MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-14212-TDI/1113

PROCEDÊNCIA DE DADOS: TEORIA E APLICAÇÕES AO PROCESSAMENTO DE IMAGENS

Juliana Cristina Braga

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Gerald Jean Francis Banon, aprovada em 20 de agosto de 2004.

INPE
São José dos Campos
2007

528.711.7
Braga, J. C.
Procedência de dados: teoria e aplicações ao processamento de imagens / Juliana Cristina Braga. - São José dos Campos: INPE, 2004.
110p. ; – (INPE-14212-TDI/1113)

1. Processamento de imagens. 2. Procedência de dados. 3. Gerenciamento computacional. 4. Biblioteca digital. 5. Protótipo. I. Título.

Aprovado pela Banca Examinadora em cumprimento a requisito exigido para a obtenção do Título de **Doutor** em **Computação Aplicada**.

Dr. Antônio Miguel Vieira Monteiro
Presidente/INPE, SJCampos-SP

Dr. Gerald Jean Francis Banon
Orientador/INPE, SJCampos-SP

Dr. João Ricardo de Freitas Oliveira
Membro da Banca/INPE, SJCampos-SP

Dra. Leila Maria Garcia Fonseca
Membro da Banca/INPE, SJCampos- SP

Dr. Maurício Gonçalves Vieira Ferreira
Membro da Banca/INPE, SJCampos- SP

Dr. Sérgio Shiguemi Furuie
Membro da Banca
Convidado INCOR, São Paulo - SP

Dr. Francisco de Assis Zampirolli
Membro da Banca
Convidado SENAC, São Paulo -SP

Candidato: Juliana Cristina Braga

São José dos Campos, 20 de agosto de 2004.

A meus pais Julião e Eunice.
Às minhas irmãs Than e Tati.
A meu amor Paulo.

AGRADECIMENTOS

À Deus por sempre iluminar e guiar meu caminho.

A meu querido pai pelo incentivo e pelas oportunidades oferecidas as quais proporcionaram a concretização deste trabalho.

A minha mãe maravilhosa que mesmo longe se fez presente através de seu amor. Seu dom em dizer as palavras certas nos momentos certos trouxe o conforto nas horas difíceis, e me ajudou a seguir em frente sem olhar para trás.

A meu admirável companheiro Paulo pelo amor, incentivo e paciência. Seu apoio me ajudou a encontrar o equilíbrio necessário para vencer essa etapa da minha vida.

A minha irmã Thanisse que trouxe alegria à minha vida em São José dos Campos. Sua presença tornou minha caminhada mais tranqüila e menos dolorosa.

A minha eterna "irmãzinha" Tati pelo orgulho e carinho que sempre demonstrou, me encorajando nos momentos que precisei.

Ao Professor Banon que com seu profissionalismo, educação e simplicidade esteve sempre disposto e paciente a me ajudar com seus inúmeros ensinamentos. Deixo registrado aqui a minha profunda admiração por ele.

Ao Miguel que me recebeu de braços abertos na Divisão de Processamento de Imagens (DPI). E por me ajudar não só com recursos para compra de livros e idas a congressos mas também me oferecendo atenção e amizade.

À Fundação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo auxílio financeiro.

Ao Instituto Nacional de Pesquisas Espaciais (INPE), pela oportunidade de estudos e utilização de suas instalações.

À Universidade Federal de Viçosa (UFV) que através de seu excelente nível de ensino me proporcionou conhecimentos suficientes para desenvolver essa pesquisa.

Ao meu orientador de mestrado Fernando Pruski, pela grande amizade e ensinamentos.

Aos docentes da Computação Aplicada (CAP), especialmente ao Haroldo, Demisio, Vijay e Carlos Ho pelo apoio e confiança depositada quando necessitei.

A todos os membros da DPI (Cartaxo, Léo, Gilberto, Mura e outros) pelo exemplo de excelente trabalho em equipe. E por me proporcionar um ambiente de trabalho do qual sentirei muita saudades.

Ao meu grande amigo Dmitri, sempre solícito a me ajudar com seus conhecimentos, e a trazer alegria contagiante para nossa salinha em frente ao cafezinho.

À minha "Big" amiga Silvely pelo seu enorme coração e amizade sincera.

À minha amiga Adriana pelos conselhos, amizade e incentivo nas corridas no horário do almoço.

Ao Arley pelo carinho e amizade na convivência diária de dois anos e meio. Pelo milagroso primeiro lugar no biatlon do INPE. E também pelos quitutes de sua avó.

Ao amigo Laercio que sempre apoiou e considerou meu trabalho.

Ao Pim (Fabrico) pela amizade e companhia nas baladas no início do curso. E pela ajuda nos momentos que precisei.

Às amigas inesquecíveis da DPI: Karine, Lubia e Marisa.

Ao grande Serginho, pela amizade e companheirismo.

À querida e doce Lise que participou e torceu por mim em vários momentos importantes da minha vida. E por estar sempre presente nas minhas apresentações.

Aos meus recentes companheiros de sala Cleber e Gilberto pela compreensão, carinho e ótima convivência na etapa final deste trabalho.

Às divertidas “DPIGirls”: Hilceia, Izabel, Janete, Jussara, Karine, Leila, Lubia, Marisa, Rosinha, Regina, Sil, Silvana, Sueli, Than.

Ao Reinaldo Rosa que me recebeu muito bem e me ajudou quando cheguei a São José dos Campos.

Ao Professor Elbert pela atenção sempre prestada.

Ao Professor José Luiz Braga, da UFV, pelo exemplo de profissional e por guiar meus caminhos na pesquisa.

À Professora Leila pelo apoio incondicional que recebi em horas difíceis.

À Professora Corina pela atenção que prestou no final do doutorado.

À Dona Amélia pela cuidado com nosso ambiente de trabalho.

Ao professor Marco Antônio Casanova (PUC - Rio) que me indicou um “paper” essencial para o início desse manuscrito.

Ao Alexandre pela atenção em solucionar dúvidas sobre o Python e o Morph.

À Thanisse por me ajudar a fazer a Interface do sistema desenvolvido neste trabalho.

Ao Julião Braga, Marcelo Tílio e Paulo Toscano pela revisão final desse manuscrito.

Aos membros da banca que aceitaram o convite.

RESUMO

A procedência é uma documentação complementar a dados. Ela contêm a descrição de "como", "quando", "onde", "porque" os dados foram obtidos e "quem" os obteve. A procedência inclui não só a origem dos dados, mas também os processos geradores dos mesmos (algoritmos e seus respectivos parâmetros). A procedência de dados pode ser muito útil em processamento de imagens para reproduzir imagens, economizar espaço de armazenamento, aumentar a velocidade de transmissão, proteger os direitos de propriedade e auxiliar na busca e recuperação de imagens. Para usufruir dos seus benefícios, a procedência de imagens deve ser gerenciada por mecanismos computacionais. O seu gerenciamento computacional inclui a criação, armazenamento, publicação, busca, recuperação e manipulação de forma automática e integrada. Motivada pelos benefícios da procedência de imagens e desafios em seu gerenciamento, o objetivo do trabalho foi: propor um modelo, uma estrutura e um tipo de armazenamento de procedência de dados para orientar a implementação de seu gerenciamento; implementar um protótipo para o gerenciamento computacional da procedência de dados; e testar o protótipo em processamento de imagens. O protótipo foi desenvolvido de forma integrada com a biblioteca digital UR*Lib*. Os testes preliminares realizados mostraram a viabilidade de um sistema de gerenciamento de procedência completo e eficiente.

DATA PROVENANCE: THEORY AND APLICATION TO IMAGE PROCESSING

ABSTRACT

The provenance is a complementary data documentation. It contains a description of "how", "when", "where", "why" the data were obtained and "who" obtained it. The provenance includes not only the origin of the data but also the description of the processes that produce them (algorithms and their respective parameters). Data provenance might be very helpful in image processing to reproduce images, save disk space, increase speed transmission, protect the copyright and retrieve images. To take advantages of image provenance, must be managed by computer systems. Its management includes the creation, storage, publication, retrieval and handling of provenance in an automatic and integrated way. Motivated the advantages of provenance and the challenges of its management, the goal of our work was: to propose a model, a structure and storage of data provenance in order to guide its implementation; to implement a prototype for the data provenance management; and to test this prototype in image processing. This prototype was integrated with the UR*Lib* digital library. The preliminary tests have shown the feasibility and efficiency of the proposed data provenance management system.

SUMÁRIO

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

Um dos problemas enfrentados por usuários que desenvolvem experimentos em sistemas computacionais, é a falta de uma ferramenta que monitore, capture e armazene automaticamente os passos, funções e parâmetros utilizados em seus experimentos. Este problema é particularmente crítico para usuários de sistemas de processamento de imagens. Neste caso, as imagens são geradas sem uma documentação completa o que, muitas vezes, inviabiliza seu reaproveitamento.

Motivados pela importância da documentação complementar ao processamento de imagens e a par dos problemas relacionados a sua criação e gerenciamento, iniciamos nosso trabalho. A seguir, apresentamos com maiores detalhes as idéias que nortearam o desenvolvimento desta pesquisa de doutorado.

1.1 Histórico da Nossa Pesquisa

O ponto de partida dessa pesquisa foi a leitura de um artigo sobre a necessidade de documentar as funções e parâmetros utilizados no processamento de imagens, ou seja, documentar o histórico do processamento de imagens. Este artigo foi apresentado em 1984 (Banon, 1984) e publicado em 1991 (Banon, 1991). O mesmo mostrou a importância da preservação do histórico de processamento de imagens e sua relevância para a comunidade científica dessa área (Seção 1.2).

O passo seguinte foi procurar na literatura trabalhos com a mesma preocupação de preservar o histórico de processamentos de imagens, porém nada foi encontrado. Concluímos então que mesmo sendo uma preocupação mais antiga, a preservação desse histórico possuía grande potencial de exploração. Motivados pela relevância do assunto e pelo potencial de exploração iniciamos nossa pesquisa nessa área.

Para familiarizarmos melhor com o histórico de processamentos, começamos a implementação de um sistema de processamento computacional para capturar e registrar históricos (funções e parâmetros) de processamentos de imagens. Essa implementação foi uma reprodução do sistema proposto no referido artigo.

Ao finalizar a implementação do sistema, observamos que somente a geração do histórico não era suficiente para o um bom reaproveitamento de processamentos. Era necessário também preservar e compartilhar o histórico com outras pessoas. Além disso, também era necessário registrar junto ao histórico a origem do processamento (data, autor, software utilizado, data) e preservar as imagens originais (Seção1.3). Na tentativa de sanar parte dessas necessidades, idealizamos armazenar o histórico e informações complementares na UR*Lib* (Banon et al, 2004).

A UR*Lib* é uma biblioteca digital de acervo distribuído que vem sendo desenvolvida desde 1995 pelo Professor Banon. Ela surgiu para solucionar problemas de integração de documentos eletrônicos armazenados por sistemas de arquivos diferentes, e atualmente armazena a memória técnico científica do INPE (Banon et al, 2004) .

Foi assim que relacionamos dois projetos independentes e com preocupações diferentes.

Para formalizar nossa pesquisa, buscamos na literatura linhas similares, porém em outras áreas de atuação. Dessa maneira descobrimos o enquadramento mais moderno para o nosso tema: Procedência de dados ou originalmente do inglês "Data Provenance".

1.2 Procedência de Dados

A procedência de dados é a documentação complementar a um dado que contém a descrição de "como", "quando", "onde", "porque" ele foi obtido e "quem" o obteve. Ela inclui não só a origem do dado (identificação, responsável pelo dado, data de criação), mas também os processos aplicados a ele (algoritmos e seus respectivos parâmetros). (Woodruff e Stonebraker, 1997).

"Linhagem" (Bose, 2002) ou "pedigree" (Brown e Stonebraker, 1995) são termos também utilizados para denotar a procedência de dados. Além disso, histórico de derivação (Hachem et al., 1993), genealogia de dados (Spery et al., 1999) e arqueologia de dados (Barkston, 1998) são outros termos relacionados encontrados na literatura.

Em uma abordagem diferente proposta por Buneman et al. (2000) o termo procedência é utilizado para fazer referência somente a origem do dado e o termo "pedigree" para fazer referência ao histórico de como o dado foi produzido. Consideramos essa abordagem válida, porém muito específica e por isso admitimos em nosso trabalho que origem e “pedigree” fazem parte da procedência.

Observamos na literatura que a procedência de dados é um campo novo e por isso ainda existe muita divergência sobre sua nomenclatura. Além disso, o conceito de procedência varia de acordo com as comunidades científicas e o tipo de aplicação. Algumas área já descobriram o potencial da procedência de dados. Abaixo algumas utilizações da procedência em cinco áreas de pesquisas distintas:

- Sistemas de Informação: juntamente com o crescimento das tecnologias Web o número de documentos eletrônicos disponíveis e a facilidade para disponibilizá-los também crescem. É difícil saber a origem desses dados e consequentemente a sua confiabilidade. Por exemplo, quando fazemos o download de um documento na Web, qual a garantia de que o autor do documento é realmente o que está indicado nele? Como ter certeza de que o documento utilizado não foi adulterado? A quem realmente pertence os direitos autorais do documento? Pesquisas em bibliotecas digitais tem tentado solucionar esse tipo de problema utilizando para isso procedência de documentos eletrônicos.

- Bioinformática: a comunidade dos biólogos é altamente fragmentada, pois diferentes campos dessa área agem independentemente, produzindo repositórios de dados e ferramentas analíticas que operam sobre os dados isoladamente. Existem vários grupos de pesquisa em biologia, cada um gerando diferentes tipos de dados e informações. Isso levou a uma desnecessária replicação de experimentos, configuração inadequada de equipamentos ou o esboço de

conclusões que não são completamente justificáveis pelas técnicas que foram seguidas. Além disso, as informações são instáveis sujeitas a constantes atualizações. Esses fatos levaram os pesquisadores a perceber que procedência de dados em biologia é um pré-requisito para o entendimento da informação nesta área. Apesar disso, poucos são os procedimentos usados para captura, manipulação e propagação da procedência nesta área. Para solucionar parte desses problemas, iniciou-se o projeto myGrid (Greenwood et al., 2003; Zhao et al., 2003). Para entender melhor o funcionamento do projeto, suponhamos um dado biológico como sendo uma seqüência de DNA. Neste caso, o myGrid gera a procedência desse dado, a qual forneceria: notas de onde o dado vem, informação sobre espécies a que pertencem, o responsável pela entrada de dados, quando, sintaxe, semântica etc. Depois de gerada a procedência, ela é armazenada e compartilhada na Internet por vários outros grupos de pesquisas. O projeto inclui não apenas a procedência mas também links para outros projetos direta ou indiretamente relacionados, indicativos de páginas de web, literaturas etc.

- Banco de dados: a procedência em banco de dados auxilia no processo de rastrear e recordar a origem dos dados e seus movimentos entre as bases de dados (Buneman et al., 2000). Com o desenvolvimento da Internet e o declínio dos custos de armazenamento de dados em hardware, tornou-se comum construir novos banco de dados usando banco de dados já existentes. Sendo assim, os dados passaram a possuir um alto nível de procedência. Isto é, passaram por vários bancos e podem ter sido modificados, traduzidos e complementados. Parte da solução para recordar a origem dos dados, é anexar à procedência em componentes dos bancos de dados. O gerenciamento destes dados traz novos problemas à comunidade científica de banco de dados, uma vez que a tecnologias de banco de dados convencionais não podem ser diretamente aplicadas (Mello, 2000). Além disso, a inserção de novos dados, neste caso a procedência, envolvem desenvolvimento de novos modelos, novas linguagens

de consulta e novas técnicas de armazenamento. Contudo, alguns esforços para incluir procedência em banco de dados estão surgindo (Buneman et al. 2001).

- Ciências químicas: O projeto chamado "The Colaboratory for the Multi-scale Chemical Sciences (CMCS) reúne colaboradores de diferentes campos de pesquisa da área química (Pancerella et al., 2003). A procedência de dados é o coração do CMCS pois é através dela que os pesquisadores definem os dados e traçam o caminho da sua origem. A procedência no CMCS inclui o nome do criador, as contribuições, palavras chaves, datas (modificação e criação), programa ou técnica usada para criar dados, comentários específicos, referências de literaturas, a proposta de continuidade do experimento e os resultados e conclusões do experimento. O projeto fornece meios de automatizar a captura dessas informações e disponibilizá-las na Web.

- Astronomia: O "Flexible Image Transport System" (FITS) (Mann, 2002) é um padrão de formato digital utilizado pelos astrônomos para transportar, analisar e armazenar dados científicos. É desenvolvido pela NASA e pela união internacional de astronomia. É considerado um formato que vai além de uma simples representação de imagens (JPG ou GIF) pois possui a procedência da imagem embutida em seu cabeçalho. Ele contém a origem da imagem, descrição do telescópio, instrumentação, observador, histórico que contém os passos e processos associados ao processamento, dentre outras informações. É utilizado para transportar, analisar, e armazenar arquivos para conjunto de dados científicos.

Nesta tese, tratamos da procedência de dados oriundos de processos computacionais, mais especificamente processamento de imagens. Sendo assim, a seguir identificamos os benefícios da procedência de dados em processamento de imagens (Banon, 1991).

- Reproduzir imagens: muitas vezes, temos a necessidade de repetir um processamento de imagens existente. Por exemplo, quando queremos comparar processamentos propostos em artigos científicos com um novo processamento,

precisamos reproduzi-los. Através da procedência das imagens, podemos obter detalhes do processamento que elas sofreram e repetir o processo com eficiência.

- Adaptar processos existentes: podemos repetir um processamento bem sucedido com novos dados de entrada, ou então testar a influência de um parâmetro sobre o resultado final. Isto é possível consultando a procedência das imagens, ajustando alguns parâmetros e executando novamente o processo. Esta é uma maneira de evitar a repetição de pesquisa e facilitar sua continuidade.

- Economizar espaço em disco: certas imagens podem ser vistas como resultados intermediários dentro de uma cadeia de processamento. Neste caso, se existir a procedência dessas imagens intermediárias, elas podem ser apagadas depois de finalizado o último tratamento que as envolve, deixando assim mais espaço para o armazenamento de futuros tratamentos. Caso necessitarmos usar uma dessas imagens novamente, basta consultarmos sua procedência para saber como a mesma foi obtida e gerá-la outra vez.

- Aumentar a velocidade de transmissão: com as tecnologias atuais, a troca de dados através de Internet cresce a cada dia. Em alguns casos, a transmissão da procedência ao invés da imagem pode ser consideravelmente mais rápida. Ou seja, a reprodução das imagens através de sua procedência pode ser mais eficaz do que sua transferência.

- Proteger direitos de propriedade: a procedência de imagens possui a informação de quem a gerou, assim ela permite identificar claramente as condições de uso, modificações e redistribuição de um dado.

- Auxiliar na busca e recuperação de imagens na Web: a procedência pode guardar de forma organizada informações a respeito das características da imagem. Mecanismos de busca da Web poderão usufruir disso para encontrar imagens através da procedência.

1.3 Gerenciamento da Procedência e Seus Desafios em Processamento de Imagens

Mostramos na Seção anterior que são grandes os benefícios da procedência para processamento de imagens porém para usufruí-los é fundamental que a procedência seja bem gerenciada. Ser bem gerenciada significa ser criada de forma adequada, armazenada em meios seguros, publicada e facilmente encontrada e aproveitada.

A principal preocupação de um usuário que realiza processamento de dados não é a documentação desse processamento, por isso acreditamos que somente um sistema computacional, com a mínima intervenção do usuário é capaz de gerenciar adequadamente a procedência (desde criação até publicação). Porém, ainda existem desafios no gerenciamento computacional da procedência. Estes estão relacionados com a criação, armazenamento, publicação, busca, recuperação e preservação da procedência e das imagens.

Na maioria dos processamentos de imagens, a procedência não existe, e quando existe é criada manualmente ou automaticamente por software de processamento (arquivos de registro ou "log"). Geralmente, quando o processo de criação da procedência é manual, as informações são incompletas, apresentam erros e sem padronização. Quando gravada automaticamente é de difícil compreensão e em formato proprietário. Esses fatores dificultam o acesso e a manipulação da procedência por sistemas computacionais.

Se a procedência for armazenada adequadamente, em meios seguros e confiáveis, a chance de sua preservação aumenta, caso contrário sua existência é comprometida. Atualmente o armazenamento de procedências de imagens é gerenciado somente pelo responsável pelo processamento. Este, pode armazená-la adequadamente mas também apagá-la ou até mesmo esquecer sua existência.

Sabemos que conduzir uma pesquisa requer publicação, troca de informações e conseqüente disseminação dos resultados. Ou seja, a publicação da procedência é tão importante quanto sua existência. Em muitos casos os pesquisadores registram a procedência de suas imagens mas raramente as publicam. Assim, a procedência torna-se

útil apenas para o próprio pesquisador sem que outros grupos de pesquisa aproveitem seus benefícios.

Como vimos, uma das grandes vantagens da procedência é podermos gerar uma imagem novamente, ou seja reproduzir uma imagem, utilizando sua procedência. Porém, em processamento de imagens, a obtenção da imagem resultante (imagem final) depende do processamento de outras imagens (imagens originais). Nesses casos, para reproduzir a imagem final somente sua procedência não é suficiente, seria necessário também as imagens originais. Portanto, a preservação das imagens originais é tão importante quanto a preservação da procedência

1.4 Objetivos

Em vista dos grandes benefícios da procedência de imagens e dos desafios em seu gerenciamento, os objetivos gerais dessa tese são:

- propor um modelo, uma estrutura e um tipo de armazenamento da procedência de dados para orientar a implementação de seu gerenciamento;
- implementar um protótipo para o gerenciamento da procedência baseado nas proposições anteriores; e
- testar o protótipo em processamento de imagens

Nesta tese, os capítulos são organizados como segue: o Capítulo 2 contém os fundamentos teóricos da tese. O Capítulo 3 apresenta um estudo da estruturação e do tipo de armazenamento da procedência. O Capítulo 4, descreve a implementação do protótipo desenvolvido. O Capítulo 5 apresenta o funcionamento do protótipo e discute os resultados dos testes realizados no mesmo. O Capítulo 6 apresenta a conclusão do trabalho, a indicação de suas contribuições e sugestões de possíveis direções para pesquisas futuras.

CAPÍTULO 2

MODELO DA PROCEDÊNCIA

Neste capítulo propomos um Modelo para a procedência de dados, no qual introduzimos os fundamentos da linhagem de dados. Para auxiliar o entendimento desses fundamentos, iniciamos esse capítulo relembrando alguns conceitos sobre teoria de Linguagens Formais.

Ressaltamos que o enfoque deste trabalho é a procedência de imagens processadas, porém os fundamentos descritos aplicam-se a dados em geral.

2.1 Linguagens Formais

Essa Seção é uma adaptação das Sessões 1.2, 1.4, 1.5 e 1.6 do livro "Introduction to Formal Language Theory" de Michael A. Harrison (Harrison, 1978).

2.1.1 Definições Básicas e Notações

Seja um conjunto não vazio Σ chamado de *alfabeto*. Suponhamos que os elementos de Σ são *símbolos indivisíveis*.

Exemplos $\Sigma = \{0,1\}$ ou $\Sigma = \{a, b\}$.

Definição Uma *palavra* sobre um alfabeto Σ é uma Σ-seqüência de tamanho finito.

Uma palavra típica pode ser escrita como $x = a_1 \ldots a_k$, $k \geq 1$, $a_i \in \Sigma$ para $1 \leq i \leq k$. Permitimos $k = 0$, neste caso a palavra é *nula* (ou *vazia*) e denotada por Λ. k é chamado de *tamanho* de x, e é o número de ocorrências das letras de Σ em x.

Seja Σ^* o conjunto de todas as palavras de Σ.

Sejam x e y duas palavras em Σ^*. A concatenação de x e y é a palavra xy.

Exemplo Sejam $\Sigma = \{0, 1\}$, $x = 0101$ e $y = 1$

então

$$xy = 01011 \qquad \text{e} \qquad yx = 1010.$$

É necessário estender a concatenação de palavras para conjuntos. Sejam X e Y dois subconjuntos de palavras. Isto é, $X, Y \subseteq \Sigma^*$. Então, o *produto* de X por Y é definido por:

$$XY = \{xy \mid x \in X, y \in Y\}.$$

2.1.2 Gramática de Frase-Estruturada

Existe um padrão para descrever "gramáticas". O conceito intuitivo de gramática é um mecanismo finito de procedimentos para produzir conjuntos de palavras. Os seguintes conceitos tem provado ser muito útil em lingüistica, linguagem de programação e biologia.

Definição Uma *gramática de frase-estruturada* é uma 4-tupla: $G = (V, \Sigma, P, S)$ onde:

V é um conjunto finito não vazio chamado de *vocabulário total*.

$\Sigma \subseteq V$ é um conjunto finito não vazio chamado de *alfabeto terminal*.

$S \in N$ é chamado de *símbolo inicial* e representa o que se deseja produzir.

em que

$N = V - \Sigma$ chamado de conjunto de *variáveis* (ou conjunto de símbolos *não terminais*).

P é um conjunto finito de regras (ou produções) da forma:

$$\alpha \to \beta \qquad \text{onde } \alpha \in V^*NV^* \qquad \text{e} \qquad \beta \in V^*$$

O operador $\to$ é chamado de operador de substituição.

A seguir, um exemplo em linguagem de programação ilustra estes conceitos.

Em linguagem de programação Σ é o conjunto de todos os símbolos da linguagem. Para algumas linguagens mais comuns, ele pode conter

A, B, C, ..., Z, **:=**, **begin**, **end**, **if**, **then**, **for, do** etc.

Neste caso, os símbolos **begin**, **end**, **if**, **then**, **for** etc, são indivisíveis.

O conjunto de variáveis, *N*, contém o símbolo inicial <program>, e outros como <compound-statement>, <block> etc. Uma regra típica pode ser:

<for statement> → **for** <control variable> **:=** <for list> **do** <statement>

2.1.3 Gramática de Livre Contexto

Definição Uma gramática de frase-estruturada $G = (V, \Sigma, P, S)$, é uma gramática de *livre-contexto* se cada regra de produção é da forma:

$$A \rightarrow \alpha$$

onde $A \in N$, $\alpha \in V^*$.

O termo "livre contexto" significa que *A* pode ser substituído por α onde quer que *A* apareça, sem se importar pelo contexto. Na gramática de livre-contexto, a substituição de A é independente dos símbolos aparecendo antes e/ou depois de A.

Existe uma forma alternativa de gramáticas de livre-contexto usada na especificação de linguagens de programação. Esse sistema é chamado de Forma Normal de Backus ou forma de Backus-Naur; ambos os casos são comumente abreviados para BNF. A forma utiliza quatro caracteres, os quais não estão presentes no vocabulários de trabalho. Esses são:

< > ::= |

As variáveis são apresentadas por caracteres entre < e >. O símbolo ::= serve como um operador de substituição assim como $\rightarrow$, e | é lido "ou".

Exemplificando, seja $G = (V, \Sigma, P, S)$ uma gramática de livre contexto para dígitos sem sinais em linguagens de programação, onde D representa a classe de dígitos e U representa a classe de inteiros sem sinal. Sendo assim,

$V = \{0, 1, ... 9, D, U\}$

$\Sigma = \{0, 1, ... 9\}$

$N = \{D, U\}$

$S = U$

O conjunto P das regras de produção é composto por:

$D \rightarrow 1$	$D \rightarrow 5$
$D \rightarrow 2$	$D \rightarrow 6$
$D \rightarrow 3$	$D \rightarrow 7$
$D \rightarrow 4$	$D \rightarrow 8$
$U \rightarrow 0$	$D \rightarrow 9$
$U \rightarrow D$	$U \rightarrow DU$

P, escrito em BNF é dado por:

<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<unsigned integer> ::= 0|<digit>|<digit><unsigned integer>

Definição Seja $G = (V, \Sigma, P, S)$ uma gramática de frase-estruturada e sejam $\alpha', \beta' \in V^*$. α' é dito *gerador direto* de β', escrevemos por $\alpha' \Rightarrow \beta'$, se existir $\alpha_1, \alpha_2, \alpha, \beta \in V^*$ tal

que $\alpha' = \alpha_1\ \alpha\ \alpha_2$, $\beta' = \alpha_1\ \beta\ \alpha_2$ e $\alpha \rightarrow \beta$ está em P. Escrevemos $x \overset{*}{\Rightarrow} y$, x é dito *gerador* de y, se e somente se existir $r \geq 0$, $z_0, z_1, \ldots z_r$ tais que $x = z_0$, $y = z_r$ e $z_i \lceil z_{i+1}$ para todo $i = 0,1,\ldots r\text{-}1$. A seqüência $z_0 \Rightarrow z_1 \Rightarrow z_2 \ldots \Rightarrow z_r$ chamamos de *derivação* .

Exemplo Sejam $\Sigma = \{a, b\}$, $N = V - \Sigma = \{A, B, S\}$ com as regras de produção:

$$S \rightarrow aAB|bBa$$

$$A \rightarrow Sa$$

$$B \rightarrow \Lambda$$

e,

$$S \Rightarrow aAB \Rightarrow aSaB \Rightarrow abBaaB \Rightarrow abaaB \Rightarrow abaa$$

$$S \overset{*}{\Rightarrow} abaa$$

Definição Seja $G = (V, \Sigma, P, S)$ uma gramática de frase-estruturada. A *linguagem gerada* por G, denotada por $L(G)$, é o conjunto:

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

Os elementos de $L(G)$ são chamados de *frases* de G.

2.1.4 Gramática de Livre-Contexto e Árvores

O uso de árvores para ilustrar derivações de gramática de livre-contexto foi uma forma importante de estimular nossa intuição para a formalização da linhagem de dados.

A Figura 2.1 mostra uma *árvore gramatical* para qualquer gramática construída a partir das regras de produção do exemplo acima.

Um exemplo de uma árvore T é dada na Figura 2.1- (a). Ela possui um ou mais *nodos* $x_0, x_1, \ldots, x_{10}$, um dos quais é a *raiz* x_0. Denotamos a relação de *descendência imediata* por $\lceil$. Na Figura 2.1(a), $x_2 \lceil x_4$, mas não $x_2 \lceil x_{10}$). $\lceil^*$ denota a relação de *descendência relativa* a

⌈. Ela é definida por: $x \lceil^* y$ se e somente se existir $r \geq 0$, $z_0, z_1, \ldots z_r$ tais que $x = z_0$, $y = z_r$ e $z_i \lceil z_{i+1}$ para todo i = 0,1,...r - 1. A seqüência ($z_0, z_1, \ldots z_r$) chama-se *caminho* de x para y (por exemplo, (x_2, x_4, x_8, x_{10}) é o caminho de x_2 para x_{10} na Figura 2.1(a), já que existe um caminho de x_2 para x_{10} então $x_2 \lceil^* x_{10}$). O nodo x é uma *folha* em T se para nenhum y em T, $x \lceil y$ (x_1, x_5, x_6, x_7, x_9 e x_{10} são folhas na Figura 2.1 (a)). Nodos que não são folhas são chamados *internos* (x_0, x_2, x_4 e x_8 são internos na Figura 2.1 (a)).

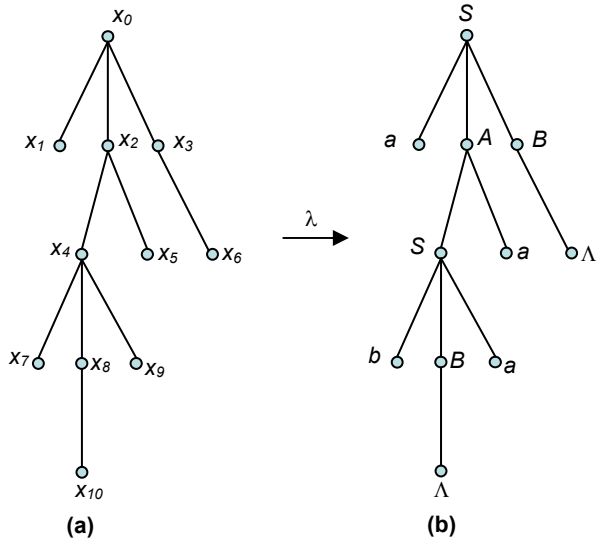


FIGURA 2.1- (a) e (b) Árvores representando uma gramática de livre-contexto.
FONTE: adaptado de (Harrison, 1978).

Devemos prestar atenção na ordem esquerda para direita (esquerda-direita) dos nodos. Seja

$$(y_1, y_2, \ldots, y_m) \qquad m \geq 1 \qquad (1)$$

uma seqüência de todas as folhas de T sem repetição. Definimos então a relação ⌊ entre certos pares de nodos de T.

Para qualquer x *e* y em T, $x \lfloor y$ se e somente se x e y não estão no mesmo caminho, e para algumas folhas y_i, y_{i+1} ($1 \leq i < m$) em (1) temos $x \lceil^* y_i$ e $y \lceil^* y_{i+1}$. Isto é, não existe folha "entre" x e y. Por exemplo, $x_2 \lfloor x_6$ mas não $x_9 \lfloor x_6$ pois a ordem dos nodos é $x_1, x_7, x_{10}, x_9, x_5, x_6$ na Figura 2.1(a)

Agora vamos relacionar o conceito de gramática de livre-contexto com árvores ou conjunto de árvores. Seja $G = (V, \Sigma, P, S)$ uma gramática de livre-contexto. Interpretamos as produções da gramática G como árvores (de altura 1) da seguinte maneira: a produção $A_0 \rightarrow A_1A_2...A_n$ corresponde, através de um mapeamento λ chamado de rotulação, à árvore com raiz x_0 e folhas $x_1, ... x_n$ (ou, formalmente, $x_0 \lceil x_1$, $x_0 \lceil x_2$, ..., $x_0 \lceil x_n$, $x_1 \lfloor x_2 \lfloor$, ..., $\lfloor x_n$, e $\lambda(x_i) = A_i$, $0 \leq i \leq n$). Faremos a convenção que, nessa correspondência, $\lambda(x_i) = \Lambda$ se e somente se $i = n = 1$ e $A_0 \rightarrow \Lambda$ está em P.

Definição Seja $G = (V, \Sigma, P, S)$ uma gramática de livre-contexto e T uma árvore com rotulação $\lambda : T \rightarrow V \cup \{ \Lambda \}$. T é chamado de *árvore gramatical* de G se e somente se para toda subárvore de T existe uma regra de produção correspondente em P. Esta árvore é chamada de *árvore de derivação*, se e somente se $\lambda(y_1)\ \lambda(y_2)\ ...\ \lambda(y_m) \in \Sigma^*$ (onde $y_1, ... y_m$ são folhas de T).

A Figura 2.1 (b) mostra uma árvore gramatical que é também uma árvore de derivação de G. Esta árvore corresponde a geração da frase *abaa* do exemplo anterior (página 29)

2.2 Modelo da Procedência de Dados

O Modelo da procedência refere-se ao tipo de informação que deve estar contida na mesma. Como visto na introdução, na literatura os autores divergem bastante com relação ao tipo de informação que pode ou não ser considerada procedência de um dado processado. Neste trabalho consideramos que a procedência de um dado processado deve conter informações suficientes para que pessoas não envolvidas na geração desse dado possam reproduzi-lo. Sendo assim, o modelo de procedência deve conter dois componentes principais: a origem e a linhagem do dado (Figura 2.2).

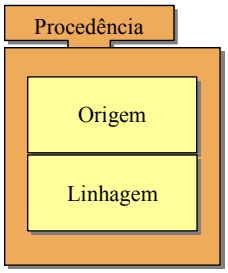


FIGURA 2.2 - Componentes da procedência de dados.

A origem do dado contém, no mínimo, identificador e data de criação do dado, nome, e contato do responsável pela criação do mesmo. Além disso, nome e versão do software em que foi realizado o processamento do dado.

Linhagem de dados é o componente da procedência de dados responsável pela descrição de como o dado foi processado. Ela contém as funções e parâmetros do processamento que o dado sofreu dispostos na ordem de aplicação (Figura 2.3).

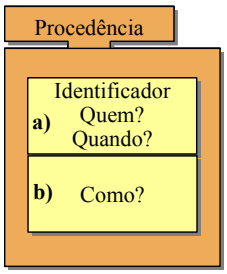


FIGURA 2.3 - a) Origem do dado; b) Linhagem do dado.

A Linhagem é especialmente importante para a procedência de imagens. Pouco adianta saber a origem de uma imagem sem saber a forma como ela foi processada. A linhagem é a fração da procedência utilizada para auxiliar na preservação de imagens, evitar repetição de esforços na tentativa de dar continuidade ou refazer processamentos existentes.

A Linhagem deve ser gerada automaticamente por programas de computadores a fim de permitir eficácia e precisão em sua manipulação. Para que sua implementação e gerenciamento computacional seja coerente é necessário a sua fundamentação.

2.3 Fundamentos da Linhagem de Dados

Baseados na teoria de linguagens formais (Seção 2.1) introduzimos cinco novos conceitos: *árvore de linhagem*, *linhagem para um dado, árvore de dados*, *nome reservado para uma linhagem* e *nome reservado para um dado*. Na formalização desses conceitos encontra-se a originalidade dessa tese.

A seguir utilizamos a noção de mapeamento matemático para introduzir as formalizações dos novos conceitos.

Seja B o conjunto de frases de uma certa linguagem de programação. Essas frases podem ser geradas por uma gramática de livre-contexto $G = (V, \Sigma, P, S)$, tem-se $B = L(G)$.

Exemplificando-se:

Σ = {1, 2, 3, 4, 5, 6, , ∧, ∨, (,), ~}[1] (alfabeto terminal)

N = { <element>, <max>, <min>, <inv>, <sentence> } (conjunto de variáveis)

$V = \Sigma + N$ (vocabulário)

S = <sentence> (símbolo inicial)

O conjunto das regras de produção, P, para geração das frases, escritas em BNF, é composta por:

<element> ::= 1 | 2 | 3 | 4 | 5 | 6 (regras 1 á 6)

<max> ::= ∨ (<sentence> <sentence>) (regra 7)

<min> ::= ∧ (<sentence> <sentence>) (regra 8)

<inv> ::= ~ (<sentence>) (regra 9)

<sentence> ::= <element> | <max> | <min> | <inv> (regra 10 á 13)

As seguintes expressões são exemplos de frases desta linguagem:

2

~(2)

[1] Ressaltamos que o sétimo elemento do conjunto Σ é um espaço em branco.

3

∧(~(2) 3)

A última frase foi gerada aplicando a seguinte derivação:

<sentence> ⇒ <min> (aplicação da regra 12)

⇒ ^(<sentence> <sentence>) (aplicação da regra 8)

⇒ ^ (<inv> <sentence>) (aplicação da regra 13)

⇒ ^ (~(<sentence>) <sentence>) (aplicação da regra 9)

⇒ ^ (~(<element>) <sentence>) (aplicação da regra 10)

⇒ ^ (~(2) <sentence>) (aplicação da regra 2)

⇒ ^ (~(2) <element>) (aplicação da regra 10)

⇒ ^ (~(2) 3) (aplicação da regra 3)

A *árvore de derivação* para a geração da frase ^(~(2) 3) é representada na Figura 2.4.

Árvore de Linhagem

Percorrendo uma árvore de derivação debaixo para cima podemos gerar uma *árvore de linhagens*. Na árvore de linhagens cada nó é rotulado por uma frase que representa uma *linhagem para um certo dado* (ver abaixo definição de uma linhagem para um dado).

A árvore de linhagem é obtida identificando na árvore de derivação os nós <sentence> (símbolo inicial). A cada um destes nós associa-se de forma biunívoca um só nó na árvore de linhagem. Cada nó é rotulado pela concatenação das folhas na subárvore de derivação correspondente (percorrendo essa árvore da esquerda para direita).

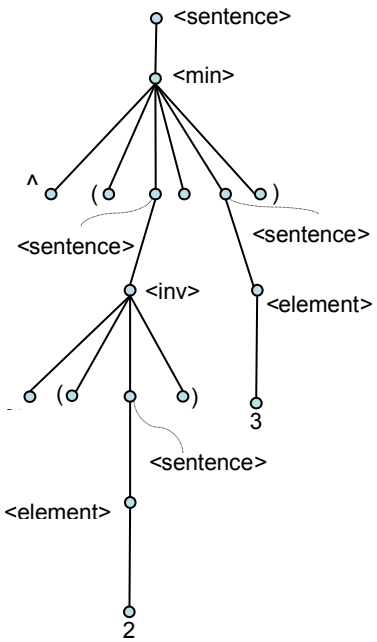


FIGURA 2.4 - Árvore de derivação para a geração da frase ^(~(2) 3).

A Figura 2.5 mostra a árvore de linhagem obtida a partir da árvore de derivação da Figura 2.4.

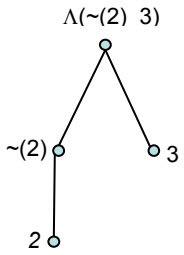


FIGURA 2.5 - Árvore de linhagem da frase ^(~(2) 3) gerada a partir da sua árvore de derivação.

Uma representação alternativa da árvore de linhagem é obtida na forma de diagrama de bloco. As entradas do diagrama de blocos são as folhas da árvore de linhagens e a saída é raiz. O bloco quadrado é usado para as regras que envolvem apenas um símbolo inicial e o bloco redondo para as que envolvem dois símbolos iniciais. A Figura 2.6 mostra o diagrama de bloco da correspondente a árvore de linhagem da Figura 2.5.

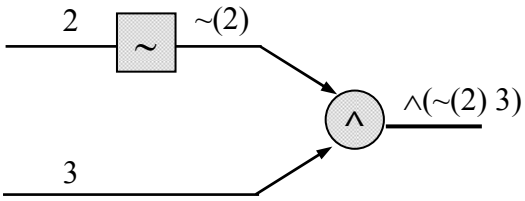


FIGURA 2.6 - Diagrama de bloco como uma representação alternativa da árvore de linhagens.

Árvore de Dados

Para formalizar o relacionamento entre, linhagem, dados e nomes reservados consideramos dois mapeamentos.

- Primeiro Mapeamento

O primeiro mapeamento, chamado de operador exec associa a cada frase da linguagem um único dado. Em outros termos, o operador exec é um mapeamento de *B* em *C*, onde *C* é o conjunto de dados.

Se $e \in C$, então por definição, uma frase *s* de *B é uma linhagem para o dado e*, ou por abuso de linguagem a *linhagem de e*, se e somente se, exec(*s*) = *e*.

A Figura 2.7 ilustra o primeiro mapeamento

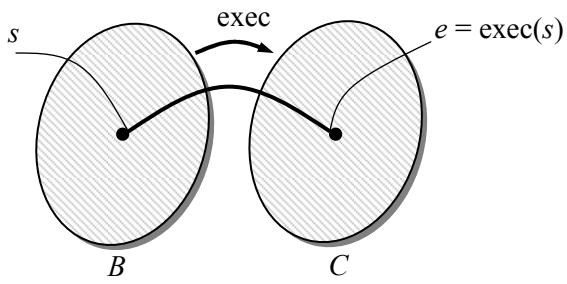


FIGURA 2.7 - Mapeamento exec.

Retornando ao exemplo anterior, admite-se que $C = \{1, 2, 3, 4, 5, 6\}$ e que as variáveis: <element>, <max>, <min> e <inv> são processadas da seguinte maneira: para qualquer $e, a, b \in$ C,

exec(e) = e

exec($\vee$ (a b)) = max{a, b}

exec($\wedge$ (a b)) = min{a, b}

exec(~(e)) = 6 - e

Desta forma o resultado do processamento do exemplo da Figura 2.5 fica:

exec(2) = 2

exec(3) = 3

exec(~(2)) = 4

exec($\wedge$(~(2) 3)) = 3

Substituindo os rótulos da árvore de linhagem da Figura 2.5 pelo resultado da aplicação do comando exec obtemos a *árvore de dados* da Figura 2.8.

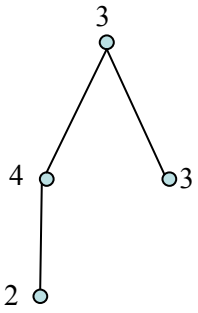


FIGURA 2.8 - Árvore de dados.

Chamamos de *dados originais* os dados representados pelas folhas da árvore de dados. Chamamos de *dado resultante* o dado representado pela raiz da árvore de dados. Chamamos de *dados intermediários* os dados representados pelos nodos internos (incluindo a raiz). No exemplo da Figura 2.8, 2 e 3 são dados originais, por sua vez 3 é também um dado intermediário junto com 4. Finalmente 3 é também o dado resultante.

Ao aplicar o comando exec na linhagem do dado resultante, conseguimos recuperar a partir dos dados originais os dados intermediários e o dado resultante. Quando dizemos recuperar um dado significa criá-lo novamente. É importante observar que para recuperação de dados através de sua linhagem é indispensável a preservação dos dados originais.

- Segundo Mapeamento

Considera-se agora o segundo mapeamento. O segundo mapeamento estabelece o critério para se dar nomes reservados às frases de uma dada linguagem. Os nomes serão úteis para referenciar frases e armazená-las.

Criado uma frase, dá-se um nome reservado, isto é, que nunca foi usado antes e que nunca mais será usado. Para isto, escolhe-se de forma apropriada um nome dentro de um conjunto infinito de nomes. Seja *linh* o mapeamento do conjunto *A* de nomes para o conjunto *B* de frases.

Seja $n \in A$, então por definição, *n* é *um nome reservado para a linhagem s* se e somente se $s = \text{linh}(n)$ e *n* é *o nome reservado de um dado e* se e somente se $\text{exec}(\text{linh}(n)) = e$.

Desta forma, a cada nome escolhido corresponderá uma única linhagem e um único dado. Procurar-se-á, na medida do possível, que nomes diferentes correspondam a linhagens diferentes mas isto não é imprescindível. Neste caso, o mapeamento linh seria injetora ("one-to-one").

Um nome para uma linhagem *s* servirá de endereço para o local onde estará sendo armazenado *s* e o resultado de sua execução.

A

FIGURA 2.9 mostra a composição dos dois mapeamentos introduzidos anteriormente.

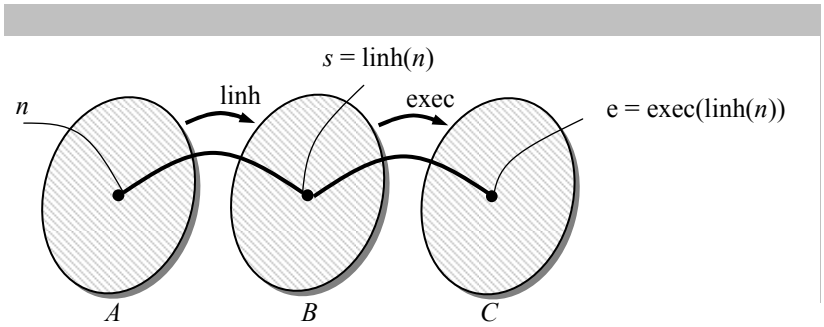


FIGURA 2.9 - Composição dos mapeamentos linh e exec.

Definido o mapeamento linh é possível dar nomes às frases consideradas no exemplo desta Seção. O diagrama de blocos da Figura 2.10 mostra a alocação de nomes para o exemplo mostrado na Figura 2.6.

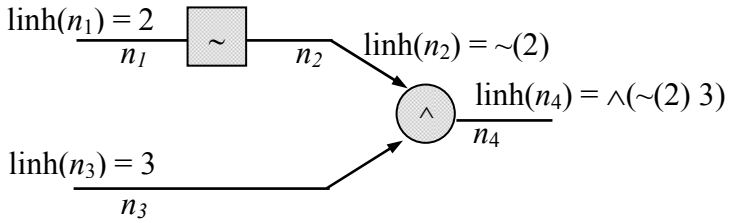


FIGURA 2.10 - Alocação de nomes reservados às linhagens.

Observamos que linh(n_4) pode ainda ser escrito como:

linh(n_4) = ∧(linh(n_2) linh(n_3))

onde

linh(n_3) = 3

linh(n_2) = ~(linh(n_1))

linh(n_1) = 2

Abusando da notação, escreveremos linh(n) como sendo n. Então a linhagem de n_4 ficaria:

n_4 = ∧(n_2 n_3)

onde

$n_3 = 3$

$n_2 = \sim(n_1)$

$n_1 = 2$

Isto é, a partir dos nomes n_1, n_2, n_3 seria possível "calcular" (efetuando substituição) a linhagem de n_4.

A Figura FIGURA 2.11 mostra a árvore de linhagem simplificada do exemplo acima.

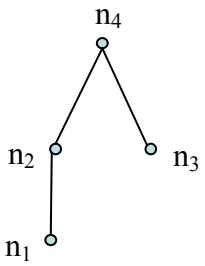


FIGURA 2.11 – Árvore de linhagem simplificada.

CAPÍTULO 3

ESTRUTURA E ARMAZENAMENTO DA PROCEDÊNCIA

Neste capítulo, relatamos dois aspectos fundamentais para que a procedência de dados seja bem gerenciada: Estrutura e Armazenamento. O primeiro aspecto define a organização da informação na procedência. O segundo aspecto propõe guardar a procedência em meios seguros que cuidam de sua integridade, preservação, persistência, assim como dos direitos da propriedade, busca e recuperação.

3.1 Estrutura

Para o gerenciamento computacional da procedência de dados, é importante que o modelo de procedência proposto (origem e linhagem) obedeça uma estrutura padrão e organizada.

A estrutura da procedência é a forma ou maneira como a informação contida na procedência deve estar organizada. Por ser um dado que descreve outro dado, podemos representar a procedência como metadados (Goble, 2002). Metadados são informações sobre informações que proporcionam a descrição abrangente de dados, contribuindo para a identificação, localização e gerenciamento destes dados por sistemas computacionais (Figura 3.1).

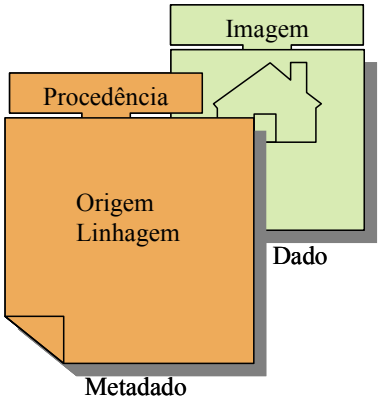


FIGURA 3.1 - Procedência de dados representada como metadados.

Para que possamos descrever as informações contidas em metadados de forma estruturada e padronizada devemos seguir um modelo ou esquema de metadados.
Esquema de metadados é um modelo para criar metadados. Para isso, o esquema fornece um conjuntos de elementos que devem obedecer uma ordem e sintaxe. Ele também define o nome e descreve o significado de cada elemento (Hodge 2001). Os valores dos elementos, chamados de conteúdo, são definidos apenas na criação dos metadados (Figura 3.2).

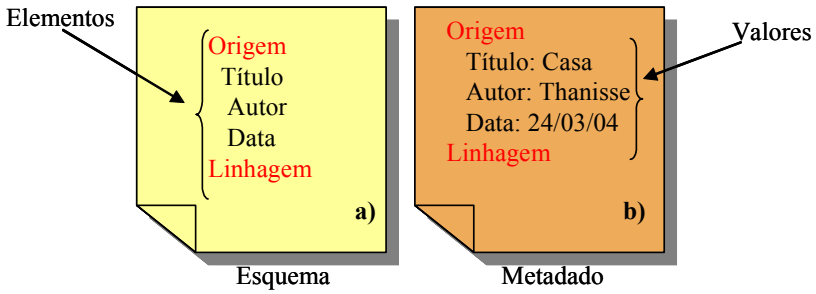


FIGURA 3.2 - a) Esquema de metadados e seus elementos. b) Metadado correspondente ao esquema e seus respectivos valores.

Facultativamente, os esquemas podem especificar regras para a formulação do conteúdo. Neste caso, existem regras de sintaxes para definir como os elementos devem ser codificados. Um esquema de metadado que não seja prescrito com regras de sintaxe é chamado de independente de sintaxe.

Atualmente os esquemas estão sendo escritos em linguagens "SGML - Standard Generalized Markup Language" (Especificação SGML) ou "XML - eXtensible Markup Language" (Holzer, 2001).

Existem diferentes esquemas de metadados para descrever dados, como por exemplo: FGDC (Federal Geographic Data Committe) para descrição de dados geo-espaciais, MARC (Machine Readable Catalogue) para catalogação bibliográfica; IAFA/WHOIS++ (Internet Anonymous Ftp Archive with whois++ protocol) para descrição do conteúdo e serviços disponíveis em arquivos ftp (file transfer protocol), TEI (Text Encoding Initiative) para representação de materiais textuais na forma eletrônica, NTD-BR para catalogação de teses e dissertações no Brasil (IBICT), DC

(**Erro! Fonte de referência não encontrada.**) para catalogação de recursos eletrônicos na Web. Uma descrição mais detalhada pode ser encontrada em Garcia et al., (1999).

Para descrição de imagens já foram desenvolvidos: metadados técnicos para imagens digitais estáticas desenvolvido pela "National Information Standards Organization" (NISO). Metadados para imagens digitais feito por "Digital Imaging Group" (DIG35, 2000). Metadados administrativos para imagens digitais estáticas desenvolvido pela Universidade de Harvard (President and Fellows of Harvard College, 2002).

Uma vez que o objetivo do nosso trabalho é propor soluções genéricas, optamos por utilizar o DC como Esquema de metadados para estruturar a procedência de imagens, pois o mesmo, apesar de não ser específico para descrição de imagens é genérico e permite extensões (inserções de novos elementos). Esquemas para estruturação de procedência de imagens de domínios específicos (imagens de sensoriamento, imagens médicas, imagens microscópicas etc) podem ser facilmente adaptados.

Dublin Core

Dublin Core é um Esquema de metadado para descrever recursos eletrônicos. A expectativa é que o DC torne os recursos descritos mais visíveis pelos mecanismos de busca e sistemas de recuperação. O DC não tem intenção de substituir modelos mais ricos e sim fornecer um conjunto básico de elementos que possam ser estendidos.

O DC possui 15 elementos sendo que todos são opcionais, podem ser repetidos e não exigem uma ordem de apresentação. Embora o DC recomende o uso de controladores de valores dos campos, estes não são obrigatórios.

Os 15 elementos do DC são adequados para descrever a origem das imagens. O elemento Relação poderia hospedar a linhagem, no entanto preferimos criar mais um elemento para conte-la, e a este elemento demos o nome de Linhagem. Na implementação, por motivos de redundância de informações ou por falta de necessidade, alguns elementos não foram implementados. Apesar disso, esses elementos podem ser facilmente inseridos na estrutura da procedência.

A seguir, apresentaremos uma versão adaptada do documento original do DC com as orientações auxiliares para a criação do conteúdo do metadado e alguns exemplos de uso. Os elementos não implementados foram indicados.

Título

Rótulo: Title

Descrição: Nome dado ao recurso que estamos descrevendo.

Exemplo de uso: Nome da imagem, Nome do processamento que a imagem sofreu.

Criador ou Autor

Rótulo: Creator

Descrição: A(s) pessoa(s) ou organização(ões) principal(is) responsável(is) pela criação do conteúdo intelectual do recurso.

Exemplo de uso: Nome da pessoa responsável pelo processamento do dado.

Palavras-chave

Rótulo: Subject ou Keywords

Descrição: Palavras que exprimam o significado do recurso.

Exemplo de uso: Nome do principal conteúdo da imagem, nome das principais funções aplicadas no processamento da imagem.

Descrição

Rótulo: Description

Descrição: Descrição textual do conteúdo do recurso.

Exemplo de uso: Descrição textual do conteúdo da imagem, do algoritmo de processamento. Resumo do que o processamento se refere.

Editor

Rótulo: Publisher

Descrição: Entidade ou pessoa responsável por tornar o recurso disponível na presente forma.

Exemplo de uso: Instituição de Pesquisa em que o processamento foi feito.

Colaborador

Rótulo: Contributor

Descrição: Pessoa ou organização não especificada no elemento Criador que tenha dado contribuição intelectual significativa para o recurso, mas cuja contribuição é considerada secundária para a pessoa ou instituição especificada no elemento Criador.

Exemplo de uso: Nome do programador. Este campo não foi implementado.

Data

Rótulo: Date

Descrição: Data em que o recurso tornou-se disponível. Uma data associada com a criação ou disponibilidade do recurso. Recomenda-se adotar a ISO 8601 (Formatos de datas e de tempo, W3C Technical Note), que inclui (entre outros) datas nos formatos YYYY e YYYY-MM-DD.

Exemplo de uso: Data em que o processamento da imagem foi realizado.

Tipo

Rótulo: Resource Type

Descrição: Categorias, funções, gênero, ou níveis agregados de conteúdos. Recomenda-se selecionar o valor de um vocabulário controlado (por exemplo, o DCMI vocabulários de tipos recomendado pela DC). Para descrição física ou digital do recurso usar o elemento Formato.

Exemplo de uso: Categoria imagem médica, categoria imagem de sensoriamento remoto. Este campo não foi implementado.

Formato

Rótulo: Format

Formato físico ou digital em que o recurso aparece.

Pode incluir formato de mídia ou dimensões do recurso. Pode ser usado para identificar os software, hardware, ou outro equipamento necessário para visualizar ou operar o recurso. Para este campo recomenda-se utilizar a lista de Tipo de Mídias da Internet (MIME - Internet Media Types)

Exemplo de uso: Tipo da imagem (JPG, GIF etc), tamanho da imagem, software em que foi desenvolvido.

Identificador

Rótulo: Resource Identifier.

Descrição: Referencia única dos recursos dentro de um dado contexto.

Recomenda-se identificar o recurso por strings ou números de acordo com sistemas de identificações formais, como por exemplo, URI (Recurso de identificação uniforme), DOI (Identificação digital de objetos), ISBN (Padrão internacional de numeração de livros) ou identificação de nomes únicos usados na UR*Lib*.

Exemplo de uso: O nome do repositório em que o dado é armazenado na URLib funciona como identificador da procedência. Exemplo: dpi.inpe.br/juliana/2004/08.04.17.20.

Fonte

Rótulo: Source.

Informação sobre um segundo recurso do qual o presente recurso é derivado. Embora seja recomendável que elementos contenham informação extraída do presente recurso apenas, o elemento Fonte pode conter data, criador, formato, identificador ou outro metadado de um segundo recurso quando este é considerado importante para a identificação do presente recurso.

Exemplo de uso: Sistema de Processamento em que a imagem foi gerada; linguagem de programação utilizada.

Idioma

Rótulo: Language

Descrição: Idioma do conteúdo intelectual do recurso. Uma prática recomendada é usar o RFC 3066 (Tags para identificação de linguagens) o qual em conjunto com a ISO639 (Códigos para representação de nomes e linguagens), define duas ou três primeiras letras como tag.

Exemplo de uso: Colocar "en" ou "eng" para o Inglês, "akk" para Akkadina, e "en-GB" para Inglês usado no Reino Unido.

Relação

Rótulo: Relation

Descrição: Possibilita relacionamento com outros recursos. A especificação desse elemento visa fornecer um meio de expressar relacionamentos entre recursos que têm relação formal com outros, mas que existem por si mesmos.

Exemplo de uso: imagens que foram derivadas de processamentos de outras imagens, ou seja imagens intermediárias. Esses dados já estão na linhagem e para evitar redundância achamos melhor não utilizar este campo.

Cobertura

Rótulo: Coverage

Descrição: Características espaciais ou temporais do conteúdo intelectual do recurso. Cobertura espacial refere-se a uma região física; uso de coordenadas ou nomes de lugares. Cobertura temporal refere-se sobre o que é o recurso, e não a uma data de criação ou identificação do recurso a um momento ou uma época.

Exemplo de uso: Localização da imagem (por exemplo: São José dos Campos); Latitude, Longitude. Este campo não foi implementado.

Direito autoral

Rótulo: Rights Management

Descrição: Uma declaração de direito sobre a propriedade, um identificador que vincula a uma declaração de direito sobre a propriedade, ou um identificador que vincula a um serviço que fornece informação sobre o direito de propriedade do recurso.

Exemplo de uso: O direito de propriedade da imagem passa a ser do autor do processamento realizado.

Linhagem

Rótulo: Lineage, History

Descrição: Procedimentos ou funções, com seus devidos parâmetros usados na geração do dado. Devem aparecer na ordem de aplicação. A linguagem utilizada pode ser a linguagem de programação (C++, Pascal, Python, MatLab etc) utilizada no processamento do dado.

Exemplo de uso: Funções e parâmetros em Python. Exemplo: img1 = img2 + img3

Observação: Este elemento não pertence ao esquema DC original.

3.2 Armazenamento

Para entender o armazenamento da procedência de dados, alguns conceitos devem ser introduzidos:

Integridade: garantia de que os dados contidos na procedência estão corretos e padronizados. A integridade é fornecida pelo uso de esquemas de metadados (Seção anterior) e geração automática dos dados.

Persistência: garantia da existência da procedência ao longo do tempo. É influenciada pela forma e local de armazenamento.

Integridade referencial entre dado e procedência: garantia de que a referência física entre a procedência e imagens nunca seja perdida. É também influenciada pela forma e local de armazenamento.

Os conceitos introduzidos mostram que o armazenamento da procedência é muito importante pois influenciam na persistência e integridade referencial.

Existem dois aspectos do armazenamento da procedência, o primeiro é o armazenamento intrínseco à procedência que chamaremos de forma de armazenamento e o segundo é o local de armazenamento. A seguir discutiremos melhor sobre esses aspectos.

3.2.1 Forma

A procedência pode ser armazenada de duas formas: embutida no próprio arquivo de dado ou como um arquivo de dados associado a ele. Se o dado for uma imagem, a procedência pode ser armazenada em seu cabeçalho (Figura 3.3) ou em um arquivo separado (Figura 3.4).

FIGURA 3.3 - Procedência embutida na imagem.

FIGURA 3.4 - Procedência em arquivo de dados separado.

Ambas as formas de armazenamento apresentam vantagens e desvantagens. No primeiro caso, não há quebra da integridade referencial entre dado e procedência, uma vez que a procedência está embutida no dado. Por outro lado, embutir novos dados significa modificar o formato do mesmo. No caso de imagens essa modificação implicaria em mais um novo formato no mercado.

No segundo caso, a relação física entre dado e procedência pode ser perdida acarretando perda da integridade referencial entre os dados. Além disso, essa dissociação pode fatalmente afetar a persistência desses dados ao longo do tempo. Por outro lado, a busca e recuperação de informações na procedência é mais fácil pois independe dos formatos dos dados. Além disso, o gerenciamento dos dados e da procedência podem ser feitos separadamente.

Analisando as vantagens e desvantagens entre as duas formas de armazenamento, optamos pela segunda forma, ou seja, dado e procedência armazenados em arquivos separados porém, associados.

3.2.2 Local

Para compensar as desvantagens que o armazenamento da procedência e dos dados em arquivos separados possuem, ambos devem ser armazenados em um sistema de armazenamento de dados (banco de dados, sistemas de arquivos, etc) que garanta a

integridade referencial entre eles nas operações de consulta, inserção, deleção, busca e recuperação.

Por razões que citaremos posteriormente, sugerimos guardar a procedência e os dados processados em uma biblioteca digital chamada UR*Lib* "Uniform Repositories for a Library" (Repositórios Uniformes para uma Biblioteca). (Banon, 2004).

De acordo com a Federação das Bibliotecas Digitais (DFL), "bibliotecas digitais são organizações que fornecem recursos, incluindo equipe especializada, para selecionar, estruturar, oferecer acesso intelectual, interpretar, distribuir, preservar a integridade, e garantir persistência integral dos acervos de trabalhos digitais de modo que estejam prontamente e economicamente disponíveis para o uso de uma comunidade".

São grandes as vantagens em armazenar a procedência e os dados na UR*Lib* dentre as quais citamos:

- Facilidade de uso: a UR*Lib* é amplamente utilizada no INPE e possui código aberto.
- Mantém a integridade referencial entre os dados e a procedência.
- Informação facilmente compartilhada: os dados depositados numa biblioteca digital são disponibilizados na Internet, o que possibilita uma ótima disseminação da procedência.
- Informação sempre disponível: a qualquer momento podemos consultar e recuperar dados e sua procedência, uma vez que os acervos ficam disponíveis 24 horas.
- Acesso persistente: qualquer que seja o acervo local onde foi depositado a procedência, seu acesso é persistente, ou seja ele sempre estará disponível mesmo que tenha mudado de acervo local.

- Respeitar o direito moral e patrimonial do autor e detentor do documento: armazenar um documento numa biblioteca digital é um ato de publicação (Banon 2004). Sendo assim, a biblioteca digital, juntamente com a procedência do dado, permitem identificar com confiança quem é o detentor do direito moral (autor) e o detentor do direito patrimonial (pessoa ou instituição que mantém o acervo local) dos processamentos descritos na procedência.

Para entender o processo de armazenamento da procedência na UR*Lib* é necessário compreender seu funcionamento. A seguir, em linhas gerais, descrevemos a arquitetura e funcionamento da UR*Lib*.

UR*Lib*

A UR*Lib* é uma biblioteca que se enquadra no conceito de biblioteca digital segundo definição da DLF. Podemos verificar isso analisando seu funcionamento. Os principais componentes da UR*Lib* são: os usuários, o acervo distribuído e o sistema UR*LibService* (Figura 3.5).

Os usuários acessam documentos armazenados no acervo distribuído. O acervo é composto por acervos locais, cada um sendo acessível a partir de seu próprio UR*LibService*.

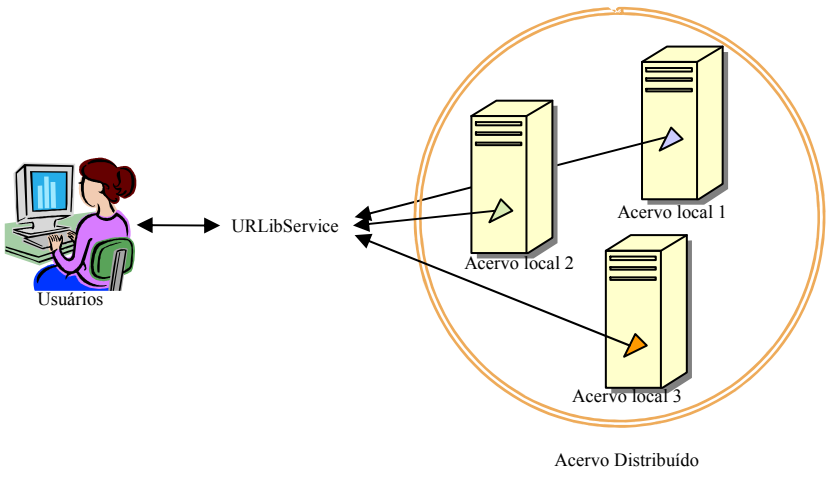


FIGURA 3.5 - Arquitetura UR*Lib*.

Usuários

A UR*Lib* possui dois tipos de usuários, os usuários da biblioteca e os bibliotecários. Os usuários da biblioteca buscam e recuperam os dados da UR*Lib*, os bibliotecários ou administradores são responsáveis pelo gerenciamento e enriquecimento dos acervos. Um navegador de Internet é usado para que os usuários interajam com os acervos. Neste caso, o navegador conecta-se com o UR*LibService*, que fornece as funções para interagir com o sistema.

Acervos e Repositórios

Como dito anteriormente, a UR*Lib* consiste em um acervo distribuído (Figura 3.6). A distribuição do acervo atende a necessidade de repartir melhor as responsabilidades de entrada dos dados e permite resolver limitações de armazenamento num determinado disco rígido ou partição.

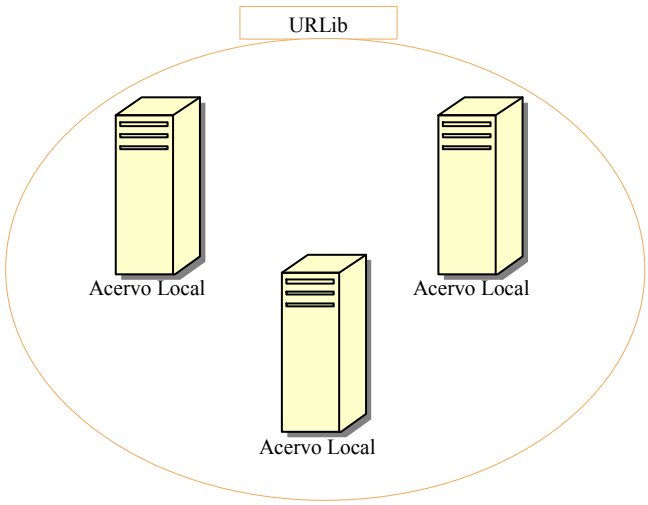


FIGURA 3.6 - Acervo distribuído.

Cada acervo local da UR*Lib* hospeda obras em espaços digitais. Esses espaços são chamados de repositórios uniformes ou simplesmente de repositórios (Figura 3.7.)

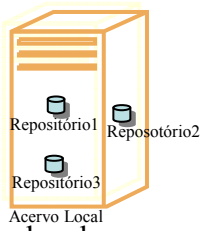


FIGURA 3.7 - Acervo local e seus repositórios uniformes.

Cada repositório contém um único material eletrônico chamado de documento. Esse documento é um conjunto de dados. Esses dados podem ter diferentes formatos (imagens, textos, sons, vídeos) e estarem localizados dentro de diretórios ou subdiretórios (Figura 3.8).

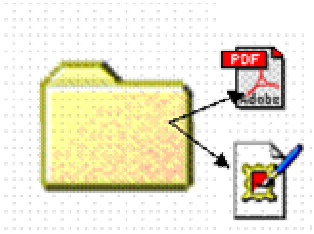


FIGURA 3.8 - Repositório uniforme contendo o documento.

Os repositórios possuem permissões próprias a serem satisfeitas antes do usuário poder acessá-los e suporta vários tipos de formatos de documentos (por exemplo: pdf, zip, doc, jpg etc).

UR*LibService*

UR*LibService* (Serviço da UR*Lib*) é o software que faz a interface entre os usuários e os acervos locais da UR*Lib*, sendo que cada Acervo possui seu UR*LibService*.

Na visão do usuário da biblioteca, a distribuição de acervos locais é transparente, ou seja é irrelevante, uma vez que através do UR*LibService*, todos estes acervos passam a ser integrados.

As principais funções do UR*LibService* são:

- Depositar e atualizar documentos remotamente utilizando o navegador;

- Buscar documentos através de palavras chaves e emitir relatórios das buscas;
- Recuperar documentos;
- Restringir o acesso aos repositórios;
- Copiar documentos de um acervo local para outro;
- Controlar as versões dos documentos;
- Exibir as estatísticas de acesso de cada documento;
- Exibir os metadados no formato BibTeX ou Refer (EndNote), XML e OAI-DC;
- Autenticar as cópias de um documento original;
- Identificar documentos originais;
- Transferir um documento original para um novo detentor de direitos autorais;
- Garantir a persistência dos vínculos (“links”).

Já que a procedência é armazenada na URLib, nos beneficiamos de todas as funcionalidades do URLibService mencionadas anteriormente. No entanto, existem duas funcionalidades que merecem destaque: Persistência de vínculos e Identificação do de documentos originais.

Persistência de vínculos

Uma vez que os acervos locais estão distribuídos em rede, assegurar a persistência dos vínculos quando um documento é passado ou copiado de um acervo para o outro é muito importante.

Na maioria dos banco de dados da Web o identificador de um documento é o seu endereço digital ou URL. É através deste endereço que os sites de busca localizam documentos procurados. Porém, se o documento mudar de local, muda-se a URL e

consequentemente todas as referências existentes no documento passam a ser desatualizadas e sem utilidade.

Para o UR*LibService* contornar esse problema e garantir a persistência dos vínculos dentro de seu Acervo distribuído ele utiliza o sistema de identificação dos documentos por nomes reservados. Neste sistema cada documento recebe um nome reservado que funciona como um identificador. Em conseqüência, o URLibService busca documentos no Acervo distribuído através do identificador do documento e não através do endereço do acervo em que o documento foi armazenado. Sendo assim, dentro da UR*Lib*, o problema de mudança de acervos locais é facilmente contornado.

Os nomes reservados dos acervos e repositórios da U*RLib* são casos particulares de *Uniform Resource Identifier* (URI). Todavia, os nomes dos acervos e repositórios seguem a seguinte regra de formação:

<domainname>/<username>/<year>/<month>.<day>.<hour>.<minute>[.<second>],

onde:

domainname é o nome de domínio no e-mail do usuário que está responsável pelo acervo local;

username é o nome do usuário no e-mail;

year, month e day formam a data de criação do acervo ou repositório; e

hour, minute e second formam o horário de criação do acervo ou repositório.

A Figura 3.9 contém exemplos de nomes de acervo e repositórios.

Identificação do original

A Identificação do original de um documento é muito importante. Somente comparando a cópia de um documento com o original é que podemos ter certeza se a cópia sofreu ou não alterações (Banon et al., 2004)

A URLibSevice trata a identificação da seguinte maneira:

- Cada acervo local recebe um nome reservado, ou seja, nunca dois ou mais acervos locais possuem o mesmo nome.
- Cada documento possui um nome reservado que será usado como nome do repositório do acervo da URLib. Documentos distintos nunca possuem o mesmo nome.
- Repositórios em acervos diferentes que possuem o mesmo nome possuem em princípio os mesmos documentos. No entanto, somente um deles é o documento original.
- Todo repositório guarda o nome do acervo hospedeiro do documento original.

A Figura 3.9 ilustra um exemplo sobre o uso de nomes reservados para identificar o original.

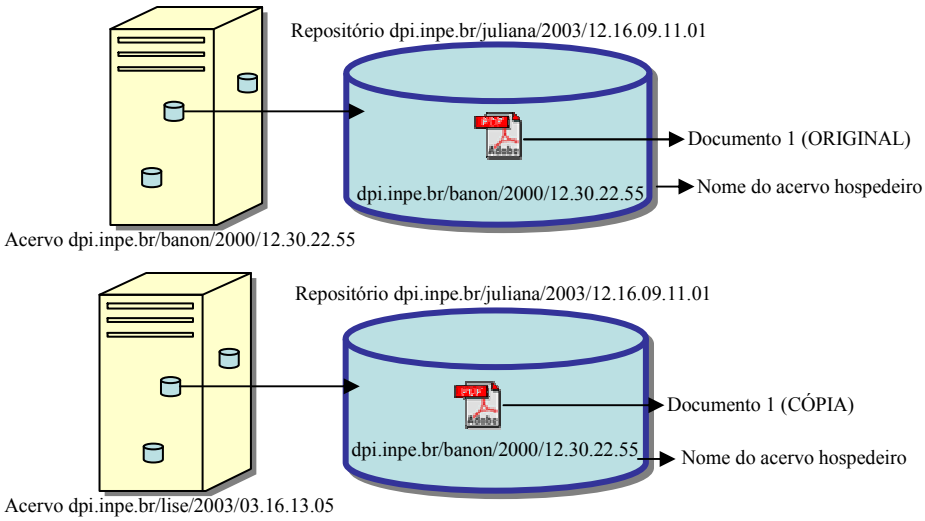


FIGURA 3.9 - Identificação dos documentos originais pela URLib.

Observamos na Figura 3.9 que existem dois acervos contendo repositórios com o mesmo nome e em princípio possuindo documentos iguais. Observamos também que os acervos possuem nomes diferentes e que todo repositório contém o nome do acervo local que hospeda o documento original. Entretanto, o acervo de nome

dpi.inpe.br/banon/2000/12.30.22.55 contém o documento original e o acervo de nome dpi.inpe.br/lise/2003/03.16.13.05 contém a cópia do original. Para chegar a esta conclusão basta consultar o nome do acervo hospedeiro em cada repositório.

Ao acessar um repositório, o UR*LibService* exibe uma capa com um cabeçalho indicando se a versão acessada é a oficial, isto é, se o documento aberto é idêntico ao original.

Com esses mecanismos, o UR*LibService* permite que os repositórios criados num determinado acervo local possam ser copiados sem problemas para qualquer outro acervo local e que, mesmo fazendo cópias de um repositório para outros acervos locais, ficamos sabendo qual é o acervo local que possui o original.

CAPÍTULO 4

IMPLEMENTAÇÃO DE UM PROTÓTIPO PARA GERENCIAR A PROCEDÊNCIA

Para mostrar a viabilidade das formalizações propostas no Capítulo 2 e Capítulo 3, desenvolvemos um sistema computacional responsável pela criação, armazenamento, preservação automática dos dados originais e reprodução de dados utilizando procedência.

Esse sistema é um protótipo e foi implementado como um módulo complementar a sistemas de processamentos já existentes.

O Protótipo é um sistema de processamento de dados que gerencia a procedência dos dados por ele processado. O gerenciamento envolve: criação, armazenamento, busca e recuperação da procedência, preservação dos dados originais e reprodução dos dados intermediários. Como o enfoque do trabalho é o gerenciamento da procedência de dados chamamos o protótipo de Sistema Gerenciador Procedência de Dados (SGPD).

Os módulos do SGPD responsáveis por gerenciar a procedência podem ser implementados embutido em sistemas de processamento existentes no mercado. Isto significa que, para implementar as formalizações da procedência proposta não é necessário desenvolver todo um sistema de processamento de dados, e sim implementar somente os processos responsáveis pelo gerenciamento da procedência.

Nos tópicos seguintes, descrevemos o funcionamento do SGPD. O propósito dessa descrição é mostrar a lógica de desenvolvimento dos fundamentos da procedência de modo a facilitar sua possível implementação em sistemas existentes. Nosso objetivo não é impor tecnologias ou linguagens de programação, sendo assim detalhes sobre as tecnologias utilizadas e código de desenvolvimento encontram-se em Apêndice A.

4.1 Arquitetura e Ambiente de Programação

O SGPD possui arquitetura Cliente/Servidor (Figura 4.1).

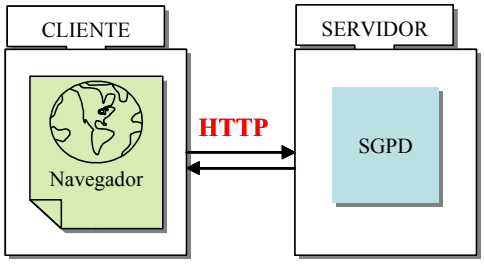


FIGURA 4.1 – Arquitetura Cliente/Servidor do SGPD.

Na Figura 4.1, é mostrado que no lado cliente encontra-se o navegador ("browser") e no lado servidor encontra-se o SGPD.

O protocolo de comunicação usado para troca de dados entre navegador e SGPD é o Hypertext Transfer Protocol (HTTP) (Orfali et al, 1996). Sendo assim, o cliente (navegador) estabelece uma conexão HTTP com o servidor remoto, e emite uma solicitação. O servidor recebe a solicitação, o SGPD processa essa solicitação e o HTTP retorna a resposta do processo para o Cliente. (Apêndice A).

O Servidor HTTP utilizado foi o software de distribuição gratuita Apache configurado automaticamente pelo UR*LibService* (Apache).

O SGPD é um script escrito em Python 2.2 (Apêndice B). Escolhemos Python, pois ele possui código aberto, distribuição gratuita, interface com uma biblioteca de alto desempenho (C/C++) e permite rápido desenvolvimento (ideal para implementação de protótipos). Além disso, ele tem excelente suporte a operações matriciais (condição esperada em um sistema de processamento de imagens) e módulos eficientes de processamento de imagens (Lundh, 2001).

4.2 Processo de Desenvolvimento

Para auxiliar na descrição do funcionamento do SGPD, apresentamos primeiro seu diagrama de fluxo de dados (DFD) (Hawryszkiewycz, 1994). Em seguida, descrevemos os processos representados no DFD. (Figura 4.2)

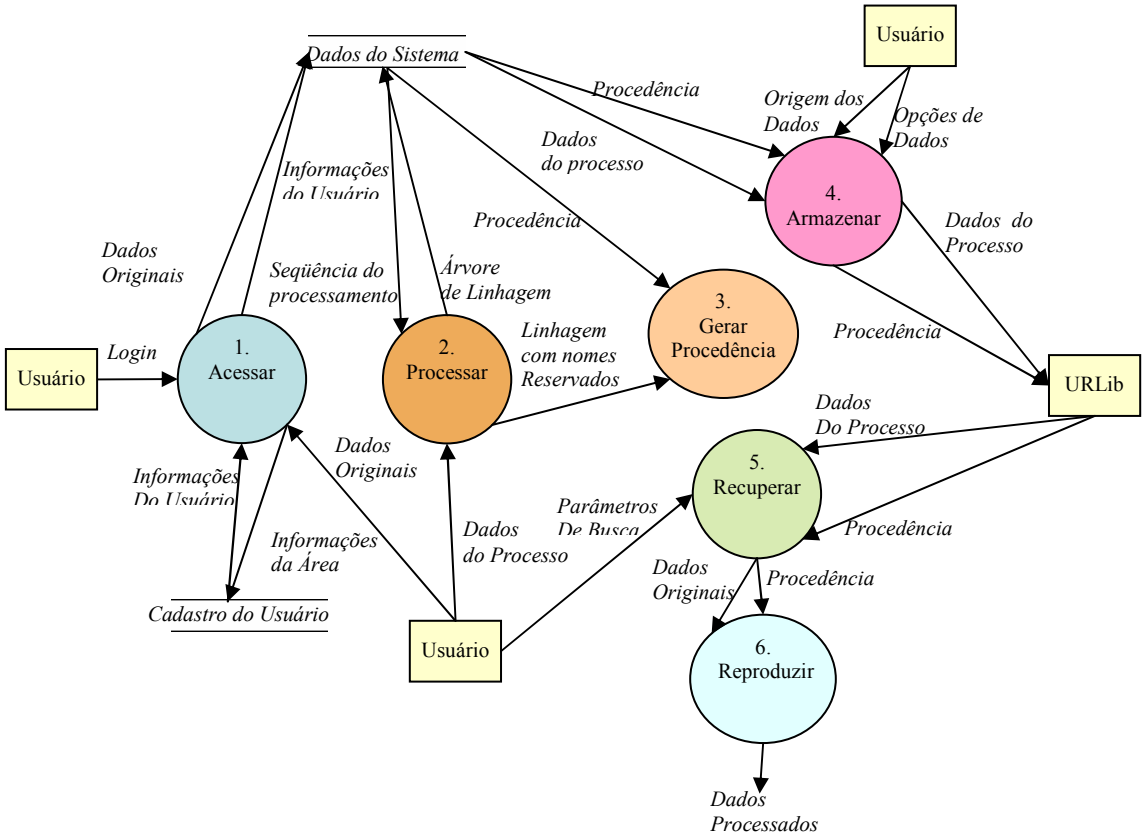


FIGURA 4.2 – Diagrama de fluxo de dados do SGPD.

O DFD mostra que o SGPD possui:

- seis Processos: *Acessar*, *Processar*, *Gerar-Procedência*, *Armazenar*, *Buscar-Recuperar*, e *Reproduzir*;
- duas entidades externas: UR*Lib* e Usuário; e
- dois depósitos de dados: Cadastro do Usuário e Dados do Sistema (Sessão e Área de Trabalho).

A seguir, explicamos o funcionamento dos processos relacionando-os com as entidades externas e os depósitos de dados. Um exemplo é mostrado ao final de cada SubSeção para auxiliar no entendimento dos processos.

4.2.1 Acessar

Para o usuário obter acesso ao SGPD é necessário que ele tenha sido previamente cadastrado pelo administrador do sistema. O processo *Acessar*, verifica na base de dados do usuário se este possui ou não um cadastro. Em caso positivo o usuário entra no sistema.

O processo *Acessar* é dividido em 3 subprocessos: *Gerenciar-Área-de-Trabalho*; *Gerenciar-Sessão*; e *Receber-Dados* (Figura 4.3).

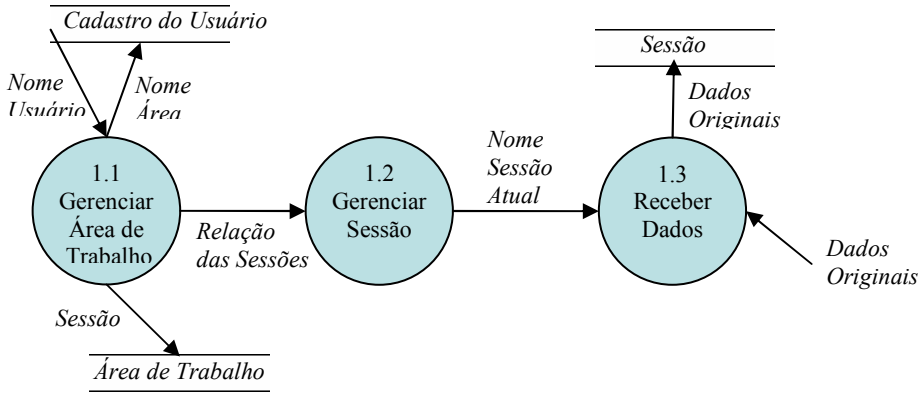


FIGURA 4.3 – Explosão do processo Acessar.

No primeiro acesso do usuário ao SGPD, o processo *Gerenciar-Área-de-Trabalho* cria um diretório de trabalho para o usuário no Servidor. Chamamos esse diretório de Área de Trabalho.

A partir do segundo acesso do usuário ao SGPD, o processo *Gerenciar-Área-de Trabalho* consulta o nome da Área do usuário em seu cadastro e o direciona para a mesma. O processo referido também é responsável por criar Sessões. As Sessões são Subdiretórios da Área de Trabalho que armazenam arquivos temporários, procedência e outros dados utilizados pelo usuário durante um processamento. Sendo assim, uma Área

de trabalho pode conter várias Sessões, cada uma delas relativas a um processamento diferente. (Figura 4.4).

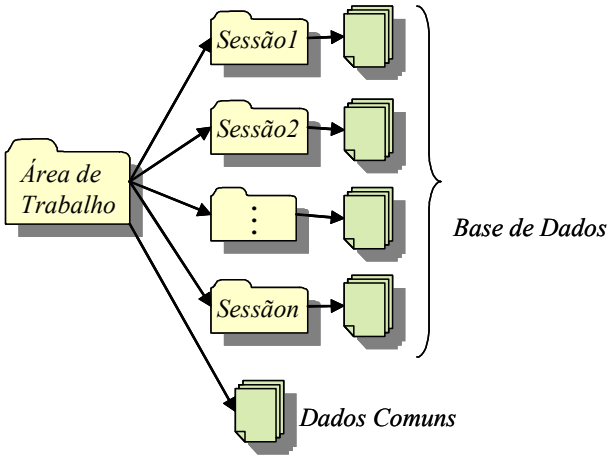


FIGURA 4.4 - Área de Trabalho e sua divisão em Sessões.

Observe pela Figura 4.4 que os dados comuns a todas as sessões são armazenados na Área de Trabalho.

Quando o usuário entra em sua Área de Trabalho, o processo *Gerenciar-Sessão* exibe o nome das sessões existentes nesta Área e permite que o usuário escolha entre criar uma nova Sessão ou utilizar uma Sessão já existente. Sessões já existentes permitem a continuidade de processos feitos anteriormente, e novas sessões permitem iniciar novos processos.

Optamos por dividir a área em sessões pois essa divisão possibilita que vários processamentos sejam realizados na mesma Área de Trabalho sem que um influencie no outro. Além disso, o uso de nomes reservados permite que o processamento realizado em uma Sessão seja importado e aproveitado em outra Sessão sem conflitos entre eles.

O processo *Gerenciar-Sessões* apaga sessões que não serão mais utilizadas pelo usuário.

O processo *Receber-Dados* permite que o usuário envie dados do computador para as Sessões localizadas no servidor. Os dados são transferidos via download, ou seja o usuário fornece o caminho dos dados e o processo transfere os dados para o servidor via HTTP.

Já que o protocolo HTTP não possui memória, as Sessões são também utilizadas para armazenar a memória do sistema.

4.2.2 Processar

Este processo captura a sequência de um processamento e constrói a linhagem dessa seqüência de dados. O processo possui 4 subprocessos (Figura 4.5): *Capturar*; *Executar*, *Gerar* -Árvore e *Alocar-Nomes*.

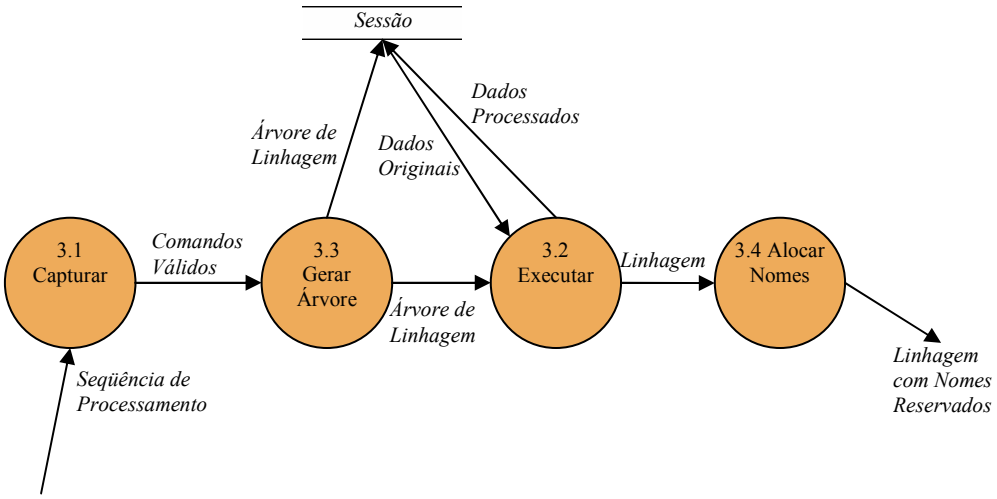


FIGURA 4.5 – Explosão do processo Processar.

O processo *Capturar* monitora as ações do usuário durante o processamento realizado em uma Sessão, e captura os comandos (funções e parâmetros) utilizados nesse processamento. Os comandos considerados válidos são enviados para o processo *Gerar -Árvore.*

Para o processo *Capturar*, um comando é válido se ele for relevante para descrever a linhagem de um dado. Ou seja, se for um comando que acarreta transformações em um dado. Por exemplo, comandos para imprimir imagens na tela não são válidos. Não são válidos também os comandos que apresentam erros de sintaxe.

Para verificar se um comando é válido ou não, utilizamos o Parser da linguagem Python (Apêndice B). O Parser constrói a árvore de derivação (2.1.4) e envia os comandos válidos para o processo *Gerar-Árvore*.

O processo *Gerar-Árvore* constrói a árvore de linhagem a partir dos comandos válidos. A árvore de linhagem é representada computacionalmente por uma estrutura de dados em árvore onde cada nó armazena a linhagem dos dados processados. A árvore de estrutura de dados é salva na Sessão, e é modificada cada vez que o usuário entra com um comando válido.

O processo *Executar* executa os comandos guardados nos nós da árvore. Na execução dos comandos são gerados os dados relativos às linhagens, ou seja, os dados intermediários. Esse processo vai executando os comandos à medida que a árvore é construída.

O processo *Alocar-Nomes* lê os nodos da árvore de linhagem gravada na Sessão e aloca nomes reservados às linhagens. Esses nomes correspondem ao local em que a procedência (origem e linhagem) pode ser armazenada na UR*Lib*.

Para alocar nomes, o módulo implementa o processo de formação de nomes similar ao implementado na UR*Lib* (Seção 3.2.2).

Para formar os nomes reservados, o processo obedece as seguintes regras:

<nickname>__<domainname>__<username>__<year>__<month>_<day>_<hour>_<minute>[_<second>],

onde: nickname é um nome curto para o dado ao qual pertence a linhagem, este nome é sugerido pelo usuário; domainname é o nome de domínio onde pertence a máquina do usuário; username é o nome do administrador do sistema; year, month e day formam a data e hour, minute e second formam o horário de criação da linhagem.

Para esclarecermos o funcionamento dos processos *Capturar*, *Executar*, *Gerar- Árvore-* e *Alocar-Nomes*, apresentamos o exemplo a seguir:

Suponhamos que um usuário cadastrado entra em sua área de trabalho e cria uma nova Sessão de processamento. Nesta Sessão, o usuário digita os seguintes comandos escritos em Python:

a1 = 2

a2 = 3

a3 = a1 + a2

a4 = a3 + a1

Neste processo, a1 e a2 são os nomes curtos dos dados originais e a3 e a4 são nomes curtos dos dados intermediários.

À medida que o processo *Capturar* lê os comandos ele os envia para o módulo *Gerar-Árvore*. Ou seja, lê o primeiro comando e envia, lê o segundo comando e envia, e assim sucessivamente. Ao enviar o último comando, o processo *Gerar-Árvore* gera a árvore de linhagem representada na Figura 4.6.

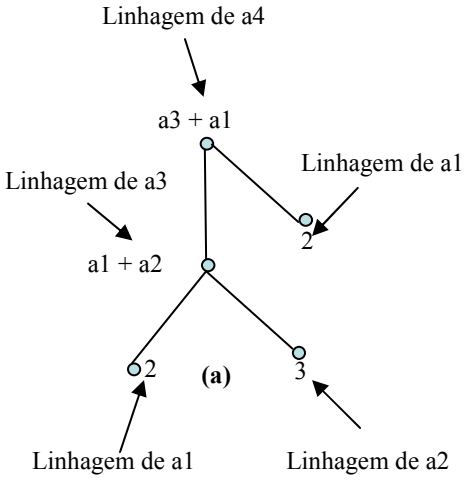


FIGURA 4.6 - Árvore de linhagem para obtenção de a4.

O processo *Executar* executa cada nodo da árvore de linhagem. Ao final da execução dos comandos, o processo *Executar* gera o dado intermediário resultante 7.

Exec(2) = 2

Exec(3) = 3

Exec(a1 + a2) = Exec(a1) + Exec(a2) = 2 + 3 = 5

Exec(a1 + a3) = Exec(a1) + Exec(a3) = 2 + 5 = 7

O processo *Alocar-Nomes*, aplica a regra de formação de nomes reservados para cada linhagem (nós das árvores de linhagem) e retorna os seguintes nomes para essas linhagens:

a1_dpi.inpe.br_juliana_2003_03_10_13_15 para linhagem 2

a2_dpi.inpe.br_juliana_2003_03_10_13_30 para linhagem 3

a3_dpi.inpe.br_juliana_2003_03_10_13_40 para linhagem a1 + a2

a4_dpi.inpe.br_juliana_2003_03_10_13_45 para linhagem a3 + a1

Ressaltamos que o uso de nomes reservados permite que a linhagem seja referenciada em sessões diferentes, sem que haja repetição de nome (ou seja, o mesmo nome utilizado para mais de uma linhagem). No entanto, como os nomes reservados são expressões extensas, o usuário faz referência às linhagens através do nome curto quanto este não apresenta ambigüidades. Estes são automaticamente renomeados pelo sistema para nomes reservados.

4.2.3 Gerar Procedência

O processo *Gerar-Procedência* recebe parte da origem de dados (Nome do usuário e Data do Processamento) e toda linhagem dos dados que foram processados (Figura 4.7).

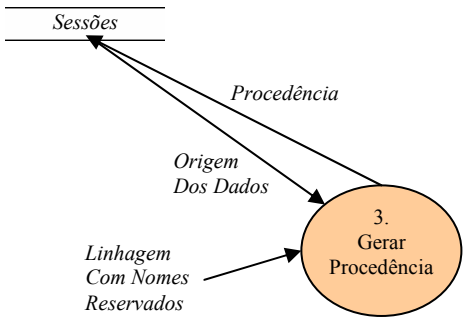


FIGURA 4.7 - Processo Gerar Procedência.

O Processo *Gerar-Procedência*, cria um arquivo texto e escreve as informações sobre origem e linhagem neste arquivo. As informações são estruturadas em XML. As tags XML são organizadas seguindo o esquema DC (Seção 0).

De acordo com o exemplo anterior, a procedência de a4 estruturada em XML é mostrada a seguir:

<data>10/03/2003</data>

<creator >Juliana Braga</ creator >

<lineage>

a4_dpi.inpe.br_juliana_2003_03_10_13_45 =
a3_dpi.inpe.br_juliana_2003_03_10_13_40 +
a1_dpi.inpe.br_juliana_2003_03_10_13_15

a3_dpi.inpe.br_juliana_2003_03_10_13_40 =
a1_dpi.inpe.br_juliana_2003_03_10_13_15 +
a2_dpi.inpe.br_juliana_2003_03_10_13_30

a2_dpi.inpe.br_juliana_2003_03_10_13_30 = 3

a1_dpi.inpe.br_juliana_2003_03_10_13_15 = 2

</ lineage>

Observe que, o sistema registrou não só a linhagem de a4 mas também a linhagem dos dados intermediários e originais relacionados à a4. Achamos que a representação mostrada é mais completa, pois exibe todo o processo que o dado sofreu, por isso optamos por ela na implementação do sistema. Essa representação é exibida para o usuário com nomes curtos para facilitar o entendimento, porém é registrada de forma completa.

4.2.4 Armazenar

O processo *Armazenar* envia dados originais, intermediários e procedência desses dados para o Acervo da UR*Lib*.

Ele é dividido em 3 subprocessos: *Identificar-Originais*, *Comunicar* e *Enviar* (Figura 4.8).

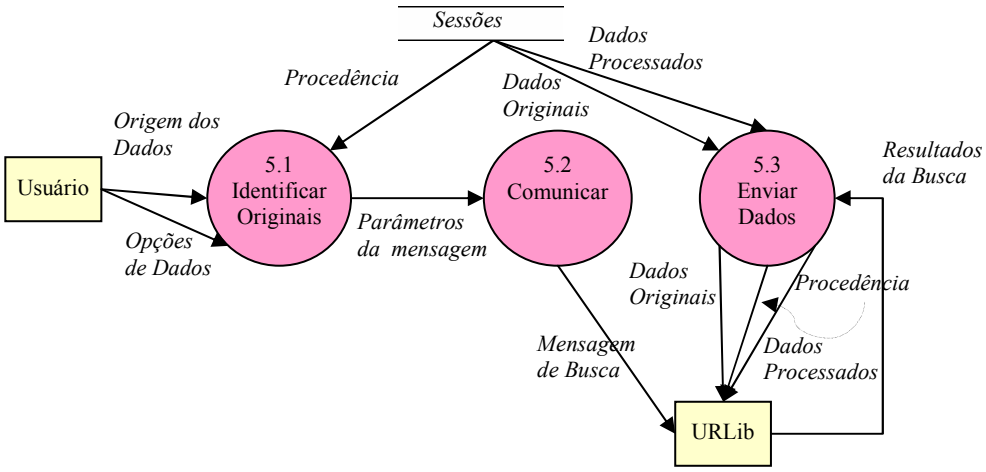


FIGURA 4.8 - Explosão do processo Armazenar.

O processo *Identificar-Originais* solicita ao usuário as seguintes informações:

- indicação do dado a ser armazenado o qual chamamos de dado final;
- local em que o dado final será armazenado, ou seja, URL do Acervo local da UR*Lib*.

- informações complementares à Origem dos dados: título do processamento que o dado sofreu, instituição em que o processo foi realizado, descrição em alto nível do processo etc.

O processo *Identificar* insere na procedência (arquivo XML) as informações complementares à Origem dos dados (Autores, Título do Processamento, Instituição, Descrição do processo e E-mail de contato). Em seguida, ele aguarda comunicação com a UR*Lib*.

O processo *Comunicar* faz a conexão entre o SGPD e o Acervo da UR*Lib* em que o dado final será armazenado (Figura 4.9).

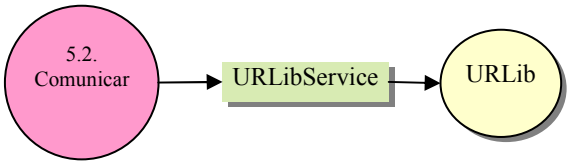


FIGURA 4.9 - Conexão entre o processo Comunicar e a UR*Lib*.

O SGPD e o UR*LibService* podem estar rodando no mesmo computador servidor ou em servidores diferentes. Devido a isso, a troca de informações entre SGPD e UR*LibService* pode ser feita via socket ou via HTTP (Orfali, 1996). O socket é uma primitiva de acesso a redes TCP/IP que permite enviar e receber mensagens (strings) entre dois processos executados no mesmo servidor ou em servidores diferentes.

Caso a comunicação SGPD/UR*Lib* seja feita para enviar dados originais para UR*Lib* o processo é feito via socket. Caso contrário o processo é feito via HTTP. Essa diferença existe porque os dados originais são enviados sem a intervenção do usuário e outros tipos de dados (dado de interesse) são enviados com a intervenção do usuário.

Para enviar os dados originais, o processo *Enviar* manda duas mensagens, via socket, para o UR*LibService*. As mensagens são strings de comandos que o UR*LibService* lê e executa sobre os Acervos da UR*Lib*.

A primeira mensagem dispara buscas em todos os Acervos da UR*Lib*, e verifica se os dados originais a serem enviados já foram ou não armazenados nesses Acervos. Em caso positivo, o envio é cancelado. Em caso afirmativo, a segunda mensagem é enviada.

A segunda mensagem cria um repositório no Acervo em que a conexão foi estabelecida. A mensagem é o comando de criação de repositórios na UR*Lib*. Os parâmetros desse comando são: nome do repositório, procedência em XML, nome e senha do usuário conectado ao SGPD. (Apêndice A).

Para enviar dados que não sejam os originais, o processo *Enviar* manda via HTTP o endereço do local (Apêndice A) em que os mesmos estão armazenados. A partir desse endereço a UR*Lib* faz o download desses dados se os mesmos já não estejam no Acervo.

Ressaltamos que, o SGPD sabe o nome do repositório em que o dado será armazenado antes do repositório ser criado. Um detalhe importante de implementação é o processo de obtenção deste nome pelo SGPD. Devido sua importância, ilustramos este processo retomando os exemplos dos tópicos anteriores.

No exemplo, existem 4 dados na Sessão:

a1 (dado original), a2 (dado original), a3 e a4.

Suponhamos que o usuário deseja armazenar a4. O processo Enviar, encontra o nome do repositório a partir do nome reservado de a4. Sendo assim, os passos para encontrar o nome do repositório são:

1. Ler o nome reservado da linhagem do dado final

   a4_dpi.inpe.br__juliana__2003__03_10_13_45

2. Retirar o nome curto do nome reservado

   _dpi.inpe.br__juliana__2003__03_10_13_45

3. Retirar o primeiro caractere "_" encontrado

dpi.inpe.br__juliana__2003__03_10_13_45

4. Substituir os caracteres "__" pelo caracter "/"

   dpi.inpe.br/juliana/2003/03_10_13_45

5. Substituir o caractere "_ " pelo caracter "."

   dpi.inpe.br/juliana/2003/03.10.13.45

Finalmente, o nome do repositório no qual a4 será armazenado é: dpi.inpe.br/juliana/2003/03.10.13.45.

Após a criação do Repositório e o envio da procedência, o UR*LibService* faz um "download" via HTTP do dado final e dos dados originais se necessário (Apêndice A). Cada dado será armazenado em seus devidos repositórios na UR*Lib*.

4.2.5 Recuperar

O processo *Recuperar* faz buscas na UR*Lib*, retorna o resultado das buscas para o usuário e permite o download desses resultados para a Sessão atual. Ele é dividido em dois subprocessos: *Buscar* e *Carregar* (Figura 4.10).

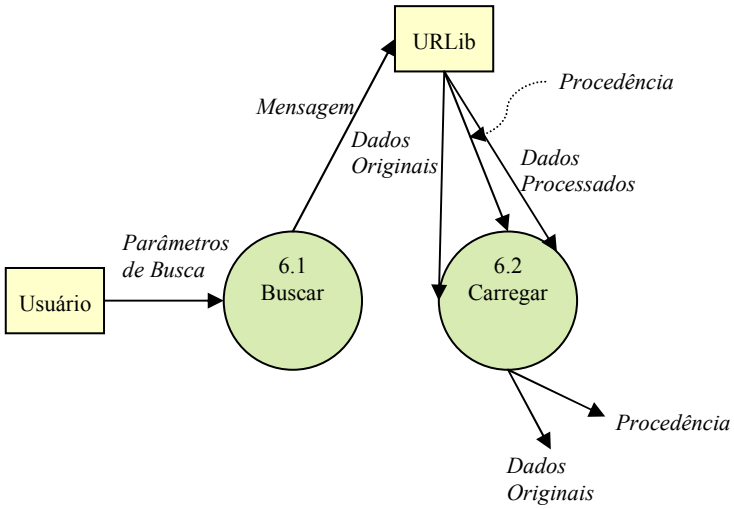


FIGURA 4.11 - Explosão do processo Recuperar.

O processo *Buscar* faz uma procura por dados armazenados na UR*Lib*. A busca é feita utilizando palavras chaves digitadas pelo usuário. Os campos de procura são os referentes à origem dos dados (Autor, Título, Data de Criação e Descrição) e/ou qualquer fração da linhagem dos dados.

O processo *Buscar* utiliza o processo *Comunicar* (Seção anterior) para obter conexão HTTP com a UR*Lib*. O processo Enviar remete os parâmetros de busca via HTTP para o UR*LibService*, e este dispara a busca nos Acervos locais da UR*Lib*. Os parâmetros de busca são os campos de procura e respectivas palavras chave (Apêndice A). O UR*LibService* retorna o resultado como uma página Web, também via conexão HTTP, para o SGPD.

Mediante os resultados retornados, o usuário indica o dado que este deseja recuperar. As opções são:

- procedência e dados originais,
- procedência e dado final, e
- procedência e todos os dados (originais, intermediários e final).

Depois de escolhida a opção, o processo *Carregar* é ativado. Este inicia o download dos dados contidos no Acervo para a Sessão atual. A informação do repositório em que o download é feito está na página de resultados que a U*RLibService* retornou (Apêndice A).

A terceira opção (carregar procedência e todos os dados), reproduz os dados intermediários e final a partir da procedência e dos dados originais. Neste caso, o processo utilizado é *Reproduzir* o qual explicamos em seguida.

4.2.6 Reproduzir

O processo *Reproduzir* gera dados processados a partir da procedência e dos dados originais. Isso significa que, não precisamos armazenar todos os dados produzidos em

um processamento. Se armazenarmos apenas os dados originais e a procedência dos dados intermediários, esses podem ser gerados novamente (ou seja, reproduzidos).

O processo *Reproduzir*, utiliza três subprocessos externos a ele: *Recuperar*, *Gerar-Árvore* e *Executar*.

Explicamos esse processo fazendo analogias com o exemplo anterior.

Suponhamos que um usuário entrou em uma nova Sessão e deseja reproduzir os mesmos dados utilizados no exemplo anterior, ou seja deseja reproduzir os dados a1, a2, a3 e a4. Para isso ele entra na UR*Lib*, e escolhe a opção recuperar todos os dados referentes a a4.

Neste ponto, o processo *Reproduzir* usa o processo *Recuperar* para fazer o "download" da procedência e dos dados originais. O processo lê a fração referente a linhagem (Figura 4.12) dos dados na procedência e envia esta linhagem para o processo *Gerar-Árvore.*

```
<data>10/03/2003</data>
<creator >Juliana Braga</ creator >
<title>Exemplo da Tese</title>
<editor>INPE</editor>
<email>juliana@dpi.inpe.br</email>
<lineage>
a4_dpi.inpe.br_juliana_2003_03_10_13_45 = a3_dpi.inpe.br_juliana_2003_03_10_13_40 +
a1_dpi.inpe.br_juliana_2003_03_10_13_15
a3_dpi.inpe.br_juliana_2003_03_10_13_40 = a1_dpi.inpe.br_juliana_2003_03_10_13_15  +
a2_dpi.inpe.br_juliana_2003_03_10_13_30
a2_dpi.inpe.br_juliana_2003_03_10_13_30 = 3
a1_dpi.inpe.br_juliana_2003_03_10_13_15 = 2
</ lineage>
```

Fração da Procedência Relativa à Linhagem

FIGURA 4.12 - Procedência em XML do dado a4.

A partir da linhagem, o processo *Gerar-Árvore* produz a árvore de linhagem (Figura 4.6).

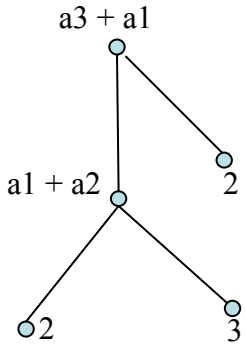


FIGURA 4.13 - Árvore de linhagem de a4.

O processo *Executar*, executa a linhagem referente aos nós intermediários da árvore, iniciando da esquerda para direita de baixo para cima. Dessa forma, conseguimos recuperar todos os dados intermediários até chegar á raiz (dado final). Observe pela Figura 4.14, que sem os dados originais, não é possível recuperar os outros dados.

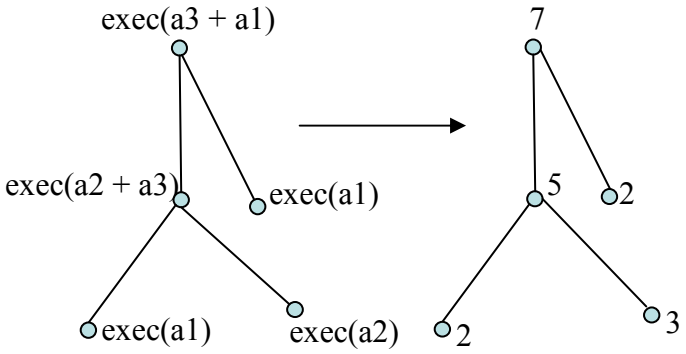


FIGURA 4.14 - Execução dos nodos da árvore de linhagem para obtenção dos dados.

Por fim, o processo *Reproduzir* concatena a árvore de linhagem recuperada com a árvore de linhagem existente na Sessão. Neste exemplo, a Sessão é nova, portanto não havia árvore de linhagem anteriormente para ser concatenada.

Os nomes reservados das linhagens já estão na procedência e, por isso não é necessário executar o processo de alocação de nomes.

Reforçamos em dizer que no SGPD todo dado pode ser reproduzido através de sua procedência, se e somente se, os originais existirem. Por isso a importância de preservar sempre os dados originais.

CAPÍTULO 5

TESTES DO PROTÓTIPO

Para mostrar os testes realizados no SGPD, dividimos este Capítulo em duas Seções. Na primeira Seção (5.1) apresentamos o funcionamento do SGPD para processamento de dados em geral. Na segunda Seção (5.2) mostramos aplicações do SGPD ao processamento de imagens.

5.1 Funcionamento

A seguir apresentamos o funcionamento do SGPD através de sua Interface. Os testes realizados foram baseados do exemplo mostrado no Capítulo 4.

A Figura 5.1 mostra a tela de entrada do SGPD. O usuário deve estar devidamente cadastrado no sistema e possuir login e senha de acesso. Com o acesso permitido o usuário entra em sua área de trabalho (Figura 5.2) e escolhe entre trabalhar em uma nova Sessão ou em sessões já existentes.

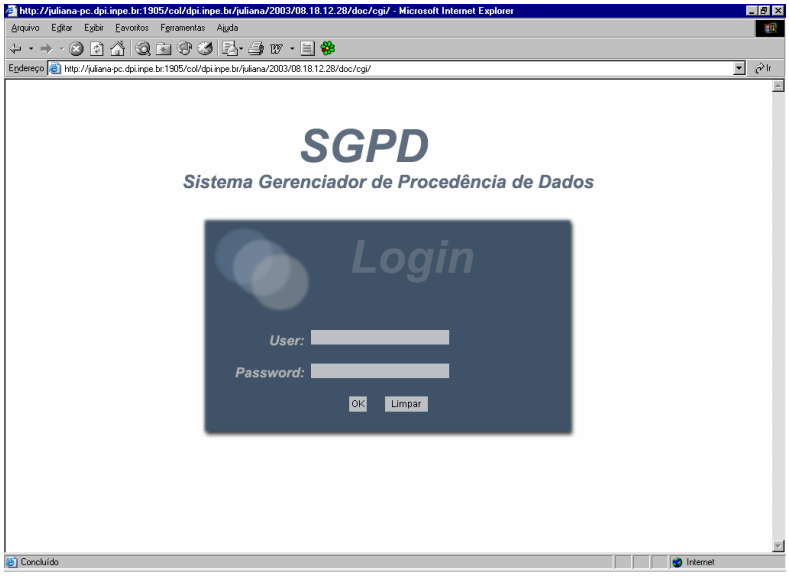


FIGURA 5.1 - Tela de entrada do SGPD.

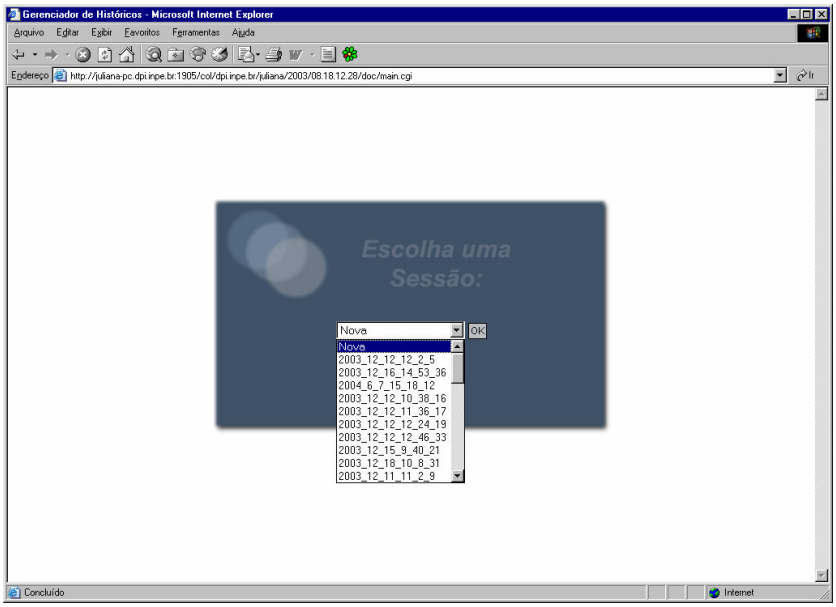


FIGURA 5.2 - Tela de escolha das sessões.

A Figura 5.3mostra o ambiente de trabalho do SGPD.

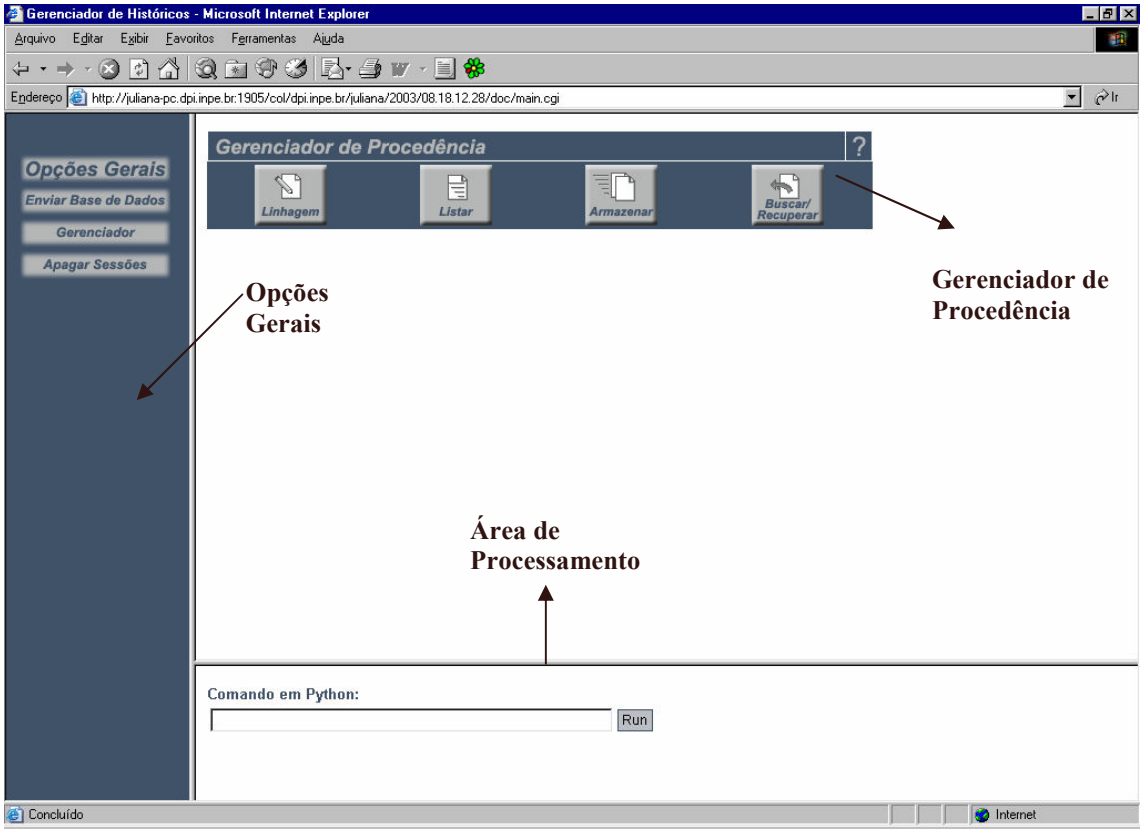


FIGURA 5.3 - Ambiente de Trabalho do SGPD.

Observamos pela Figura 5.3 que à esquerda da tela aparece o menu das funcionalidades gerais do sistema: *Enviar base de dados*, entrar no *Gerenciador* e *Apagar Sessões*. Na porção superior da tela é mostrado o menu das funcionalidades específicas do gerenciador de procedência: mostrar *Linhagem* dos dados processados na Sessão, *Listar* as características dos dados processados, *Armazenar*, *Buscar* e *Recuperar* dados e procedência da UR*Lib*. Na porção inferior da tela aparece o ambiente de processamento de dados monitorada pelo SGPD. Observe o local reservado para entrada das linhas de comando em Python.

As próximas figuras mostram a Interface das funcionalidades do gerenciador de procedência. As figuras estão relacionadas com o exemplo mostrado do Seção **Erro! Fonte de referência não encontrada.**.

A Figura 5.4 exibe a execução do comando *Linhagem* após ter entrado as seqüências de processamento de a1, a2, a3 e a4 na linha de comando em Python . Neste caso, os dados processados e suas respectivas linhagens são mostrados na tela. No exemplo, os dados processados são: a1, a2, a3 e a4. Observe que a linhagem resumida contém apenas os nomes curtos e a linhagem completa contém os nomes reservados.

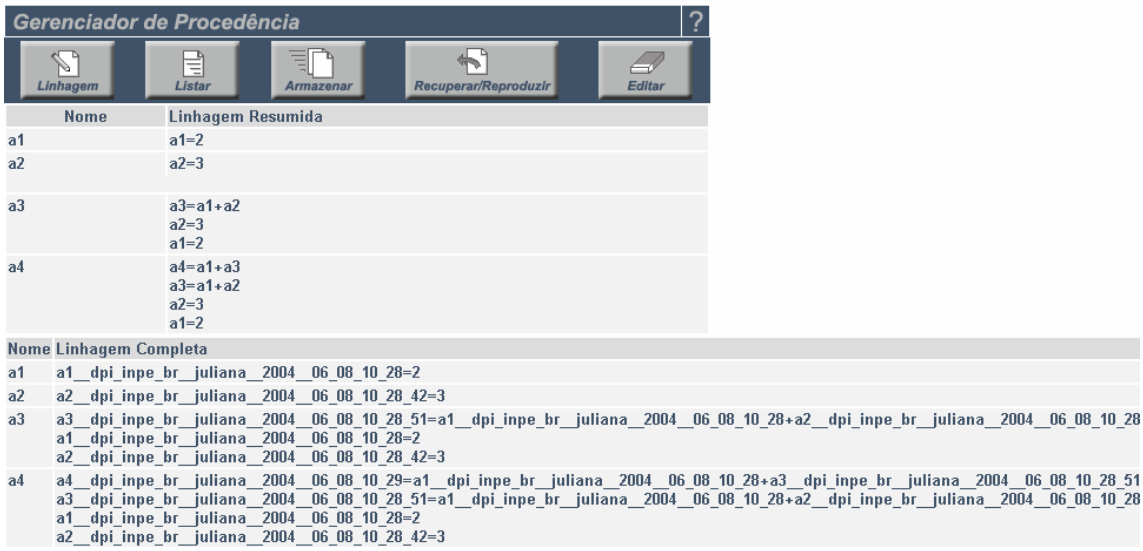
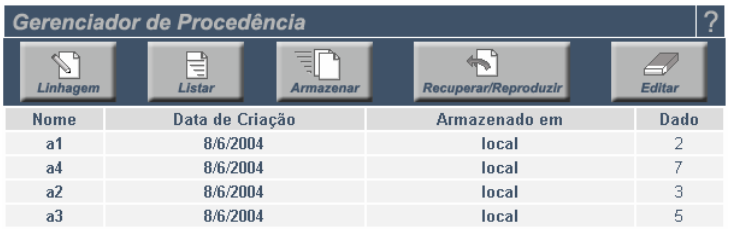


FIGURA 5.4 - Linhagem dos dados processados.

A Figura 5.5 mostra a execução do comando *Listar*. Neste caso são exibidos o nome e valor dos dados processados na Sessão, a data de processamento dos dados, e o Acervo em que esses foram armazenado na URL*ib*. Observe que no exemplo, os dados estão armazenados localmente pois ainda não foram enviados para UR*Lib*.



Nome	Data de Criação	Armazenado em	Dado
a1	8/6/2004	local	2
a4	8/6/2004	local	7
a2	8/6/2004	local	3
a3	8/6/2004	local	5

FIGURA 5.5 - Listagem do nome, data de criação, local de armazenamento e valor dos dados processados na Sessão.

A Figura 5.6 mostra a execução do comando *Armazenar*. A tela solicita a indicação do nome do dado a ser armazenado e o acervo local no qual este deve ser guardado na UR*Lib*.

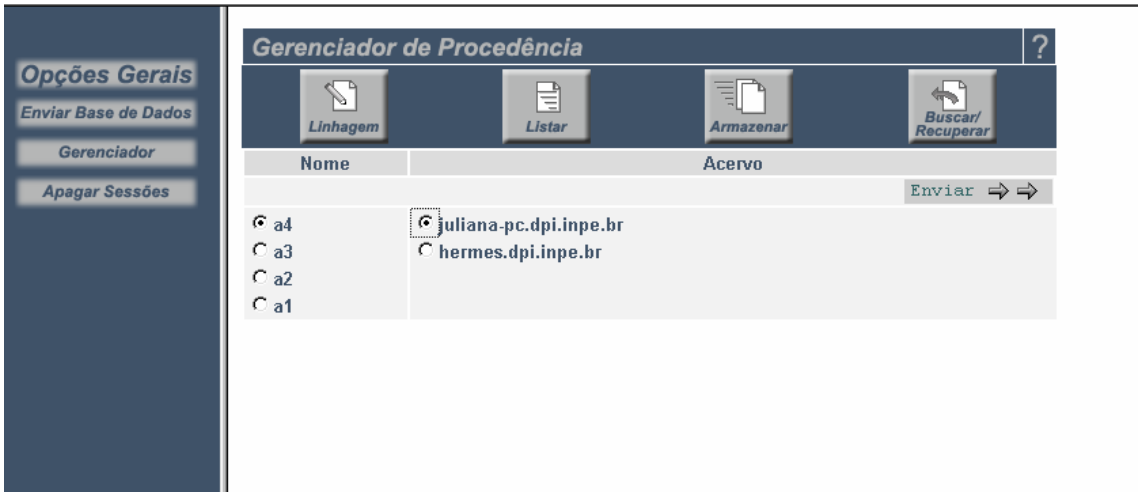


FIGURA 5.6 – Tela para escolha do dado e acervo local para o qual o dado será enviado.

Neste exemplo, indicamos o dado a4 e o acervo juliana-pc.dpi.inpe.br. Após indicação dos dados o comando *Enviar* é executado e a tela mostrada na Figura 5.7 exibida.

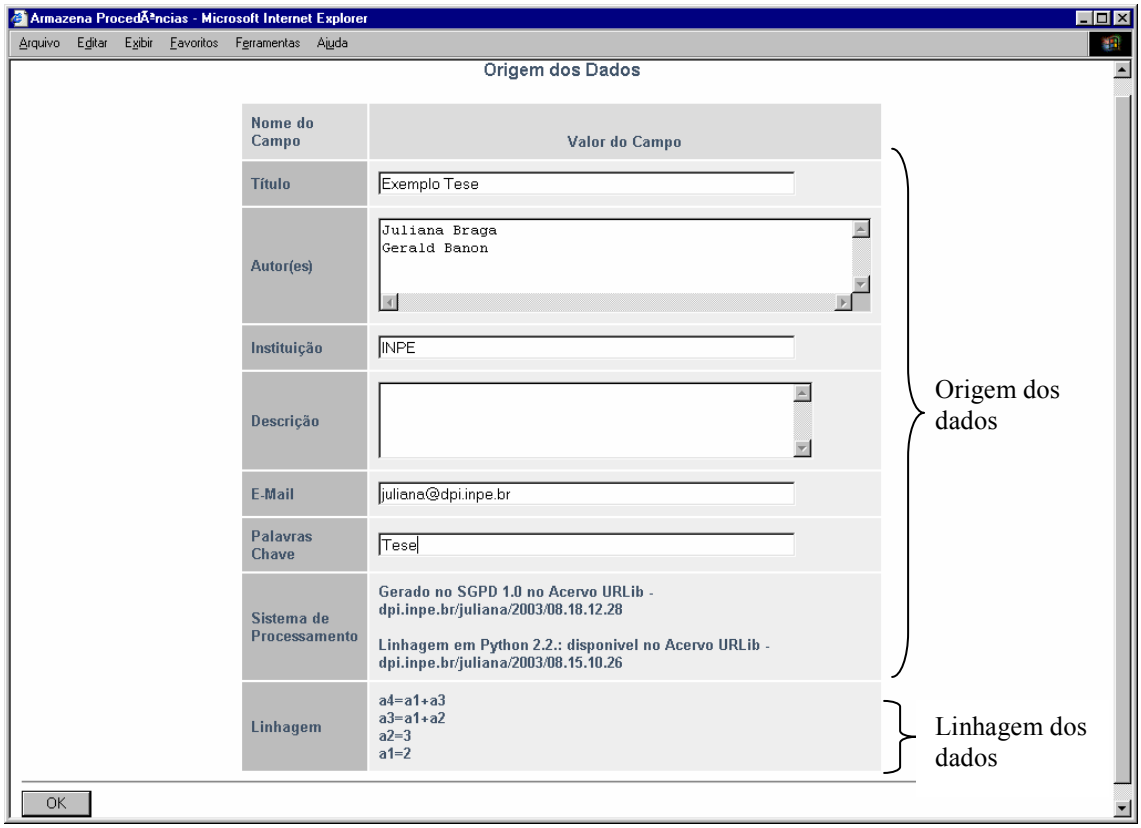


FIGURA 5.7 - Tela para o preenchimento da Origem do dado processado.

A Figura 5.7 mostra a tela de preenchimento da Origem dos Dados a serem armazenados. No exemplo, preenchemos a origem do dado a4. O campo de Linhagem e Sistema de Processamento são criados automaticamente e os outros campos são preenchidos pelo usuário.

Após preenchimento dos campos o usuário clica no botão OK e o SGPD envia os dados para a UR*Lib*. Se a operação for realizada com sucesso, a UR*Lib* envia a mensagem mostrada na Figura 5.8.

Observe que para o usuário final a integração do SGPD com a UR*Lib* é transparente. Já para a implementação do SGPD, essa integração permitiu que o armazenamento de dados e procedência usufrua de todas as vantagens que uma biblioteca digital de acervo distribuído oferece.

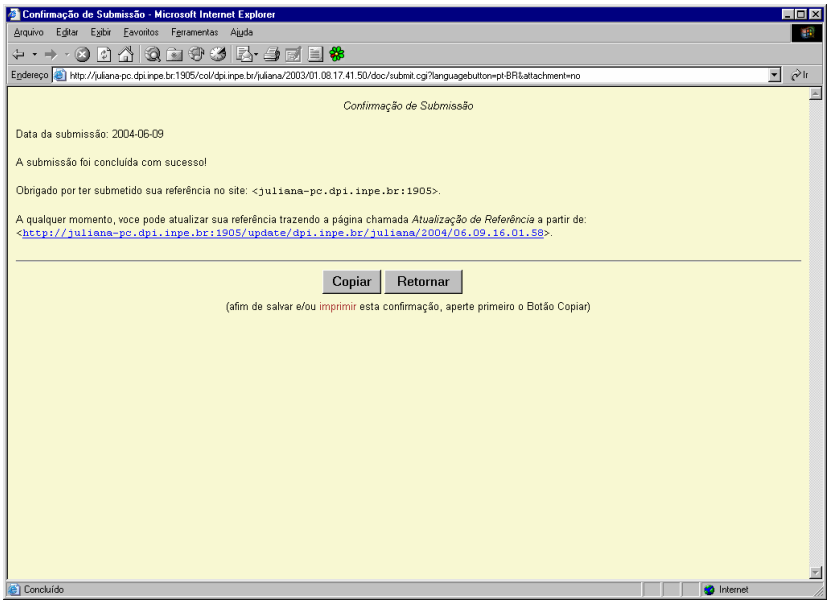


FIGURA 5.8 – Retorno do UR*LiService* para o SGPD após o armazenamento de um dado na mesma.

Observe pela FIGURA 5.8 que o UR*LibService* retorna o nome do repositório o qual o dado foi armazenado e um "link" para o mesmo.

A Figura 5.9 mostra a execução do comando *Buscar/Recuperar*. Observe as opções para buscar um dado e/ou procedência na UR*Lib*. Neste caso, escolhemos a opção "Linhagem" e entramos com a palavra chave "a4". Após a entrada das opções de busca, a UR*LibService* retorna o resultado para o SGPD (Figura 5.10).

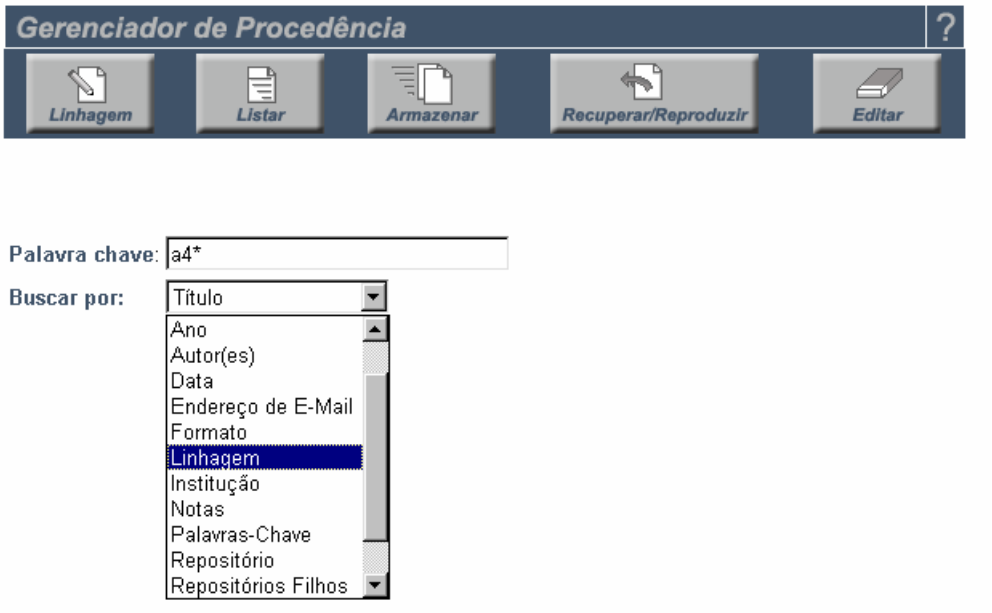


FIGURA 5.9 - Busca de dados na UR*Lib*.

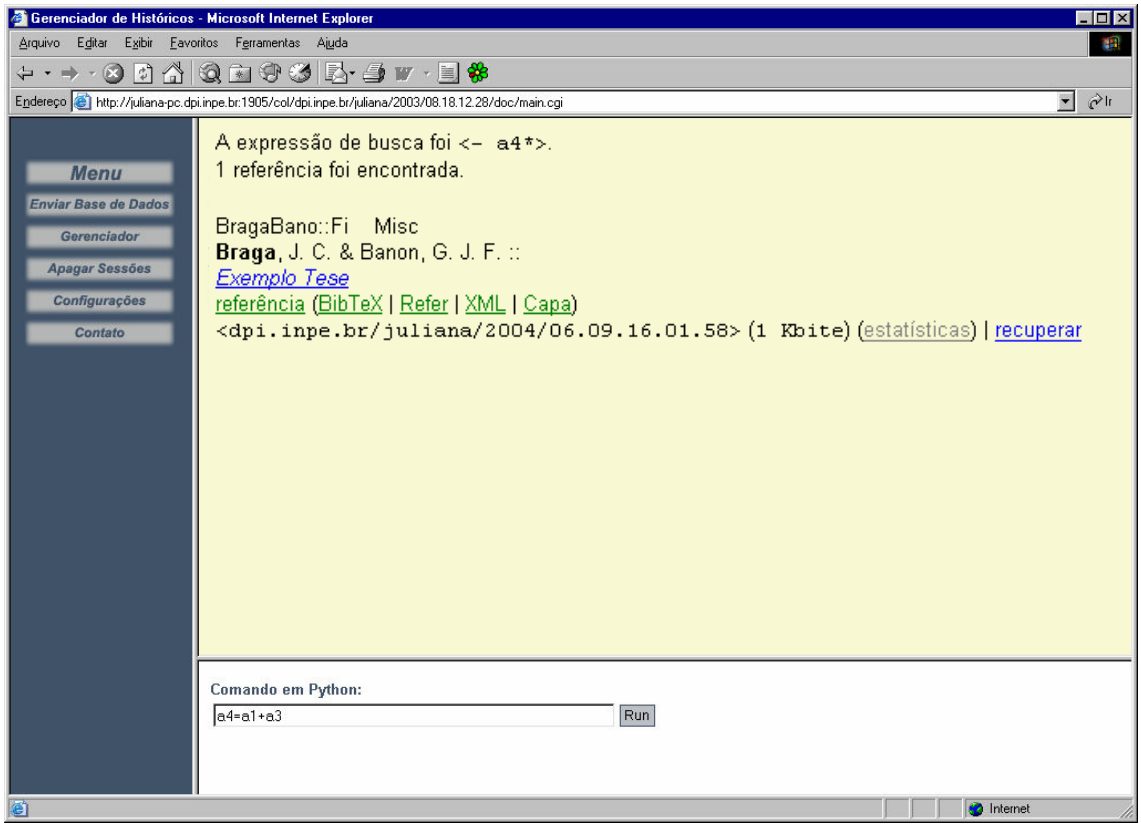


FIGURA 5.10 – Retorno da UR*Lib* para o SGPD após realização de uma busca.

A Figura 5.10 mostra algumas opções de metadados em que a procedência de a4 aparece (XML, BibText, Refer). A Figura 5.10 também mostra o link "Recuperar" . Este link permite a recuperação do dado através de sua linhagem.

A Figura 5.11 mostra a tela gerada a partir do "link" *Recuperar*. Observe que através da linhagem de a4 podemos recuperar os dados originais a1 e a2, o dado resultante a4 e/ou o dado intermediário a3. Ressaltamos que o dado intermediário a3 não se encontra na URL*ib* mas assim mesmo pode ser recuperado através da linhagem.

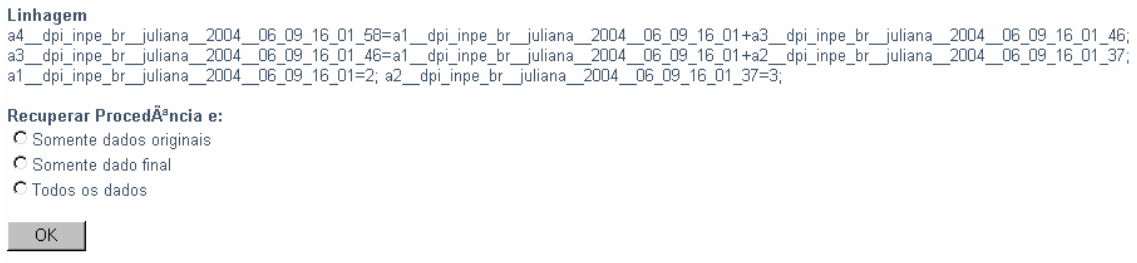


FIGURA 5.11 - Escolha do tipo de recuperação/reprodução de dados.

Para ilustrar o potencial que os nomes reservados fornecem ao SGPD, fechamos a Sessão anterior e abrimos uma nova Sessão. Criamos nessa Sessão um novo dado a partir da seqüência de processamento 3 + 3 e demos novamente o nome a4 para o resultado (Figura 5.12).

Fizemos uma busca na UR*Lib* pelo dado a4 da Sessão do exemplo anterior. Posteriormente, acionamos a opção "Somente dado Final" mostrada na Figura 5.10. Essa opção reproduz para a Sessão atual o dado a4 sem problemas de ambigüidade de nomes.

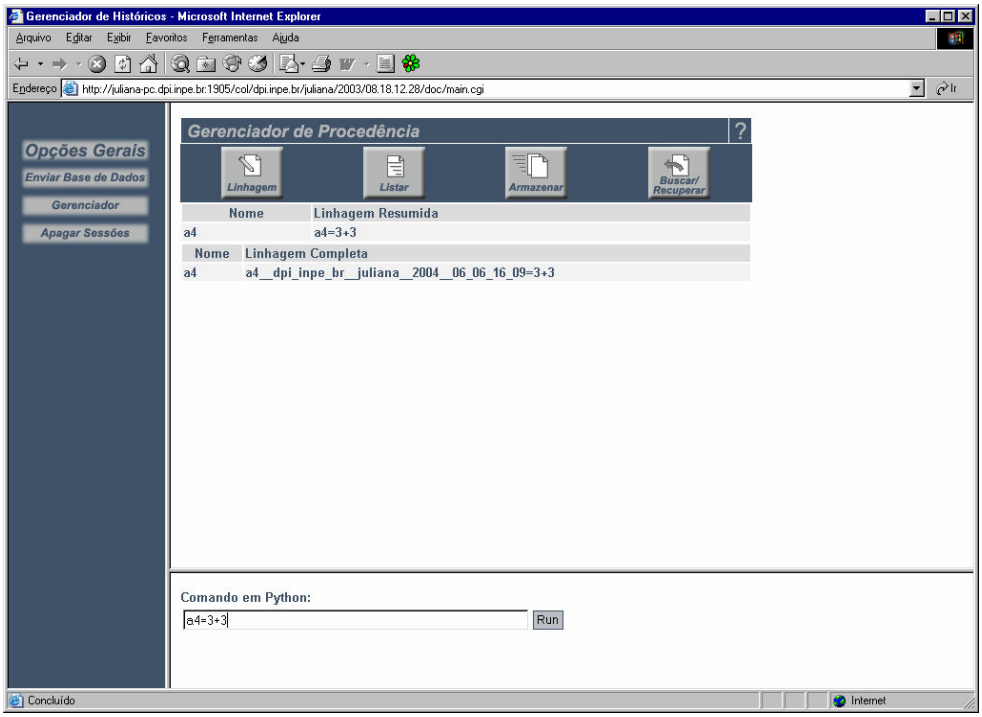


FIGURA 5.12 - Nova Sessão contendo o dado a4 que possui valor 6.

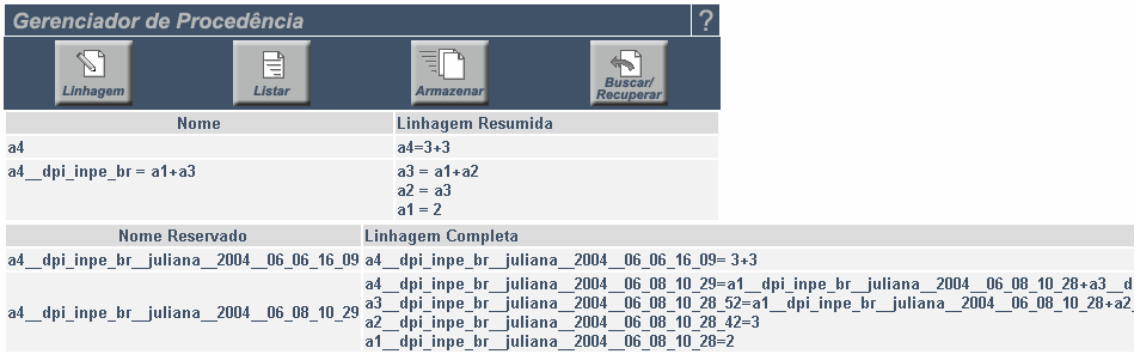


FIGURA 5.13 - Reprodução do dado a4 através de sua procedência sem ambigüidade de nomes.

Observamos na Figura 5.13 que o primeiro dado a4 é reproduzido pelo SGPD, porém para a nova Sessão o nome curto para a4 passa a ser a4__dpi_inpe_br. Contudo, seu nome reservado continua sendo a4__dpi_inpe_br__juliana__2004__06_08_10_29. Essa mudança de nome curto ocorreu para que não houvesse duplicação de nomes curtos. A mudança não afeta no gerenciamento da procedência. Para dar continuidade ao processamento, devemos utilizar o nome curto a4__dpi para referenciar o dado de valor 8 e a4 para referenciar o dado de valor 6.

5.2 Aplicação ao Processamento de Imagens

Para testar o uso do SGPD em processamentos de imagens realizamos um experimento nesta área. O experimento é dividido em duas etapas. Na primeira etapa o SGPD monitora um processamento de imagens e a partir dele cria e armazena a procedência dos dados gerados no mesmo. Na segunda etapa o SGPD busca, recupera e reproduz dados a partir da procedência armazenada na UR*Lib*.

Na primeira etapa do experimento refizemos o processamento de imagens realizado no artigo "Restauração de imagens NOAA por Morfologia Matemática" apresentado no VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens (Banon e Candeias 1993). O artigo utiliza a Morfologia Matemática para reduzir o efeito de listras em imagens NOAA.

A Figura 5.14 mostra a imagem original com uma listra horizontal. Este defeito ocorreu devido uma falha no sensor no momento da captura da imagem.

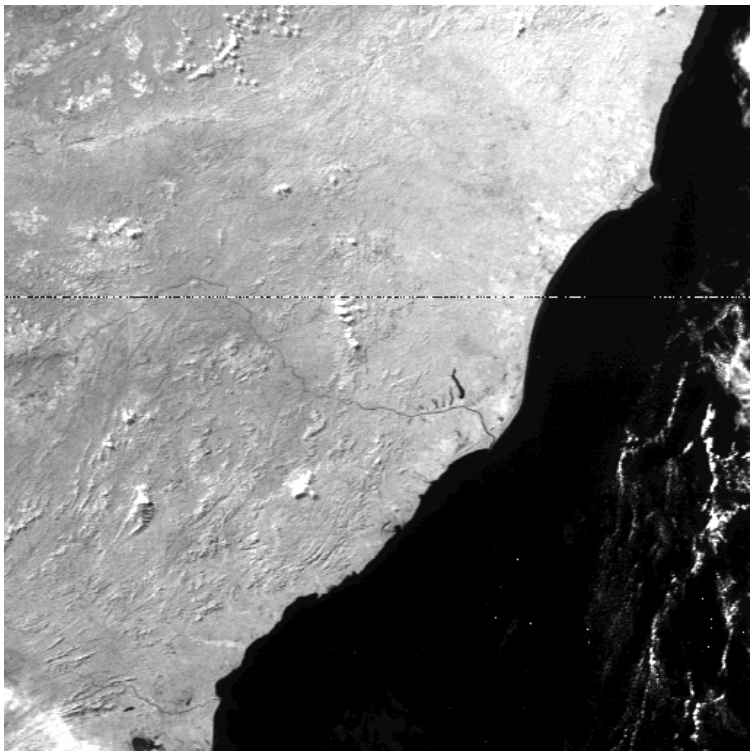


FIGURA 5.14 - Imagem NOAA original (com um defeito na forma de uma listra horizontal).

Para reproduzir o processamento relatado no artigo em estudo, utilizamos o SGPD e o módulo de morfologia matemática chamado morph. O morph é um módulo integrante de uma Caixa de Ferramentas de Morfologia Matemática (Barrera et al, 1998). A versão da Caixa de Ferramentas que utilizamos está escrita na linguagem Python (SDC, Morfology toolbox).

Iniciamos uma nova Sessão no SGPD e enviamos a imagem original (Figura 5.14) para a área de trabalho (FIGURA 5.15). No ambiente de trabalho realizamos o processamento descrito no artigo. No primeiro estágio do processamento, os pixels corrompidos são localizados e, no segundo estágio, seus valores são interpolados a partir dos valores dos pixels dos vizinhos não corrompidos.

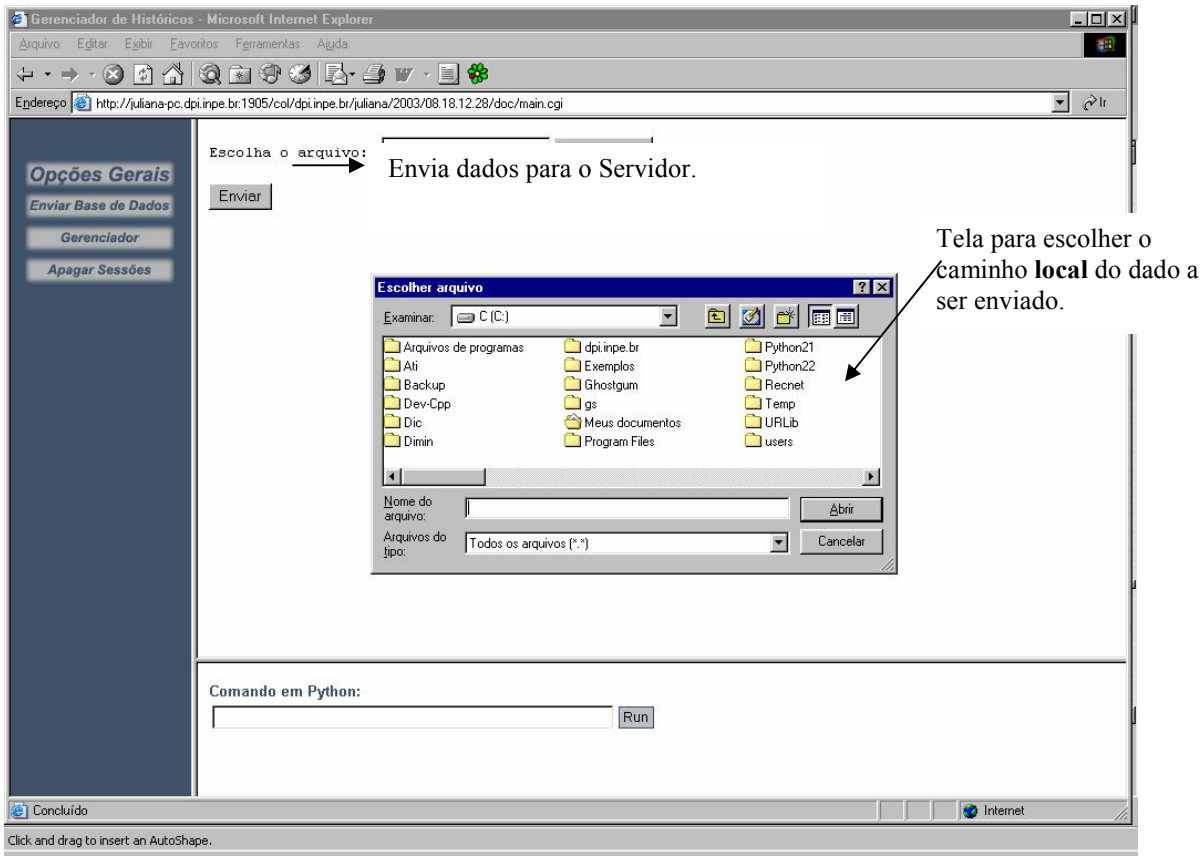


FIGURA 5.15 - Tela para enviar dados que estão no servidor para o cliente.

Os comandos em Python para a realização do processamento estão listados a seguir.

```
import adpil
import morph
```

```
f1=adpil.adread('f02.gif')
f2=morph.mmclose(f1,morph.mmsesum(morph.mmimg2se(morph.mmbinary([[0,0,0],[
1,1,1],[0,0,0]])),30))
f2c=morph.mmero(f2,morph.mmimg2se(morph.mmbinary([[0,0,0],[0,0,0],[0,1,0]])))
f2cc=morph.mmcmp(f2c,'~=',f2)
f2b=morph.mmero(f2,morph.mmimg2se(morph.mmbinary([[0,1,0],[0,0,0],[0,0,0]])))
f2bb=morph.mmcmp(f2b,'~=',f2)
f2a=morph.uint8(morph.mmdil(f2,morph.mmimg2se(morph.mmbinary([[0,1,0],[0,1,0],[
0,1,0]]))))
f2aa=morph.mmcmp(f2a,'==',f2)
f3=morph.mmintersec(f2aa,f2bb,f2cc)
f4a=morph.mmopen(f3,morph.mmsesum(morph.mmimg2se(morph.mmbinary([[0,0,0],[
1,1,1],[0,0,0]])),150))
f4b=f4a*255
f4=morph.uint8(f4b)
f5c=morph.mmero(f1,morph.mmimg2se(morph.mmbinary([[0,0,0],[0,1,0],[0,1,0]])))
f5b=morph.mmero(f1,morph.mmimg2se(morph.mmbinary([[0,1,0],[0,0,0],[0,1,0]])))
f5a=morph.mmero(f1,morph.mmimg2se(morph.mmbinary([[0,1,0],[0,1,0],[0,0,0]])))
f5=morph.mmunion(f5a,f5b,f5c)
f6=morph.mmunion(morph.mmintersec(f1,morph.mmneg(f4)),morph.mmintersec(f5))
```

Observe pelos comandos que o nome curto da imagem original é f1 e o da imagem resultante é f6. Os comandos mostram que foram geradas algumas imagens intermediárias (f2, f3, f4 etc). Essas imagens podem ser visualizadas pela execução do comando *Listar* (Figura 5.16).

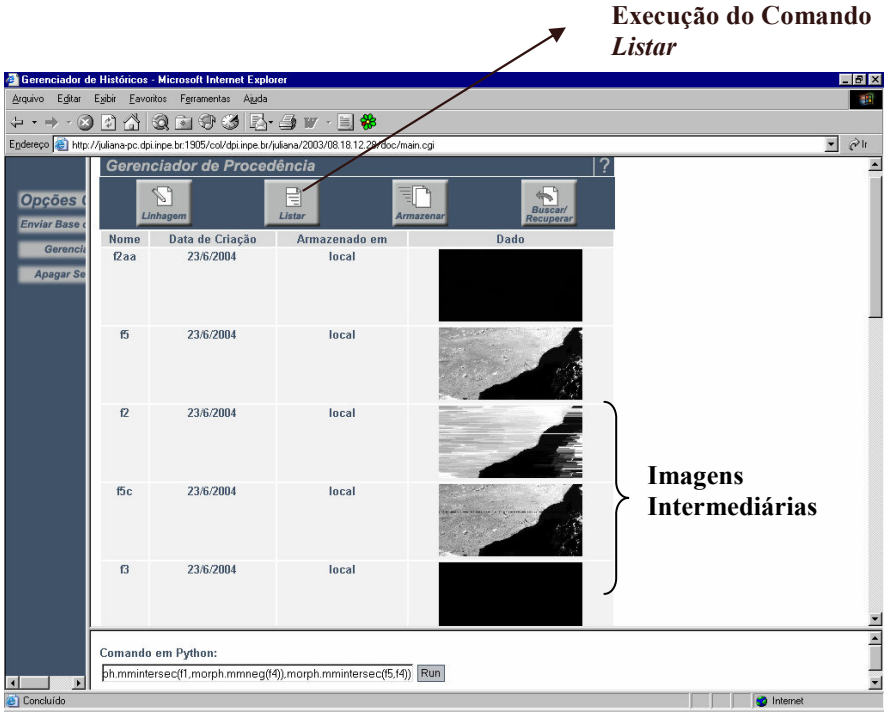


FIGURA 5.16 - Imagens intermediárias visualizadas pelo comando *Listar*.

A imagem Resultante pode ser visualizada na Figura 5.17. Nela observamos a eliminação da linha defeituosa sem alteração do defeito da imagem.

Enviamos a imagem resultante para a UR*Lib*, sendo assim a imagem original também foi armazenada automaticamente.

O armazenamento de f6 e sua procedência permite que os processamentos realizados sejam compartilhados com toda a comunidade científica evitando uma possível repetição de esforços e facilitando a continuidade desse trabalho.

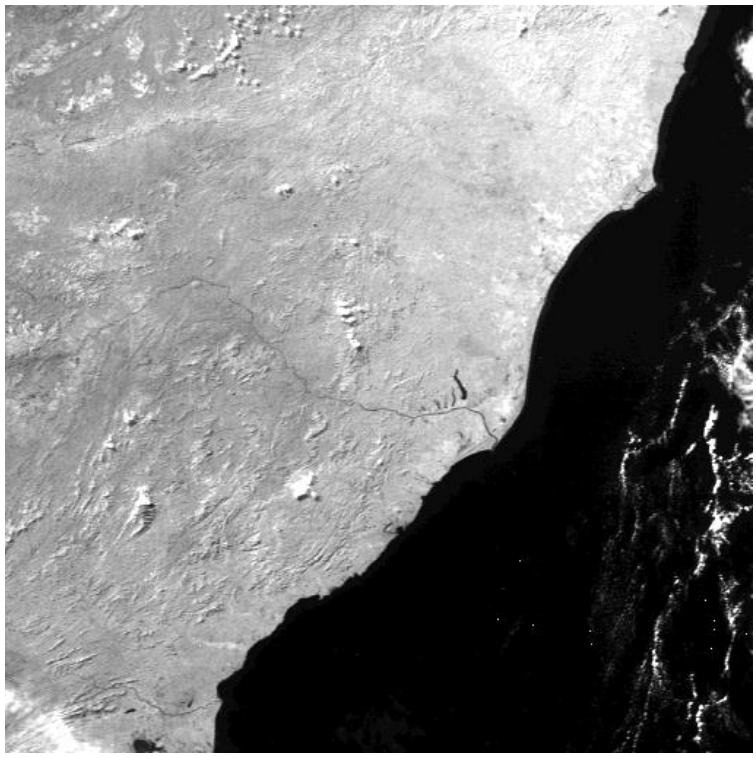


FIGURA 5.17 - Imagem Resultante.

Deste ponto em diante vamos mostrar os resultados provenientes da segunda etapa do experimento. Para isso, fechamos a Sessão atual e abrimos uma nova Sessão a qual não existe nenhuma imagem. Essa Sessão pode ser aberta por outro usuário que não seja o responsável pelo processamento de f6.

O objetivo desta etapa é recuperar as imagens geradas na primeira etapa. Para isso, procuramos a imagem resultante na UR*Lib* executando o comando de busca. A busca é feita pelo campo Título e a palavra chave é NOAA.

A UR*Lib* retornou com sucesso o resultado da busca. A partir dele escolhemos a opção "Recuperar todos os dados". Essa opção trouxe para a Sessão atual não só a imagem original e final como também as imagens intermediárias (Figura 5.18).

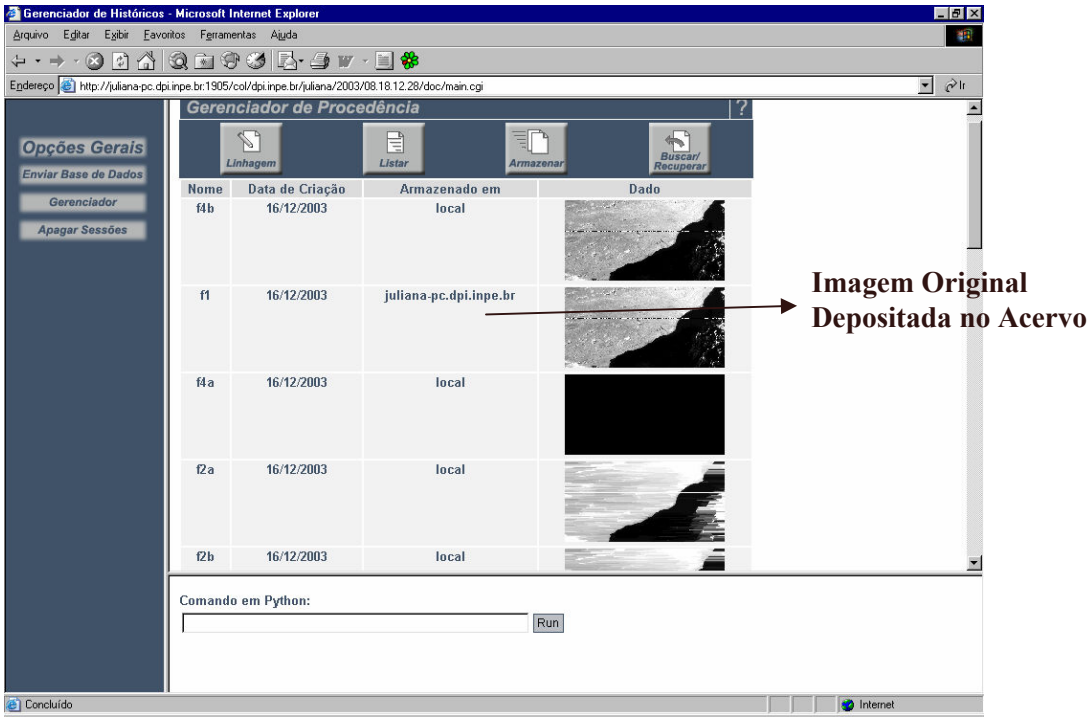


FIGURA 5.18 - Imagens reproduzidas a partir da procedência da imagem resultante armazenada na UR*Lib*.

Observe pela Figura 5.18 que a imagem original f1 está armazenada em um acervo da UR*Lib*, porém as imagens intermediárias estão armazenadas localmente. Significa que as imagens intermediárias não foram trazidas da UR*Lib* e sim reproduzidas em tempo real a partir da procedência da imagem resultante f6.

A publicação da procedência de f6 na UR*Lib* permite que outras pessoas possam reproduzir o processamento realizado nessa imagem utilizando ou não o SGPD. Além disso, o e-mail do autor do processamento está anexado a procedência de f6 facilitando a troca de informações.

A FIGURA 5.19 mostra a árvore de linhagem gerada para o processamento da imagem NOAA.

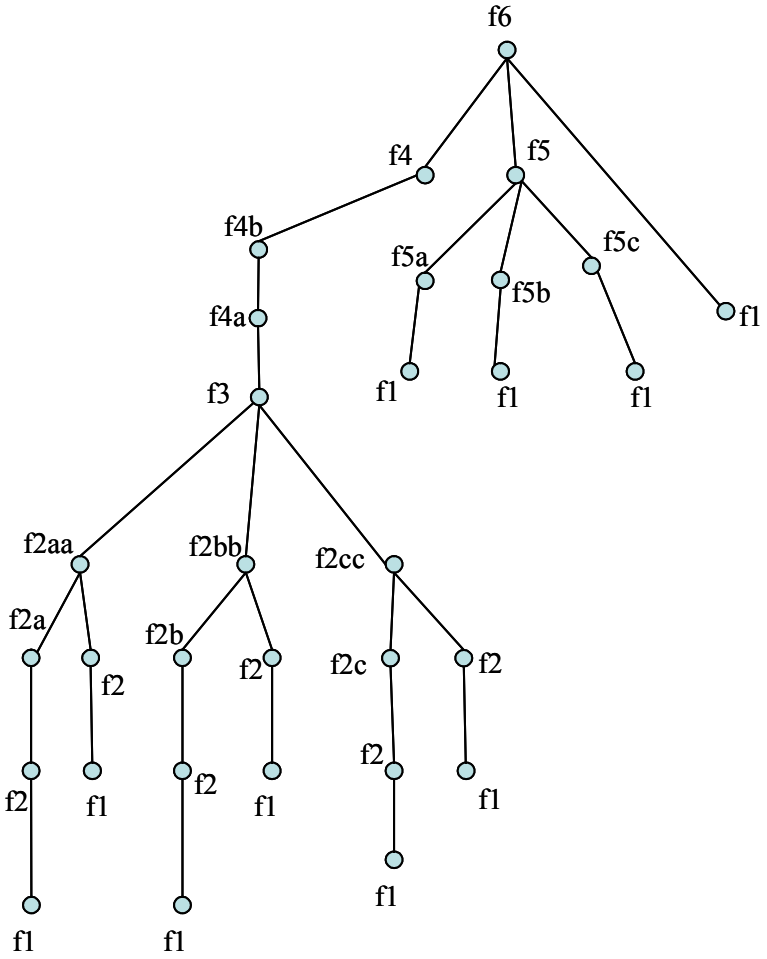


FIGURA 5.19- Árvore de linhagem gerada no processamento da imagem NOAA.

CAPÍTULO 6

CONCLUSÕES, CONTRIBUIÇÕES E PERSPECTIVAS FUTURAS

Propomos neste trabalho um modelo, uma estrutura e um tipo de armazenamento para a procedência de dados processados. Baseado nessa proposição, implementamos um protótipo para gerenciar a procedência o qual chamamos de SGPD (Sistema Gerenciador de Procedência de Dados). O SGPD utiliza a UR*Lib* para armazenar dados e sua procedência. Testamos o protótipo em processamentos de imagens.

O modelo de procedência proposto possui dois aspectos: Origem e Linhagem dos dados. A origem contém informações relativas ao processamento sendo elas: nome e email do autor, data, nome e versão do software usado, descrição em alto nível e linguagem de programação utilizada. Por conter essas informações a origem permite: contato com o autor do processamento, identificar a versão do processamento (data), entender como e porque o processamento foi realizado (descrição), reproduzir o processamento utilizando o mesmo software e versões indicados. A linhagem contém as funções e parâmetros utilizados no processamento, e por isso permite a reprodução automática de dados.

Devido a sua importância, a linhagem foi o enfoque dessa pesquisa. Introduzimos neste trabalho alguns novos conceitos, dentre eles os principais são: *árvore de linhagem*, *linhagem para um dado* e *nome reservado para um dado*. Tais conceitos auxiliaram no entendimento da linhagem e orientaram a implementação do SGPD. Dentre outros fatores, *a árvore de linhagem* e a *linhagem para um dado* permitiram que o SGPD identificasse a ordem das funções e parâmetros aplicadas a um dado e também os dados originais de um processamento. Com isso, o SGPD reproduziu um dado a partir de sua linhagem. O nome *reservado para um dado* garante que dois dados diferentes possuam nomes diferentes. Este fator aumentou o potencial do SGPD, uma vez que podemos reproduzir para nossa Sessão de trabalho qualquer dado armazenado na UR*Lib,* independente do autor e sem que haja conflitos de nomes.

Especificamente a procedência é um metadado escrita em XML seguindo o Esquema Dublin Core adaptado (15 elementos mais o elemento linhagem). Pela sua característica interoperável a XML permitiu a troca de informações entre o SGPD e UR*Lib* . E pelo mesmo motivo permite que qualquer outro sistema, que tenha suporte ao XML, também extraia facilmente as informações da procedência de dados.

A procedência é um arquivo associado ao dado a que ela se refere. Para garantir sua preservação, manter a integridade referencial entre procedência e dado e preservar seus direitos autorais a procedência foi armazenada no acervo distribuído da biblioteca digital UR*Lib*. Por estar na UR*Lib* a procedência foi facilmente publicada, encontrada e recuperada. Por estar armazenada em acervo distribuído o espaço de armazenamento não é um fator limitante do nosso projeto. O UR*LibService* permitiu a busca pela procedência tanto pela sua origem como por sua linhagem. O mesmo mostrou ser um sistema de fácil instalação e comunicação, além de flexível as necessidades do SGPD.

O SGPD foi desenvolvido em Python. O Python demostrou ser de fácil acesso e instalação e permitiu rápido desenvolvimento. Utilizamos o suporte técnico do Python que foi muito eficiente. O ambiente de processamento de imagens utilizado, Morph, atendeu muito bem aos testes realizados em processamento de imagens. Esse ambiente também foi de fácil instalação e recebemos apoio dos seus desenvolvedores (Silva, 2003). O Morph possui excelente documentação disponível na Web, fato que facilitou o entendimento dos comandos.

O enfoque da tese é o processamento de imagens por isso testamos o SGPD para este fim. O SGPD monitorou o processamento de imagens realizado, criou e armazenou a procedência dessas imagens com sucesso. O SGPD reproduziu imagens a partir de sua linhagem e dos dados originais. Dessa forma, podemos apagar imagens intermediárias e reproduzi-las novamente quando necessário. Mesmo não fazendo uso do SGPD é possível reproduzir uma imagem a partir de sua procedência. Basta fazer uma busca na UR*Lib* encontrar a procedência, consultar a linhagem, fazer o download dos dados originais e refazer o processo em ambientes que suportem a linguagem python. Nada impede que o processamento seja refeito em outra linguagem que não seja o Python,

para isso é suficiente conhecer e interpretar os comandos contidos na linhagem. Neste caso, a documentação dos comandos disponível on-line auxilia muito.

Contudo, o SGPD mostrou a viabilidade de uso de um sistema Gerenciador de Procedência de Dados.

A primeira das quatro contribuições do nosso trabalho foi o desenvolvimento dos fundamentos da procedência de dados. Esses fundamentos são muito importantes pois auxiliam no entendimento da procedência e orientam sua implementação por mecanismos computacionais. Dessa maneira, qualquer outro sistema de processamento de imagens poderá implementar os fundamentos propostos.

A segunda contribuição foi a implementação de um gerenciador de procedência de dados. O SGBD possui documentação e código aberto podendo ser útil na implementação de outros sistemas afins.

A terceira contribuição foi mostrar a viabilidade do uso de bibliotecas digitais num escopo diferente do usual. Geralmente bibliotecas digitais são utilizadas para acessar ou armazenar publicações, livros etc. Em nosso trabalho optamos, com sucesso, em usar a UR*Lib* como um banco de dados para armazenar de forma automatizada imagens e suas procedência. Contudo, outros tipos de aplicações poderão reutilizar essa solução.

A quarta contribuição foi apresentar soluções modernas para uma antiga preocupação em documentação de processamento de dados.

Pretende-se após o término deste trabalho dar continuidade nos seguintes aspectos:

Permitir a alteração consistente da linhagem dos dados para posterior processamento. Isso significa poder editar a linhagem, modificar alguns parâmetros e executar novamente o processamento com a linhagem modificada.

Registrar a linhagem em uma linguagem genérica, como por exemplo XML. Dessa maneira, a linhagem ficaria independente da linguagem de programação. A partir da linhagem em XML e utilização de folhas de estilo, poderíamos traduzir a linhagem para

outras linguagens (c++, pyhton, linguagem próxima a humana etc) e/ou formas de apresentação (texto, html, doc, pdf). Iniciativas a esse respeito, aplicado a morfologia matemática, pode ser vista em (Zampirolli, 2003).

Testar o uso do SGPD para outras aplicações além do processamento de imagens.

Complementar a fração do metadado que define a origem de imagens de acordo com o tipo de aplicação (exemplo: imagens médicas, sensoriamento remoto, microscópios). Sugerimos a utilização de ontologias para definir a estrutura a ser utilizada.

Transformar o SGPD em um sistema colaborativo onde grupos de pesquisa de processamento de imagens em diferentes locais possam compartilhar os resultados de suas pesquisas.

Acrescentar ao modelo de procedência proposto informações a respeito do conteúdo das imagens. Neste ponto o conteúdo pode ser extraído automaticamente abrindo espaço para linhas de pesquisas nessa área. A linhagem poderá também ser utilizada para registrar como esse conteúdo foi extraído.

Vincular os comandos da linhagem a uma ajuda on-line facilitando o entendimento do usuário.

Investigar a utilização de linhagem de imagens para persistência de objetos durante as conexões HTTP.

REFERÊNCIAS BIBLIOGRÁFICAS

Apache HTTP Server Version 1.3. San Jose, CA: The Apache Software Foundation. 2004. Disponível em: http://www.apache.org/. Acesso em: 20 jul. 2004.

Banon, G. J. F. Implementação de um sistema de tratamento de imagens usando uma definição ampla para imagem. In: Simpósio Brasileiro de Sensoriamento remoto, 1984, Rio de Janeiro. **Anais**... São José dos Campos: INPE, 1984.

Banon, G, J, F. Implementação de um sistema de tratamento de imagens usando uma definição ampla de imagens. In: Simpósio de Sensoriamento Remoto, 1991, Rio de Janeiro. **Anais...** São José dos Campos: INPE, 1991. p.28-30.

Banon, G. J. F.; Candeias, A. L. B. Restauração de imagens NOAA por morfologia matemática. In: Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, SIBGRAPI 93, 6., 1993, Recife. **Anais...** [S.1]: Wagner, 1993. p. 139-145.

Banon, G. F. **What is URLib?** Disponível na UR*Lib*: <http://iris.sid.inpe.br:1905/rep-/iconet.com.br/banon/2001/02.10.22.55.05>. Acesso em 27 abr. 2004.

Banon, G.J.F.; Ribeiro, M.L.; Banon, L.C. Preservação digital da memória técnico-científica do INPE. In: Simpósio Internacional de Biblioteca Digital, 2, 2004, Campinas. **Anais...** 2004. Disponível na UR*Lib*: < http://ePrint.sid.inpe.br:80/rep-/dpi.inpe.br/lise/2004/03.02.15.20>. Acesso em 27 abr. 2004.

Barrera, J.; Banon, G. J. F.; Lotufo, R. A.; Hirata Junior, R. MMMach: a mathematical morphlogy toolbox for the KHOROS system. **Journal of Eletronic Imaging**, v. 7, n. 1, p. 174-210, Jan. 1998.

Bose, R. A Conceptual Framework for Composing and Managing Scientific Data Lineage. In: International Conference on Scientific and Statistical Database Management, 14., July, 2002, Edinburgh, Scotland. **Proceedings**... 2002. p. 15-19.

Brown, P.; Stonebraker, M. A System for the Management of Earth Science Data. In: International Conference of Very Large Data Bases, 21., 1995, Zurich, Switzerland. **Proceedings**... 1995. p. 720-728.

Buneman, P.; Khanna, S.; Tan, W. Data Provenance: Some Basic Issues. In: Foundations of Software Technology and Theoretical Computer Science (FST TCS), December 13-15, 2000, New Delhi, India. **Proceedings**... Springer-Verlag, 2000. v. 1974.

Buneman, P.; Khanna, S.; Tan, W. Why and Where: A Characterization of Data Provenance. In: International Conference on Database Theory, 4-6 January, 2001, London, United Kingdom. **Proceedings**... 2001. p. 15. On-line.

Buneman, P.; Khanna, S; Tan, W. Where was your data yesterday, and where will it go tomorrow? Data Annotation and Provenance for Scientific Applications In: NSF Workshop on Information and Data Management (IDM '00), 5-7 March, 2000, Chicago, Illinois. **Proceedings**... 2000. p. 18. On-line.

Garcia, S. S.; Moura, A. M. C.; Campos, M. L. M. **Metadados para documentação e recuperação de imagens**. Rio de Janeiro: Instituto Militar de Engenharia (IME), 1999. 19 p.

Goble, C. Position Statement: Musings on Provenance, Workflow and (Semantic Web) Annotations for Bioinformatics. In: Workshop on Data Derivation and Provenance, 17-18 October, 2002, Chicago, Illinois. **Proceedings**... 2002.

Greenwood, M.; Goble, C.; Stevens, R.; Zhao, J.; Addis, M.; Marvin, D.; Moreau, L.; Oinn, T. Provenance of e-Science Experiments - Experience from Bioinformatics. In: UK OST e-Science second All Hands Meeting 2003. (AHM'03), 2-4 Sept, 2003, Nottingham, UK. **Proceedings**... 2003. p. 4.

Hachem, N.; Gennert, M.; Ward, M. A Spatio-Temporal Database System for Global Change Studies. In: AAAS Workshop on Advances in Data Management for The Scientist and Engineer, February, 1993, Boston, Massachusetts. **Proceedings.**.. 1993. p. 84-89.

Harrison, M. A. **Introduction to formal language theory**. [S.l]: Addison-Wesley Publishing, 1978. 594 p.

Hawryszkiewycz IT. **Introduction to systems analysis and design.** 3.ed. Sydney: Prentice Hall, 1994.

Hodge, G. Metadata Made Simpler.. Bethesda, MD: National Information Standards Organization (NISO), 2001. (ISBN: 1-880124-50-5). Disponível na biblioteca digital URLib: <dpi.inpe.br/banon/2004/04.21.12.47>. Acesso em: 22 abr. 2004.

Holzer, S. **Desvendando XML**. Rio de Janeiro: Campus, 2001. 857 p.

Lundh, F. **Python standart lybrary**. Sebastopol: O'REILLY, 2001. 281 p.

Mann, B. Some Data Derivation and Provenance Issues in Astronomy. In: Workshop on Data Derivation and Provenance, October, 2002, Chicago, IL. **Proceedings**... 2002. p. 17-18.

Mello, R. S. **Aplicação de ontologias a bancos de dados semi-wstruturados**. Porto Alegre: Universiddade Federal de Porto Alegre, 2000. 150 p. (EQ-45 PGCC-UFRGS).

NISO DRAFT STANDARD; **Data dictionary — Technical metadata for digital still images**. Maryland: © NISO and AIIM, 2000. 45 p.

Orfali, R.; Edwards, D. H. **The essential client/server survival guide**. Canadá: Wiley Computer Publishing, 1996. 676 p.

Pancerella, C.; Hewson, J.; Koegler, W.; Leahy, D.; Lee, M.; Rahn, L.; Yang, C. Metadata in the Collaboratory for Multi-Scale Chemical Science. In: DC-2003: the 2003. Dublin Core Conference, 27 September - 2 October, Seattle, Washington, USA. **Proceedings**... 2003.

Rossum G. V; Drake, F. **Python Tutorial release 2.2.1**. Amsterdan: Python Software Foundation, 2002. 82 p.

SDC Morphology Toolbox for Python; Disponível em: http://www.mmorph.com/pymorphpro/. Acesso em: 07 jun. 2004.

SGML; Standard Generalized Markup Language [online]. <www.w3.org/MarkUp/SGML/>. Acesso em: 15 abr. 2004.

Silva, A. G.; Lotufo, R. A. **Ambiente de suporte ao ensino de processamento de imagens usando a linguagem Python**. 2003. 100p. Dissertação (Mestrado em Engenharia Elétrica e de Computação) - Universidade Estadual de Campinas (UNICAMP), Campinas, 2003.

Spery, L.; Claramunt, C.; Libourel, T. A Lineage MetaData Model for the Temporal Management of a Cadastre Application. In: International Workshop on Database and Expert Systems Applications, 10. September 01 - 03, 1999, Florence, Italy. **Proceedings...** [S.l]: IEEE Computer Society, 1999. p. 466-474.

Woodruff, A.; Stonebraker, M. Supporting Fine-Grained Data Lineage in a Database Visualization. In: International Conference on Data Engineering, 13. 7-11 Apr. 1997, Birmingham, UK. **Proceedings**... 1997. p. 15.

Zampirolli, F. A.**Transformada de distância por morfologia matemática**. 2003 . 161p. Tese (Doutorado em Engenharia Elétrica e de Computação) - Universidade Estadual de Campinas (UNICAP), Campinas, 2003.

Zhao, J.; Goble, C.; Greenwood, M.; Wroe, C.; Stevens, R. Annotating, linking and browsing provenance logs for e-Science. In: Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, Oct. 2003, Oxford Road, Manchester. **Proceedings**... 2003. p. 6.

APÊNDICE A

CONEXÃO COM A U*RLIB*

Interface UR*Lib* e SGPD

A Interface entre o SGPD e a UR*Lib* é feita via conexão HTPP ou via socket. Nesta Sessão descrevemos os comandos utilizados nessas conexões.

Conexão HTTP

O SGPD utiliza conexão HTTP para fazer pesquisa e armazenar dados na UR*Lib*. Para realizar essas ações algumas informações devem ser enviadas nessa conexão. A seguir descrevemos essas informações.

Buscar dados na UR*lib*

A string associada ao atributo "ACTION" da tag "FORM" para disparar a busca na UR*Lib* via HTPP é:

http://<dominio>**:**<porta>**/col/**<repositório>**/doc/mirrorsearch.cgi**

em que:

<domínio> nome do domínio do acervo local;

<porta>: porta de acesso ao acervo local;

<repositório>: Repositório do espelho bibliográfico, sendo que espelho bibliográfico é o endereço que indica em quias acervos a busca é disparada.

Um exemplo da string:

http://juliana-pc.dpi.inpe.br:1905/col/dpi.inpe.br/juliana/2003/01.08.17.41.50/doc/mirrorsearch.cgi

Utilizamos o método GET para essa conexão http.

As variáveis que devem ser enviadas junto à conexão HTTP estão descritas na Tabela 0.1.

Nome da variável	Descrição
languagebutton	pt-BR para receber o resultado da busca em português. en para receber o resultado da busca em inglês.
prefixquery	Nome do campo de busca. Podendo receber os seguintes valores: hostcollection, para Acervo Hospedeiro year, para Ano author, para Autor(es) date, para Data e-mailaddress, para Endereço de E-Mail format, para Formato lineage, para Linhagem institution, para Instituição keywords, para Palavras-Chave repository, para Repositório title, para Título

(continua)

	lastupdate, para Ultima Atualização - para Todos os Valores citados anteriormente
query	Valor do campo de busca
sessionId	Número da Sessão atual

TABELA 0.1 - Variáveis relacionadas a busca de dados enviadas do SGPD para a UR*Lib*.

Armazenar dados na* UR*Lib

Utilizamos a conexão HTTP para armazenar o dado resultante na UR*Lib*. Para isso informamos o local em que o dado resultante está armazenado e a UR*Lib* faz o download desse dado.

A string associada ao atributo "ACTION" da tag "FORM" para disparar o download na UR*Lib* via HTPP é:

http://<dominio>**:**<porta>**/col/**<repositório>**/doc/submit.cgi**

em que:

<domínio> nome do domínio do acervo local;

<porta>: porta de acesso ao acervo local;

<repositório>: Repositório do espelho bibliográfico, sendo que espelho bibliográfico é o endereço que indica a partir de quais acervos o download é disparado.

Um exemplo da string é:

http://juliana-pc.dpi.inpe.br:1905/col/dpi.inpe.br/juliana/2003/01.08.17.41.50/doc/submit.cgi

O método utilizado é o GET.

As variáveis que devem ser enviadas junto à conexão HTTP estão descritas na Tabela 0.2.

Nome	Descrição
%0 referencetype	Tipo do metadado. Neste caso o valor é sempre "Misc".
__e-mailaddress_e-mailaddress	E-mail do responsável pelo processamento.
__format_format	Versão do software em que o processamento ocorreu.
__sourcerepositories_sourcerepositories	Linhagens dos dados originais de um outro dado qualquer. A partir dessa linhagem encontro o repositório da imagem original.
_1_lineage	Linhagem do dado.
_3_targetfile	Nome do arquivo alvo no repositório.
_A_author	Nome do autor responsável pelo processaemento.
_E_institution	Nome da instituição.
_K_keywords	Palavras chaves do processamento.
_T_title	

(continua)

	Título do dado processado ou do processamento.
_U_url	Local no cliente onde encontram-se os dados (procedência e dado) para que a UR*Lib* possa fazer o donwload desses dados. *OBS*: os dados originais são enviados via socket.
_X_abstract	Descrição do processamento.
download	O valor deve ser "geturltransfertcopyright". A UR*Lib* faz uma cópia dos dados que estão na URL e indicada e considera que os dados originais passam a ser aqueles armazenados na UR*Lib*.
languagebutton	pt-BR para receber o resultado da busca em português. en para receber o resultado da busca em inglês.
metadatarepository	Nome do repositório em que será armazenada a procedência.
password1	Senha do usuário que está enviado os dados. Essa senha deve estar criptografada.
Repository	(continua)

	Nome do Repositório onde será armazenado o dado.
username	Nome do usuário que está enviado os dados. Este usuário deve estar previamente cadastrado na UR*Lib*.

TABELA 0.2 - Variáveis relacionada ao armazenamento de dados enviadas do SGPD para a UR*Lib* via HTTP.

Ler procedência

urllib.urlopen(path1)

O comando urllib cria uma conexão socket, lê uma URL na UR*Lib* e retorna o valor lido.

Neste caso, a variável path1 é:

http://juliana-pc.dpi.inpe.br:1905/col/dpi.inpe.br/juliana/2003/01.08.17.41.50/doc/mirrorget.cgi?languagebutton=pt-BR&metadatarepository=metadatarepository&index=0&site=site&choice=fullXML.

Observe que a string http contém o nome do repositório de metadados, o nome do site e o tipo de documento que será lido.

Neste caso, o valor de retorno é o arquivo XML contendo a procedência do dado de interesse.

A partir da procedência conseguimos recuperar os dados.

Conexão Sockets

O SGPD utiliza sockts para fazer pesquisa e armazenar dados na UR*Lib*. Os processos que disparam essa conexão são para Pesquisar Repositório, Armazenar Dados Originais e Ler procedência.

Para realizar essas ações algumas informações devem ser enviadas nessa conexão. A seguir descrevemos essas informações.

Pesquisar Repositório

Os comandos utilizados na conexão socket para pesquisar um repositório na UR*Lib* são especificados a seguir.

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

Este comando cria um objeto socket. Os argumentos especificam a família de protocolos a serem utilizado com o socket, isto é, especifica como interpretar endereços quando eles são fornecidos.

s.connect((HOST, PORT))

Este comando estabelece a conexão com a UR*Lib*. O primeiro argumento é o nome do Acervo local e o segundo a sua porta. Exemplo: HOST = juliana-pc.dpi.inpe.br e PORT = 19050

s.send(STRING)

O comando *s.send* envia uma string para a UR*Lib*. A UR*Lib* recebe essa string e a executa. Neste caso, enviamos a string: *"SearchRepository {} {repos nomerep}\n"*

SearchRepository é a função em TCL para buscar um repositório na UR*Lib*. Após o parâmetro *repos* enviamos o nome do repositório que estamos procurando. A string "\n" deve ser obrigatoriamente envida para indicar o final da string.

data = s.recv(1024)

Este comando recebe o retorno da UR*Lib* após disparada a busca. Caso já exista esse repositório a UR*Lib* retorna a string "\r\n". Caso contrário retorna uma string contendo o erro.

Armazenar Dados Originais na URLib

Os comandos para criação e conexão do socket são os mesmos mostrados anteriormente.

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((HOST, PORT))

b = s.send("StoreRepository Rep {} MetaRep {Metadata} UserName Password \n")

O comando *s.send* envia uma string para a UR*Lib*. A UR*Lib* recebe essa string e a executa. Neste caso, enviamos a string: *" StoreRepository Rep {} MetaRep {Metadata} UserName Password \n"*

StoreRepository é a função em TCL que cria um repositório e armazena dados neste repositório. O argumento Rep deve ser o nome do repositório de armazenamento do dado a ser criado, o argumento MetaRep é o nome do repositório de armazenamento do metadadoa a serem criados, Metadata deve ser o metadado escrito em XML, UserName e Password devem ser o nome e senha do usuário responsável pela criação dos repositório. A string "\n" deve ser obrigatoriamente envida para indicar o final da string de envio.

O Metadado em XML deve possuir a seguinte estrutura:

Metadata = <metadata ReferenceType="Misc">

<author>Nome do Usuário</author>

<title>Título do Processamento</title>

<lineage>linhagem do dado</lineage>

<url>URL local em que a UR*Lib* deve fazer o donwload dos dados e da procedência</url>

</metadata>

APÊNDICE B

SOBRE O PYTHON

Python

Python foi desenvolvida no final de 1990, pelo holandes Guido Van Rossun e foi escrita em C ANSI. É uma linguagem genérica, capaz de ser aplicada a uma expressiva quantidade de problemas, superior a de outras linguagens no mesmo nível, como por exemplo, Awk, Perl ou Tcl. E ainda assim, apresenta uma sintaxe bastante atraente pela elegância e simplicidade. Estas características proporcionam, em geral, redução de esforço na manutenção de código, desenvolvimento de aplicações de forma mais rápida que o de linguagens como C, C++ ou Java.

Em Python, o código fonte resultante normalmente é menor por três motivos: os tipos de dados de elevado nível podem traduzir expressões complexas em comandos bastante simples; Python é uma linguagem não declarativa, ou seja, não é necessário unir variáveis ou argumentos antes de utiliza-los e os mesmos podem assumir tipos diferentes conforme o fluxo do programa (tipagem dinâmica); um bloco de comandos, é marcado simplesmente pela indentação, forçando o desenvolvedor a manter um código visualmente limpo. Por estes motivos também, um programa Python se adapta facilmente na descrição de algoritmos por se apresentar, em geral, quase como um pseudocódigo.

O Python oferece estruturas de dados que dão bastante flexibilidade aos usuário, com exemplo citamos as listas e dicionários. Em C, estruturas de dados equivalentes e de mesmo rigor de otimização custaria bom tempo de implementação. Outra característica importante é que, além destas estruturas, existe um grande conjunto nativo de bibliotecas implementadas em C (built-in), para Python, que torna o processo de compilação/correção/recompilação totalmente descartado (a menos que se queira

estender a linguagem com uma API). Com isto, há um ganho em produtividade (Rossum, 2002 citado por Silva 2003).

O Python também possui uma forma eficiente de acesso e reutilização de código com o uso de módulos. Os módulos em Python são coleções de funções. Sua versão 1.5.2, de 1999, já incluía mais de 140 módulos. Muitos módulos que podem ser encontrados a partir do site oficial, são quase todos livres e gratuitos. Como exemplo, citamos os principais módulos usados no desenvolvimento do protótipo (Tabela 0.3). Maiores detalhes sobre os módulos basta acessar a página oficial do Ptyhon.

Módulo	Descrição
cgi	Programação de páginas dinâmicas.
code	Simulação de uma console em Python.
parser	Geração da árvore de derivação da linguagem Python.
re	Busca de texto com expressões regulares.
socket	Estabelecimento de conexões sockets.
string	Operações com strings
symbol e token	Geração da árvore de símbolos em Python e obtenção de suas tokens.
time	Obtenção de hora atual e conversão de formatos de data.
xml.dom.minidom	Interpretação de arquivos em formato XML utilizando DOM.

TABELA 0.3 - Descrição dos principais módulos em Python utiliza.