UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DENISE MATTÉ GIACOMOLLI

METADATAPROV – uma interface web para a consulta da proveniência de metadados de bibliotecas digitais

Trabalho de Graduação.

Profª. Drª. Renata de Matos Galante
Orientadora

Msc. Eduardo Nunes Borges
Co-orientador

Porto Alegre, novembro de 2007.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a minha família. Aos meus pais, Nédio Giacomolli e Teresinha Maria Matté Giacomolli, por estarem sempre ao meu lado, me apoiando em todas as minhas decisões e pela compreensão e paciência em todos os momentos em que estive ausente por causa de trabalhos ou provas da faculdade. À minha irmã Cláudia por ter me dado o exemplo de profissional na informática e ser a principal responsável pela minha decisão de fazer Ciência da Computação. Á minha irmã Carina, por todo o carinho e momentos divertidos que me ajudaram a relaxar e afastar por tantas vezes as preocupações da faculdade.

Ao Tiago, meu namorado que eu amo muito. Por todos os momentos felizes vividos, pelo carinho e amor e por tantas vezes ter me ajudado com problemas encontrados na faculdade e trabalho.

Às empresas em que passei, Procempa e ADP, por terem me ensinado a por em prática os ensinamentos recebidos na faculdade.

À minha orientadora Renata Galante, sempre disposta a responder meus emails ou me receber em sua sala. Sem seu empenho este trabalho, com certeza, não estaria finalizado. Ao meu co-orientador Eduardo Borges, por todos os ensinamentos e ajuda nas decisões de implementação deste trabalho.

A todos vocês, o meu muito obrigada!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

MetadataProv	*Metadata Provenance*
GAP	*Gráfico Acíclico Dirigido*
CMCS	*Collaboratory for Multi-scale Chemical Science*
SAM	*Scientific Annotation Middleware*
ESSW	*Earth System Science Workbench*
BDBComp	*Biblioteca Digital Brasileira de Computação*
BDLP	*Digital Bibliography & Library Project*
XML	*eXtensible Markup Language*
DOM	*Document Object Model*
HTML	*HyperText Markup Language*
JSP	*Java Server Pages*
API	*Application Programming Interface*
UFRGS	Universidade Federal do Rio Grande do Sul
UML	*Unified Modeling Language*
W3C	*World Wide Web Consortium*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Bibliotecas digitais tem se tornado um mecanismo muito utilizado por estudantes e cientistas para a consulta na Internet a artigos científicos já publicados. Em sistemas de integração de diversas bibliotecas digitais é necessária a identificação de quais artigos representam a mesma instância, mesmo que os metadados que os descrevem sejam diferentes. Outra questão em aberto identificada em sistemas de integração de bibliotecas digitais é a inexistência de um mecanismo eficiente para a consulta da proveniência dos dados, de forma a identificar a origem dos dados e o processo percorrido pelos mesmos até a versão atual.

O presente trabalho apresenta o projeto e a implementação da ferramenta MetadataProv (Metadata Provenance), que é uma interface gráfica para a consulta de artigos científicos oriundos de bibliotecas digitais integradas e visualização das informações de proveniência de dados que descrevem o artigo consultado.

Palavras-chaves: bibliotecas digitais, metadados, proveniência.

MetadataProv - a graphic web tool for consulting digital library metadata provenance

ABSTRACT

Digital Libraries has been the most used tool by students and scientists to find published scientific articles. In integration of digital libraries systems is necessary to identified what articles represents the same instance, even when the metadata that describe the article are not equals. The problem identified in integration of digital libraries systems is the inexistence of an efficient tool to find articles and visualize the metadata provenance data about it.

This manual presents the project and implementation of MetadataProv, a Web graphic tool for search an article in an integration of digital libraries system and the visualization of the information about the metadata provenance that describes an article.

Key-Words: digital libraries, metadata, provenance

1 INTRODUÇÃO

Hoje em dia, com os avanços tecnológicos cada vez mais dinâmicos na área da informática, o armazenamento da informação utiliza cada vez menos a tinta e o papel. Com a informação digitalizada se tornando a cada dia mais freqüente e usada, muitas vezes substituindo por completo os meios clássicos de armazenamento, surgem propostas de organização desses dados digitalizados, para facilitar sua busca e recuperação. É nesse contexto que surgem as bibliotecas digitais.

Bibliotecas digitais tem se tornado o mecanismo mais utilizado por cientistas ou estudantes para a busca de artigos científicos em diferentes áreas. Um artigo científico indexado por uma biblioteca digital é representado por metadados que descrevem todo o seu conteúdo e principais informações. Estes metadados em geral são representados em documentos XML, que permitem a representação de informações estruturadas e semi-estruturadas, formadas por entidades hierárquicas que representam de forma simples e eficaz os dados contidos no documento.

Em um sistema de integração de artigos científicos indexados por diferentes bibliotecas digitais, como o mecanismo proposto em [1], é necessário detectar repetições de um mesmo artigo. Porém um artigo pode variar em conteúdo ou em forma de representação entre diferentes bibliotecas digitais. Logo, é necessário que o modelo de armazenamento do sistema de integração consiga estruturar os metadados relativos a um artigo de forma que não seja perdida nenhuma informação contida em alguma versão do documento.

Com o aumento de volume de dados disponibilizados na internet é de vital importância o conhecimento sobre as informações de proveniência de dados, para poder-se definir quais informações podem ser consideradas seguras. Proveniência, segundo Buneman [2] são informações que descrevem a origem dos dados e o processo percorrido pelos dados originais até a versão atual. Os dados de proveniência podem ser representados de duas formas. A primeira é armazenando as informações de proveniência à medida que os dados originais são alterados. A segunda é realizar um processo de inversão para cada vez que uma informação de proveniência for requisitada pelo usuário. O grande problema com relação às informações de proveniência armazenadas com anotações é que estas ocupam grande espaço de armazenamento, ou para o caso de inversão o tempo de processamento pode ser muito alto.

Em sistemas de integração de bibliotecas digitais, as informações sobre o local de onde um artigo provém e suas informações sobre cada metadado precisam ser armazenadas. Documentos XML oferecem uma estrutura ideal para o armazenamento destas informações, pois as informações sobre proveniência são armazenadas não necessitando a replicação de um artigo em um modelo, só sendo armazenadas as informações que diferem de uma representação para outra.

O trabalho de Borges e Galante [1] propõe um mecanismo para identificação, representação e consulta de versões de objetos XML oriundos de bibliotecas digitais. Neste trabalho é proposto também um modelo para armazenamento de informações sobre artigos de diferentes bibliotecas digitais, onde são armazenados os conteúdos dos metadados que possuem diferentes formas de representação entre as bibliotecas digitais consultadas. Este modelo de armazenamento é um documento XML, livre de redundância, e que mantém as informações a respeito da origem dos dados recuperados.

O presente trabalho foi desenvolvido em como um módulo para o mecanismo proposto em [1]. O principal objetivo é oferecer ao usuário uma ferramenta Web para acesso à proveniência de dados de bibliotecas digitais, sem a necessidade de instalação ou atualizações por parte do usuário. O usuário pode utilizar a ferramenta para consulta de artigos científicos, recebendo como retorno uma interface simples para a visualização da proveniência das informações consultadas. A partir da ferramenta desenvolvida é possível visualizar informações sobre a origem dos dados recuperados, assim como as diferentes representações de dados de um mesmo artigo nas diferentes bibliotecas digitais às quais este é indexado.

Este trabalho está estruturado da seguinte forma:

- O Capítulo 2, Proveniência, apresenta um estudo sobre a importância da recuperação da proveniência de dados disponíveis na Web. Também são apresentadas as arquiteturas existentes para a recuperação e o armazenamento das informações de proveniência e exemplos de ferramentas desenvolvidas neste contexto;
- O Capítulo 3, Um Modelo de Proveniência de Dados Aplicado a Objetos Oriundos de Bibliotecas Digitais, apresenta o trabalho no qual a ferramenta proposta está inserida apresentando detalhadamente cada módulo presente no modelo de armazenamento utilizado. Em seguida, uma forma de representação de documentos XML é apresentada. Por fim, é realizado um estudo para a decisão de qual algoritmo de similaridade utilizar para a realização das operações de consulta;
- O Capítulo 4, METADATAPROV: Uma Interface Web para a Consulta da Proveniência de Metadados de Bibliotecas Digitais, apresenta a ferramenta desenvolvida para a solução do problema proposto para o trabalho. Neste capítulo cada módulo desenvolvido é apresentado e um caso de utilização da ferramenta é simulado;

- O capítulo 5, Conclusões, apresenta as conclusões do trabalho e as possibilidades de extensões em trabalhos futuros.
- O Anexo A, apresenta a modelagem e o projeto da ferramenta METADATAPROV desenvolvidos a partir do padrão UML, como forma de documentar todo o trabalho desenvolvido, possibilitando também um melhor entendimento das funcionalidades implementadas para a ferramenta.

2 PROVENIÊNCIA

A proveniência, também referenciada como *provenance*, *lineage* ou *pedigree*, é uma informação que descreve a origem de algum dado e pode ser descrita de diferentes formas, dependendo da área em que será aplicada.

Neste capítulo são apresentados os principais conceitos de proveniência de dados relevantes para o trabalho desenvolvido, considerando: a importância da proveniência em diferentes áreas de aplicação, as arquiteturas e as regras que definem os modelos de proveniência, além dos projetos de pesquisa que estão sendo desenvolvidas atualmente.

2.1 Definições Básicas

Com a proliferação de cópias de informações retiradas de bases de dados disponíveis na Web se torna difícil determinar se a informação capturada provém de uma base de dados confiável, ou se os dados copiados sofreram alguma modificação. Determinar a veracidade da informação se torna, então, um trabalho crucial, especialmente para a criação de uma base de dados com fundamentos científicos.

Dependendo do contexto, diferentes definições podem ser tomadas. Buneman [2], por exemplo, descreve a proveniência como a descrição da origem dos dados e o processo percorrido pelo dado até chegar à base de dados. Lanter [3] define o conceito de proveniência como informações que descrevem as transformações sofridas pelos dados. Greenwood [4] expandiu a definição de Lanter descrevendo o conceito de proveniência como metadados que armazenam processos de *workflows* e anotações sobre experimentos.

Buneman define duas principais características que compõe a proveniência dos dados: *where-provenance* e *why-provenance.* O primeiro diz respeito à localização dos dados que geraram o resultado. O segundo se refere ao processo de derivação sofrido pelos dados originais até chegar à presente informação.

Alguns exemplos da importância do conhecimento da proveniência dos dados são descritos em [5]. A importância para engenheiros de material, por exemplo, é crucial para a seleção de materiais que serão usados em projetos críticos, como a construção de um avião. Sabendo a proveniência de análises já realizadas com diversos materiais a escolha do que melhor se adequar pode ser feita sem a necessidade de realização de uma nova análise. Outro exemplo muito importante de proveniência é a sua utilização para determinar a autoria original de uma informação, ou seja, determinar o local de onde uma

informação provém, o que facilita a verificação da veracidade dos dados e também ajuda na questão de direitos autorais.

2.2 Arquiteturas

Há duas principais arquiteturas utilizadas para representar a proveniência: anotação e inversão.

Anotações são metadados que representam a derivação sofrida por um dado desde a sua origem e são apresentados a partir dos dados e das transformações sofridas em cada derivação. A maior vantagem deste método é que o armazenamento destas anotações são pré-computados, ou seja, ao realizar uma consulta a uma base onde a proveniência é armazenada a partir de anotações o tempo de processamento é apenas o necessário para acessar essas informações na base de dados. Porém, um grande problema gerado por este método se refere ao espaço de armazenamento. Se a base de dados e as informações sobre proveniência forem bem detalhadas este método precisará de um grande espaço para armazenamento das informações. Para alguns dados pequenos, por exemplo, os metadados que representam a proveniência podem ocupar mais espaço de armazenamento do que o dado propriamente dito. Outro problema com a representação por anotações ocorre na atualização dos metadados. A base de dados que contém os metadados com as anotações de proveniência precisam ser constantemente atualizados, para garantir que se algum dado em uma base de dados origem for alterado o dado que referencia este na atual base também sofrerá as transformações necessárias. Por isso é necessário definir qual o melhor intervalo para a atualização que consiga obter um menor número de registros consultados incorretos com o menor número possível de atualizações na base de dados em um certo período de tempo.

Outra arquitetura para representação da proveniência é a inversão. Nesta forma representação, o processo de derivação é invertido a fim de determinar os dados originais. Este processo de inversão pode ser executado automaticamente ou a partir de funções pré-definidas. Para executar uma busca de registro, neste caso, é necessário executar um algoritmo baseado nas regras de mapeamento para cada registro que se desejar recuperar A grande vantagem deste método é que o sistema não precisa descobrir a proveniência de todos os dados existentes na base de dados. Apenas quando o usuário deseja consultar a proveniência de algum dado que esse processo de inversão precisa ser executado. Dessa forma não é necessário um grande espaço de armazenamento para guardar a proveniência, pois esta é calculada no momento da consulta. Um problema que surge com isso é que as consultas podem se tornar muito demoradas, especialmente se o dado a ser consultado sofreu muitas derivações desde o dado original. E, além disso, por falta de informações suficientes, o processo pode não conseguir definir de qual base de dados o dado sofreu derivação. Um ponto positivo, adverso ao problema das anotações, é que não é necessário realizar atualizações na base de dados que contém as informações de proveniência, não correndo o risco de o usuário realizar buscas de proveniência erradas.

A figura 2.1 ilustra os problemas e vantagens das duas arquiteturas. No caso da inversão, os dados são consultados diretamente nas bases de dados, porém mais consultas são necessárias, enquanto na arquitetura de anotações

apenas uma consulta à base de dados de proveniência é realizada, porém esta base precisa ser constantemente atualizada com as informações das bases de dados à que se refere.

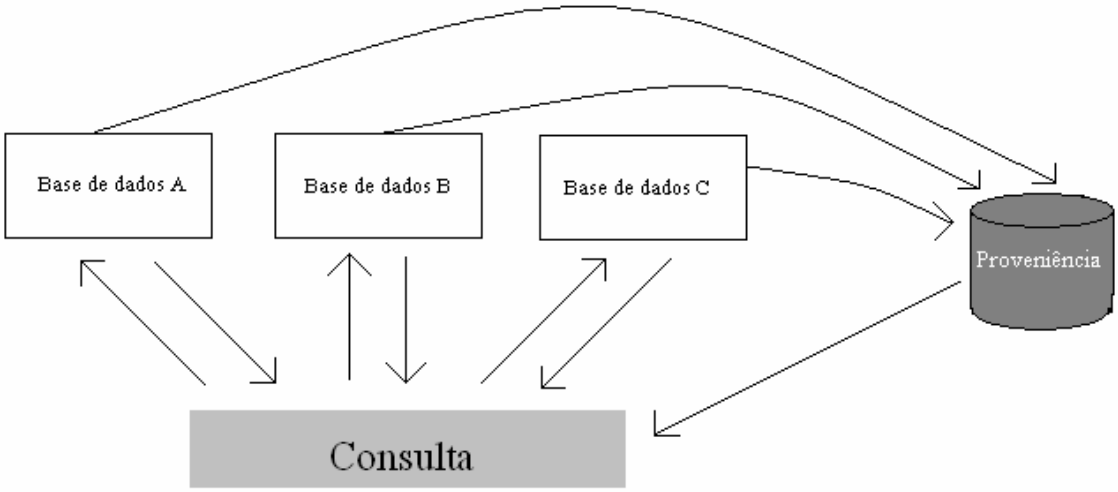

Figura 2.1 – Arquiteturas: inversão e anotação

2.3 Taxonomia

Nesta seção é apresentada uma taxonomia a respeito das técnicas de proveniência. A figura 2.2 sumariza esta taxonomia e logo após cada tópico será apresentado detalhadamente.

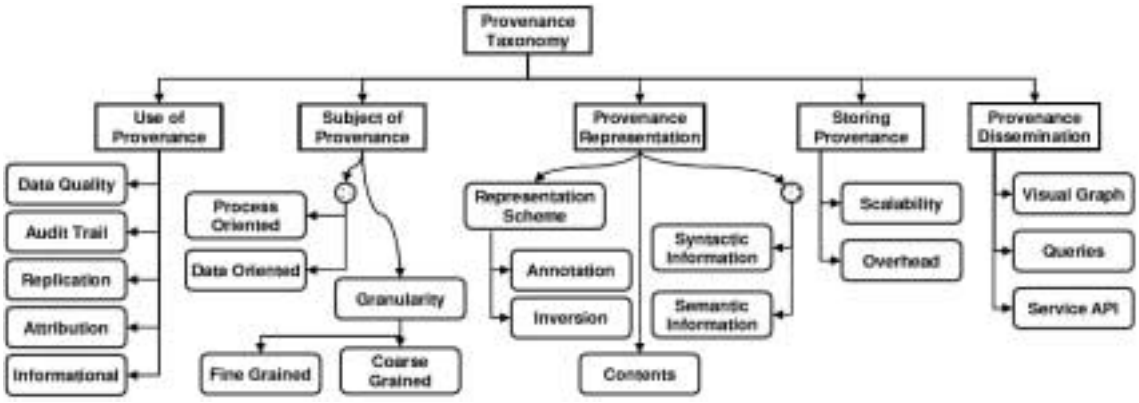

Figura 2.2 - Taxonomia

2.3.1 Uso da proveniência

Sistemas de proveniência podem ser utilizados para diferentes propósitos. As principais áreas de aplicações são [6]:

- *Qualidade de dados* – Determinar a qualidade da base de dados é importante para transmitir segurança ao usuário da aplicação. As informações sobre proveniência, neste caso, determinam se dados maliciosos foram inseridos em algum passo de derivação e assim pode-se avaliar a qualidade da base de dados a partir das métricas

de qualidade, que geralmente levam em consideração o número de transformações sofridas pelos dados desde o documento original.

- *Caminho de derivação* – Um uso comum da proveniência dos dados é para voltar nos passos de derivação até encontrar a origem dos dados a serem analisados determinando assim o caminho percorrido desde o dado original até o presente na base de dados analisada. Traçar esta rota é importante, por exemplo, ao se estabelecer patentes de algum medicamento descoberto, para determinar a contabilidade de um negócio ou para se coletar estatísticas para estimar o uso do recurso ancestral após cada passo da derivação.
- *Replicação* – a proveniência pode ser pensada como uma receita para recriar dados a partir de informações sobre sua derivação. Se a proveniência possuir informações suficientes sobre as operações e sobre os dados originais é possível repetir os passos de derivação, para manter a veracidade dos dados quando alguma alteração na base é efetuada.
- *Autoria* – proveniência pode ser usada para determinar a autoria de algum dado, ou a responsabilidade pela publicação de informações erradas em uma base de dados. Neste caso os usuários da aplicação podem ou olhar na árvore de derivação para descobrir de quem é a autoria do dado utilizado, o que ajuda no trabalho de definição da bibliografia, ou consultar se alguém está utilizando os dados criado por este usuário.
- *Informacional* – A proveniência pode ser definida a partir da descrição dos metadados que representam as informações da base de dados, ou das suas anotações, que ajudam a interpretar os dados no seu contexto, especialmente os que serão usados por muito tempo após a sua criação.

2.3.2 Abordagens

A proveniência pode ser coletada de diferentes formas e em diversos níveis de detalhes. Pode-se determinar quais os tipos de dados a serem coletados baseados na importância destes dados e no custo para extraí-los. A seguir, a proveniência é classificada quanto ao assunto ao qual ela representa e sua granularidade.

2.3.2.1 Modelo Orientado a Dados X Modelo orientado a processos

As informações sobre proveniência podem estar disponíveis explicitamente ou podem ser deduzidas indiretamente. O modelo onde os dados são disponíveis explicitamente é conhecido como modelo orientado a dados. Neste modelo temos um GAD (Grafo Acíclico Dirigido) onde os nodos representam os dados e as arestas representam as transformações sofridas em um passo de derivação. Este grafo representa a proveniência estando associado apenas com uma base de dados, sem a necessidade de fazer uma derivação com uma base de dados externa. Este modelo contrasta com o modelo indireto, onde os processos de derivação são as entidades primárias pela qual a proveniência é coletada, e as informações sobre a proveniência são

coletadas em workflows e utilizadas para derivar as informações de proveniência que são geradas de uma base de dados criada durante a execução do workflow. Dependendo do contexto no qual a proveniência é capturada, quaisquer dos dois modelos podem ser utilizados (ou ainda pode-se utilizar outro modelo que adota outras formas de extração da proveniência).

2.3.2.2 Granularidade

A utilização de proveniência em certo domínio depende do seu nível de granularidade, ou seja, do nível de detalhamento do comportamento de um objeto. Alguns arquivos em uma coleção podem ser gerados a partir do mesmo experimento, e, então, podem ter a proveniência similar, podendo ser traçada uma estatística de aproximação entre os objetos da coleção. Os dados que são subconjuntos de outros dados podem ter sua proveniência compartilhada com os dados parentes na árvore de derivação, e mesmo assim possuir uma proveniência diferente com relação à informação como um todo. Alguns domínios exigem que a proveniência seja armazenada em diversos níveis de granularidade, enquanto outras bases de dados abstratas que se referem aos dados independente da granularidade estão crescendo em uso e fornecendo uma maior flexibilidade. Dependendo do nível de detalhamento a granularidade pode ser considerada fina ou grossa. Em um modelo de armazenamento de proveniência de metadados de artigos oriundos de bibliotecas digitais é possível ter duas situações possíveis onde a granularidade é classificada como fina ou grossa. Estas situações são:

- Granularidade Fina: o objeto de dados são os metadados que descrevem um artigo. Para cada metadado são armazenadas as informações da biblioteca digital proveniente, além do rótulo e do valor (representação quanto à terminologia e conteúdo) utilizados por esta biblioteca. Este caso gera bastante espaço de armazenamento. Supondo que cada artigo possui 10 metadados, serão 30 informações de proveniência para cada artigo.
- Granularidade Grossa: o objeto de dados é um artigo científico. Para cada artigo é necessário armazenar a biblioteca digital que armazena o artigo e o link para o texto completo deste artigo. Para este caso apenas 2 informações de proveniência são armazenadas para cada artigo, ocupando pouco espaço de armazenamento.

2.3.3 Armazenamento

Duas são as principais formas para armazenamento da proveniência. Na primeira as informações de proveniência são armazenadas junto dos os dados que representam. A vantagem desta forma é que é fácil de manter a integridade dos dados, porém se torna difícil a realização de uma busca a estes dados. Outra forma é armazenar a proveniência de forma independente das informações que representam. Neste caso é preciso considerar se a proveniência é inalterada, versionada ou se pode ser alterada para refletir o estado dos predecessores.

Um grande problema do armazenamento da proveniência é a mudança dos dados aos quais as informações de proveniência se referem. Para prevenir

que as informações de proveniência permaneçam desatualizadas é necessário garantir que quando as fontes de onde elas foram obtidas mudem o repositório da proveniência também mude e seja atualizado.

2.3.3.1 Escalabilidade

Sistemas de proveniência podem ser escaláveis de acordo com diferentes perspectivas: granularidade, quantidade de proveniência, localização geográfica e uso da base de dados. Se a granularidade de uma base de dados diminui, a quantidade de base de dados que a proveniência está relacionada diminui. Quando o número de derivações necessárias para derivar uma base de dados aumenta, a quantidade de proveniência aumenta. Muitas vezes as informações de proveniência são maiores do que os dados que elas representam. Para resolver este problema técnicas que utilizam a recursividade para descobrir a origem dos dados são utilizadas. Porém se a quantidade de proveniência é muito grande e se encontra em diferentes bases o custo para realizar a pesquisa pode ser muito alto. Uma saída é utilizar anotações para armazenar as informações dos dados que precedem imediatamente a base atual e recursivamente a partir destas informações chegar até a origem.

2.3.3.2 Overhead

Grandes quantidades de proveniência podem causar overhead de armazenamento ou de tempo para realizar pesquisas a partir da derivação dos dados. Uma solução adotada em muitos sistemas é a utilização de modelos que arquivam as informações de proveniência não muito consultadas e retêm as informações mais consultadas.

2.3.4 Disseminação

Sistemas de proveniência podem permitir diversas formas para o usuário poder acessar as suas informações. A forma mais comum é a partir de um grafo de derivação onde o usuário pode navegar analisando as alterações de um estado para outro.

2.4 Técnicas

Diversos trabalhos sobre o problema de proveniência já foram desenvolvidos. Em [5] são selecionados 5 trabalhos para serem discutido e os mesmos serão brevemente descritos neste trabalho com o intuito de obter exemplos reais que auxiliem em um melhor entendimento da manipulação de informações sobre proveniência. São estes: Chimera [6], MyGrid [7], CMCS [8, 9], ESSW [10] e Trio [11, 12].

Chimera [6] coleta a proveniência a partir dos passos da derivação dos dados, com uma precisão suficiente para que outros dados possam ser recriados a partir desta informação. A arquitetura do Chimera é constituída basicamente de dois componentes: o primeiro conhecido como VDC é um catálogo de dados virtual que fornece uma representação compacta dos procedimentos computacionais utilizados para derivar os dados e das chamadas destes procedimentos. O segundo componente é o *virtual data language interpreter*, que implementa as chamadas e requisições à base de dados.

O MyGrid [7] fornece um middleware para experimentos na área de biologia, modelados em workflows em um ambiente de Grid. As informações de proveniência são armazenadas em workflows escritos na linguagem Xsculf. Um log de proveniência é automaticamente armazenado durante a execução do workflow, que contém os serviços invocados e seus parâmetros, o tempo de início e de fim e os dados derivados.

O projeto CMCS [8, 9] (Collaboratory for Multi-scale Chemical Science), utilizado para armazenar informações de proveniência para a área de química usa o SAM (Scientific Annotation Middleware), que é um middleware que serve como repositório para o armazenamento de arquivos referenciados na Web. No CMCS é utilizado um portal para possibilitar a comunicação entre membros da comunidade, assim como a procura por dados já registrados, que podem ser públicos para toda a comunidade ou restritos a um determinado número de membros.

O ESSW [10] (Earth System Science Workbench) é um sistema de gerenciamento de metadados e armazenamento de dados para cientistas que trabalham com informações e imagens provenientes de satélites. A proveniência nesta área é importante para detectar erros provenientes da derivação dos dados e também para determinar a qualidade da base de dados gerada.

Trio [11, 12] é um sistema de base de dados centralizado que gerencia os dados, sua proveniência e precisão. Algumas características positivas do trio são:

- os valores dos dados podem ser incorretos, incompletos ou aproximados;
- a proveniência pode ser usada para determinar modificações em dados, determinando a precisão dos dados derivados deste;
- o Trio trabalha com dados inexatos e dados incompletos, exibindo ao usuário as respostas em função da precisão dos dados. Quanto mais inexatos os dados(quanto menor a confiança dos dados) menor a exatidão da resposta a uma determinada consulta.
- podem ser feitas consultas sobre a proveniência e precisão dos dados.

2.5 Considerações finais

A disponibilização das informações de proveniência no contexto de sistemas de integração de bibliotecas digitais é muito importante pois possibilita ao usuário visualizar as diferenças dos dados armazenados de um mesmo artigo em diferentes bases de dados. Este trabalho utiliza um modelo de armazenamento baseado em anotações das proveniências dos metadados de um artigo científico. A granularidade dos dados armazenados é fina, pois o modelo armazena todas as informações que diferem em um metadado de um artigo em uma biblioteca digital para o mesmo indexado por outra biblioteca digital.

3 UM MODELO DE PROVENIÊNCIA DE DADOS APLICADO A OBJETOS ORIUNDOS DE BIBLIOTECAS DIGITAIS

O objetivo deste capítulo é inicialmente apresentar a ferramenta para detecção de réplicas de informações sobre artigos em bibliotecas digitais, contexto no qual a ferramenta implementada neste trabalho está inserida. Em seguida é apresentado o modelo de proveniência implementado pela ferramenta proposta. Nesta seção é explicado o significado de cada atributo contido no modelo e o problema proposto para este trabalho. Em seguida é mostrada a API utilizada para a consulta do modelo a ser utilizado. Por fim é apresentado o algoritmo de similaridade escolhido para a execução das operações de consulta ao modelo de proveniência.

3.1 Um ferramenta para a detecção de metadados replicados em bibliotecas digitais através do uso de proveniência

Esta seção apresenta uma visão geral do mecanismo proposto em [1], cujo objetivo é detectar automaticamente duplicatas de artigos científicos provenientes de diferentes bibliotecas digitais, criando um modelo de representação único, sem redundância e sem perda de informações. A figura 3.1 ilustra os principais módulos do mecanismo proposto, que são:

- Identificação: responsável por coletar, com um intervalo de tempo *t*, os metadados XML referentes a sistemas de integração de bibliotecas digitais, com a finalidade de detectar novas versões do mesmo objeto XML referente a um artigo. Para isso são utilizadas técnicas de similaridade aplicadas ao conteúdo e à estrutura dos documentos.
- Proveniência: este módulo é o responsável por rastrear as informações de proveniência e criar um modelo de armazenamento das anotações referentes às diferentes versões dos objetos XML, onde são armazenadas informações sobre a origem de cada metadado do objeto XML retornado pelo módulo *Identificação*.
- Consulta: permite ao usuário realizar uma operação de pesquisa a um artigo cientifico no modelo criado pelo módulo *Proveniência*, obtendo uma resposta única e sem perda de informações.

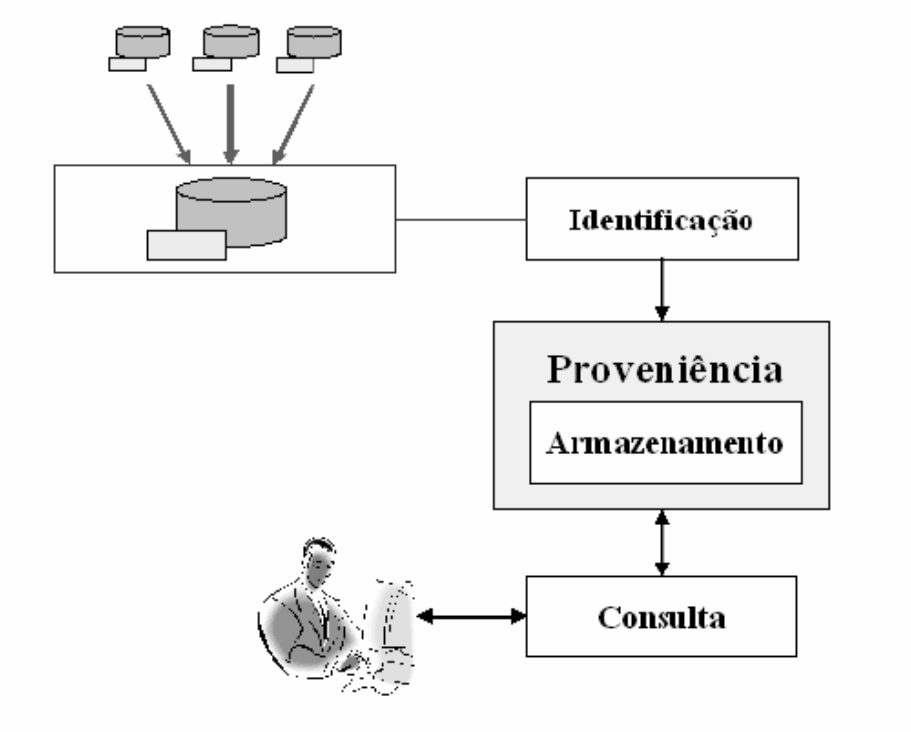

Figura 3.1 – Módulos da ferramenta

A ferramenta proposta está inserida no módulo proveniência onde será implementada uma interface para a visualização das informações de proveniência de artigos oriundos de diferentes bibliotecas digitais. A interface terá como principal funcionalidade a consulta ao modelo criado no módulo de proveniência para o retorno das informações referentes a um artigo cientifico a partir dos dados submetidos pelo usuário.

3.2 Modelo de proveniência

Nesta seção é apresentado o modelo de proveniência especificado para o módulo *Proveniência* descrito na figura 3.1. É neste modelo que serão realizadas todas as operações necessárias para a implementação do presente trabalho não sendo necessária nenhuma consulta à base de dados.

Tomando como exemplo um artigo científico presente em duas bibliotecas digitais: BDBComp (Biblioteca Digital Brasileira de Computação) e DBLP (Digital Bibliography & Library Project). Ambas contém informações a respeito do artigo escrito, por Edleno Silva de Moura e Altigran Soares da Silva, cujo o título é “Detecção de Réplicas Utilizando Conteúdo e Estrutura”. Porém na BDBComp estas informações de autor são representadas por um elemento com nome “creator”, enquanto a DBLP representa as mesmas informações por um elemento de nome “author”. O modelo de proveniência proposto tem o objetivo de armazenar estas informações de como são representados os diversos elementos de um artigo em cada biblioteca digital. Além disso, se alguma informação diferir de uma biblioteca digital para outra o conteúdo dos metadados será armazenado no modelo de forma replicada, visando garantir a proveniência de cada informação distinta.

O formato adotado para representar os metadados no modelo de Borges e Galante[1] é o XML (eXtensible Markup Language). O XML também é o formato adotado pelas bibliotecas digitais utilizadas pois permite uma fácil representação dos metadados para um artigo e posterior utilização pelos programas de aplicação. Um exemplo de representação de artigo no modelo utilizado é apresentado na Figura 3.2.

O modelo adotado é composto por um arquivo xml, que armazena todos os metadados referentes aos artigos provenientes de todas bibliotecas digitais consultadas. O modelo está estruturado da seguinte forma. Cada artigo é representado pela tag *digitallibrary*. Cada tag *digitallibrary* contém todos os metadados provenientes de diferentes bibliotecas digitais. Os metadados geralmente freqüentes em todos os artigos são representados pelas tags: *title, author, year, fulltext e booktitle*. Cada metadado possui uma tag *value* que possui o a informação do metadado sobre o artigo. Além da tag *value* cada metadado possui um elemento *provenance* que contém as informações de proveniência das diferentes bibliotecas digitais onde o artigo se encontra. Cada elemento *provenance* possui 3 tags que representam os valores na biblioteca digital à qual a proveniência se refere:

- url - contém a url da biblioteca digital de onde a versão do artigo provém, ou seja, o endereço da biblioteca digital responsável pelas informações de proveniência definidas nos tags abaixo;
- derivedlabel - contém o rótulo do metadado do qual o rótulo exibido ao usuário foi derivado;
- derivedvalue - contém o valor do metadado do qual o valor exibido ao usuário foi derivado.

Analisando o primeiro elemento autor, presente na linha 17 do exemplo da Figura 3.1, que possui como conteúdo de sua tag *value* o valor “Antonio Leitao”, presente na linha 18, e possui dois elementos de proveniência. O primeiro provém da bdbcomp e nesta biblioteca digital o rótulo que representa o nome do autor é denominado “creator”. Na segunda proveniência, que referencia as informações da biblioteca digital bdlp, o label derivado está vazio pois nesta biblioteca digital também é utilizado o rótulo *author.* Quanto ao nome do autor foi armazenado como “Antonio Menezes Leitao”.

Percebe-se que as bibliotecas digitais possuem informações sobre o nome do autor que diferem na representação tanto no conteúdo quanto em terminologia. Portanto é necessária uma ferramenta que manipule e rastreie a proveniência destes metadados levando em consideração esta múltipla representação. O problema proposto para o presente trabalho é desenvolver uma ferramenta visual para realizar consulta sobre a proveniência de artigos científicos baseada no modelo apresentado. Será implementada uma ferramenta Web onde o usuário poderá pesquisar por artigos científicos a partir do valor de um ou mais metadados. Como resultado deverão ser apresentadas as informações que descrevem os artigos que satisfazem a consulta, bem como as informações de proveniência.

```xml
<?xml version="1.0" encoding="utf-8"?>
<MetadataProv>
  <digitalobject>
    <title>
      <value>SNePS and semi-structured databases</value>
      <provenance>
        <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
        <derivedlabel> </derivedlabel>
        <derivedvalue> </derivedvalue>
      </provenance>
      <provenance>
        <url>http://dblp.uni-trier.de</url>
        <derivedlabel> </derivedlabel>
        <derivedvalue>SNePS and Semi-Structured Databases.</derivedvalue>
      </provenance>
    </title>
    <author>
      <value>Antonio Leitao</value>
      <provenance>
        <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
        <derivedlabel>creator</derivedlabel>
        <derivedvalue> </derivedvalue>
      </provenance>
      <provenance>
        <url>http://dblp.uni-trier.de</url>
        <derivedlabel> </derivedlabel>
        <derivedvalue>Antonio Menezes Leitao</derivedvalue>
      </provenance>
    </author>
    <author>
      <value>Joao Pavao Martins</value>
      <provenance>
        <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
        <derivedlabel>creator</derivedlabel>
        <derivedvalue> </derivedvalue>
      </provenance>
      <provenance>
        <url>http://dblp.uni-trier.de</url>
        <derivedlabel> </derivedlabel>
        <derivedvalue>Joao P. Martins</derivedvalue>
      </provenance>
    </author>
    <year>
      <value>1999</value>
      <provenance>
        <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
        <derivedlabel>date</derivedlabel>
        <derivedvalue> </derivedvalue>
      </provenance>
      <provenance>
        <url>http://dblp.uni-trier.de</url>
        <derivedlabel> </derivedlabel>
        <derivedvalue> </derivedvalue>
      </provenance>
    </year>
    <booktitle>
      <value>sbbd</value>
      <provenance>
      <provenance>
    </booktitle>
  </digitalobject>
</MetadataProv>
```

Figura 3.2 – Modelo XML de proveniência de metadados

3.3 Representação de arquivos XML em árvores DOM

Antes do estudo dos algoritmos de similaridade, é necessário mostrar como o modelo XML proposto na seção anterior será manipulado e representado. O Document Object Model (DOM) em [13] é uma API que foi criada para ser utilizada por qualquer linguagem de programação e representa um documento XML ou HTML na forma de uma árvore, preservando sua estrutura e seu conteúdo. Com esta API todos os componentes de um XML são transformados em nodos de uma árvore, permitindo que um documento XML possa ser acessado e alterado dinamicamente tanto no seu conteúdo como na sua estrutura, podendo criar novos elementos, modificar ou alterar o já existentes.

Um esboço da forma como o DOM representará a estrutura do XML proposto em [1] é apresentado na figura 3.3. Para a resolução do problema proposto será necessária a manipulação dos valores dos elementos encontrados no XML, não sendo necessárias modificações tanto de estrutura quanto de conteúdo do arquivo XML. Esta manipulação é feita dinamicamente, à medida que operações forem requisitadas no sistema.

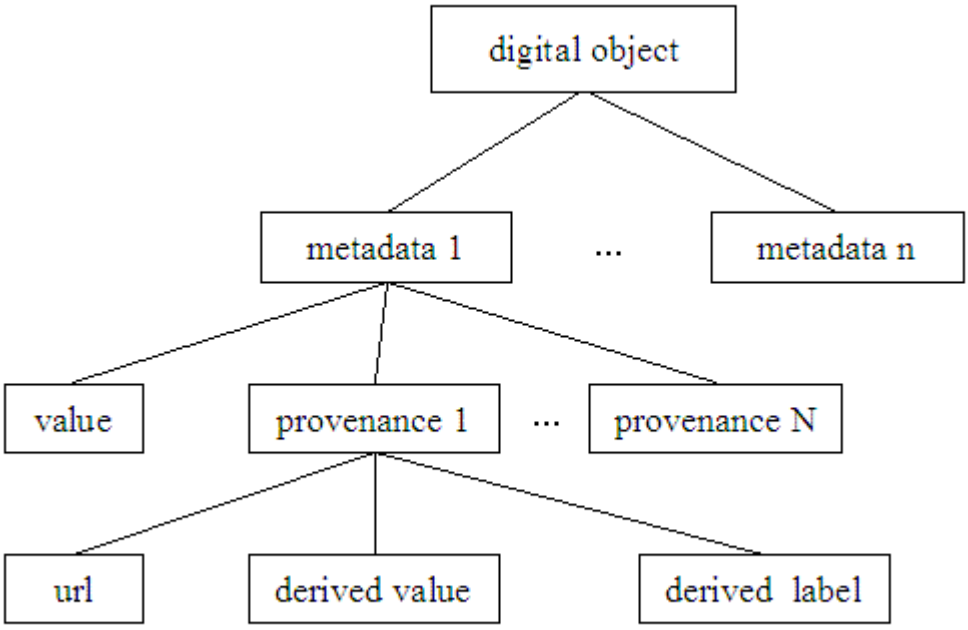

Figura 3.3 – Representação do modelo de proveniência como árvore DOM

3.4 Algoritmo de similaridade

Esta seção estabelece uma solução para o problema de similaridade entre os dados submetidos na tela pelo usuário e os dados armazenados no XML de proveniência a ser utilizado. Um estudo sobre algoritmos de similaridade foi realizado e uma comparação prática entre os algoritmos mais comuns será apresentada.

Algoritmos de similaridade possuem a finalidade de comparar duas strings e retornar um número que representa o quanto estes dois objetos são semelhantes. Há diversos algoritmos que atuam com este propósito, porém cada um atua de forma diferente. Para os cálculos de qual função de similaridade mais se adequa aos objetivos desejados são utilizados os índices de precisão e revocação. Estes índices são calculados a partir dos retornos dos testes com cada função de similaridade e com diferentes limiares. O índice de precisão é o número de registros relevantes recuperados sobre o número de registros recuperados, e o índice de revocação é o número de registros relevantes recuperados sobre o número total de registros relevantes. O melhor algoritmo a ser encontrado é o que possui um limiar onde ambas as taxas de precisão e revocação são altas.

Supondo que o XML de proveniência tenha um artigo com o título: *Um mecanismo para identificação, representação e consulta de versões de objetos XML oriundos de bibliotecas digitais.* Seria interessante que ao entrar com os dados na interface o usuário não precisasse escrever todo o título para fazer a consulta. O problema dos algoritmos de similaridade já existentes é que o cálculo de similaridade é feito integralmente, ou seja, para o título exemplificado acima, se um usuário entrasse com os dados *mecanismo para identificação localização objetos XML* a função de similaridade não retornaria um valor satisfatório. O único algoritmo que não atua integramente sobre uma lista de strings é o *Monge Elkan*, porém ele atua sobre caracteres, ou seja, detecta os nomes *Ana* e *Tiziana* como 100% similares.

Logo para o presente trabalho será necessário realizar uma função de similaridade que atue sobre *tokens* e não sobre toda a String. A função deve comparar cada *token* da string de entrada com cada *token* da string armazenada no modelo utilizado. Assim é garantido que ao submeter apenas uma parte do valor de um metadado de algum artigo este artigo será detectado como similar.

Além do problema de entradas parciais de dados para consulta, há também o problema de erros gramaticais ou de digitação por parte do usuário. É comum que nomes de pessoas sejam digitados de forma incorreta, ou que ao buscar informações que se encontram em outras línguas o usuário erre ao escrever alguma palavra. A ferramenta a ser desenvolvida precisa então prever que alguns erros podem ser cometidos pelo usuário. Para isto uma função de similaridade precisa ser utilizada na comparação de cada token de entrada com cada token armazenado.

Para resolver este problema foram testados os seguintes algoritmos de similaridade: Levenstein, Jaro Winkler, Distância Euclideana, NeedlemanWunch e Qgrams Distance. O objetivo dos testes realizados é simular entradas levemente diferentes de strings na interface e achar uma função com um limiar adequado para poder retornar o objetivo.

Foram realizadas 60 simulações para cada algoritmo proposto. As simulações incluíam testes entre strings levemente diferentes, como por exemplo *anelise* e *anelize*, e também entre strings parecidos, mas de significados diferentes como por exemplo *local* e *logical*.

Para cada algoritmo foi calculada a precisão e a revocação dos testes

para limiares entre 0.5 e 1.0, com variação de 0.1. Os resultados para cada limiar são apresentados nas tabelas 3.1, 3.2, 3.3, 3.4, 3.5 e 3.6.

Tabela 3.1: Precisão e revocação para limiar 0.5

	Levenstein	*Euclideana*	*Jaro Winkler*	Needleman Wunch	Qgrams Distance
Precisão	0,555	1.0	0,520	0.5	0,63
Revocação	1.0	0.033	1.0	1.0	0,966

Tabela 3.2: Precisão e revocação para limiar 0.6

	Levenstein	*Euclideana*	*Jaro Winkler*	Needleman Wunch	Qgrams Distance
Precisão	0,731	1.0	0,526	0,517	0,725
Revocação	1.0	0.033	1.0	1.0	0,966

Tabela 3.3: Precisão e revocação para limiar 0.7

	Levenstein	*Euclideana*	*Jaro Winkler*	Needleman Wunch	Qgrams Distance
Precisão	0,714	1.0	0,545	0,681	0,758
Revocação	1.0	0.033	1.0	1.0	0,733

Tabela 3.4: Precisão e revocação para limiar 0.8

	Levenstein	*Euclideana*	*Jaro Winkler*	Needleman Wunch	Qgrams Distance
Precisão	0,75	1.0	0,576	0,682	0,857
Revocação	0,9	0.033	1.0	0,933	0,4

Tabela 3.5: Precisão e revocação para limiar 0.9

	Levenstein	*Euclideana*	*Jaro Winkler*	Needleman Wunch	Qgrams Distance
Precisão	1.0	1.0	0,657	0,823	1.0
Revocação	0,233	0.033	0,833	0,466	0,033

Tabela 3.6: Precisão e revocação para limiar 1.0

	Levenstein	*Euclideana*	*Jaro Winkler*	Needleman Wunch	Qgrams Distance
Precisão	1.0	1.0	1.0	1.0	1.0
Revocação	0,033	0.033	0,033	0,033	0,033

A tabela 3.6 que representa os resultados para um limiar 1.0, apresenta as mesmas taxas de precisão e revocação para todos os algoritmos. Isso se

deve ao fato de que nos testes realizados apenas 1 dos testes foi realizado com 2 palavras idênticas. A partir dos testes realizados conclui-se que o algoritmo *Distancia Euclideana* não apresentou resultado satisfatório, pois considera similar apenas strings idênticas. O algoritmo *QGrams Distance* não apresenta uma taxa de revocação de 100% nem mesmo para o menor limiar testado, porém já apresentava uma taxa de precisão não satisfatória.

Levando em consideração os resultados conclui-se que o melhor algoritmo é o que possui a precisão mais alta para o mais alto número de revocação possível. O algoritmo de *Levenstein* possui 100% de revocação para no limiar 0,7, porém o índice de precisão é de 0,714. Este método foi o que apresentou comportamento mais próximo do desejado para a implementação da ferramenta.

Para a implementação proposta da ferramenta apresentada neste trabalho não há problemas se mais de um artigo é retornado para uma entrada não exata. Porém seria ruim se ao submeter com uma informação com 100% de certeza o usuário recebe como retorno diversos resultados, logo seria interessante que o usuário pudesse escolher o limiar ideal para utilizar a cada consulta.

O algoritmo a ser utilizado para o desenvolvimento da ferramenta será o *Levenstein*. Isto porque é o algoritmo que retorna com garantia o resultado esperado com o menor número de elementos não corretos.

3.5 Considerações finais

Neste capítulo foi apresentada a estrutura do modelo de proveniência a ser utilizado para a implementação da interface a ser desenvolvida. Para ajudar a manipulação do modelo pela ferramenta foi estudada a biblioteca DOM, que manipula arquivos XML ou HTML. Também foram apresentados alguns algoritmos de similaridade a serem utilizados na ferramenta. Dentre eles o que mais se adequou aos objetivos da interface foi o Levenstein.

O estudo destes três tópicos é de vital importância para o desenvolvimento da interface. O modelo XML precisa ser corretamente utilizado para a consulta e para o retorno dos artigos encontrado para a interface, e a biblioteca DOM facilitará este trabalho. A partir do algoritmo de similaridade encontrado serão realizadas as operações necessárias para a comparação entre os dados inseridos pelo usuário na aplicação e os dados armazenados no modelo de proveniência.

4 METADATAPROV: UMA INTERFACE WEB PARA A CONSULTA DA PROVENIÊNCIA DE METADADOS DE BIBLIOTECAS DIGITAIS

Este capítulo descreve a ferramenta METADATAPROV que é uma ferramenta desenvolvida para a visualização de dados oriundos de um modelo de metadados a respeito de artigos científicos disponíveis em bibliotecas digitais. A ferramenta tem como objetivo oferecer ao usuário uma interface web para a realização de consultas à artigos científicos oriundos de bibliotecas digitais integradas e visualização dos metadados relativos a proveniência em diferentes bibliotecas digitais.

O capítulo inicia com uma apresentação geral da ferramenta. Em seguida, é apresentada a arquitetura escolhida para o desenvolvimento da ferramenta e os métodos utilizados para a realização da consulta e visualização dos resultados. Por fim, são apresentados alguns exemplos práticos de utilização do programa, onde são descritos passo a passo o funcionamento de cada módulo da ferramenta.

A modelagem e projeto da ferramenta, desenvolvidos nos padrões UML estão detalhadamente descritos no Apêndice A.

4.1 Visão geral da ferramenta

MetadataProv, sigla de *metadata provenance*, ou seja, proveniência de metadados. A ferramenta implementada é uma aplicação web desenvolvida com o objetivo de oferecer aos usuários uma interface amigável para a consulta e visualização de proveniência de metadados de artigos científicos oriundos de diferentes bibliotecas digitais. Para oferecer uma interface de fácil visualização para os usuários MetadataProv utiliza um esquema de árvores para a visualização das informações de proveniência para cada metadado, sendo estas carregadas juntas após a operação de pesquisa, não necessitando submeter uma nova operação a cada evento de visualização selecionado na tela.

4.1.1 Tecnologias utilizadas para a implementação

As linguagens de programação utilizadas para a implementação da ferramenta são Java 1.5[1] e JSP. A linguagem Java foi escolhida pelas seguintes qualidades:

- É gratuita;

- Portabilidade – ou seja, independente de plataforma;
- Orientada a objetos – facilidade de manipulação dos resultados recuperados do nosso modelo XML;
- Bibliotecas – possui bibliotecas para manipulação de arquivos XML e também bibliotecas para funções para diversos algoritmos de similaridade;

A escolha de se desenvolver uma aplicação Web se deu ao fato de possibilitar uma futura disponibilização da ferramenta em um servidor, para poder ser acessada por qualquer usuário. Para a implementação da interface foi utilizada então o JSP (Java Server Pages), que é um código que integra HTML (HyperText Markup Language) com Java. O JSP é interpretado e transformado em servlets que interagem com a aplicação Java.

Para a implementação da ferramenta foi utilizado o software Eclipse 3.3.0 em conjunto com o Tomcat 4.1, ambos de software livre e código aberto. O Eclipse foi escolhido como ferramenta de desenvolvimento por possuir plugins para auxilio no desenvolvimento, compilação automática de código com opções para depuração e reconhecimento de documentos JSP. O Tomcat é um servidor de aplicações Java para Web de fácil instalação e utilização, robusto o suficiente para ser utilizado por uma aplicação em nível de produção.

4.2 Visão Geral do Projeto METADATAPROV

A função da ferramenta é permitir ao usuário realizar uma consulta ao modelo de proveniência. Esta consulta deve identificar o objeto no modelo, a partir dos dados de entrada submetidos pelo usuário. Após, os resultados são exibidos hierarquicamente para a melhor compreensão e visualização pelo usuário. O modelo de casos de uso é único, sendo apresentado na Figura 4.1.

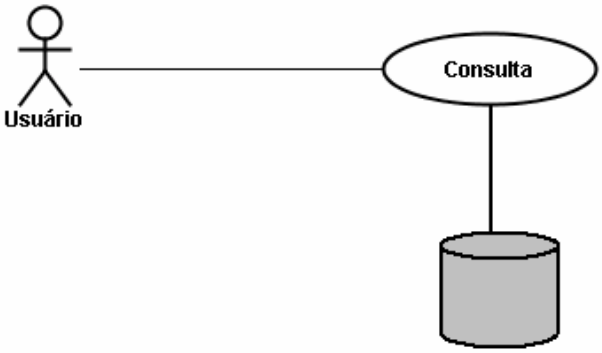

Figura 4.1 – Diagrama de caso de uso

Para o desenvolvimento da ferramenta foram necessárias seis classes, como apresentado no diagrama de classes da figura 4.2. Estas seis classes são:

- ServletPrincipal: esta classe é a responsável por fazer a comunicação entre a tela e o serviço. É ela que recupera os

dados e eventos submetidos pela interface e os encaminha para a classe ReadXML. Após o processamento, esta classe recebe os resultados das operações exercidas e encaminha o resultado para a interface;

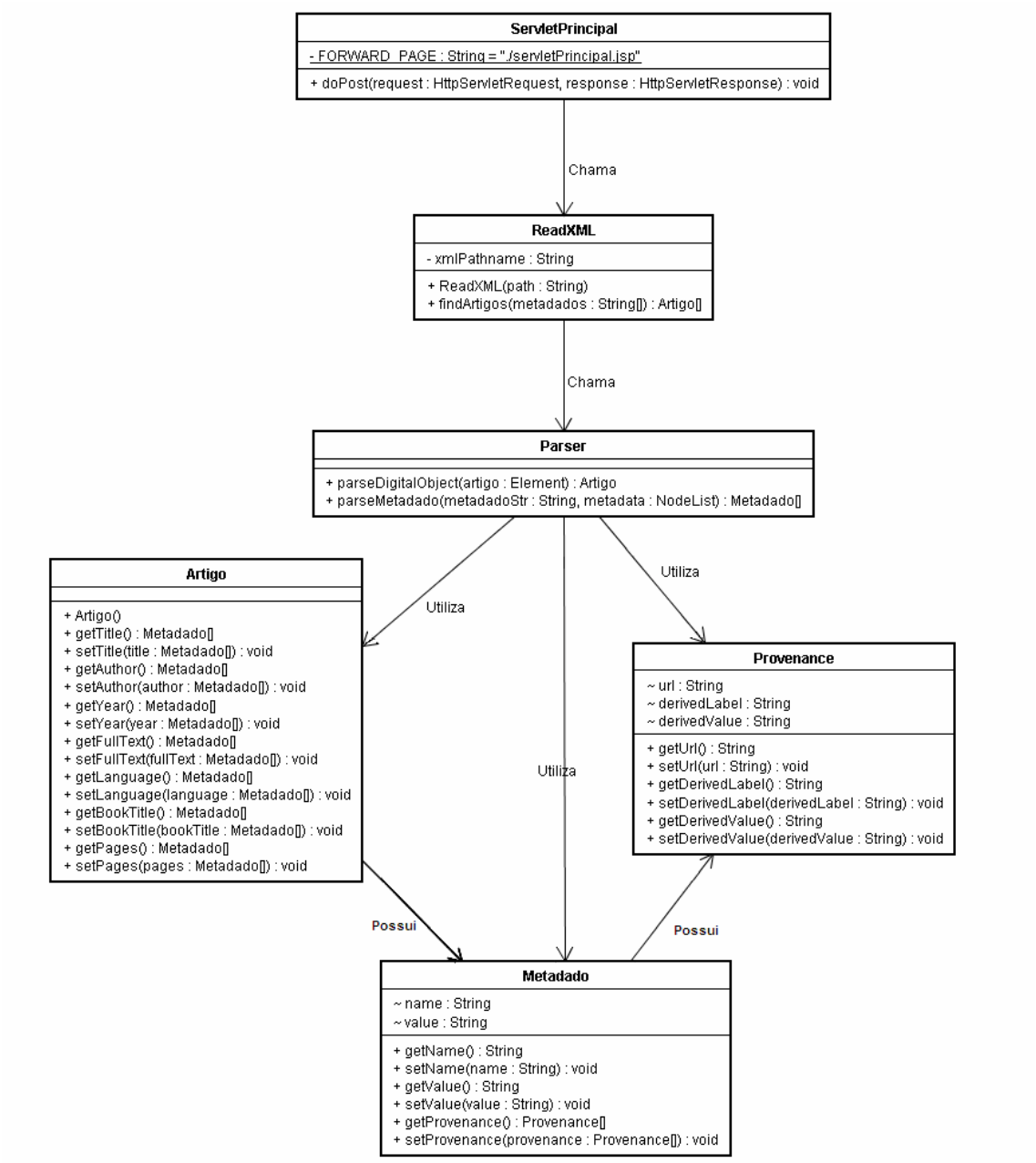

Figura 4.2 – Diagrama de Classes

- ReadXML: e a classe que faz as comparações necessárias para encontrar um artigo na estrutura de dados que implementa o modelo de proveniência que é similar ao submetido pela interface. Ao encontrar um artigo similar, esta classe encaminha os dados para a classe Parser e recebe como retorno um objeto artigo;
- Parser: nesta classe é feita a transformação de um elemento XML que representa um artigo em um objeto do tipo Artigo;

- Artigo: esta classe representa um artigo. Possui um array do tipo Metadado para cada metadado que pode existir em um artigo científico;
- Metadado: esta classe é genérica e representa qualquer metadado de um artigo. O atributo *name* define qual metadado o objeto representa. Possui um array do tipo Provenance que armazena todas as proveniências de um metadado;
- Provenance: representa um elemento proveniência de qualquer metadado.

4.3 Descrição da ferramenta

Esta seção apresentada as principais funcionalidades da ferramenta implementada e a maneira encontrada para resolver as questões em aberto identificadas nos capítulos anteriores. Primeiro é apresentada a tela inicial da interface. Em seguida é simulado um caso real para a consulta de um artigo existente na base XML. Nesta etapa são explicados os passos tomados para a execução das consultas. No final, são apresentados os resultados da consulta e definidos os componentes utilizados para visualização do resultado.

A tela inicial da interface é apresentada na figura 4.3. A interface simples disponibiliza um ícone ao lado do botão “Pesquisar”, que abre uma tela de ajuda. A ajuda, figura 4.4, possui uma relação de perguntas e respostas mais freqüentes pelos usuários da ferramenta. Além disso, na tela inicial, cada campo texto ou botão disponível possui um ToolTip de ajuda. Tooltip é um evento HTML que abre um texto de ajuda ao se posicionar o cursor acima de algum componente da tela. Na figura 4.3 o cursor aponta para o ícone de tela de ajuda, que possui o Tooltip com o string “Tela de ajuda”.

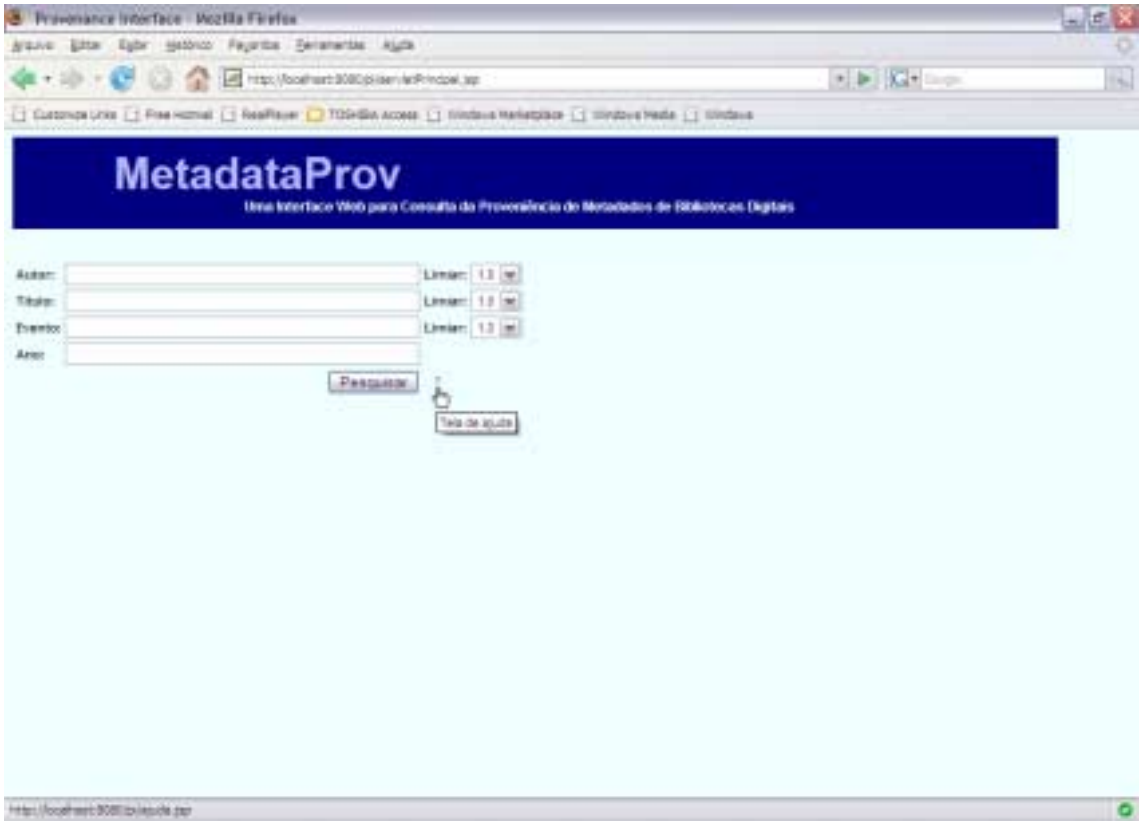

Figura 4.3 – Tela inicial

Figura 4.4 – Tela de ajuda

4.3.1 Realizando uma pesquisa

A tela inicial da interface possui 4 campos de preenchimento, sendo que nenhum deles de preenchimento obrigatório. O usuário pode optar em preencher apenas 1 campo ou mais, conforme as informações que souber sobre o artigo que deseja encontrar. Os campos de entrada são:

- Autor: se refere ao elemento *author* do documento XML. Esta entrada possibilita ao usuário entrar com o nome do autor do artigo que deseja pesquisar.
- Título: se refere ao elemento *title* no documento XML. Nesta área de texto o usuário pode submeter parte ou todo o título do artigo que deseja pesquisar.
- Ano: se refere ao elemento *year* do documento XML. Ao preencher esta entrada o usuário especifica o ano de publicação do artigo que deseja submeter para pesquisa.
- Evento: se refere ao elemento *booktitle* do documento XML. Esta informação se refere ao ano de publicação em que o artigo a ser pesquisado foi submetido.

Outro campo que pode ser selecionado é a *combobox* de seleção para o Limiar a ser utilizado para a pesquisa. O limiar é o valor limite utilizado pela função de similaridade. Se uma comparação entre dois strings retornar um valor superior ou igual ao limiar, então as strings são detectadas como similar, caso contrário não serão similares. Para a comparação quanto maior o limiar, menos suscetível a erros deve ser a entrada. Para o caso de um usuário de MetadataProv, quanto menos certeza um usuário tiver sobre alguma informação de algum artigo, menor deve ser o limiar utilizado para a pesquisa. A opção de seleção do limiar é possível para os campos autor, título e evento. O *combobox* possui os valores de 0,5 a 1.0, com variação de 0,1, como apresentado na Figura 4.5.

A figura 4.6 simula uma pesquisa onde o usuário tem informações sobre o nome de um autor e o ano de publicação. Para o metadado Autor o usuário optou por selecionar um limiar de valor 0,8, pois não tem certeza em relação à grafia do sobrenome do autor.

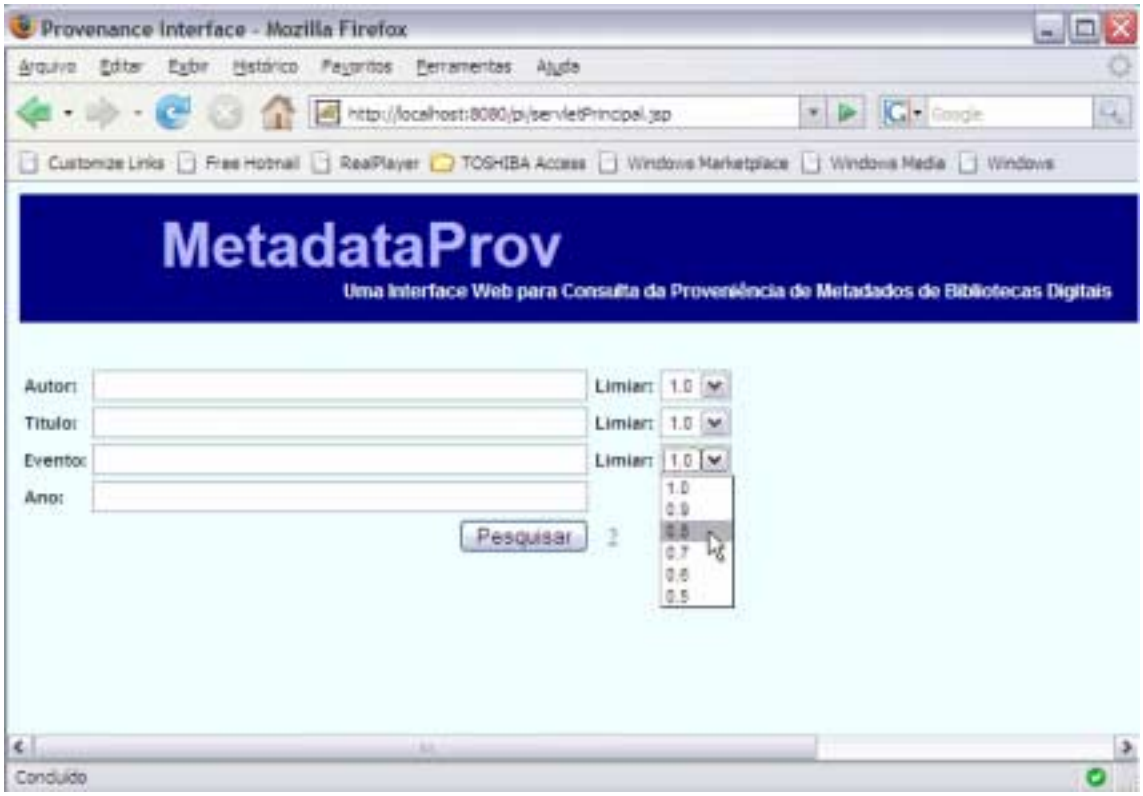

Figura 4.5 – *Combobox* com limiares

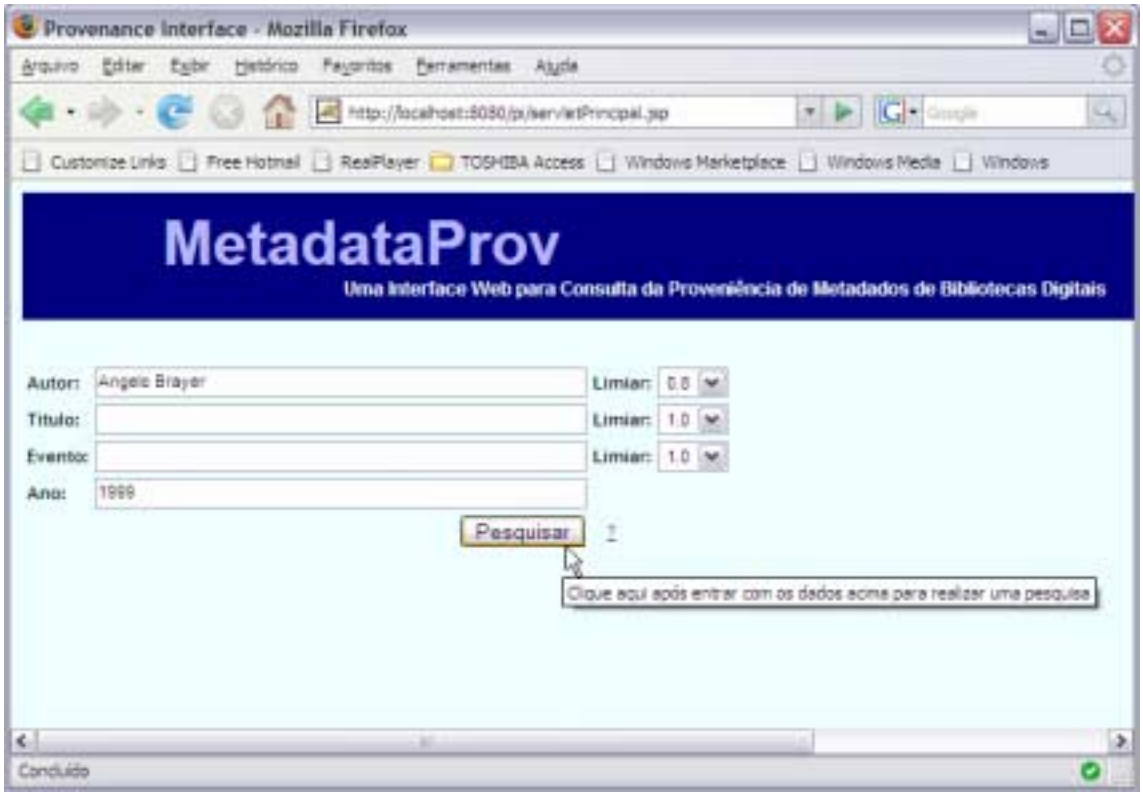

Figura 4.6 – Pesquisa por nome e ano de publicação

O servlet Java recebe do JSP, após o evento do botão Pesquisar, o conteúdo de cada um dos campos de pesquisa. Então, cria um array de 4

posições e o passa como parâmetro para o método findArtigos(String[] metadados) da classe ReadXML.java.

4.3.1.1 Detectando artigos similares

O método findArtigos(String[] metadados), da classe ReadXML.java, é o responsável por detectar similaridade entre o conteúdo do array *metadados*[], submetido pelo usuário na interface e os metadados de um artigo representado pelo elemento *digitalobject* do modelo XML. Este método possui um array, nomeMetadados[], de 4 posições que possui um label para cada posição. Este label corresponde em posição ao metadado que cada elemento do array *metadados[]* que contém as strings submetidas na interface. Para o nosso exemplo temos os arrays definidos na figura 4.7.

```
metadados[0] = "Angelo Brayer";
metadados[1] = " ";
metadados[2] = " ";
metadados[3] = "1999";
```

```
nomeMetadados[0] = "author";
nomeMetadados[1] = "title";
nomeMetadados[2] = "booktitle";
nomeMetadados[3] = "year";
```

Figura 4.7 – Arrays associados de metadados

O método que procura por artigos similares percorre cada um dos artigos contidos no documento XML. Para cada um dos artigos o programa recupera, a partir do array nomeMetadados[], o valor dos elementos passados como parâmetro na tela. Em seguida recupera o conteúdo da tag *value* e faz uma comparação por similaridade ao valor submetido pela interface. Se não for similar ainda compara com cada uma das tags *derivedvalue* de cada elemento de proveniência. Se a similaridade for maior do que o limiar selecionado na interface pelo usuário em cada um dos casos então passa para o próximo metadado a ser analisado. Se todos os valores de *metadados[]* não nulos recebidos por parâmetro forem similares aos correspondentes metadados recuperados do arquivo XML então o artigo analisado é um forte candidato a ser o artigo especificado pelo usuário, e deverá ser retornado para o resultado da consulta.

No caso do exemplo da figura 4.6 os elementos *author* e *year* de cada *digitalobject* são recuperados e comparados com os dados entrados pelo usuário na interface. A figura 4.8 apresenta o elemento *digitalobject* de um artigo com metadados similares aqueles informados pelo usuário na tela de pesquisa. O metadado *author* informado pelo usuário é comparado com o valor da tag *value*, na linha 432 e a função de similaridade detecta 100% de similaridade para o token “Angelo“ e 85% de similaridade para o token “Brayer”, ambos submetidos pelo usuário para a consulta. Para o valor do campo ano uma comparação de igualdade é realizada com a tag value da linha 458, e as duas strings são detectadas como idênticas. Logo, o artigo será retornado como resultado da pesquisa.

4.3.1.2 Transformando elemento digitalobjetc em Objetos Java

Antes de retornar o artigo para a tela é necessário manipular o seu conteúdo. O método parseDigitalObject(Element tagArtigo), da classe Parser.java, é responsável por transformar o elemento *digitalobject,* detectado como similar, em um objeto do tipo Artigo.

```
<digitalobject>
  <title>
    <value>Recovery in multidatabase systems</value>
    <provenance>
      <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
      <derivedlabel> </derivedlabel>
      <derivedvalue> </derivedvalue>
    </provenance>
    <provenance>
      <url>http://dblp.uni-trier.de</url>
      <derivedlabel> </derivedlabel>
      <derivedvalue>Recovery in Multidatabase Systems.</derivedvalue>
    </provenance>
  </title>
  <author>
    <value>Angelo Brayner</value>
    <provenance>
      <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
      <derivedlabel>creator</derivedlabel>
      <derivedvalue> </derivedvalue>
    </provenance>
    <provenance>
      <url>http://dblp.uni-trier.de</url>
      <derivedlabel> </derivedlabel>
      <derivedvalue> </derivedvalue>
    </provenance>
  </author>
  <author>
    <value>Theo Haerder</value>
    <provenance>
      <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
      <derivedlabel>creator</derivedlabel>
      <derivedvalue> </derivedvalue>
    </provenance>
    <provenance>
      <url>http://dblp.uni-trier.de</url>
      <derivedlabel> </derivedlabel>
      <derivedvalue>Theo Hauml;rder</derivedvalue>
    </provenance>
  </author>
  <year>
    <value>1999</value>
    <provenance>
      <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
      <derivedlabel>date</derivedlabel>
      <derivedvalue> </derivedvalue>
    </provenance>
    <provenance>
      <url>http://dblp.uni-trier.de</url>
      <derivedlabel> </derivedlabel>
      <derivedvalue> </derivedvalue>
    </provenance>
  </year>
  <language>
  <booktitle>
    <value>sbbd</value>
    <provenance>
      <url>http://www.lbd.dcc.ufmg.br/bdbcomp</url>
      <derivedlabel>source</derivedlabel>
      <derivedvalue> </derivedvalue>
    </provenance>
    <provenance>
      <url>http://dblp.uni-trier.de</url>
      <derivedlabel> </derivedlabel>
      <derivedvalue> </derivedvalue>
    </provenance>
  </booktitle>
  <versions>yes</versions>
</digitalobject>
```

Figura 4.8 – Elemento *digitalobject* similar ao pesquisado

Para isto este método recupera os elementos de cada metadado de *digitalobject* e chama o método parseMetadado(String metadadoSTR, Nodelist metadata) para cada metadado. Este método é o método que cria um objeto do tipo Metadado para cada elemento que representa um metadado em *digitalobject*. Além disso, cria objetos Provenance para cada elemento *provenance* recuperado nos metadados. Se um artigo não possuir um ou mais dos metadados associados, o método a ser chamado é parseMetadadoVazio(), que cria um elemento metadado com conteúdo nulo. Este procedimento foi especificado para evitar problemas no momento da manipulação dos resultados na tela. Um esboço dos objetos criados a partir do exemplo de pesquisa ilustrado na figura 4.6 é apresentado na figura 4.9.

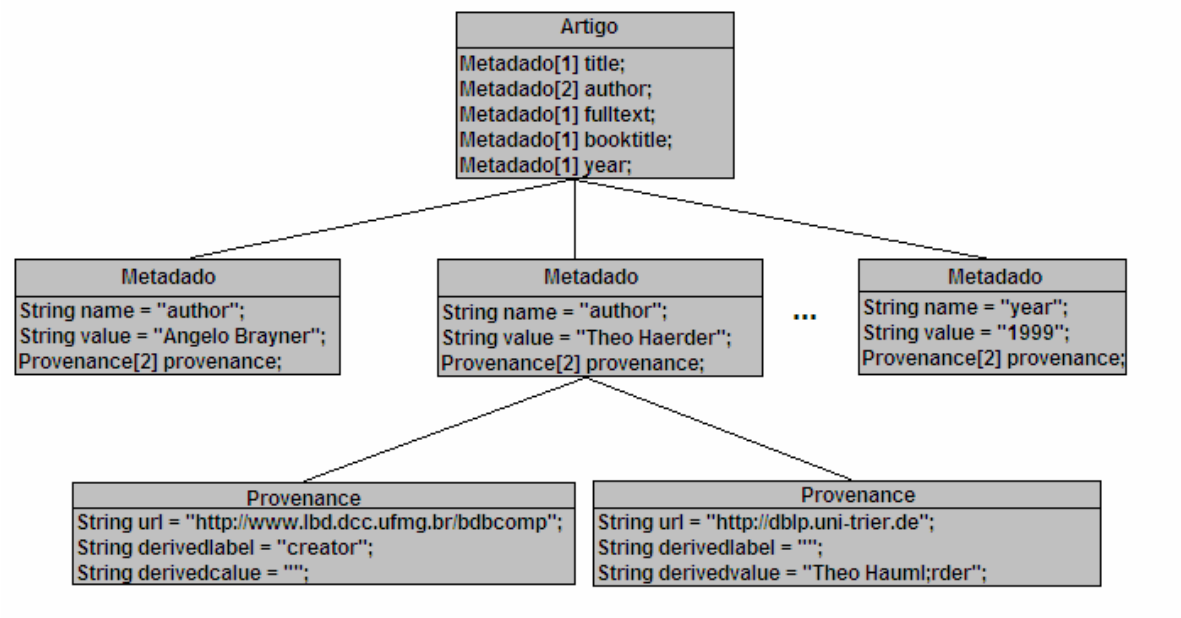

Figura 4.9 – Esboço dos Objetos Java criados

Finalizada a tag *digitalobject* todos os objetos disponíveis para uma manipulação dos resultados na tela estão criados. O artigo retornado é então armazenado no método findArtigos(String[] metadados), e o processo se repete para todos os artigos disponíveis no documento XML. O retorno deste método é um array de objetos do tipo Artigo, que é retornado por servlet para o JSP que ira manipular os resultados na tela.

O fluxo dos métodos descritos é apresentado na figura 4.10. A interface encaminha 2 arrays de strings com as informações submetidas pelo usuário para o método *findArtigos(String[]metadados, String[] limiares)*. Este método encontra um elemento *digitalobject* similar ao array de metadados e encaminha *digitalobject* para *parseDigitalObject(Element artigo)* que percorre o elemento chamando o método *parseMetadado(String metadadoSTR, Nodelist metadata)* para cada metadado encontrado. Para metadados não presentes em *digitalobject* o método chamado é *parseMetadadoVazio()* que retorna um objeto Metadado sem valores. A partir dos objetos Metadados retornados pelos dois métodos o método *parseDigitalObject(ElementArtigo)* cria um objeto do tipo Artigo e retorna para o método *findArtigos(String[]metadados, String[] limiares)* que armazena os resultados em um array do tipo Artigos e após toda sua execução retorna para a interface este array criado.

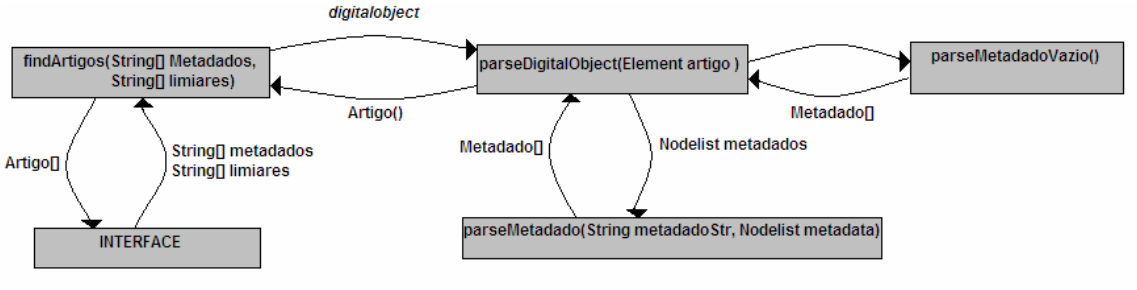

Figura 4.10 – Fluxo dos métodos para realização de uma pesquisa

4.3.2 Apresentando os resultados na tela

Após a submissão e recuperação dos metadados de entrada da pesquisa um array de objetos Artigo é retornado para a interface. Este array tem tamanho máximo de 50 objetos. Se esta cota for ultrapassada é retornado para a tela uma mensagem de erro, pedindo ao usuário para ser mais específico em relação aos critérios de pesquisa. Esta restrição foi necessária pois para o caso de muitos elementos *digitalobject* necessitarem de manipulação para serem transformados em objetos do tipo Artigo o programa perderia muito em performance. Além disso, uma grande quantidade de dados necessitaria ser trafegada pela rede, o que implicaria em uma grande demora por parte do usuário. Um exemplo de pesquisa que retorna mais de 50 artigos é apresentado na figura 4.11, onde um usuário utiliza apenas o ano como dado de entrada. Para o caso de uma pesquisa retornar um array de tamanho zero, a interface imprime uma mensagem na tela avisando o usuário que não há nenhum artigo na base de dados similar aos metadados submetidos para Pesquisa. Um exemplo desta tela se encontra na Figura 4.12.

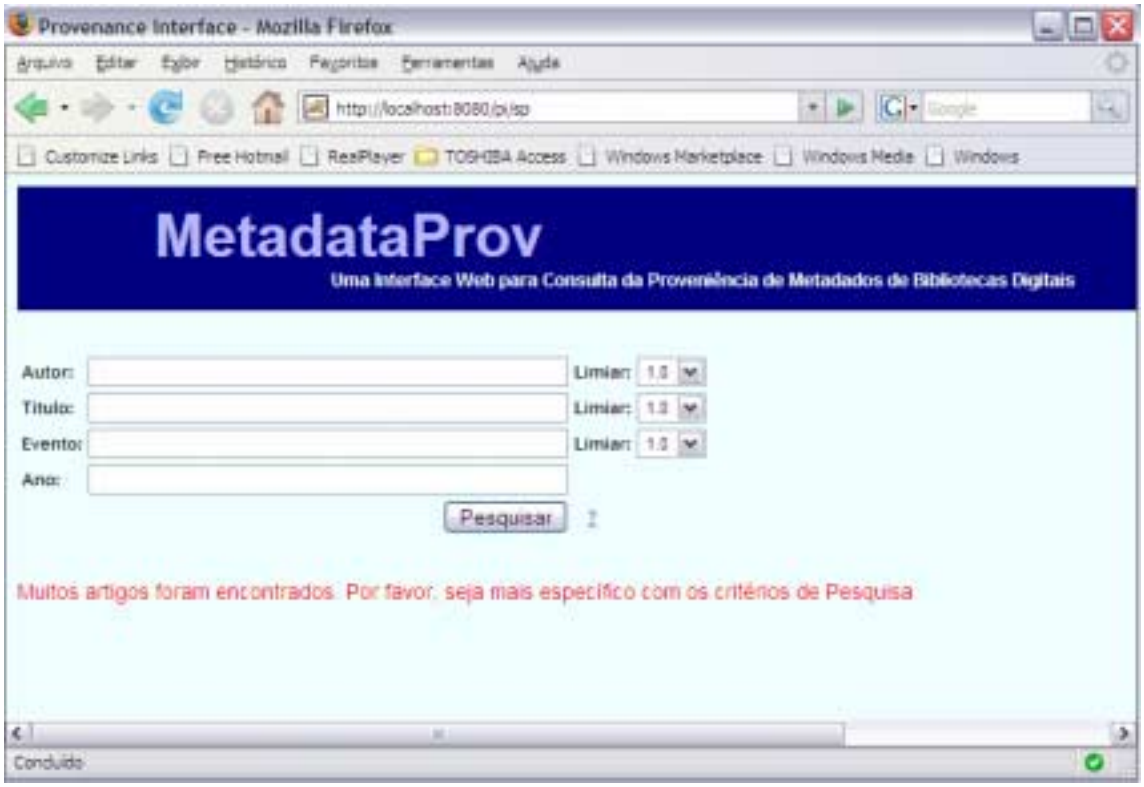

Figura 4.11 – Tela de alerta para retorno grande

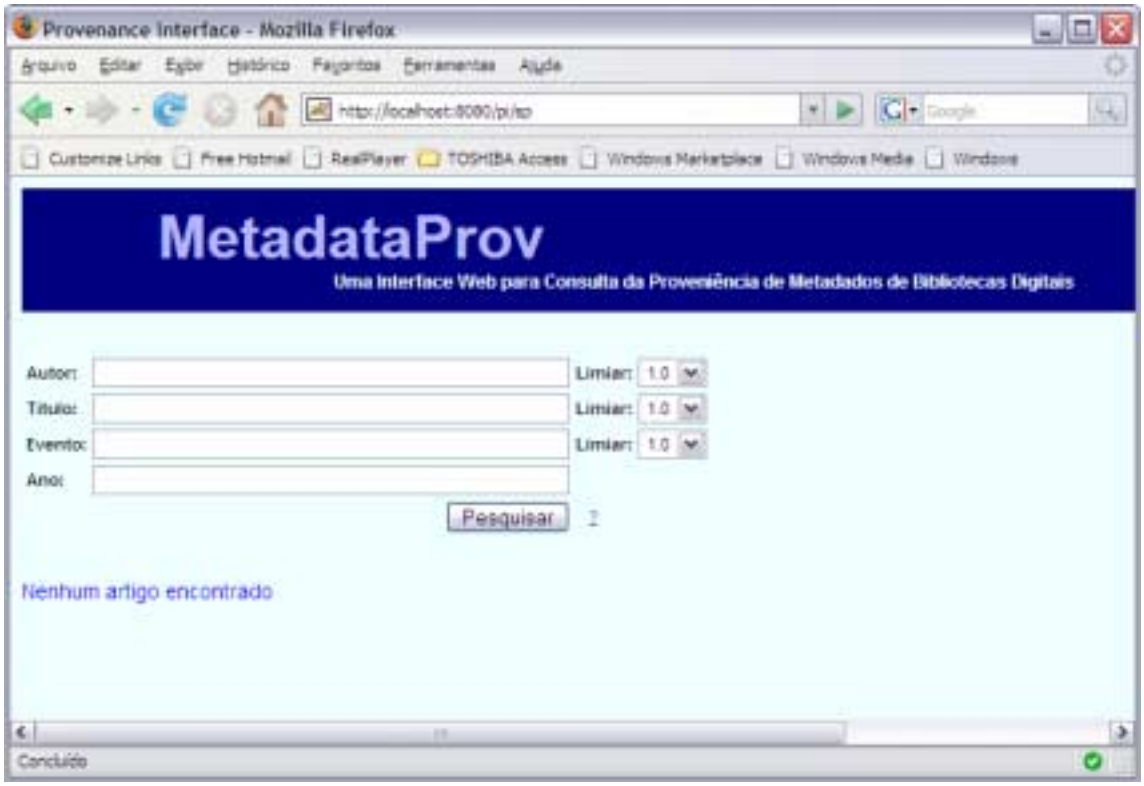

Figura 4.12 – Tela com aviso de nenhum artigo encontrado

Quando o array não ultrapassa os 50 artigos os artigos retornados são apresentados na tela de resultado da pesquisa. O retorno do exemplo proposto na figura 4.6 é apresentado na figura 4.13. Pode-se notar que os valores dos metadados são os mesmos das tags *value* apresentadas no XML da figura 4.8. Para esta consulta, a base de dados apresentava apenas 1 artigo similar aos metadados de entrada. O número de artigos retornados é exibido na tela em um label que aparece antes da apresentação dos resultados. Para uma melhor disposição do retorno das informações de proveniência para cada metadado foi utilizada uma applet JavaScript que monta uma árvore, onde é possível expandir ou ocultar algumas informações retornadas na tela.

4.3.2.1 TreeView Menu JavaScript Applet para apresentação dos metadados de proveniência

TreeView [14] é um menu de controle, suportado pela maioria dos browsers, onde as informações são apresentadas em ordem hierárquica, com o tópico principal no topo e os itens subordinados abaixo. O sinal (+) indica que uma informação possui filhos e pode ser expandida. O sinal (-) indica que se pode retrair as ramificações exibidas pelo tópico principal.

O menu em árvore possui um arquivo chamado ftiens4.js, que contém todas as funções JavaScript necessárias para a construção das árvores. Este arquivo, a cada evento, executa uma condição para detectar em qual browser a árvore está sendo executada. Assim pode-se decidir qual método utilizar, pois os browser utilizam diferentes funções para manipulação de seus componentes. TreeView também possui diversos frames de figuras que são utilizados para cada um dos estados de uma árvore.

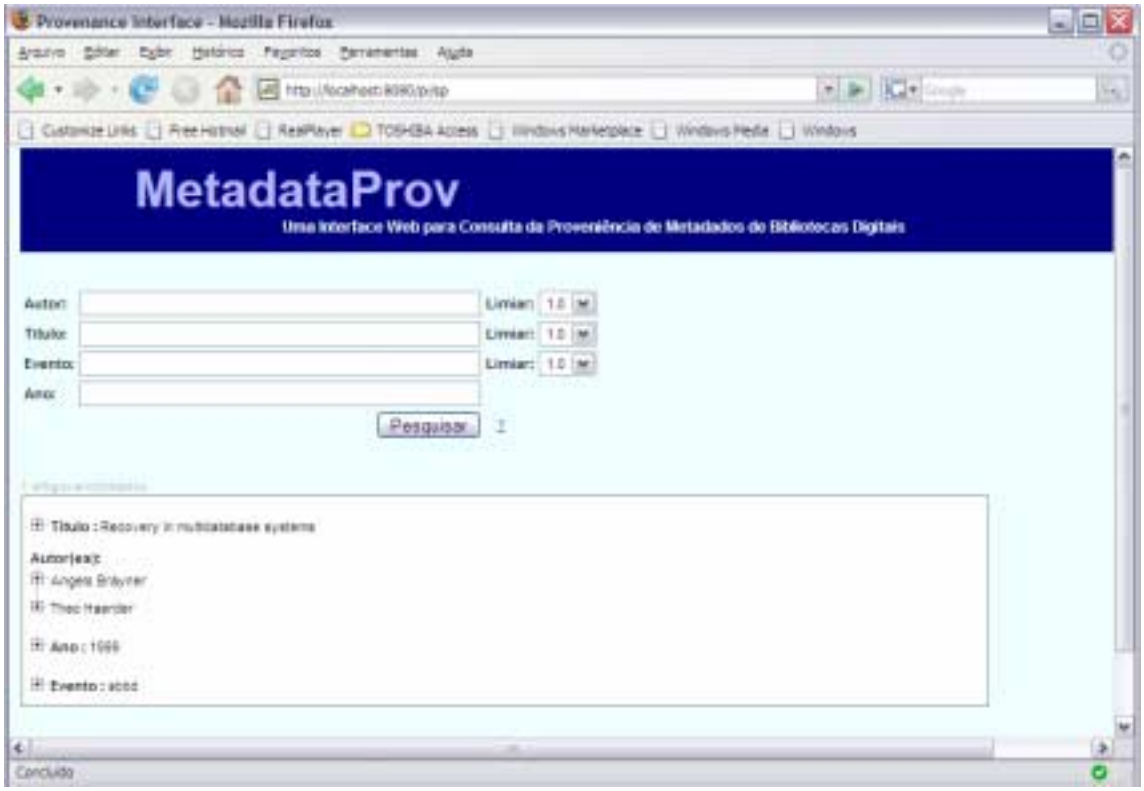

Figura 4.13 – Artigo retornado na pesquisa

Para montar uma árvore com a applet TreeView na interface do trabalho o arquivo ftiens4.js foi adicionado ao projeto Java, juntamente com os arquivos de imagem necessários para a criação da árvore. No servlet JSP é realizada uma operação de inclusão do arquivo ftiens4.js para a utilização das funções disponíveis para a criação da árvore.

Para a apresentação dos resultados na tela de retorno é criada uma árvore para cada metadado do artigo retornado. Para o metadado "*autor"* uma árvore é criada para cada autor do artigo. Antes da criação de cada árvore uma lista de atributos que representam o seu comportamento inicial é setada. O atributo "STARTALLOPEN" é inicializado com o valor 0, ou seja, a árvore é inicializada no modo não expandido.

Ao criar uma árvore para um metadado o valor recebido pelo nodo raiz da árvore é o conteúdo da tag "*value*" do modelo de proveniência. Cada árvore contém um nodo folha, que contém as informações de proveniência sobre o metadado que é apresentado na raiz. O array de proveniência de cada metadado é iterado e exibido com cores e posicionamento intercalados. As informações sobre a proveniência apresentados na tela são as das tags "url", "derivedlabel" e "derivedvalue". Um esboço da árvore de proveniência expandida é apresentado na figura 4.14. O esboço se refere ao *digitalobject* do exemplo do documento XML apresentado na figura 4.8.

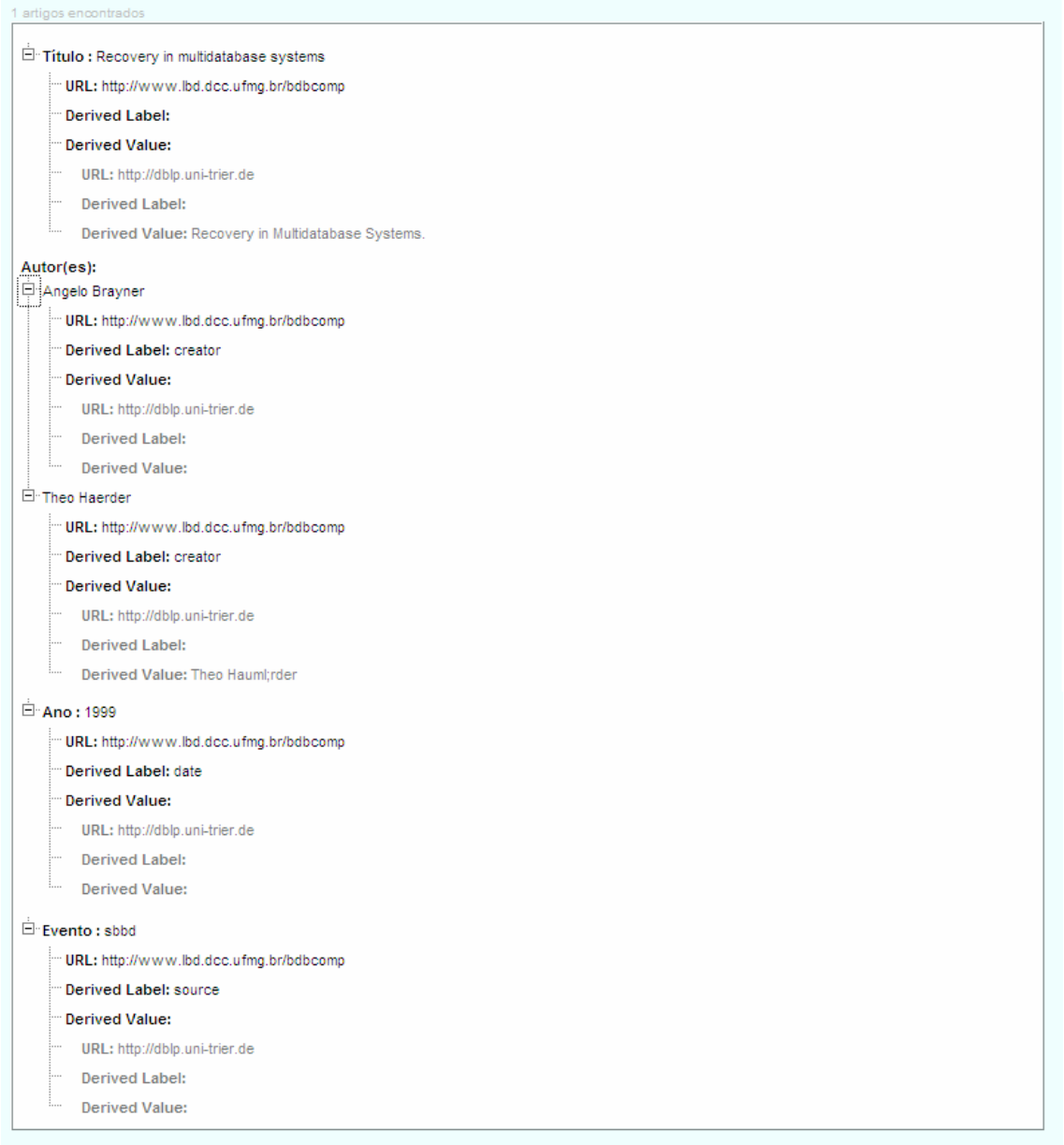

Figura 4.14 – Árvore de proveniência expandida

4.3.2.2 Paginação

Pode ter casos onde ao submeter alguns critérios de pesquisa, o usuário receba como retorno um array com dezenas de artigos. Por exemplo, se forem exibidos 40 artigos, fica visualmente difícil para o usuário analisar todos os resultados. Para resolver este problema é proposto um modelo para paginação dos resultados. Se o array de artigos retornado for maior do que 10 e menor do que 50 os resultados serão paginados no retorno para a interface. Um exemplo de consulta que retorne os resultados com paginação é apresentado nas figuras 4.15 e 4.16. Neste exemplo, 23 artigos são retornados em 3 páginas após uma consulta onde o único dado de entrada é a string “sbbd” para o metadado evento.

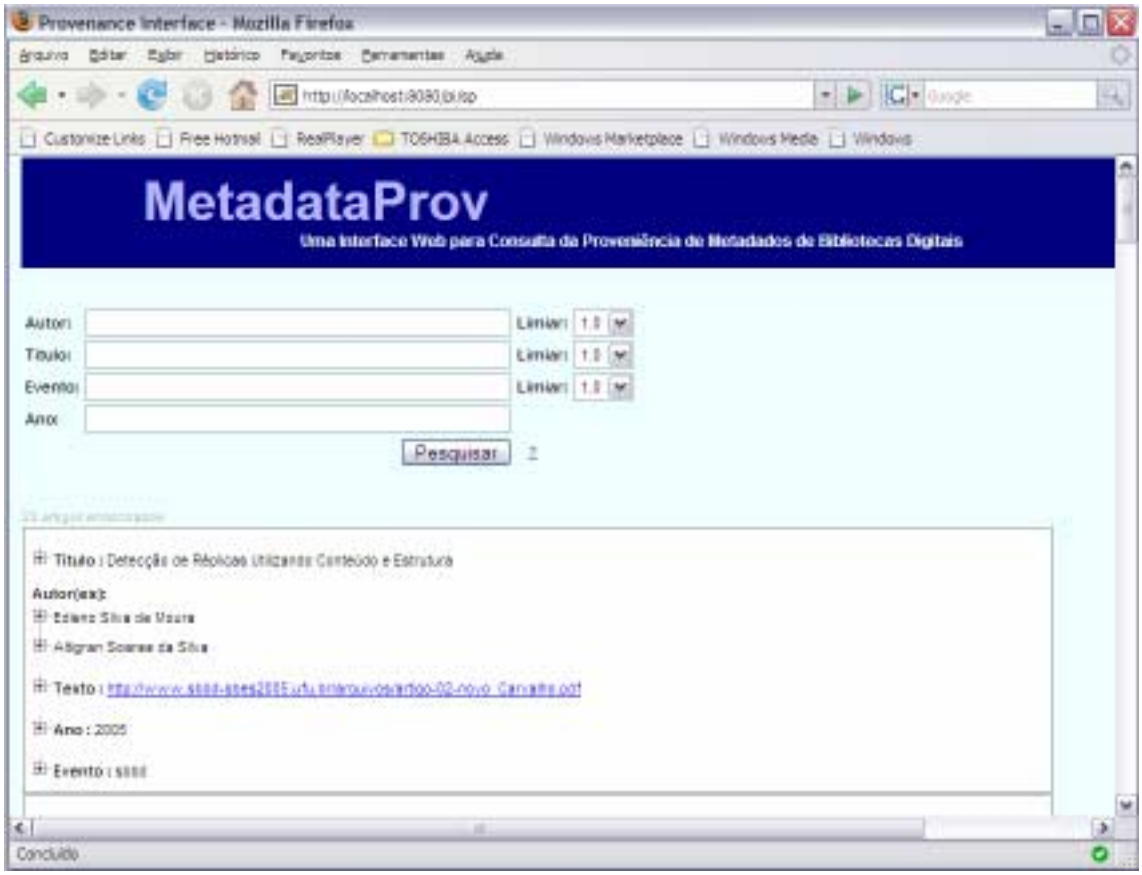

Figura 4.15 – Exemplo que retorna 23 artigos como resultado da consulta

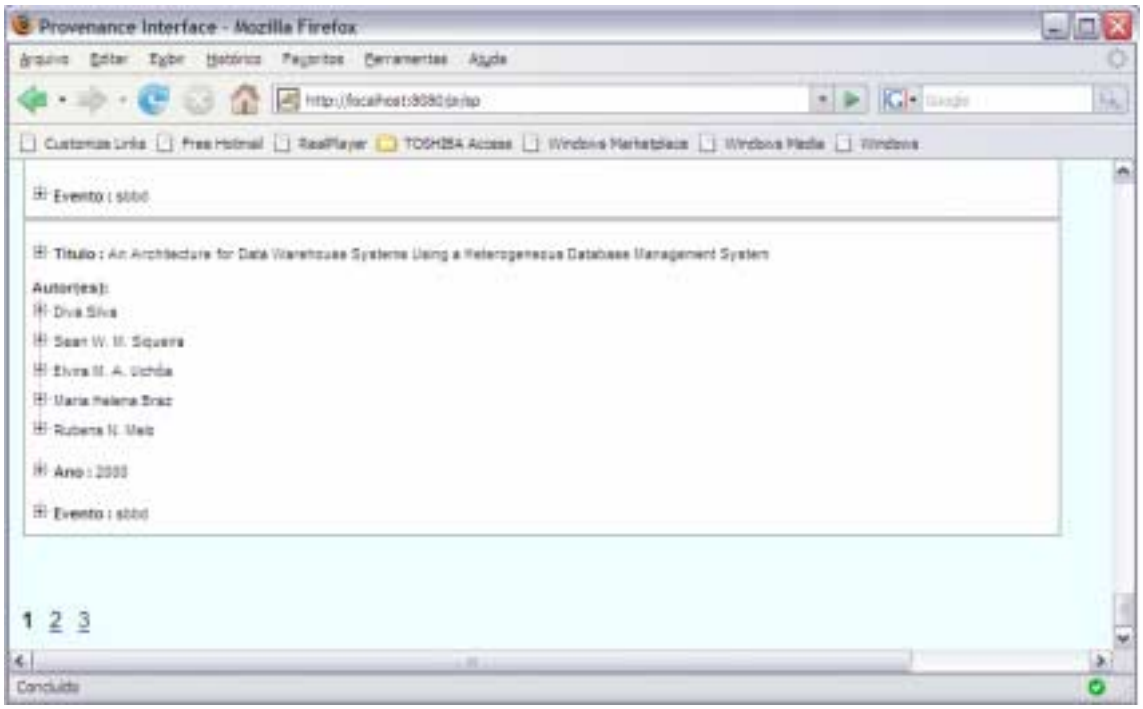

Figura 4.16 – Exemplo onde 3 páginas são utilizadas para paginação dos resultados

4.4 Considerações finais

MetadataProv, a ferramenta desenvolvida para consulta e visualização de informações sobre proveniência de metadados oriundos de diferentes bibliotecas digitais, gera uma interface simples e completa para a realização de consultas a artigos científicos oriundos de diferentes bibliotecas digitais e visualização das informações sobre as diferentes versões do objeto encontrado. Para a consulta de artigos o que já existe são ferramentas de consulta em cada biblioteca digital, porém ao procurar na Web por um artigo é difícil detectar quais artigos retornados são a mesma versão, apenas indexados por bibliotecas digitais diferentes. Com o modelo único de armazenamento todas as referências de um mesmo artigo são armazenadas em conjunto, e o trabalho desenvolvido permite a consulta a todos os metadados recuperados de cada versão.

A ferramenta foi desenvolvida para Web, para que qualquer usuário possa realizar consultas para a visualização das informações de proveniência ou apenas para encontrar referências do artigo desejado, ou de artigos de um mesmo autor, ou mesmo de algum evento.

5 CONCLUSÃO

Este trabalho apresentou a ferramenta MetadataProv, uma ferramenta Web para consulta e visualização de informações de proveniência de metadados de bibliotecas digitais. Usando a ferramenta MetadataProv o usuário pode realizar uma consulta à base de dados a partir de uma informação sobre o artigo que deseja pesquisar, e recebe como retorno o artigo desejado, visualizando tanto o conteúdo dos seus metadados quanto as suas proveniências.

A ferramenta tem como objetivo oferecer uma interface simples onde o usuário pode realizar consultas a artigos científicos a partir do conteúdo dos seus metadados e receber como retorno todos os artigos que possuem metadados similares aos metadados de entrada. Após o retorno da pesquisa a ferramenta apresenta os resultados estruturados em forma de árvore, o que facilita a visualização uma vez que as infomações sobre a proveniência de cada metadado são retornadas colapsadas, ou seja, o usuário só visualiza as informações que forem de seu interesse.

A principal contribuição da ferramenta é disponibilizar uma forma simples e eficaz para a consulta do modelo de armazenamento de um sistema de integração de bibliotecas digitais. A partir desta ferramenta Web é possível disponibilizar o acesso para qualquer usuário de bibliotecas digitais, sem a necessidade de instalação ou atualizações do software e com a garantia que terão uma forma segura de consulta à base de dados de integração de artigos, sem perda de informação relevante e livre de redundância.

O desenvolvimento da ferramenta MetadataProv abre possibilidade para diversos trabalhos futuros. Um trabalho futuro que se pretende desenvolver é adaptar o modelo em árvore TreeView para possibilitar uma quantidade não limitada de níveis de hierarquia. Com um número ilimitado de níveis seria possível adicionar mais um nível para as informações de proveniência, sendo apresentado colapsado os valores dos atributos *derivedlabel* e *derivelvalue* para cada biblioteca digital á que o artigo analisado provém.

Outra melhoria que se pretende realizar é a paginação por funções JavaScript. Atualmente o processo é realizado por operações de *request*, porém com o processo realizado por JavaScript seria necessário o carregamento único dos resultados na tela, não necessitando executar uma nova pesquisa para cada mudança de página.

Outra melhoria ainda é realizar uma conversão para caracteres que possuem acentos, antes de submeter os dados da interface para o programa Java. Se um usuário submete uma consulta com uma palavra com acentos o

JSP retorna para o Java uma string desconfigurada, que não segue um padrão detectado até o momento. Realizando uma conversão na interface é garantido que os caracteres chegarão corretamente codificados no programa Java.

6 REFERÊNCIAS

[1] Eduardo Nunes Borges, Renata de Matos Galante. **Um mecanismo para identificação, representação e consulta de versões de objetos XML oriundos de bibliotecas digitais.** In: VI Workshop de Teses e Dissertações em Bancos de Dados - WTDBD, October 2007, João Pessoa, PB, Brazil. (in conjunction with XXII Simpósio Brasileiro de Banco de Dados - SBBD).

[2] P. Buneman, S. Khanna, and W. C. Tan, **Data provenance: Some Basic Issues**, in FSTTCS, 2000.

[3] D. P. Lanter, ***Design Of A Lineage-Based Meta-Data Base For GIS***, in Cartography and Geographic Information Systems, vol. 18, 1991, pp. 255-261.

[4] M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn, **Provenance of e-Science Experiments - experience from Bioinformatics**, in Proceedings of the UK OST e-Science second All Hands Meeting, 2003.

[5] Yogesh L. Simmhan, Beth Plale, Dennis Gannon, ***A Survey of Data Provenance Techniques***, Computer Science Department, Indiana University, Bloomington IN 47405.

[6] J. Zhao, C. A. Goble, R. Stevens, and S. Bechhofer, **Semantically Linking and Browsing Provenance Logs for Escience**, in *ICSNW*, 2004.

[7] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao, **Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation,** in *SSDBM*, 2002.

[8] C. Pancerella, J. Hewson, W. Koegler, D. Leahy, M. Lee, L. Rahn, C. Yang, J. D. Myers, B. Didier, R. McCoy, K. Schuchardt, E. Stephan, T. Windus, K. Amin, S. Bittner, C. Lansing, M. Minkoff, S. Nijsure, G. v. Laszewski, R. Pinzon, B. Ruscic, Al Wagner, B. Wang, W. Pitz, Y. L. Ho, D. Montoya, L. Xu, T. C. Allison, W. H. Green, Jr, and M. Frenklach, **Metadata in the collaboratory for multi-scale chemical science**, in *Dublin Core Conference*, 2003.

[9] J. Myers, C. Pancerella, C. Lansing, K. Schuchardt, and B. Didier, **Multi-Scale Science, Supporting Emerging Practice with Semantically Derived Provenance**, in *ISWC workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, 2003.

[10] J. Frew and R. Bose, **Earth System Science Workbench: A Data Management Infrastructure for Earth Science Products**, in *SSDBM*, 2001.

[11] Y. Cui and J. Widom, **Practical Lineage Tracing in Data Warehouses**, in *ICDE*, 2000.

[12] Y. Cui and J. Widom, **Lineage tracing for general data warehouse transformations**, in *VLDB Journal*, vol. 12, 2003.

[13] Philippe Le Hégaret, W3C; Lauren Wood, SoftQuad Software Inc., WG Chair; Jonathan Robie, Texcel , **What is the Document Object Model?;** Nov. 2000. Disponível em : <http://www.w3.org/TR/DOM-Level-2-Core/introduction.html>. Acesso em setembro de 2007.

[14] Gubusoft, **Tree View**. Disponível em: <http://www.treeview.net/>. Acesso em: set. 2007.

APÊNDICE: ARQUITETURA DO XVERSION

Este apêndice tem por objetivo apresentar a arquitetura do MetadataProv e suas principais funções.

Para descrever as ações que a ferramenta utiliza, serão mostradas as seguintes estruturas da UML (LARMAN, 2004): diagramas de caso de uso, descrição de casos de uso, diagramas de seqüência e diagramas de classe.

A.1 Diagrama de casos de uso

A figura A.1 apresenta os casos de uso do MetadataProv. São duas as funcionalidades principais da ferramenta. A primeira é a realização de uma operação de consulta a um artigo, e a segunda é a visualização dos resultados.

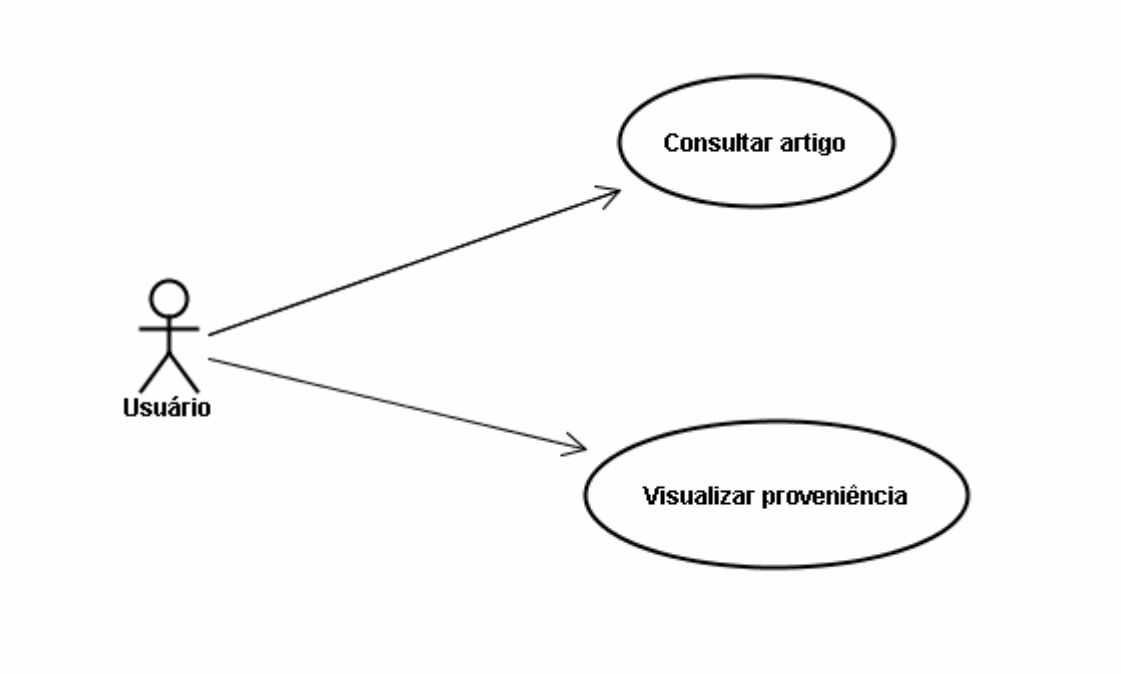

Figura A.1 – Diagrama de casos de uso do MetadataProv.

A.2 Diagrama de classes

A figura A.2 apresenta o diagrama de classes do MetadataProv. Logo após cada classe da ferramenta é brevemente descrita.

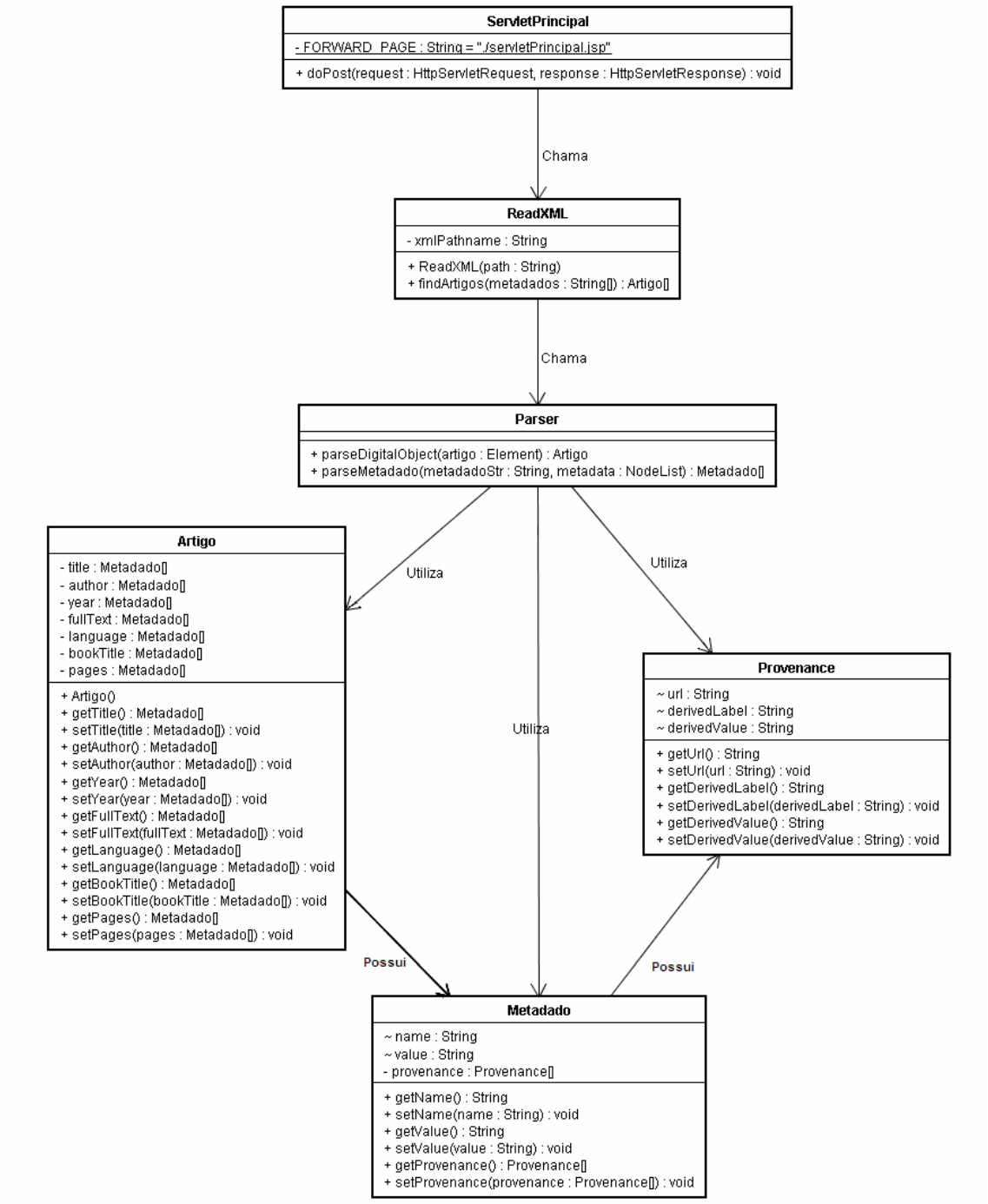

Figura A.2 – Diagrama de classes do MetadataProv

A.2.1 Classe: Provenance

Contém as informações sobre a proveniência de um metadado de um artigo.

A.2.2 Classe: Metadado

Contém as informações sobre um metadado e um array de proveniência.

A.2.3 Classe: Artigo

Contém um array para cada metadado de um artigo.

A.2.4 Classe: ReadXML

Lê o XML e calcula em todos seus artigos a similaridade entre as informações passadas na tela e o valor de cada metadado correspondente. Se a silimaridade for alta para todas as informações passadas na tela então chama o Parser passando o correspondente artigo como parâmetro, recebe um objeto Artigo no retorno e o adiciona a um array de objetos Artigo. Após ler todo o XML encaminha o array de objetos Artigo para o Servlet.

A.2.5 Classe: Parser

Recebe o artigo passado como parâmetro e cria objetos Provenance, Metadado e Artigo a partir dele, e retorna o objeto Artigo.

A.2.6 Classe: ServletPrincipal

Recebe os metadados passados pelo usuário na tela e encaminha para o ReadXML. Recebe de volta um array de objetos Artigo e encaminha para o JSP que apresentará ele na tabela de resultados.

A.3 Casos de uso e diagramas de seqüência

A.3.1 Consultar Artigo

Ator: Usuário do sistema.

Finalidade: Recebe os parâmetros passados pelo usuário e consulta no arquivo XML qual artigo possui metadados similares aos de entrada.

Visão Geral: Usuário entra com os metadados do artigo que deseja pesquisar e após as operações de consulta o programa devolve para a tela a lista com os artigos que possuem metadados similares aos de entrada.

Tipo: Primário.

Tabela A.1 – Seqüência típica de eventos do caso de uso Gerar Delta Script entre dois Documentos

Ação dos Atores	**Resposta do Sistema**
1. Usuário preenche os campos de entrada e pressiona o botão “Pesquisar”	2. Exibe os resultados encontrados

A.3.1.1 Diagrama de seqüência do caso de uso Pesquisa a um Artigo

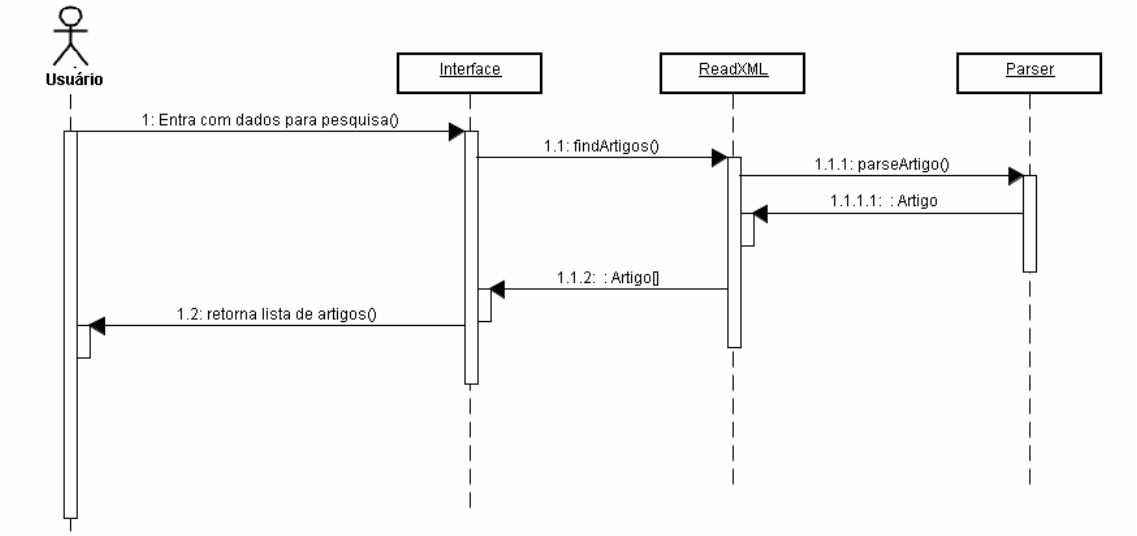

Figura A.3 – Diagrama de seqüência de uma operação de consulta

A.3.2 Visualizar proveniência

Ator: Usuário do sistema.

Finalidade: Visualizar informações de proveniência para um metadado.

Visão Geral: Após uma operação de consulta usuário expande a árvore de proveniência de um metadado que deseja analisar e uma lista de proveniências é apresentada.

Tipo: Primário.

Tabela A.2 – Seqüência típica de eventos do caso de uso Expandir árvore de proveniência

Ação dos Atores	**Resposta do Sistema**
1. O usuário clica no botão “+” de algum metadado.	2. Exibe na tela a lista de proveniências do metadado

A.3.2.1 Diagrama de seqüência do caso de uso Expansão da Proveniência

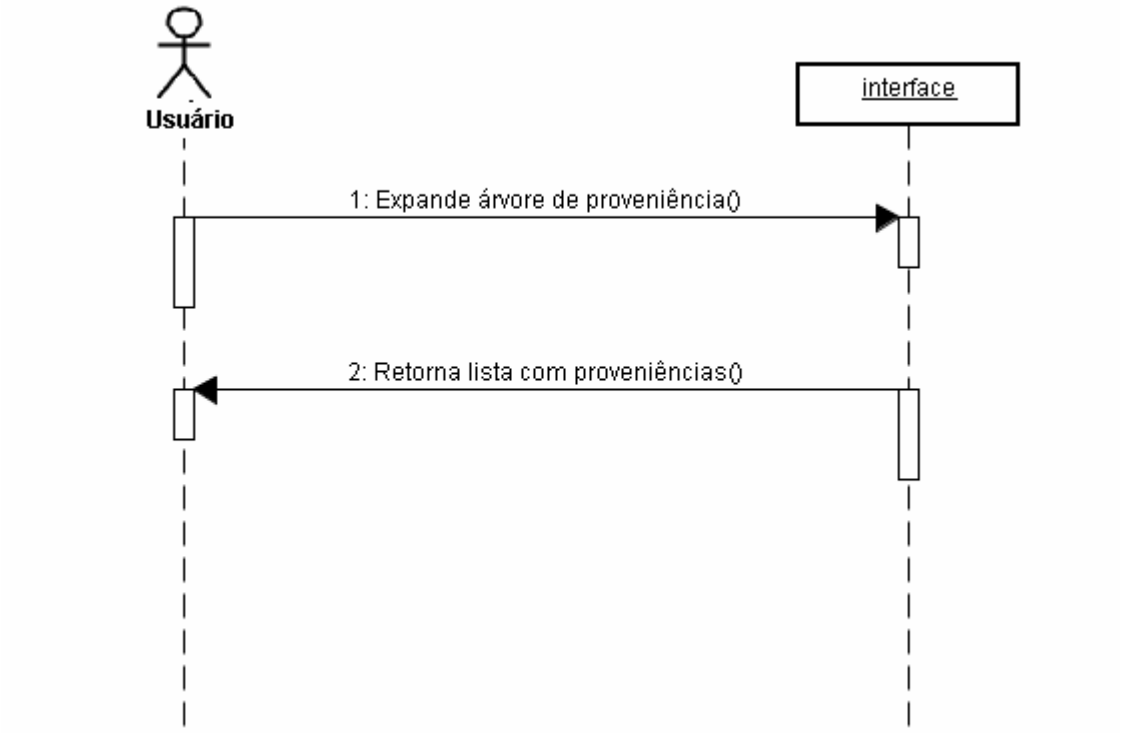

Figura A.4 – Diagrama de seqüência para a visualização das informações de proveniência de um metadado