra espécie, mas na relação entre esta espécie e um conjunto de outras. Para ajudar na comparação de diferentes conjuntos, é útil ter disponível um medidor de similaridade baseado no número de membros compartilhados.

5.1.1.2. Ferramentas para tarefas taxonômicas

5.1.1.2.1. Bancos de Dados

Bancos de dados relacionais substituíram os antigos cartões perfurados e oferecem operações sobre conjuntos e o potencial para computar correlações que o raciocínio taxonômico requer. De fato, alguns sistemas que utilizam banco de dados foram projetados especialmente para a recuperação de dados taxonômicos. Matematicamente, é simples modelar operações de conjuntos com um banco de dados relacional, e a tecnologia de banco de dados pode servir efetivamente como a camada de armazenamento de um sistema hipermídia, este estando por sua vez baseado em grafo ou conjuntos.

Entretanto, como interface para o usuário, bancos de dados convencionais têm 2 desvantagens:

- O modo de interação com um banco de dados é tipicamente através de linguagem de formulários ou *query*, onde o usuário descreve os itens desejados definindo restrições aos valores dos diversos campos. Essa abordagem é apropriada para alguns tipos de aplicações, porém geralmente quando um usuário está utilizando o raciocínio taxonômico, primeiramente ordena os nós por similaridade e então examina os vários campos para entender qual tipo de similaridade. Assim, este raciocínio requer a capacidade de designar e não somente de descrever; para apontar um item e associá-lo com outros itens, mais do que uma interface de *query*.

- Bancos de dados assumem que os campos podem ser definidos em um processo de análise separado, antes dos dados serem carregados e manipulados. Adicionar novos campos a um arquivo antigo normalmente requer a definição de uma nova estrutura de arquivo e a cópia do arquivo antigo nesta. O raciocínio taxonômico é muito mais eficiente se os campos puderem ser definidos dinamicamente e incrementalmente à medida que o raciocínio evolui.

5.1.1.2.2. Hipermídia baseada em grafos

Dado um ambiente hipermídia baseado em grafos que suporte links binários, pode-se construir uma primeira aproximação para o modelo baseado em conjuntos através da criação de um nó para representar cada conjunto e então linkar cada nó-conjunto para seus respectivos nós membros. Neste tipo de estrutura é difícil adaptá-la para que aceite operações de conjuntos. Na hipermídia convencional é simples visualizar nós que estão linkados a um dado conjunto, mas complexo de encontrar os que não estão. Não existe maneira dentro desse tipo de abordagem para definir operações entre pares de conjuntos, como: união e interseção.

Alguns estudos e implementações permitem a agregação de objetos que possibilitam que um grupo de nós seja manipulado como um simples nó. Por

exemplo, pode-se querer manipular nós que representam seções como um único capítulo, ou diversos capítulos como um único livro. Essas agregações não conseguem facilmente ser estendidas para atender necessidades cognitivas e operações de conjuntos, exemplo: encontrar as entidades do conjunto A que estão também no conjunto B, mas não no conjunto C.

5.1.1.2.3. HyperSets

Um usuário começa uma seção típica no HyperSet, modelo proposto por Parunak (PARUNAK, 1991), visualizando uma lista em uma janela com scroll. A lista contém ou todos os objetos de uma coleção ou todos os subconjuntos que foram definidos para esta coleção. Então o usuário seleciona um objeto e o browser segue com a interação.

Essa operação básica não é simplesmente de se mover de um nó para outro em uma arquitetura baseada em grafos, mas sim mover de um objeto para um dos conjuntos que ele é membro e então para algum outro membro deste conjunto.

Quando está em um artefato, o usuário pode adicioná-lo ou removê-lo de um conjunto. Quando está um conjunto o usuário pode remover qualquer objeto que é membro deste conjunto ou adicionar um objeto do universo a esse conjunto. Embora essas operações têm o mesmo efeito que adicionar origens e destinos para múltiplos links em um sistema baseado em grafo, a imagem cognitiva é diferente. No HyperSets não se pensa em nós que se pode alcançar a partir do nó atual e sim em um conjunto ao qual se deseja que este nó pertença.

HyperSets suporta um repertório completo de operações de conjuntos que geram novos conjuntos, incluindo união, interseção, complemento e diferença simétrica.

5.1.1.3. Comparação dos dois modelos

Um sistema baseado em grafos é uma tripla ordenada H = <N, A, L> de nós, âncoras e links, onde:

- $N = \{N_1...N_m\}$ é o conjunto de nós de informação.
- $A = \{A_{11}...A_{mn}\}$ é o conjunto de âncoras, ou sub-regiões dentro dos nós, tal que cada nó N_i tem uma ou mais âncoras $\{A_{i1}...A_{in}\}$.

- $L = <A_{ij}, A_{kl}>$ é um conjunto de links, ou par ordenados de âncoras.

A operação de browsing básica é a função *b: A Ω A* que mapeia a primeira âncora de um link para a segunda. Em sistemas de cartão, a segunda âncora é a sub-região de um nó que inclui o nó todo. Um possível efeito colateral desse mapeamento é mostrar ao usuário a âncora do range da função. A operação básica de autoria é a definição de links identificando duas âncoras. Isso significa que links são definidos explicitamente ao se caminhar pelo processo. Alguns dispositivos comuns para hipermídia baseada em grafos incluem mapas que mostram a disposição da rede de links e caminhos que identificam as sub-redes de uma topologia reduzida dentro de um grafo.

Já um modelo simples de hipermídia baseado em conjuntos é um par ordenado H = <N, S> de nós e conjuntos, onde:

- $N = \{N_1...N_m\}$ é um conjunto de nós de informação, exatamente como no modelo de grafo.
- $S \subseteq 2^N$ é um conjunto de subconjuntos de N.

A navegação neste tipo de modelo utiliza duas funções: *f: N Ω S* que mapeia um nó a um dos conjuntos a que pertence e *g: S Ω N* que mapeia um conjunto em um de seus membros. Essas funções geralmente são aplicadas em alternância. As operações básicas de autoria são habilitar e desabilitar um a pertinência de um nó em um conjunto e gerar novos conjuntos a partir de conjuntos já existentes através de operações de conjuntos. Dispositivos incluem a visualização de todos os nós em um conjunto; todos os conjuntos aos quais um nó pertence e medida de correlação entre os conjuntos.

Ambas as abordagens utilizam o mesmo conceito de nó, porém cada um requer algum tipo de alteração não requerida pela outra.

Para construir um sistema baseado em grafos a partir de um baseado em conjuntos, é necessário adicionar o conceito de âncora, assim múltiplos links podem originar de diferentes pontos em um nó.

Para construir um sistema baseado em conjuntos a partir de um baseado em grafos, é necessário permitir links com aridade arbitrária, e então à interação não leva um nó ao outro e sim a uma lista de nós acessíveis (conjunto), a partir do qual

o próximo nó pode ser escolhido. Também são necessários operadores para realizar operações de conjunto em pares destes links estendidos.

5.1.1.4. Implementação Híbrida

Tanto o modelo de grafos quanto o de conjuntos satisfazem diferentes tipos de tarefas, mas não são inconsistentes entre si. É desejável a implementação de ambos os modelos em um único sistema, com conjuntos manipulando coleções de nós similares e links juntando esses nós a outros (HyperSet é apenas um protótipo para explorar interfaces baseadas em conjuntos, não suporta links convencionais).

Ajustes simples nos modelos básicos em cada tipo de hipermídia permitem esse tipo de aplicação híbrida. Em particular, um sistema híbrido que pode suportar tanto operações em grafos quanto em conjuntos é uma tripla $H = <N, A, S>$ de nós, âncoras e conjuntos, onde:

- $N = \{N_1...N_m\}$ é um conjunto de nós de informações.
- $A = \{A_{11}...A_{mn}\}$ é um conjunto de âncoras, ou sub-regiões dentro dos nós, tal que cada nó N_i tem uma ou mais âncoras, $\{A_{i1}...A_{in}\}$ (aplicações baseadas em conjuntos restringem âncoras a serem nós completos).
- $S \subseteq 2^A$ é um conjunto de subconjuntos de A (aplicações baseadas em grafos restringem esses conjuntos para serem binários e ordenados).

Modelos abstratos existentes de hipertexto diferem no tipo de suporte que eles dão para os requerimentos. Por exemplo, (LANGE, 1990) requer que os links sejam binários e direcionados, enquanto (GARG, 1988) e Ham modelam os links como binários e sem direção; nenhum deles pode facilmente suportar sistemas baseados em conjuntos. O Dexter Reference Model (HALASZ, 1990), em contraste, é suficientemente rico para suportar HyperSet.

5.1.2
Conjuntos induzidos a partir de um modelo

5.1.2.1.
RDFReactor

Atualmente a maioria dos dados são armazenados em bancos de dados relacionais, documento e em dados semi-estruturados.

O RDF é um modelo de dados orientado a grafo, projetado para armazenar todos os tipos de modelos de dados. O RDF Schema atribui tipos a um nó de um grafo e associa-os às semânticas pretendidas. RDF armazena dados, estruturas de dados e associações de ambos. Para interligar RDF ao mundo Java, o RDFReactor (RDFR) utiliza dois passos importantes: primeiramente é criada uma API (RDF_2Go) para manipulação de triplas, depois as semânticas RDFS são mapeadas para semânticas Java.

O RDF_2Go (RDFR) permite que os desenvolvedores programem em API's que utilizam triplas para representar seus RDF's sem ter que decidir por uma implementação específica para a maioria das operações de modelos RDF. Pode-se estender o RDF_2Go para outros tipos de armazenamentos em triplas. Ele não possui nenhum tipo de estado interno, agindo simplesmente sobre as triplas armazenadas.

O módulo descrito acima é muito útil, mas ainda requer um conhecimento sobre os modelos de dados de um RDF. O RDFReactor facilita o uso de RDF por um desenvolvedor Java, permitindo seu acesso através de orientação a objetos. Classes Java são geradas automaticamente a partir de um RDF Schema. Basicamente as classes RDFS são mapeadas para classes Java e propriedades do RDFS são mapeadas para chamadas de métodos. O ciclo utilizado pelo RDFReactor por ser melhor visualizado pela imagem abaixo:

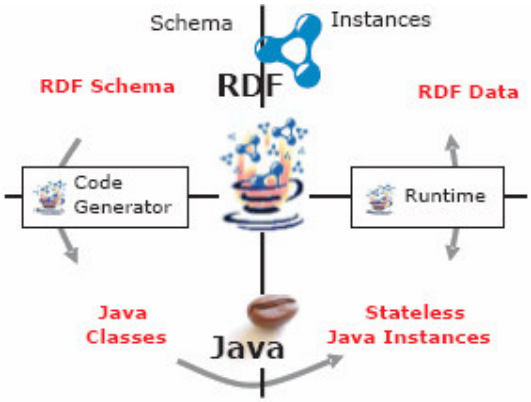

Figura 31: Ciclo de utilização do RDFReactor.

Para cada propriedade *p* no domínio *A* e do tipo *B* o seguinte conjunto de métodos é gerado:

- Public B getP() – retorna null, um valor simples ou então uma exceção RDFDataException se a propriedade tiver múltiplos valores.
- Public void setP(B value) – remove todos os valores existentes e conjuntos dado um valor.
- Public void removeP(B value) – remove o valor passado como parâmetro.
- Public void addP(B value) – adiciona o valor.
- Public B[] getAllP() – retorna todos os valores.

Todas as chamadas aos métodos acima resultam em manipulações imediatas ou consultas no modelo RDF_2Go. Todos os dados escritos no modelo são assegurados que estão escritos de acordo com o RDF Schema.

RDFReactor permite ao desenvolvedor que quando estiver programando sempre pense em objetos e não em triplas de dados. Ele pode acessar o RDF utilizando o vocabulário Java, por exemplo: utilizar person.setName(“Max”) ao invés de utilizar addTriple(personURI, nameURI, “Max”).

O RDFReactor interpreta o modelo de dados do RDF através de procedimentos de orientação a objetos e torna mais simples o uso de RDF para desenvolvedores que utilizam a linguagem Java.

Geralmente um RDF é definido através de triplas compostas por sujeito, predicado e objeto. Utilizando o RDFReactor um RDF pode ser definido através de objetos Java, assim como a leitura e escrita de dados RDF.

Também é possível utilizar o RDFReactor para gerar interfaces Java automaticamente a partir de um RDF Schema.

Segue um exemplo:

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix test: <http://purl.org/net/xamde/ns/test#> .
test:Person a rdfs:Class.
test:Name a rdfs:Property
;   rdfs:domain test:Person
;   rdfs:range rdfs:Literal
.
test:knows a rdfs:Property
;   rdfs:domain test:Person
;   rdfs:range test:Person
.
```

Quadro 30: Definição RDFS de uma classe no RDFReactor

Acima foi feita a definição da classe *Person* com dois atributos, *Name* que pertence ao domínio *Person* e seu tipo é *Literal*, ou seja, recebe um valor como um *string* por exemplo e a propriedade *knows* que pertence ao domínio *Person* e seu tipo é *Person*, ou seja, uma pessoa está relacionada a outra pessoa pelo atributo *knows*.

Após a definição do RDFS o RDFReactor deve ser configurado para que possa ser utilizado mais a frente.

```
// cria o modelo de dados Jena, o qual contém todos o dados em
// tempo de execução
Model datamodel = ModelFactory.createDefaultModel();
// carrega o mapeamento, gerado pelo InterfaceGenerator
Mapping m = new MemoryMapping("loaded", "./src/org/ontoware/rdfreactor/
example/mapping.properties");
// cria um objeto RDFReactor com o modelo e o mapeamento
// definidos
RDFReactor rr = new RDFReactor(datamodel, m);
```

Quadro 31: Configuração do RDFReactor

Por fim os objetos podem ser instanciados e utilizados para as interações necessárias e definidas pelo usuário.

```
Person p1 = (Person) rr.createInstance(Person.class, "http://www.example.com/ns
/2005/#person1");
Person p2 = (Person) rr.createInstance(Person.class, "http://www.example.com/ns
/2005/#person2");
Person p3 = (Person) rr.createInstance(Person.class, "http://www.example.com/ns
/2005/#person3");
p1.setName("Joe");
p1.addKnows(p2);
p1.addKnows(p3);
p2.setName("Jim");
p3.setName("Jon");

Person[] p1friends = p1.getAllKnows();
for (int i = 0; i < p1friends.length; i++) {
  System.out.println(p1.getName()+" knows "+p1friends[i].getName() );
}
```

Quadro 32: Exemplo de uso do RDFReactor

Nesse exemplo foram definidos três objetos do tipo *Person*, com os respectivos atributos *Name* definidos como: Joe, Jim e Jon, sendo que Joe conhece Jim e Jon. Então é criado um vetor com todos os amigos de Joe, no caso, Jim e Jon, que são impressos.

Esse sistema se assemelha a esta dissertação, pois da mesma forma a partir de uma definição feita em RDF gera as classes Java para serem utilizadas pelo usuário programador que por sua vez é capaz de utilizá-las para instanciar os objetos e manipulá-los através das peculiaridades de orientação a objeto. Porém não é capaz de realizar manipulações com estes objetos, como buscas e operações, só é possível criar instâncias e mudar seus atributos.

5.1.2.2. ActiveRDF

ActiveRDF (ACTIVERDF) é uma biblioteca para o acesso a repositórios RDF em programas implementados em Ruby. Pode ser utilizado como uma camada de dados no Ruby-on-Rails, similar ao ActiveRecord que provê mapeamentos para banco de dados relacionais). O ActiveRDF utilizado no Ruby-on-Rails permite a criação rápida de aplicações com web semântica e também disponibiliza uma DSL para o modelo RDF em questão, ou seja, é possível acessar recursos RDF, classes, propriedades, etc. programaticamente, sem o uso de queries.

Vejamos algumas considerações sobre o ActiveRDF:

- Pode ser utilizado com vários repositórios RDF e adaptadores para outros repositórios podem ser escritos rapidamente.
- Está em fase de desenvolvimento, portanto inúmeros bugs podem ser encontrados.
- Possui código aberto, está sob a licença LGPL.

Segue um exemplo:

```
NodeFactory.connection( :adapter => :yars, :host => 'browserdf.org' )

eyal = Person.create 'http://eyaloren.org/#me'
eyal.firstName = 'eyal'
eyal.lastName = 'oren'

armin = Person.create 'http://armin-haller.com/#me'
armin.firstName = 'armin'
armin.age = 30

eyal.knows ... [Andreas, Knud]
eyal.knows << armin
armin.save
eyal.save
```

Quadro 33: Exemplo de uso do ActiveRDF: criar e editar pessoas

No exemplo acima, primeiramente é definida a conexão com o repositório YARS e então criados dois objetos pessoa, *eyal* e *armin*. *Eyal* já conhece Andréas e Knud. Podemos dizer que uma nova pessoa que é conhecida de *eyal*, neste caso *armin*. Por fim são salvos os dois objetos para serem utilizados no próximo exemplo.

```
require 'activerdf'

eyal = IdentifiedResource.create 'http://eyaloren.org/#me'
eyal.class ... Person
eyal.firstName ... eyal
eyal.knows ... [Andreas, Knud, Armin]
eyal.knows[0].firstName ... Andreas
eyal.knows[0].age ... 30
```

Quadro 34: Exemplo de uso do ActiveRDF: ler recursos

Neste exemplo, carregamos o objeto já definidos anteriormente (*eyal*) e então conseguimos recuperar todos as suas propriedades, como *class*, *firstName* e *knows*. Podemos observar que esta última propriedade já mostra *armin* como conhecido de *eyal*, e também é possível acessar os atributos de *armin*.

Para finalizar, vamos analisar o próximo exemplo:

```
require 'activerdf'
all_people = Person.find
all_resources = Resource.find

[1]
Person.find.each do |person|
  p person.to_s + ' knows ' + person.knows.to_s
end

[2]
johns = Person.find_by_firstName('john')

[3]
thirties = Person.find_by_age(30)
```

Quadro 35: Exemplo de uso do ActiveRDF: encontrar recursos

No primeiro trecho de código conseguimos recuperar todas as pessoas e seus respectivos amigos. No segundo trecho, encontramos todas as pessoas que têm *john* como primeiro nome. No terceiro e último trecho, encontramos todas as pessoas com 30 anos.

Pudemos observar que a partir de um repositório RDF, são geradas classes que permitem o acesso às suas informações através de linhas de comando. Como, por exemplo, o método *find_by_firstName*, que é criado de acordo com um atributo da classe *Person* que é o *firstName*. É um dos métodos que compõem a DSL gerada.

O ActiveRDF se assemelha a esta dissertação exatamente neste aspecto de ter uma DSL gerada automaticamente e também permitir que sejam formados conjuntos induzidos a partir de um modelo, porém não é capaz de realizar operações entre seus objetos, somente a busca é permitida.

5.1.2.3. HyperDE

O HyperDE é a combinação de um framework no padrão *Model-View-Controller* e um ambiente de desenvolvimento visual para a construção de protótipos de aplicações hipermídia, modeladas através dos métodos OOHDM ou SHDM. Como framework MVC, o HyperDE fornece componentes reutilizáveis e extensíveis para as camadas de modelo, visão e controle, especificados como ontologias em RDFs.

Como ambiente de desenvolvimento visual, o HyperDE fornece, através de sua interface gráfica e ferramentas auxiliares, uma forma interativa e dinâmica de construir e prototipar uma aplicação hipermídia, com a possibilidade de visualizar imediatamente o resultado de cada passo do processo de desenvolvimento.

A arquitetura de desenvolvimento promovida pelo ambiente é orientada a modelos, onde a definição dos modelos navegacionais efetivamente gera a implementação da aplicação. Utilizando um modelo de dados baseado em RDF e RDFs, os modelos navegacionais produzidos no HyperDE podem ser utilizados como ontologias, fazendo-se uso de tecnologias e linguagens da Web Semântica.

Além disso, a utilização de uma linguagem de programação dinâmica permite que o HyperDE construa dinamicamente linguagens específicas de domínio para cada aplicação desenvolvida, o que resulta em um modelo de programação mais conciso e natural.

Este trabalho se relaciona com esta dissertação no aspecto de poder construir dinamicamente uma DSL para a aplicação em questão e de possuir um ambiente de desenvolvimento.

5.1.3 Facetas induzidas a partir de um RDF

5.1.3.1. Longwell

Longwell é um browser para RDF baseado em web (LW). É possível navegar e buscar arbitrariamente complexos conjuntos de dados definidos em RDF. Foi idealizado e projetado pelo projeto SMILE, que tem como objetivo aplicar tecnologias de web semântica em bibliotecas digitais e também gerar

protótipos de cenários de interface que possam ser úteis aos usuários finais, bibliotecários e analistas de meta-dados.

Outras tecnologias para navegação em RDF existem, como o HayStack Project, porém a principal característica do projeto SMILE é a integração com o DSpace que é uma aplicação servidor baseada em web. Para isso foi necessária uma abordagem web e uma simples arquitetura configurada para atender os objetivos finais, o que contribuíram para uma prototipação mais rápida.

O DSpace é um repositório digital que captura, armazena, indexa, preserva e distribui material digital para pesquisa. Instituições de pesquisa por todo o mundo o utilizam como um repositório institucional, como um repositório de *learning objects*, para manipulação de registros, etc.

Vejamos um exemplo de iteração no sistema:

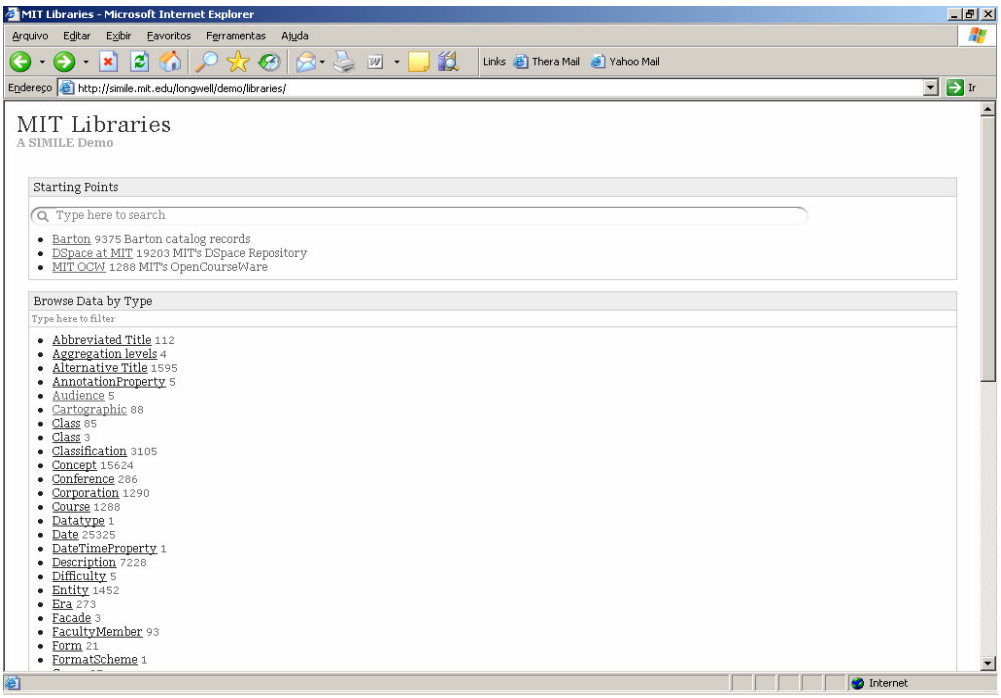

Figura 32: Tela do Longwell

O usuário final é capaz de selecionar uma categoria e esta será adicionada como uma restrição na busca. Também pode ser adicionada uma restrição digitada pelo usuário. Essas restrições podem ser removidas caso não sejam mais necessárias.

Inúmeras restrições podem fazer parte de uma busca e serão todas relevantes igualmente para a filtragem dos itens resultantes. Novamente a operação de interseção é utilizada.

5.1.4 Navegação Facetada

Os modelos utilizados pelas aplicações abaixo não permitem outras operações sobre conjuntos além da operação de interseção.

5.1.4.1. Flamenco Search

Flamenco Search é um framework de interface de busca com o objetivo principal de permitir que os usuários naveguem dentro de um grande espaço de informações de maneira flexível sem se sentirem perdidos (FLASEARCH). A propriedade chave da interface é a exposição explícita dos metadados facetados hierarquicamente, tanto para guiar o usuário entre as possíveis escolhas como para organizar os resultados de buscas por palavras-chave.

A interface usa metadados de maneira que os usuários possam refinar e expandir uma busca, enquanto mantém-se consistente a representação da estrutura de facetas. O uso dos metadados é completado pela busca através de texto livre, permitindo que o usuário siga através dos links e adicione termos na busca e então seguir or mais links sem interromper seu fluxo de busca.

Abaixo segue um screenshoot de exemplo da interface:

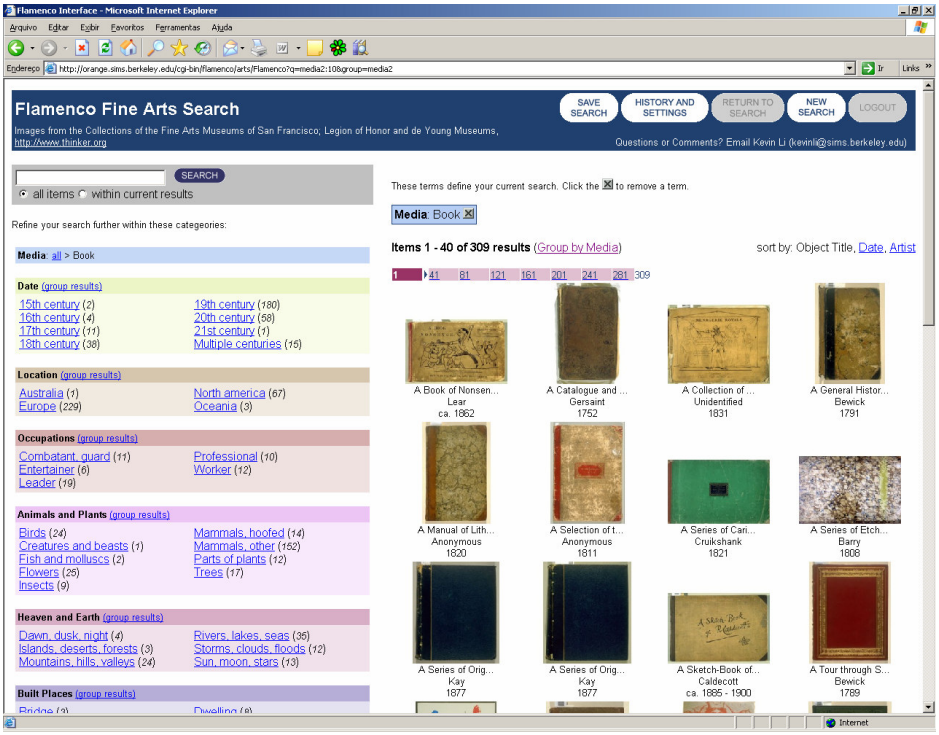

Figura 33: Tela do Flamenco Search

Nessa imagem nota-se o conteúdo das facetas representadas hierarquicamente na esquerda, onde cada grupo de faceta tem em um cor diferente. E no lado direito, os objetos encontrados com a busca realizada, que no caso foi um filtro aplicado, ou seja, foi selecionada a faceta Media > Book.

Essa interface permite somente que o usuário vá adicionando novos itens de filtro, isto é, buscando por mais de uma faceta, fazendo a interseção entre os elementos de duas ou mais facetas. Não é possível realizar nenhum outro tipo de operação entre os objetos e as facetas.

5.1.4.2. FacetMap

O Facetmap usa o mesmo princípio do Flamenco Search. O usuário entra com um arquivo XML contendo a definição de todas as facetas hierarquicamente e de todos os recursos que estarão sendo buscados, associando-os apropriadamente aos seus conjuntos (FACETMAP).

Segue abaixo uma imagem do FacetMap utilizando tipos de vinhos como XML de entrada:

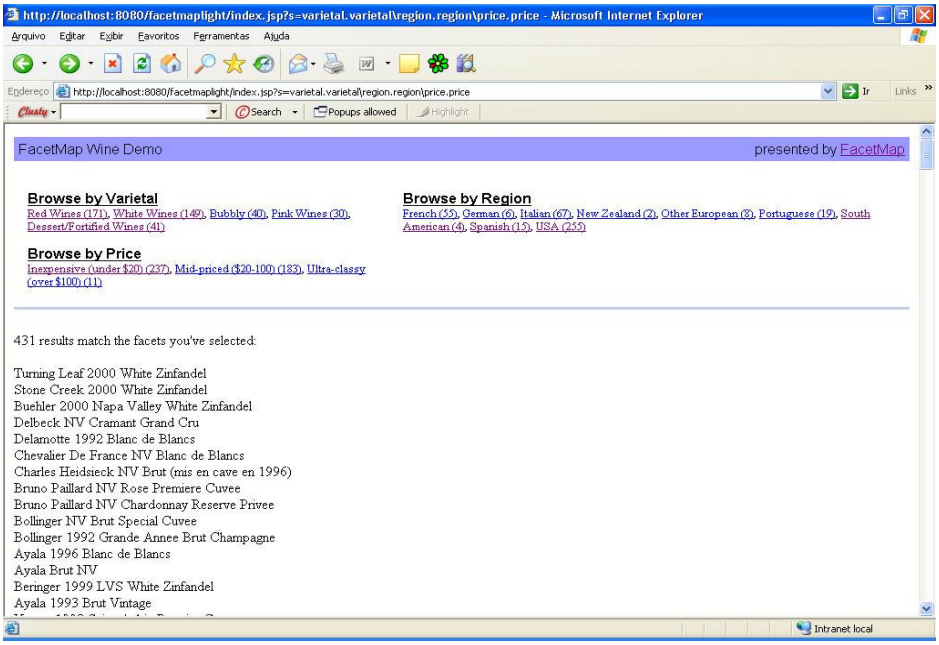

Figura 34: Tela do FacetMap

Nesse caso estão sendo mostrados todos os vinhos disponíveis no catálogo e suas facetas organizadas hierarquicamente na parte superior.

O usuário pode ir filtrando faceta após faceta e ir refinando a busca ou então retirar alguma faceta já adicionada anteriormente na busca. Não é possível realizar nenhum outro tipo de operação entre os objetos e as facetas.

5.1.4.3. Endeca

Endeca é um outro tipo de framework que cria uma interface baseada em navegação facetada, que é chamada pelos seus criadores de Guided Navigation. É aplicado o mesmo princípio dos demais exemplos, onde ao invés de realizar a busca simplesmente através de um campo de texto livre e mostrar inúmeras ocorrências de resultados, uma tela como a abaixo é mostrada ao usuário (ENDECA).

O usuário, por sua vez, pode refinar a busca, pois o Endeca sabe todos os metadados que contêm a palavra buscada e então pode agrupar em facetas as possíveis categorias em que o produto, que o usuário procurou, se encontra.

Através dos metadados o Endeca elimina as categorias que não servem para a primeira busca feita, por exemplo, e evita assim o retorno de facetas sem itens. Posteriormente é feito um agrupamento automático das categorias relevantes e mostradas ao usuário para que o mesmo possa prosseguir com a busca.

Da mesma forma que os anteriores, o Endeca faz a interseção entre os objetos das facetas e os retorna para o usuário. Não é possível realizar nenhum outro tipo de operação entre os objetos e as facetas.

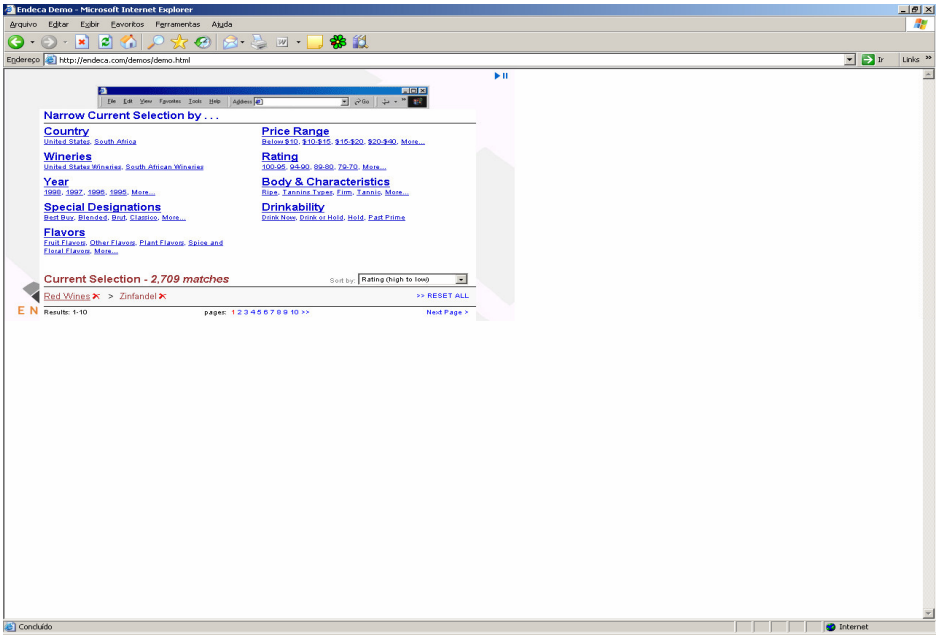

Figura 35: Tela do Endeca

5.2 Contribuições

As principais contribuições deste trabalho são:

- Criação de um modelo de informações com um nível mais alto e mais próximo ao que o usuário está acostumado a interagir. O universo de informações é interpretado como um modelo baseado em conjuntos e não um modelo tradicional, baseado em grafo. É um modelo capaz de

expressar tarefas típicas facilmente em muitos tipos de navegação, que unifica o modelo clássico com o modelo de conjuntos.

- Criação de uma DSL para poder ser utilizada pelo usuário programador para que este possa usufruir de todas as operações que foram implementadas e dinamizar o processo de manipulação de dados. Esta DSL é responsável por representar as tarefas do usuário em métodos e comandos.

- Novas operações podem ser executadas sobre um conjunto de dados, contribuindo para que um usuário final possa fazer buscas em um grande repositório de dados de maneira mais simples e rápida e muitas vezes encontrar ou realizar buscas por dados que não eram possíveis de ser feitas através de operações clássicas em conjuntos, como por exemplo, a interseção que é muito utilizada na navegação facetada. Estas operações abrangem desde operações clássicas até operações que são induzidas pelo modelo de domínio.

- Possibilidade de utilização das funcionalidades desenvolvidas como uma camada de manipulação de dados por uma camada de interface, criando assim uma maneira de executar as operações propostas através de manipulação direta.

- Possibilidade da utilização do padrão de arquitetura MVC. O componente *model* é representado pelo modelo proposto. O componente *view* alguma interface que renderize o ambiente mais próximo ao do domínio em questão, deixando assim cada componente independente em termos de código. E o componente *controller* é composto pelo mapeamento realizado entre o modelo e a interface juntamente com a DSL.

5.3
Trabalhos futuros

Algumas idéias e sugestões para outros trabalhos que poderão contribuir para melhorar o que foi proposto nesta dissertação:

- Realização de alterações no gerador de classes. Atualmente nem todos os tipos de dados são reconhecidos. Os tipos *string* e *int* são interpretados pelo gerador e classes são geradas para abranger esses dados. Também são reconhecidos tipos referentes a outras classes, onde vetores do tipo da classe são gerados. Porém outros tipos como *boolean*, *ArrayList*, etc seriam bastante úteis ao se definir uma classe.
- Novos métodos podem ser formulados para melhorar a utilização das classes geradas, tornando-as mais rica. Como, por exemplo, um método que retorna os *n* primeiros elementos de um conjunto sendo passada uma configuração para a busca e que nesta própria configuração outros parâmetros já possam ser definidos.

- Melhor utilização dos métodos de adicionar, alterar e remover associações a uma classe. Por exemplo, um objeto do tipo *vinho* contém por definição do usuário *n* categorias associadas, porém ao longo de uma interação o usuário pode necessitar acrescentar uma nova *faceta* a esse *vinho*, como também removê-la. Atualmente somente métodos para consulta e manipulação de conjuntos estão sendo utilizados.

- Ampliação dos tipos de entrada de dados. Atualmente só é possível definir as classes e objetos através de RDFS/RDF. Para tornar o ambiente mais rico seria interessante que outras entradas como, por exemplo, banco de dados e XML, fossem aceitas. Dessa maneira o usuário poderia definir seus elementos em um banco de dados ou um XML que seria interpretado pelo sistema e carregado no seu repositório central para posterior utilização das operações baseadas em conjuntos.

- Melhorar a linguagem de consulta desenvolvida. Desenvolver um validador para verificação dos parênteses das cláusulas do *WHERE*. Incluir outros tipos de operadores como, por exemplo, o *like*, muito utilizado no Microsoft SQL Server. Também é interessante o comando *IS NULL* que permitirá retornar, por exemplo, todos os vinhos que não tenham facetas associadas.

- Modificar os métodos propostos para que possam virar WebServices e ser acessados de qualquer lugar. Dessa forma diferentes interfaces poderiam utilizar a biblioteca desenvolvida ao mesmo tempo, sem precisar incluí-la como parte do seu código fonte.

- Criar um modelo de conjuntos distribuídos, de forma que cada *peer* tenha seu próprio conjunto de dados. Teríamos então um servidor WebService apenas que seria responsável pelas operações entre os conjuntos distribuídos. A vantagem seria que cada *peer* poderia disponibilizar seus próprios repositórios de dados, não havendo a necessidade de estarem centralizados no servidor de aplicação, como ocorre atualmente.

- Utilizar mais funcionalidades para a aplicação criada segundo o modelo MVC, atualmente somente algumas estão sendo aplicadas. Métodos que podem ser aplicados sobre um conjunto de itens ou sobre um conjunto não foram demonstrados.