UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

HENRIQUE GREGIANIN TESTA

Um Modelo Hierárquico para Sistemas de Tipos de Arquivos Implementado como um Middleware

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. José Palazzo Moreira de Oliveira
Orientador

Porto Alegre, agosto de 2007

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Testa, Henrique Gregianin

Um Modelo Hierárquico para Sistemas de Tipos de Arquivos Implementado como um Middleware / Henrique Gregianin Testa – Porto Alegre: PPGC da UFRGS, 2007.

64 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2007. Orientador: José Palazzo.

1. Tipos de arquivo. 2. Middleware. 3. Grafo Hierárquico. I. Oliveira, José Palazzo Moreira. II. Título.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
DOM	Document Object Model
FOAF	Friend of a Friend
FTP	File Transfer Protocol
GTK	The Gimp Toolkit
GUI	Graphical User Interface
HTML	HyperText Markup Language
IANA	Internet Assigned Numbers Authority
IDE	Integrated Development Environment
IETF	The Internet Engineering Task Force
KDE	K Desktop Environment
MIME	Multipurpouse Internet Mail Extensions
N3	Notation3
OWL	Web Ontology Language
RTF	Rich Text Format
RAM	Random Access Memory
RDF	Resource Description Framework
RSS	Really Simple Syndication
SMIL	Synchronized Multimedia Integration Language
SVG	Scalable Vector Graphics
UTI	Universal Type Identifier
W3C	World Wide Web Consortium
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

LISTA DE FIGURAS

RESUMO

Com a finalidade de facilitar a manipulação de documentos pelos usuários e aplicações, foram desenvolvidas no decorrer do tempo diversas formas para classificar e organizar esses documentos, incluindo a criação de diferentes mecanismos pertinentes a formatos de arquivos, os chamados *sistemas de tipos de arquivos*. É possível perceber nesse esforço uma crescente preocupação com a correta identificação dos arquivos e com a obtenção de maiores informações sobre seu conteúdo. Afinal, com esse conhecimento as aplicações e ambientes gráficos dos sistemas operacionais podem adicionar novas funcionalidades na manipulação de documentos e arquivos em geral.

Neste trabalho se propõe um novo modelo e arquitetura para sistemas de tipos de arquivos. O modelo se baseia em um grafo de tipos representando relações hierárquicas e um índice de metadados de arquivos, usados em conjunto por funcionalidades que são disponibilizadas através de uma API. A construção do grafo de tipos, que desempenha papel central no modelo proposto, envolveu o estudo dos tipos de arquivos registrados nos principais sistemas operacionais, os tipos definidos pelo IANA (*Internet Assigned Numbers Authority*) e outros tipos comumente utilizados, com a finalidade de selecioná-los e interrelacioná-los em uma hierarquia.

A arquitetura proposta para o sistema de tipos é um *middleware* que oferece os serviços abstraindo operações de baixo nível específicos do sistema operacional. O modelo e arquitetura apresentados neste trabalho beneficia principalmente *softwares* que trabalham intensamente com arquivos, como IDEs (*Integrated Development Environment*) e ambientes gráficos.

Palavras-chave: Tipos de Arquivo, Middleware, Gráfico Hierárquico

A Hierarchical Model for File Type Systems Implemented as a Middleware

ABSTRACT

Several strategies to classify and organize documents by users and applications has being developed in order to facilitate its manipulation, including creation of different mechanisms related to file formats, known as *file type systems*. It is possible to notice in these works an increasing concern about file identification correctness and obtaining further information regarding its contents as well. This knowledge allow application and graphical environments of operating systems adding new functionalities on manipulation of documents and files.

The present work proposes a new model and architecture to file type systems. The model is based on a file type graph representing hierarchical relationships and an index of file metadata. Both are used by functionalities provided through an API. To build the graph of file types, central piece on the proposed model, it was necessary to study file types registered on the most imporant operating systems, types defined by IANA and other types commonly used. The goal is to collect file types and interlink them in a hierarchy.

The proposed architecture for the file type system is a middleware which provides services hiding native operating systems low level operations. The model and architecture presented in this work is specially useful to softwares that handle files intensely, like IDEs (Integrated Development Environment) and graphical environments.

Keywords: Filetypes, Middleware, Hierarchical Graph

1 INTRODUÇÃO

Desde o surgimento das primeiras *Graphic User Interfaces* (GUIs), a forma como os usuários organizam e buscam seus documentos permanece essencialmente a mesma. Documentos e pastas (mapeadas respectivamente para arquivos e diretórios do sistema) são manipulados com o auxílio de janelas e ícones, e são visualizados pelo usuário seguindo a mesma estrutura da árvore de diretórios do sistema de arquivos (REIMER, 2005). De fato, os gerenciadores de arquivos dos principais sistemas operacionais apenas percorrem a árvore de diretórios e os exibem de forma gráfica. Apesar desse método tornar mais fáceis e intuitivas operações com arquivos em relação aos comandos de texto de *shells*, a organização dos recursos continua sendo limitada por estruturas de baixo nível, provocando uma dependência da visualização dos documentos pelo usuário final com a implementação do sistema de arquivos.

Métodos alternativos de organização de documentos requerem a abstração da estrutura física, e inevitavelmente incorrem na necessidade da utilização de índices para mapeamento entre recursos (TANENBAUM, 2001). Com o aumento do número de documentos do usuário, principalmente arquivos multimídia, como imagens de câmeras fotográficas digitais e músicas provenientes de programas de compartilhamento, mas também documentos em geral, numerosos devido à grande capacidade de armazenamento dos computadores atuais, a rápida localização dos recursos se faz cada vez mais necessária. Novos métodos de organização, portanto, ganham importância nesse contexto, e podem auxiliar significativamente na tarefa de classificação automática e busca por documentos.

Este trabalho apresenta uma hierarquia entre tipos de arquivo, um artefato teórico que estende ou substitui os atuais sistemas de tipos de arquivo e habilita uma série de serviços que podem ser facilmente incorporados às aplicações que manipulam documentos. A arquitetura sugerida para a implementação é baseada em um *middleware*, atuando como intermediário entre o sistema de arquivos do sistema operacional e as aplicações.

Tirando proveito da hierarquia de tipos de arquivo, o *middleware* disponibiliza um serviço de classificação de arquivos por tipos baseado em *tags* adaptadas para uso em sistemas *desktop*. As *tags* constituem uma forma bastante popular de marcar recursos na Web, como imagens e artigos de *blogs*, mas são praticamente inexistentes em *desktops*. Apesar de teoricamente diferentes, classificações por *tags* e por tipos de arquivo compartilham grandes semelhanças estruturais. Tais semelhanças serão exploradas neste trabalho a fim de criar, por meio da hierarquia, novos e importantes métodos de classificação e busca de documentos.

2 CONCEITOS BÁSICOS E ESTADO DA ARTE

Com o passar do tempo foram surgindo novas formas de identificação de documentos e conseqüentemente novos sistemas de tipos. É possível perceber uma evolução em tais sistemas pelo aumento de complexidade e funcionalidades presente nos mesmos. Este capítulo apresenta os conceitos básicos importantes utilizados nesta dissertação e os principais sistemas de tipos que mais influenciaram os ambientes gráficos e aplicações.

2.1 Middleware

Middleware é um termo com significado amplo, sendo utilizado principalmente para designar o *software* que conecta componentes de *software* ou aplicações. O ObjectWeb Consortium (OW2, 2006) define *middleware* como "*a camada de software presente entre o sistema operacional e as aplicações*". Nesse sentido, além de ser um agente integrador, ele conecta camadas de diferentes níveis, abstraindo operações específicas e detalhes de implementação em serviços de alto nível.

Adicionalmente, a menção a *sistema operacional* na definição reforça o caráter de serviço básico, auxiliando aplicativos nas operações que envolvem chamadas ao sistema. Exemplos de softwares freqüentemente referenciados como *middleware* incluem servidores web, sistemas de comunicação orientado a mensagens, sistemas de gerenciamento de conteúdo, etc. (OW2, 2006).

2.2 Sistemas de tipos de arquivo

Convém neste ponto definir os termos *documento* e *arquivo*, importantes para a compreensão da proposta e sutilmente diferentes no significado. O termo *documento* será usado nesta dissertação como sendo um objeto contendo informação e produto de aplicativos, conforme detalhado em (PÈDAUQUE, 2003). E arquivo é um documento que é armazenado em mídia persistente (TANENBAUM, 2001). Portanto, os termos diferem pela maneira como são armazenados. Enquanto um documento pode existir em memória, em disco ou até mesmo em uma rede, um arquivo é armazenado em uma mídia persistente, como em um disco rígido ou um DVD. Quando um navegador é utilizado para visitar um site na Internet, ele recebe pela rede diversos documentos, como páginas HTML, imagens e documentos pdf. Os documentos que inicialmente existiam apenas no servidor, passaram a existir na rede e, por fim, na memória do computador cliente. No

momento em que o usuário do navegador salvar uma página do site em disco, os documentos envolvidos passarão a ser arquivos.

É importante salientar que um documento na memória RAM do computador pode diferir significativamente em sua representação e estrutura de quando está armazenado em disco. Um exemplo são os documentos HTML. Na memória RAM, seguindo a especificação DOM (W3C, 1998) definido pelo W3C (*World Wide Web Consortium*), são estruturados como uma árvore de elementos, mas são armazenados em disco em formato XML (W3C, 1006). Essa diferença ocorre porque as características desejáveis diferem para cada representação: enquanto os documentos em memória devem ser flexíveis e rapidamente manipulados, para os arquivos se busca principalmente clareza e simplicidade, facilitando seu processamento.

Muitos tipos de arquivos não se originaram de aplicativos específicos, mas sim da popularização da sua própria estrutura, como LaTeX, RTF e XML. Outros tiveram sua origem em um aplicativo, como dBASE (conhecido pela extensão `.dbf`) e shapefile, mas posteriormente foram adotados por outros programas, popularizando-os. Existe uma grande quantidade de tipos de arquivos, pois muitos programas criam seu próprio formato, apenas utilizado por ele mesmo; entretanto, a tendência é que aplicativos reaproveitem cada vez mais formatos pré-existentes, com isso favorecendo o intercâmbio de informações entre programas e com serviços on-line.

O projeto Web Semântica (BERNERS-LEE, 2005), idealizado e atualmente coordeado por Tim Berners-Lee, criador da World Wide Web, procura justamente padronizar o formato da informação promovendo a utilização de tecnologias pré-estabelecidas. Baseada nas linguagens XML, RDF e OWL projetadas pelo W3C, a Web Semântica envolve um grande número de subprojetos, alguns deles já bastante populares, como o RSS (WINTER, 2005) e o FOAF(BRICKLEY, Dan; MILLER, Libby; 2005). A hierarquia de tipos de arquivo é ainda mais relevante se considerarmos a estrutura em rede e o reaproveitamento de formatos presentes nas tecnologias da Web Semântica, embora o foco neste documento esteja voltado para plataformas *desktop*.

Vale ressaltar, enfim, que tipos de arquivos diferem de tipos de documentos. Um tipo de arquivo se refere a um grupo de arquivos com o mesmo formato, conforme classificado pelo sistema operacional ou ambiente em questão. Já o conceito de tipo de documento é inevitavelmente subjetivo, atribuído a um grupo de documentos semanticamente relacionados. A definição é genérica porque diferentes aplicativos podem representar um documento em memória de formas variadas. Por exemplo, enquanto uma imagem está sendo construída em um editor gráfico, ela não é facilmente classificável nos tipos de documentos existentes, pois depende da implementação do editor sendo utilizado. No momento em que a imagem é armazenada em disco ou enviada para alguém por e-mail, ela toma a forma de arquivo, possivelmente assumindo um tipo de arquivo pré-existente.

Para determinar o tipo de um arquivo, é preciso que o mesmo seja identificado de forma automática, pois a classificação dos arquivos em tipos tem justamente o propósito

de tornar mais fácil e prática a manipulação de documentos para do usuário. Diversas técnicas são empregadas na identificação de arquivos, variando desde a classificação pelo nome até métodos heurísticos para análise de conteúdo (MCDANIEL, Mason; HEYADRI, M.Houssain; 2003). Algumas dessas técnicas são usadas complementarmente, ou seja, se não for possível identificar o arquivo por um método, outro será utilizado, segundo uma ordem de prioridade.

2.2.1 Extensões de nomes de arquivo

A primeira forma encontrada para identificação de arquivos foi o reconhecimento de um padrão no próprio nome do arquivo, conhecido como extensão de nome de arquivo (*filename extension*), ou simplesmente extensão de arquivo. O caractere ponto (`.`) é adicionado ao final do nome do arquivo e à sua direita anexado o identificador de tipo (TANENBAUM, 2001). Ainda hoje essa é a forma mais empregada pelos principais sistemas operacionais. Exemplos de nomes de arquivos comuns são `contatos.txt`, `dissertacao.doc`, `config.xml`, `data.dbf`, `index.html`, `foto13.jpg`, `fibonacci.java`. A ocorrência mais numerosa de extensões com três caracteres tem origem histórica, mais especificamente em uma limitação do sistema de arquivos integrado ao sistema operacional DOS (*Disk Operating System*), da Microsoft, embora sistemas atuais não sofram dessa limitação. Alguns sistemas operacionais e ambientes gráficos, como o Mac OS X, Gnome e KDE, permitem classificar arquivos não só por meio de sua extensão, mas também pelo seu enquadramento em expressões regulares.

Um dos principais problemas das extensões de nomes de arquivos é que elas não são únicas. Uma mesma extensão pode ter mais de um significado e formatos diferentes provenientes de vários aplicativos. Mesmo a extensão `.doc`, mais conhecida por indicar que o arquivo foi criado pelo aplicativo Microsoft Word, possui outros aplicativos que a utilizam, como o FrameMaker, WordStar e WordPerfect. O inverso também ocorre: um formato de arquivo pode ser representado por duas ou mais extensões. É o caso das extensões `.htm` e `.html`, ambas indicando um documento HTML.

A simplicidade das extensões de nomes de arquivos facilita a identificação de tipos por parte das aplicações (é necessário apenas checar a seqüência de caracteres após o ponto contra uma tabela de tipos conhecidos) e até mesmo para os usuários, afinal, o tipo do arquivo está tão visível quanto seu próprio nome. Porém, essa simplicidade existe ao custo de uma série de limitações que conduziram à criação de novos sistemas de tipos de arquivos. Além do problema relativo à unicidade mencionado anteriormente, a identificação de tipos pela extensão do nome do arquivo não é expressiva, ou seja, não é fornecida nenhuma informação além de uma breve seqüência de caracteres que, sem conhecimento prévio, não indica coisa alguma. Outro problema é a diferenciação completa entre os formatos. Entre tipos de arquivo, existe reaproveitamento de formatos e padrões de formatação que não são expressos pela extensão do arquivo, favorecendo a impressão de que cada tipo de arquivo se diferencia completamente dos demais.

2.2.2 OSCodes

O sistema operacional Mac OS, da Apple Computer Inc, emprega outro método para identificar formatos de arquivos. Quando um arquivo é criado, um código, conhecido como OSType, ou FourCC (*Four Character Code*) é armazenado no descritor do arquivo. O OSType é formado por quatro bytes, usualmente caracteres ASCII, e é escolhido de forma a ser facilmente identificado por humanos, contendo seqüências como `TEXT` e `PICT`. Além do identificador de tipo, também é armazenado o código do aplicativo usado para criar o arquivo, chamado de *creator code* (código do criador). Com ele, arquivos de mesmo tipo podem ser associados a aplicativos distintos. Quando o usuário disparar a ação *abrir* sobre um arquivo no Mac OS, o documento será aberto no mesmo aplicativo que o criou. O OSType e o *creator code* são chamandos de OSCodes (SINGH, Amit, 2006).

OSCodes são armazenados em uma base centralizada no sistema operacional, com descrições dos tipos e caminhos para os aplicativos. Com a popularização do Microsoft Windows e a necessidade cada vez maior em manter compatibilidade com esse sistema operacional, o Mac OS X (última versão do sistema operacional da Apple) suporta, além de OSType e *creator code*, identificação de tipo pela extensão do nome do arquivo (SINGH, Amit, 2006), assim como o Windows e o Linux (nos seus principais ambientes gráficos).

Apesar de armazenar a informação a respeito do aplicativo criador no *creator code*, o sistema de identificação de tipos especificamente, através de OSTypes, não se distancia muito das extensões de arquivos em suas qualidades. A expressividade do OSType é comparável ao das extensões de arquivos, e nele também não existe relacionamentos de similaridade entre os tipos.

2.2.3 Tipos de mídia MIME

Mais recentemente, os *tipos de mídia MIME* (*MIME media types* - *Multipurpose Internet Mail Extensions media types*), também chamados de *tipos MIME*, sistema de tipos originalmente criado para ser usado em e-mails, começaram a ser empregados em outros domínios, de forma mais aparente na Web. Sua sintaxe é marcada pela forma hierárquica em dois níveis. O primeiro nível, mais significativo, possui algumas poucas combinações: `application`, `audio`, `example`, `image`, `message`, `model`, `multipart`, `text` e `video`. Já o segundo nível conta com um grande número de opções, mais específicas, que pertencem a uma das categorias de primeiro nível. Uma barra é utilizada para separar o tipo (primeiro nível) do subtipo (segundo nível). Exemplos típicos são `image/jpeg`, `application/msword`, `application/x-gzip`, `text/html`, `audio/x-wav`, `text/plain` (IANA, 2006).

Um dos primeiros principais sistemas operacionais a empregar tipos MIME como um sistema de tipos para *desktop* foi o BeOS, seguido do Linux com os ambientes gráficos KDE e Gnome. Os tipos MIME possuem uma grande vantagem sobre as

extensões de arquivos: são o padrão de tipos na web, possibilitando a utilização de um só sistema para os dois domínios, web e *desktop*. Essa unificação tem reflexos na complexidade das aplicações, que podem então classificar documentos remotos da mesma maneira que documentos locais.

Adicionalmente, os tipos MIME são mais expressivos do que extensões de arquivos, principalmente se considerarmos o grande número de extensões com três caracteres. Os caracteres `application/msword` expressam muito mais do que `.doc`, e `image/svg+xml` é muito informativo para alguém que desconhece o formato do que simplesmente `.svg`.

Infelizmente, porém, os tipos MIME sofrem algumas das limitações existentes nas extensões de nomes de arquivos. Em primeiro lugar, eles não são únicos, ou seja, é possível que um mesmo tipo MIME se refira a dois formatos diferentes. Um conjunto de tipos é padronizado pelo IANA (*Internet Assigned Numbers Authority*), o mesmo órgão responsável pela gestão de domínios na internet. Estes tipos são garantidamente únicos. Mas quando uma aplicação atribui um formato próprio para o documento, a especificação define que este deverá possuir o prefixo `x-` (por exemplo, `application/x-meusubtipo`, ou `x-meutipo/x-meusubtipo`), o que leva a inevitáveis conflitos potencias, afinal, nada garante que dois produtos utilizem caracteres diferentes após o prefixo `x-` (IANA, 2006).

Em segundo lugar, a hierarquia dos tipos MIME é limitada em dois níveis. Segundo o IANA, documentos XML são classificados como sendo do tipo `text/xml`, uma vez que documentos XML são documentos texto. Porém, surge a dúvida de qual seria o tipo MIME de documentos baseados em XML, já que estão envolvidos três níveis de hierarquia. Para contornar a limitação, o operador de soma (+) tem sido empregado no subtipo para os formatos padronizados, adicionando uma terceira informação. Mas, ao contrário do que se poderia esperar, o tipo MIME definido para a linguagem de descrição multimídia SMIL é `application/smil+xml`, em vez de `text/smil+xml`, e o tipo associado à linguagem RDF é `application/rdf+xml`, em vez de `text/rdf+xml`.

A forma não intuitiva dos tipos definidos para documentos baseados XML é uma conseqüência da limitação da estrutura em que os tipos MIME são baseados. Segundo a especificação RFC2046:

> *"O tipo de mídia `application` existe para ser usado para dados discretos que não se enquadram nas outras categorias, e particularmente para dados a serem processados por algum tipo de programa aplicativo."*

Por esta definição, é correto categorizar os tipos SMIL e RDF no tipo `application`. A especificação também informa que o tipo `text` é reservado para documentos legíveis mesmo sem a utilização de ferramentas ou aplicativos, que não é o caso de SMIL e RDF, projetados para serem tratados por computadores.

Segundo essa classificação, porém, uma informação importante foi perdida: que os tipos SMIL e RDF são baseados no formato texto. Sem esse conhecimento, uma série de aplicações é prejudicada. Ambientes gráficos não disponibilizam editores de texto para esses formatos, clientes de FTP não sabem se trata-se de um formato binário ou texto, ferramentas de busca por conteúdo ignoram documentos desses tipos, etc. Para contornar esses problemas, os aplicativos utilizam uma tabela própria, que indica quais formatos são baseados em texto.

Para indicar que o formato SMIL e RDF são baseados no formato texto e, adicionalmente, não são suficientemente legíveis para serem tratados sem uma ferramenta específica, é necessário uma forma de herança múltipla, recurso inexistente nos tipos MIME. Os problemas com herança existentes nesse sistema influenciam na criação de novos tipos. Na especificação da linguagem OWL, criada pelo W3C (W3C, 2004), seção 2.3 consta que:

> *"O Web Ontology Working Group não requisitou um tipo MIME separado para documentos OWL. Em vez disso, recomendamos o tipo MIME requisitado pelo RDF Core Working Group, chamado* `application/rdf+xml`*, ou, alternativamente, o tipo MIME do XML,* `application/xml`*."*

Essa opção, porém, possui o custo de que as aplicações não terão conhecimento se determinado documento está no formato OWL, apenas que se trata de um documento RDF, exceto se a própria aplicação implementar um mecanismo próprio de identificação.

2.2.4 Uniform Type Identifiers (UTI)

Recentemente, em 2005, a Apple introduziu um novo sistema de tipos de arquivo na versão mais recente do sistema operacional MacOS X, chamado de *Uniform Type Identifiers*. Procurando solucionar os problemas já conhecidos relacionados a sistemas de tipos de arquivo, os UTIs foram projetados do zero, e resultaram em um sistema de tipos com características modernas, muito superior em recursos que os demais (APPLE, 2006).

A sintaxe de um UTI é baseada em um nome de domínio invertido (*reverse-DNS*), inspirada na hierarquia de classes e organização de pacotes da linguagem Java. Exemplificando, o tipo de arquivo identificando um vídeo do aplicativo Quicktime é `com.apple.quicktime.movie` (o domínio `apple.com` é invertido). As palavras são separadas por ponto (.), e quanto mais a direita se encontram, mais específicas são. Alguns outros exemplos de UTIs são apresentados abaixo:

```
com.apple.quicktime.movie
com.mycompany.myapp.myspecialfiletype
public.html
com.apple.pict
public.jpeg
```

Empresas e desenvolvedores são encorajados a utilizar a convenção de domínios invertidos para designar seus próprios tipos de arquivo. Para identificar tipos comuns ou padronizados é reservado o domínio `public`, e o domínio `dyn` é atribuído a tipos dinâmicos (solução encontrada para mapear tipos de arquivos de classificação desconhecida). Ao contrário dos OSTypes, as UTIs não possuem limitação de tamanho e suportam caracteres Unicode.

A principal inovação dos UTIs é a sua estrutura. Os tipos possuem conexões entre si formando um grafo direcionado transitivo e hierárquico, chamado de *conformance hierarchy* (hierarquia de conformidade). A figura 2.1 ilustra uma parte dessa hierarquia. Suportando herança múltipla, o grafo não necessariamente forma uma árvore, e pode até apresentar subgrafos desconexos. Isso implica que a hierarquia formada por UTIs não garante a existência de um nodo principal, ascendente de todos os outros. De fato, no próprio conjunto inicial de tipos do MacOS X não há um ascendente superior.

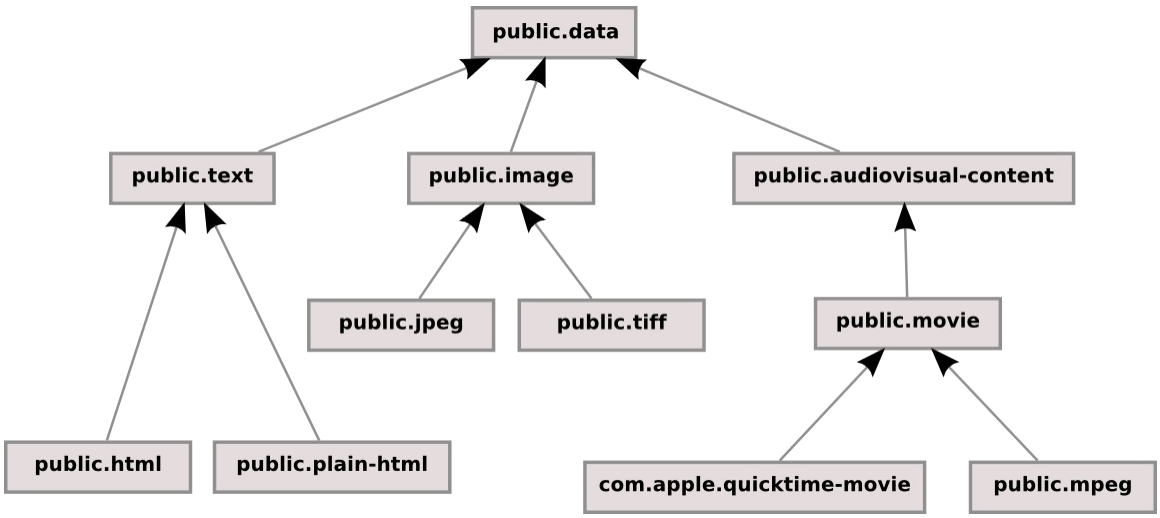

Figura 2.1: Hierarquia formado por UTIs

Outra característica importante do novo sistema de tipos da Apple é a sua flexibilidade. UTIs não foram projetados para identificar apenas arquivos e documentos, mas sim qualquer "*classe de entidade considerada possuidora de um tipo*" (APPLE, 2006), como arquivos, atalhos ou apenas uma série de dados. De fato, um dos principais objetivos dos UTIs foi criar uma arquitetura consistente para o *pasteboard*, área de transferência de dados, semelhante ao *clipboard* (área de transferência) do sistema operacional Windows, podendo conter qualquer conjunto de dados. Com a hierarquia formada por UTIs, o *pasteboard* conta com mais informações acerca do objeto sendo copiado, aumentando o número de operações possíveis e aprimorando as já existentes.

O grafo formado pelos UTIs pode ser dividido em duas hierarquias: física e funcional. Tirando proveito da abrangência do sistema, a hierarquia física classifica o tipo como diretório, volume, atalho, pacote, etc., ou seja, pelas suas características estruturais (figura 2.2). Já a hierarquia funcional diz respeito aos propósitos e à semântica do tipo (figura 2.3). Se for possível, é recomendado que novos tipos sejam inseridos em ambas hierarquias, disponibilizando assim informações complementares.

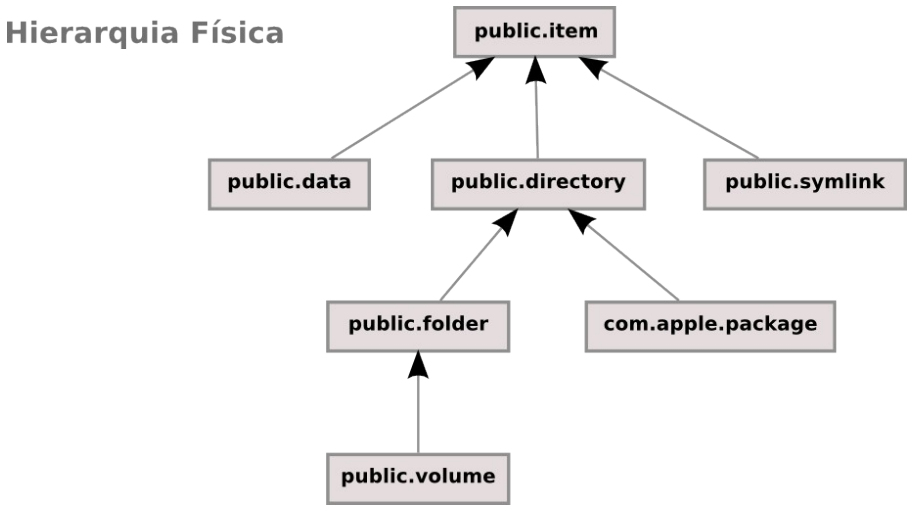

Figura 2.2: Principais tipos da hierarquia física

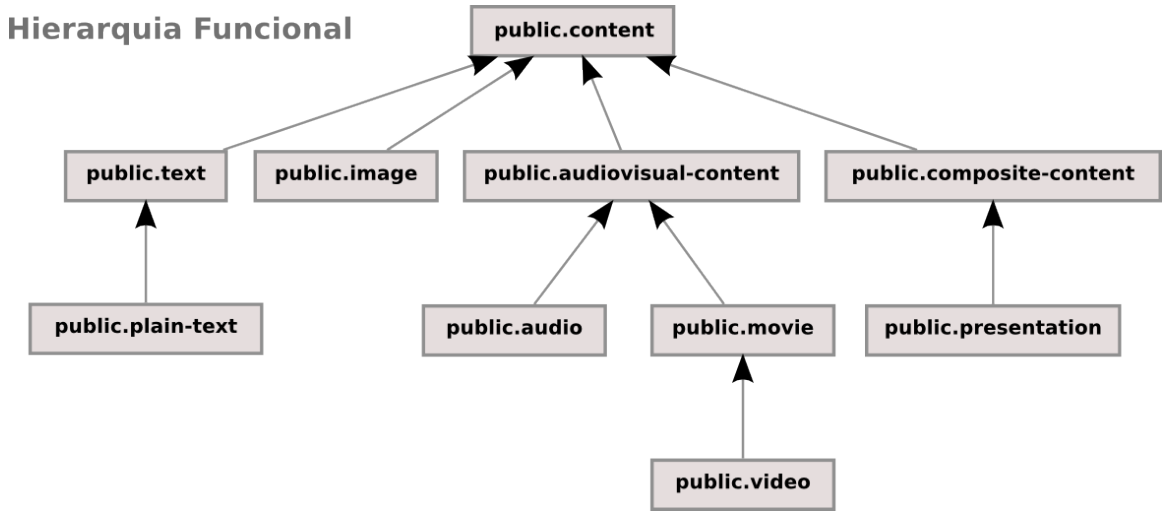

Figura 2.3: Principais tipos que formam a hierarquia funcional

A hierarquia física e a hierarquia funcional são casos de diferentes classificações em um mesmo grafo hierárquico, segundo determinado critério. Uma das propriedades da

herança múltipla são justamente classificações alternativas, inexistentes em uma hierarquia estruturada em árvore.

2.2.5 Outros sistemas de tipos de arquivo

Existem outras maneiras de identificar arquivos, que geralmente complementam os sistemas de tipos já mencionados. Além do nome e metadados associados, é possível inferir o tipo de um arquivo analisando seu conteúdo. A princpal técnica é analisar os primeiros bytes do arquivo, investigando a existência de um *magic number* (número mágico) ou de um padrão conhecido. Um *magic number* é uma seqüência de alguns poucos bytes presentes no início do arquivo que identificam seu tipo (LI, Wei-Jen. et al.). Uma imagem GIF, por exemplo, possui o *magic number* `GIF87a` ou `GIF89a`, dependendo da versão da imagem.

Mesmo na ausência de um *magic number*, é possível classificar um documento verificando seus primeiros caracteres. Documentos HTML, por exemplo, iniciam com `<html`, enquanto que documentos XML, com `<?XML`. Porém, assim como existem muitos documentos HTML mal-formados que são apresentados corretamente nos navegadores, vários outros documentos apresentam problemas semelhantes, impedindo sua identificação. Além disso, muitos arquivos não possuem um padrão diferenciado nos seus dados iniciais.

Apesar desses problemas, a identificação por análise de conteúdo é um método bem-sucedido, sendo empregado em muitos aplicativos, além de ambientes gráficos e sistemas operacionais. Por questões de desempenho, ocasionado pela complexidade dos testes envolvidos na detecção do *magic number* e de padrões conhecidos, ele é usado geralmente complementando outros métodos. Se a extensão de um arquivo é desconhecida, por exemplo, talvez seja possível identificá-lo analisando-se seu conteúdo.

No Gnome, para a identificação de determinado arquivo, é verificado se seu nome possui uma das extensões listadas ou se possui um dos padrões das expressões regulares (GNOME, 2006). Caso o arquivo não seja identificado pelo seu nome, o sistema então analisa seu conteúdo em busca de *magic numbers* ou de uma seqüência de caracteres conhecida. Em versões recentes do ambiente gráfico, ele faz as duas verificações: por padrão de nome de arquivo e por análise de conteúdo, independente do resultado da primeira verificação. Se houver discordância entre os dois resultados, o usuário é avisado com uma mensagem alertando sobre o problema.

Uma outra forma de identificação, pouco utilizada, é o armazenamento do tipo do arquivo em campos adicionais do seu descritor. Alguns sistemas de arquivos (como NTFS, FAT12, FAT16, ext2, etx3, ReiserFS, entre outros) suportam *extended attributes* (atributos estendidos), atributos anexados ao descritor do arquivo que permitem armazenar informações adicionais além dos atributos pré-definidos (TANENBAUM, 2001). Dessa forma, aplicativos podem gravar informações específicas em um arquivo sem interferir em seu conteúdo. Alguns sistemas operacionais, como o OS/2 (descontinuado), utilizaram *extended attributes* a fim de identificar o arquivo, armazenando um código ou um pequeno texto de acordo com o seu tipo.

2.2.6 Estado da arte e tendências

Entre os diversos sistemas de identifcação de arquivos, atualmente prevalece a identificação por extensões, apesar de existirem soluções muito superiores, como o sistema da Apple baseado em UTIs. Na verdade, as extensões funcionam como um máximo divisor comum entre tipos de arquivo. O MacOS, que utilizava apenas OSTypes em suas versões anteriores, passou progressivamente a também suportar extensões de nomes de arquivos. O Gnome e KDE, baseados em tipos MIME, fazem mapeamento de extensões para tipos MIME. Já o Mac OS X, com seu novo sistema de tipos, é capaz de mapear extensões, tipos MIME e OSTypes para UTIs. Assim, a idenficação de documentos por extensão de nome de arquivo se tornou um padrão *de facto*, sendo o mais portável método, apesar de suas grandes deficiências. Com o recente lançamento dos UTIs, possivelmente outros sistemas operacionais e ambientes gráficos priorizem o sistema de tipos, desenvolvendo um sistema semelhante ou mesmo adotando a solução da Apple (embora a transição entre os sistemas ocorra provavelmente de forma bastante gradual).

É possível perceber uma evolução dos sistemas de tipos de arquivos e da forma empregada para identificar documentos. Muitos problemas existentes com os métodos mais antigos foram estudados e corrigidos, com isso aumentando a complexidade dos sistemas. Tal complexidade, no entanto, se mostrou necessária, pois acompanha a semântica existente nas relações entre os documentos e dados em geral. Os documentos não possuem estruturas arbitrárias e completamente diferentes entre si, mas seguem formatos pré-existentes e padrões bem estabelecidos.

Felizmente, o usuário final não precisa lidar diretamente com a complexidade dos sistemas de tipos de arquivo, pois eles são utilizados quase que exclusivamente pelas aplicações. Na verdade, o objetivo de um sistema de tipos é justamente propiciar maior conforto ao usuário, procurando extrair a maior quantidade possível de informação de um arquivo por meio de seu tipo. Assim, ferramentas e aplicativos podem auxiliar o usuário sugerindo ações sobre documentos, organizando documentos automaticamente, exibindo ícones diferenciados, realizando buscas por documentos semelhantes, entre outras aplicações.

2.3 Tags

Nos últimos anos, uma nova forma de organizar documentos ganhou grande popularidade na Web, principalmente em *blogs* e em *sites* com grandes quantidades de documentos. As *tags*, como são conhecidas, têm sido cada vez mais empregadas na classificação dos documentos devido a sua simplicidade e eficácia. Uma *tag,* neste contexto, é uma palavra-chave ou termo descritivo associado a um item com o objetivo de classificá-lo. Elas geralmente são escolhidas informalmente pelo autor do item, ou seja, não são atribuídas automaticamente (SHIRKY, 2006). Por exemplo, os termos *Linux*, *Gnome*, *KDE*, *GUI* e *sistema operacional* poderiam ser *tags* atribuídas a um artigo sobre uma comparação entre ambientes gráficos no Linux.

As *tags* são empregadas em um grande número de aplicações na web. No site *del.icio.us* (http://del.icio.us), *tags* são utilizadas pelos usuários na classificação de links para páginas web. Em combinação com a popularidade do link (número de usuários que o contém), as *tags* funcionam como uma forma interessante de se localizar informações relacionadas a um determinado assunto. Já no seriço de armazenamento e compartilhamento de imagens Flickr (http://flickr.com), pertencente à empresa Yahoo!, imagens podem ser organizadas e localizadas por meio de *tags* atribuídas pelo próprio usuário, além da usual organização em pastas, chamadas pelo site de *sets* (conjuntos). Outro exemplo de utilização de *tags* é encontrado no sistema de correio eletrônico on-line GMail (http://mail.google.com), do Google, que possibilita a marcação das mensagens com *tags* (chamadas de *labels* pelo sistema), visando facilitar posteriormente sua localização.

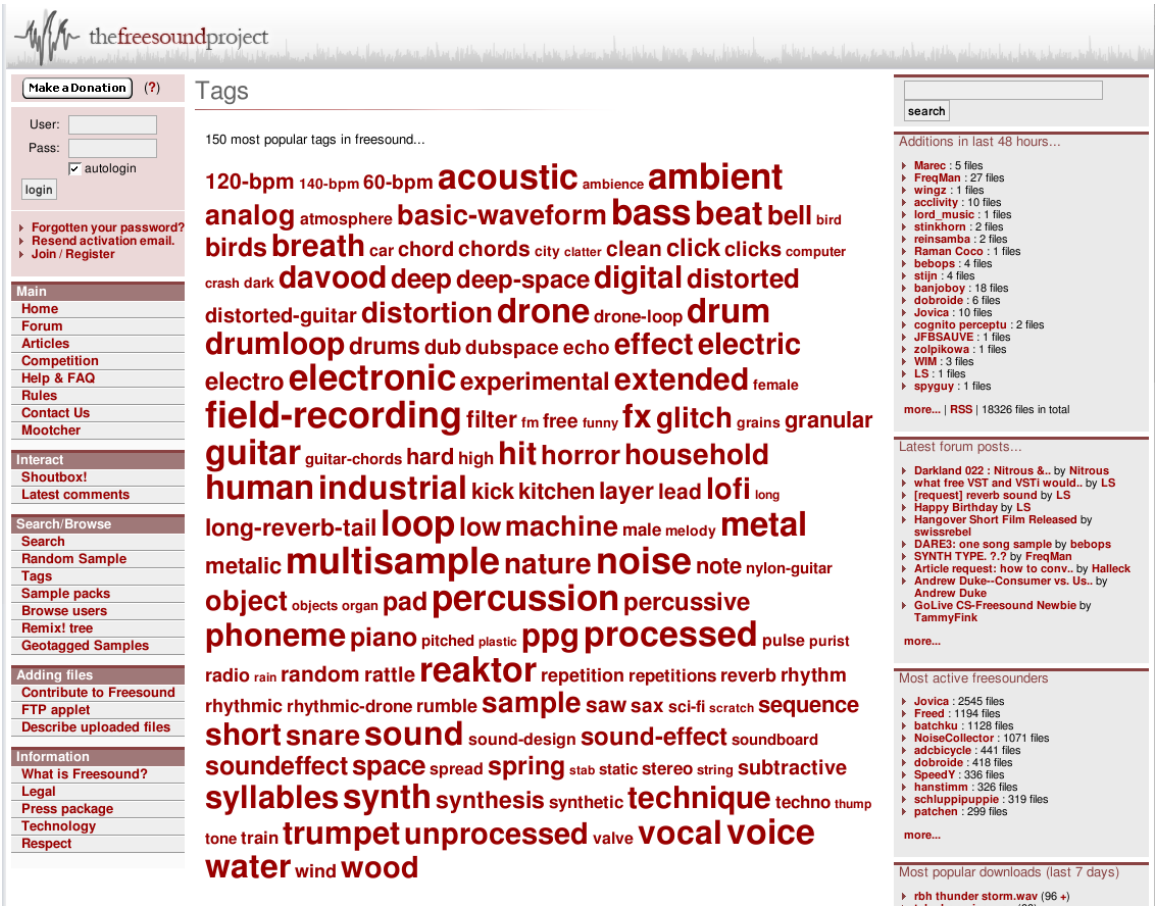

Figura 2.4: *Tags* classificando sons no projeto Freesound

Uma forma de visualização que se tornou comum na Web é o emprego de fontes de tamanhos variados de acordo com a quantidade de itens de uma *tag*. Quanto mais itens uma *tag* possui, maior é o tamanho da fonte utilizada, conforme ilustrado na figura 2.4,

site do projeto Freesound. Cada uma das *tags* apresentadas na figura representa uma categoria de sons, mas um som pode estar relacionado a mais de uma *tag*. Um dos sons, por exemplo, o som de uma impressora ao ser inicializada, está marcada com as tags *computer*, *digital*, *field-recording*, *industrial*, *machine* e *printer*. Fornecer uma informação visual desse tipo é importante quando é de interesse do usuário conhecer as quantidades envolvidas. Essa idéia pode ser reaproveitada para classificações em sistemas *desktop.*

2.3.1 Tags em sistemas desktop

O conceito de *tags* não foi muito explorado para classificação de documentos em sistemas *desktop*. Na verdade, a maneira como os arquivos são exibidos ao usuário é a mesma de como os arquivos são armazenados fisicamente. Para que novas formas de organização e categorização de arquivos se concretizem, é necessário abstrair o nível físico, constituído de arquivos, diretórios e descritores em uma camada de mais alto nível. Como conseqüência, é muito provável que seja necessária a utilização de índices, considerando que, em última instância, os recursos exibidos devem ser mapeados para a estrutura física. Portanto, a implementação de *tags* para marcação de documentos locais precisa contar com um mecanismo próprio para mapeamento de recursos para o nível físico.

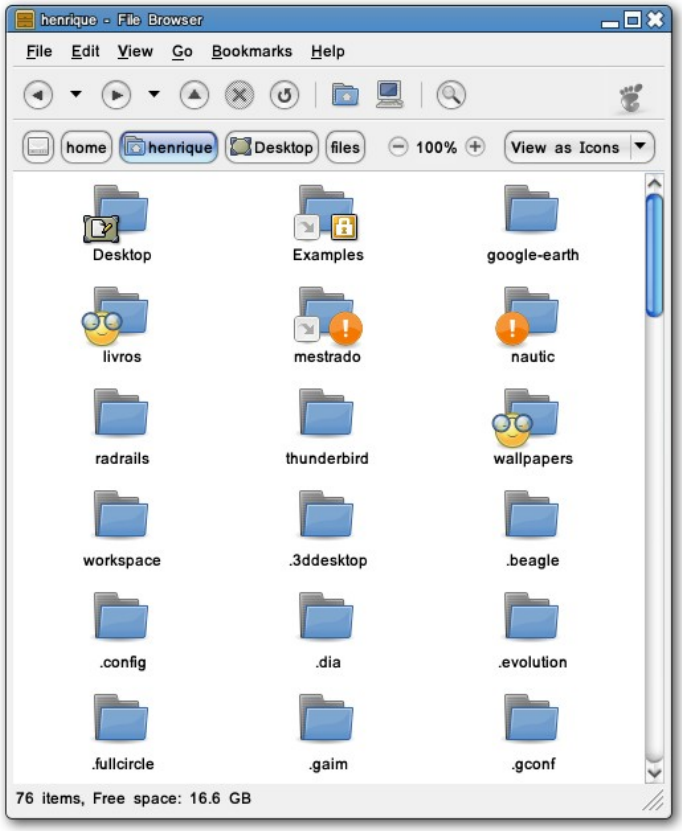

Figura 2.5: Emblemas de ícones no gerenciador de arquivos Nautilus

Em uma versão recente (GNOME, 2006), o ambiente gráfico Gnome introduziu o conceito de *emblemas* (*emblems*) através de seu gerenciador de arquivos. Emblemas são *tags* de marcação que são visualizados como pequenos ícones próximos ao ícone principal do recurso (figura 2.5). Documentos com emblemas se distinguem dos outros e são facilmente identificados, pois cada emblema possui uma imagem diferente das demais. Os recursos são marcados com emblemas pelo próprio sistema (como em recursos controlados por *cvs* ou em arquivos protegidos) ou pelo usuário, através do menu de contexto do gerenciador de arquivos. Praticamente qualquer tipo de recurso, incluindo arquivos, diretórios, *weblinks*, computadores remotos e unidades de disco podem ser marcados.

Atualmente, a única utilidade dos emblemas é auxiliar o usuário na distinção dos documentos marcados. Porém, é fácil perceber que muitos outros serviços poderiam ser agregados a essa funcionalidade, se observarmos as *tags* utilizadas na Web. A disponibilização de funcionalidades como classificação por emblemas e busca de documentos com determinado emblema possibilitariam uma nova maneira de organizar arquivos. O projeto de código aberto Leaftag (TROWBRIDGE, David), iniciado no final do ano de 2005 e agora pertencente ao projeto Gnome, implementa esses serviços por meio de *tags* textuais.

Existe uma discussão não encerrada entre os participantes do projeto, no entanto, no momento em que esta dissertação está sendo escrita. Ela diz respeito ao escopo das *tags*: se serão individuais (diferentes para cada usuário) ou globais (*system-wide*), válidas para todos usuários. Cada abordagem possui vantagens e desvantagens, e talvez seja cedo para determinar qual a melhor escolha, apesar do coordenador do projeto estar mais inclinado a optar por *tags* individuais. A principal diferença entre as duas formas é o reaproveitamento das *tags* por outros usuários, no caso delas serem globais, e a possibilidade de marcar quaisquer recursos, inclusive recursos remotos e documentos que o usuário não possui permissão, no caso de *tags* individuais.

No caso das *tags* globais, elas seriam armazenadas em atributos estendidos (*extended attributes*), por isso a restrição apenas aos recursos que o usuário tem acesso. Como os atributos estendidos são os mesmos para todos os usuários, e são movidos e copiados juntamente com os arquivos, as *tags* seriam preservadas e compartilhadas entre os vários usuários e até mesmo entre diversos computadores. De qualquer maneira, as duas abordagens requerem um banco de dados de *tags*, mesmo com a utilização de atributos estendidos, pois a verificação das *tags* nos arquivos percorrendo toda a árvore de diretórios do sistema, a cada busca, seria demasiadamente lenta.

O banco de dados do Leaftag pode ser visto essencialmente como um índice: para cada *tag*, existe um conjunto de referências contendo a localização dos arquivos. Como um arquivo pode ser marcado com múltiplas *tags*, sua localização física pode possuir mais de uma referência no banco de dados. A figura 2.6 apresenta um exemplo de *tags* e suas referências para os arquivos.

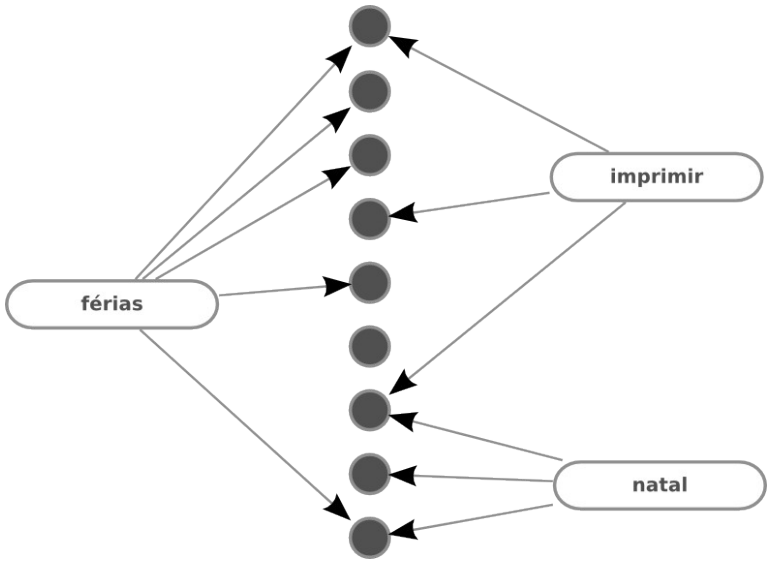

Figura 2.6: *Tags* para *desktop* como índices de arquivos

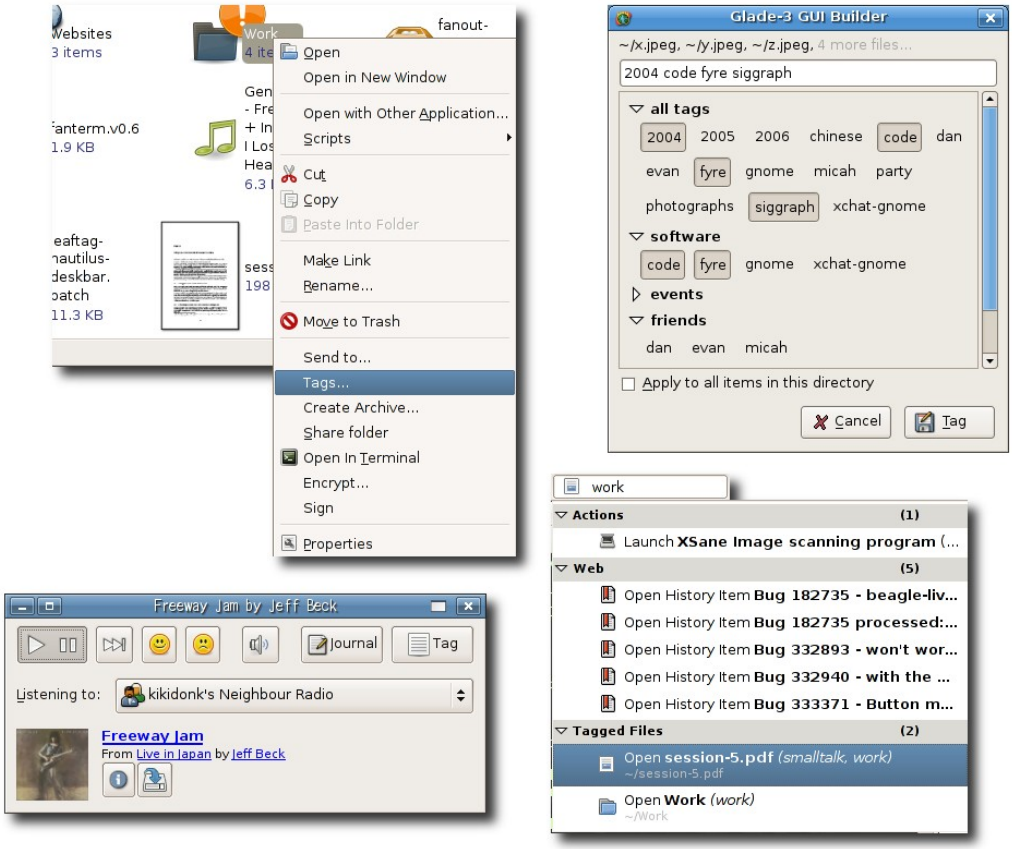

Figura 2.7: Suporte a *tags* no ambiente gráfico Gnome através do projeto Leaftag

O Leaftag foi adotado pelo projeto Gnome, com o objetivo integrá-lo nas próximas versões de seu ambiente gráfico. Diversas partes do ambiente, como o gerenciador de arquivos, sistema de busca, barra de ícones e alguns aplicativos instalados juntamente com o ambiente, estão sendo estendidos para suportar *tags* (figura 2.7). A ubiquidade das *tags* e o esforço dos desenvolvedores indicam que as *tags* possuirão um suporte adequado, e talvez se tornem em uma alternativa viável na organização e classificação de documentos em sistemas *desktop*.

3 A HIERARQUIA DE TIPOS DE ARQUIVO

No projeto de sistemas de tipos de arquivos se procura abordar questões como ambigüidades, expressividade e estensibilidade, implicando em grandes conseqüências para aplicações que trabalham com diferentes formatos de arquivos. Mas eles não são projetados para serem diretamente manipulados por usuários finais. Quando um arquivo é criado por um aplicativo, este já marca seu tipo, que permanece até que a estrutura de seu conteúdo seja suficientemente alterada (o que raramente ocorre). Portanto, o usuário não escolhe qual tipo de arquivo atribuir a determinado documento, e tampouco o modifica posteriormente. Os sistemas de tipos de arquivos servem às aplicações, e estas repassam indiretamente os benefícios do sistema de tipos ao usuário.

Embora também ofereça serviços às aplicações através de uma API, o grafo de tipos de arquivos apresentado neste trabalho possui sua estrutura voltada também para a manipulação direta do usuário final, implementado como uma arquitetura mista envolvendo *tags* e tipos de arquivos. Existe um grande número de aplicações potenciais ao combinar esses dois conceitos, como permitir ao usuário incorporar tipos às suas *tags*, criar novos tipos de alto nível (tipos classificadores, que serão apresentados mais adiante), ou visualizar arquivos pertencentes a determinada *tag* classificados de acordo com seus tipos. Essas aplicações apenas são possíveis devido à existência de uma hierarquia de tipos de arquivos.

A hierarquia entre tipos de arquivos não é uma idéia nova, e vários softwares já incorporaram, de uma forma ou de outra, seus próprios sistemas hierárquicos. A plataforma Eclipse (ECLIPSE FOUNDATION, 2006) oferece um framework genérico sobre a qual centenas de aplicativos já foram desenvolvidos, de diferentes portes, como a família de produtos Rational Rose XDE e Lotus Notes. Todas as funcionalidades são disponibilizadas na forma de componentes, que são selecionados e reutilizados pelos softwares de acordo com seu propósito. Entre os componentes mais centrais da arquitetura, estão aqueles que gerenciam e manipulam documentos.

Uma série de problemas ocorreram em versões anteriores do Eclipse na identifcação de arquivos e associação com editores. Editores específicos eram associados a extensões de nomes de arquivos, mas às vezes os componentes responsáveis por determinados editores não estavam disponíveis, impossibilitando a edição de certos documentos. Na ausência de um editor XML, por exemplo, editores de texto não eram associados a documentos baseados nessa linguagem. Além disso, mesmo na presença de um editor XML, para documentos baseados nessa tecnologia mas com extensão diferente de `.xml` era necessária a associação manual a dois editores, o editor XML e o editor de texto (o *software* desconhecia a natureza de um documento XML). Com a versão 3.2, o

Eclipse solucionou os problemas relacionados a tipos de arquivos adicionando relações de herança, permitindo que todos os aplicativos baseados nessa versão tirassem proveito da nova estrutura.

De maneira semelhante, o *middleware* disponibiliza uma série de serviços sobre a hierarquia de tipos de arquivos. Através de uma API os aplicativos podem consultar ou manipular a hieraruia, adicionando, alterando ou excluindo tipos e associações.

3.1 Conceitos e propriedades

A idéia central do grafo apresentado neste trabalho é estender os tipos de arquivos adicionando relacionamentos de herança para refletir o aspecto hierárquico existente nos formatos de arquivos. Dessa forma, um determinado tipo de arquivo herda as características de seus tipos ascendentes e as propaga aos seus descententes, de forma análoga ao paradigma de orientação a objetos. Não sujeitos às restrições de uma estrutura tabular, os tipos de arquivos podem em um grafo representar um modelo muito mais rico, contendo relações semânticas antes inexistentes, ideais para identificação e classificação de documentos. Por meio de inferências, é possível extrair mais informações dos tipos de arquivos, solucionando os problemas apresentados na introdução deste capítulo.

3.1.1 Escolha do sistema de tipos de arquivo

A hierarquia de tipos de arquivo forma um grafo direcionado transitivo com arestas rotuladas, onde os nodos são tipos de documentos e as arestas representam a relação de herança. Foram utilizados tipos MIME para identificar os tipos de arquivo. Esta escolha foi feita levando em consideração a documentação disponível, a grande quantidade de sistemas que o utilizam e o fato de que seu formato é bastante conhecido, além de ser padrão na identificação de documentos na Web.

Embora com a hierarquia e os novos serviços oferecidos pelo *middleware* algumas limitações dos tipos de mídia MIME como sistema de tipos sejam eliminadas, ainda persistem alguns problemas inerentes à especificação desse sistema. É fácil para o *middleware* evitar, por exemplo, que dois tipos possuam o mesmo nome, pois se trata de uma questão de implementação. Porém, não é apresentada na especificação dos tipos MIME um padrão ou uma forma que deve ser seguida na criação de nomes para evitar ambigüidades de nomes (IANA, 2006), ao contrário de UTIs, que adotam domínio invertido.

Mas vale ressaltar que o objetivo deste trabalho é apresentar conceitos e uma hierarquia de tipos de arquivo que pode ser usada como referência. Assim, considerando também os argumentos apresentados anteriormente, a opção por tipos MIME se mostrou mais adequada, mesmo na existência de um sistema superior. Muitos dos principais princípios, porém, como a arquitetura baseada em um *middleware*, a junção de tipos e

tags, e indexação de arquivos poderiam também ser aplicados em uma solução baseada em UTIs. Quanto à hierarquia de tipos, especificamente, o conjunto inicial presente no MacOS X poderia ser estendido, utilizando o próprio mecanismo de herança dos UTIs, para acomodar tipos classificadores (APPLE, 2006).

Outra alternativa seria a elaboração de um novo sistema de tipos, destinado a ser utilizado pelo *middleware*. Como um dos objetivos do *middleware* é justamente ser multiplataforma, esse novo sistema estaria disponível para aplicativos e ferramentas produzidas em qualquer plataforma, sem problemas de compatibilidade. O esforço de criação e adoção de um novo sistema, porém, talvez não seja justificado frente a opção de adotar um sistema completo e funcional como UTI ou uma variação deste.

3.1.2 Herança múltipla

No grafo implementado no *middleware*, os tipos MIME são estendidos de forma a oferecer um sistema de tipos com mais recursos, entre os quais a herança entre tipos de arquivos. A figura 3.1 ilustra os relacionamentos existentes entre quatro tipos de arquivos. Os nodos `text/html` e `text/xml` possuem relacionamento de herança com o nodo `text/plain`, mais genérico. Assim, ambos possuem um mesmo formato comum derivado de `text/plain`. Já o tipo `application/xhtml+xml` herda características tanto de `text/html` quanto de `text/xml`, ocasionando uma herança múltipla. Nesse caso, um arquivo XHML pode ser considerado ao mesmo tempo um arquivo HTML e XML. Essa informação permite que aplicativos preparados para trabalhar com documentos pertencentes a um desses tipos possam também operar arquivos do tipo XHTML.

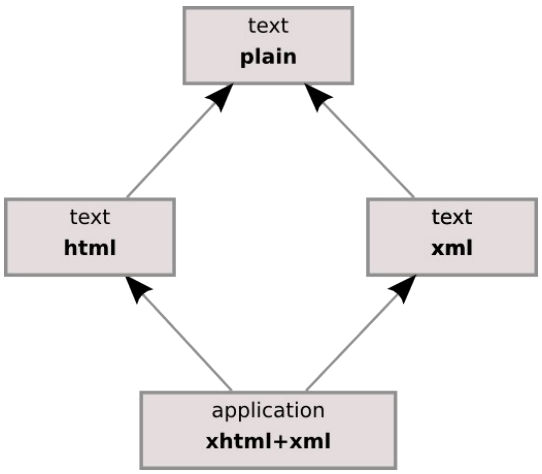

Figura 3.1: Herança múltipla em tipos de arquivos

3.1.3 Tipos classificadores

Além dos tipos de arquivo comuns, a hierarquia de tipos prevê nodos especiais, chamados de tipos classificadores. Trata-se de identificadores de um grupo de documentos semanticamente relacionados no mundo real, mas que não possuem correspondência nos tipos de arquivos existentes. Eles têm como objetivo completar o grafo, enriquecendo o modelo e permitindo que se faça buscas e classificações mais precisas e úteis. Exemplos de tipos classificadores típicos são `image`, representando qualquer imagem gráfica e `source_code`, abrangendo o código-fonte texto de qualquer linguagem de programação. Na figura 3.2, é apresentado o tipo classificador `source_code` com seus descendentes `text/x-c++` e `text/x-java` (assim como os tipos comuns, os tipos classificadores também suportam herança).

Tipos classificadores participam da hierarquia juntamente com os tipos comuns, mas não possuem extensões mapeadas e tampouco arquivos diretamente associados. Esses nodos especiais apenas se associam com outros tipos de arquivos (comuns ou classificadores), representando um grupo de arquivos semanticamente similares segundo determinado aspecto. Conforme analisado mais adiante, esses tipos especiais inspirados em *tags* desempenham um papel chave na arquitetura como um todo, integrando tipos de arquivos, *tags* para *desktop* e indexação de arquivos por tipo, habilitando uma nova classe de aplicações.

É importante lembrar que podem existir várias extensões ou expressões regulares associadas a cada tipo de arquivo. No Gnome, por exemplo, arquivos com extensão `.cc`, `.C`, `.cpp` e `.c++` são associadas ao tipo MIME `text/x-c++` (o Gnome emprega tipos MIME como tipos de arquivos), e a extensão `.java` é associada a `text/x-java` (GNOME, 2006). Ao fazer uma busca pelos arquivos de código-fonte, ou seja, encontrar arquivos dos tipos descententes de `source_code`, são retornados todos os documentos com extensão `.cc`, `.C`, `.cpp`, `.c++` e `.java`. O grafo poderia conter um tipo de arquivo que fosse descentente de `source_code` e de `text/xml` (constituindo uma herança múltipla), como a linguagem o:XML (KLANG, Martin), uma linguagem de programação cuja sintaxe é baseada em XML. Documentos do tipo o:XML seriam então retornados tanto em buscas por arquivos de código-fonte quanto por arquivos XML, comportamento que provavelmente é o esperado pelo usuário.

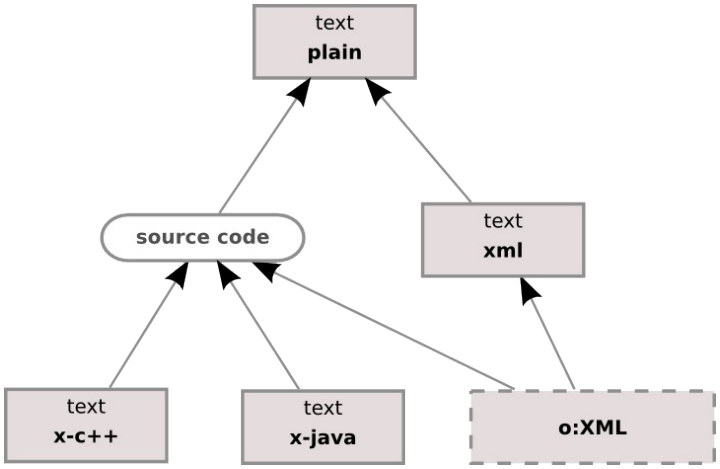

Figura 3.2: Tipos classificadores possuem as mesmas propriedades dos tipos de arquivos comuns, mas não possuem arquivos diretamente associados

É importante salientar que a maneira como são identificados hoje os arquivos quanto ao seu tipo não é exata, e pode levar a ambigüidades ou classificações incorretas, além de não garantir sua identificação. Apesar disso, na grande maioria dos casos os arquivos seguem as extensões corretamente ou seu tipo é inferido com sucesso pela análise de conteúdo. De qualquer forma, a hierarquia de tipos depende da eficácia do sistema de identifcação da plataforma em questão, e apenas os arquivos com identifcação problemática serão classificados incorretamente.

3.2 A construção da hierarquia

O objetivo de criar uma hierarquia de tipos de arquivos neste trabalho não é elaborar uma ontologia completa dos formatos existentes, mas sim selecionar um conjunto representativo de tipos que possa atender as expectativas das ferramentas e aplicativos. Para essa função, foram escolhidos os tipos de arquivos iniciais que são oferecidos pelo ambiente gráfico Gnome, alguns tipos oficiais padronizados pelo IANA e outros criados exclusivamente para a hierarquia, mas que seguem a sintaxe dos tipos MIME, conforme será detalhado mais adiante. Esse grupo deve ser suficiente para abranger uma grande parte dos arquivos e permitir a validação do *software* experimental.

O sistema de tipos do Gnome é baseado em tipos MIME, que são armazenados em um arquivo lido durante a inicialização do ambiente. Contendo 265 tipos (na versão 2.14.2), com suas respectivas extensões e expressões regulares, sua sintaxe é simples e facilmente processável. Os primeiros dez tipos são apresentados abaixo.

```
application/andrew-inset
      ext: ez

application/msword
```

```
      ext: doc

application/octet-stream
      ext: bin

application/oda
      ext: oda

application/ogg
      ext: ogg

application/pdf
      ext: pdf

application/pgp
      ext: pgp

application/postscript
      ext: ps eps

application/qif
      ext: qif

application/rtf
      ext: rtf
```

Além dos tipos existentes inicialmente nesse arquivo, foram adicionados alguns formatos padronizados pelo IANA, como `application/rdf+xml`. Outros formatos bastante conhecidos não possuem um tipo MIME correspondente devido à limitação em dois níveis de herança desse sistema. É o caso de RSS, RDF Schema e OWL. A recomendação feita pelo W3C, autor dessas tecnologias, é que se utilize o tipo `application/rdf+xml` para esses formatos, evitando complicações no processamento de navegadores web, servidores de aplicação, servidores web e outros aplicativos. Esse é um problema cuja solução envolve a adoção generalizada de um novo sistema de tipos para a web ou no aperfeiçoamento do sistema atual, os tipos de mídia MIME.

Como aplicações para *desktop* utilizam o sistema de tipos através de uma API e a interface do *middleware* fornece acesso às funcionalidades sobre a hierarquia, não há necessidade de se restringir a criação de um tipo devido à limitação em dois níveis mencionada anteriormente. A hierarquia de tipos utiliza o nome do tipo MIME apenas como identificador, e não distingue entre *tipo* e *subtipo*. Ambos, juntamente com o caractere separador, a barra (/), formam um único nome que é usado para fins de identificação. Por esses motivos, foram adicionados novos tipos mais específicos à hierarquia. Por exemplo, para documentos RSS, RDF Schema e OWL, foram criados os tipos `application/rss+xml`, `application/x-rdfs` e `applicaton/x-owl`.

Já os relacionamentos e tipos classificadores são armazenados em um arquivo separado em formato texto simples (até certo ponto semelhante ao arquivo contendo tipos MIME do Gnome). Os nodos classificadores diferem dos tipos comuns por possuirem o

sinal de dois-pontos (:) imediatamente antes do seu tipo. Embora esse arquivo utilize um formato texto próprio, a estrutura interna do *software* que comporta o grafo é bastante genérica, e implementa no seu núcleo os conceitos de trincas, com sujeito, predicado e objeto, provenientes da linguagem RDF. No arquivo contendo os relacionamentos, a expressão `:can_be` se refere ao predicado da sentença, e possui o significado inverso da relação *é-um* (também suportada pelo *middleware*, como `:is`). Poranto, embora todos os relacionamentos criados para o grafo inicial sejam de herança, outros tipos de relações também são possíveis. A listagem abaixo apresenta um trecho do arquivo em questão, apresentando a definição dos descendentes do tipo classificador `:source_code`.

```
#=========== SOURCE CODE ============#
:source_code :can_be
      text/x-c
      text/x-c++
      text/x-c-header
      text/x-emacs-lisp
      text/x-fortran
      text/x-haskell
      text/x-java
      text/x-javascript
      text/x-lua
      text/x-objc
      text/x-pascal
      text/x-perl
      text/x-python
      text/x-sh
      application/x-php
      application/x-asp
      text/x-asm
      text/x-tcl
```

Neste exemplo, `:source_code` é o sujeito, `:can_be` é o predicado e os demais tipos abaixo são os objetos. A sintaxe usada nesse arquivo pode ser visto como uma representação alternativa para recursos RDF, específica para o *middleware*. A escolha pela sintaxe baseada em XML ou pela notação N3 (BERNERS-LEE, 1998) para definir os tipos de arquivos e associações implicaria na utilização de um *parser* mais complexo, além de prejudicar a manipulação manual do arquivo (principalmente no caso da sintaxe XML). Com o objetivo de manter o protótipo claro e compreensível, optou-se por um formato mais simples. Porém, é recomendada uma linguagem padronizada para versões de produção do *middleware* devido principalmente aos benefícios proporcionados pela interoperabilidade com ferramentas e outros sistemas

3.2.1 Hierarquia estrutural

Para cada tipo de arquivo foi realizado um estudo de seu formato e uma persquisa dos aplicativos que o utilizam, com o objetivo de categorizar o tipo MIME estruturalmente e semanticamente. Algumas relações de herança na hierarquia indicam informações estruturais, que dizem respeito ao formato (sintaxe) do documento. Esses nodos formam entre si uma hierarquia de formatos, um subconjunto da hierarquia completa de tipos de arquivo. Os tipos `text/plain` e `application/rtf` (*Rich Text Format*), por exemplo, possuem uma relação hierárquica estrutural, ou seja, documentos RTF são baseados na estrutura de um documento texto. O mesmo ocorre com os tipos `text/xml` e `image/svg`, pois o formato dos arquivos SVG é baseado na sintaxe XML. A hierarquia estrutural é equivalente à hierarquia física de tipos UTIs (apresentada na subseção 2.2.4), porém menos abrangente, por apenas comportar tipos de arquivos (UTIs identificam não só arquivos, mas também diretórios, atalhos, volumes, entre outros elementos).

Em documentos de tipos que possuem herança estrutural entre si, o tipo descendente pode realizar todas as operações de seus ascendentes, pois os dois tipos compartilham uma sintaxe comum. Um documento SVG, portanto, de acordo com a hierarquia de tipos de arquivos, pode ser tratado da mesma forma que um documento XML ou que um arquivo texto. Os tipos centrais da hierarquia estrutural inicial do *middleware* são `text/plain`, `text/xml` e `text/rdf`. Isso ocorre porque esses tipos possuem uma estrutura que é reutilizada por vários outros formatos. Os principais elementos da hierarquia estrutural são apresentados na figura 3.3. A hierarquia estrutural é importante para determinados tipos de aplicações, como ações sobre arquivos, descrita na seção 5.1.

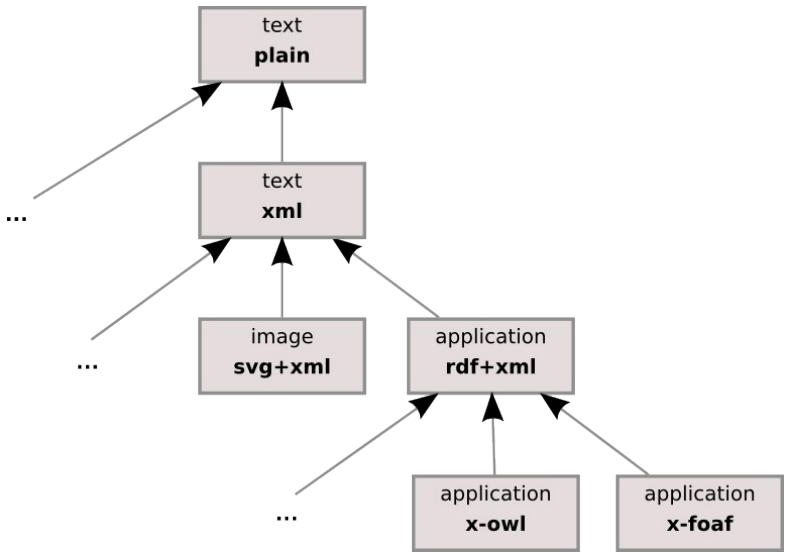

Figura 3.3: Parte da hierarquia estrutural

3.2.2 Hierarquias semânticas

Para a criação das relações estruturais entre arquivos foi necessário essencialmente estudar o formato de seu conteúdo, sendo praticamente um processo de descoberta. Já a criação das relações não estruturais envolve a pesquisa do significado (semântica) do tipo, que deve ser adequadamente associado às várias classificações inter-relacionadas existentes.

Os tipos classificadores, juntamente com tipos comuns, formam uma estrutura que permite praticamente qualquer classificação. Além da hierarquia estrutural, que é mais útil aos aplicativos e ferramentas do que ao usuário final, existem outras classificações úteis, que geralmente empregam como critério um aspecto semântico dos documentos. Assim, os mesmos arquivos podem ser visualizados e pelo usuário organizados de diferentes maneiras, conforme a aplicação.

Por exemplo, a visualização de um conjunto diversificado de documentos de escritório podem ser visualizados pela sua função, classificados em textos, planilhas e apresentações, ou segundo a sua origem, classificados como documentos do Microsoft Word, Lotus Notes, Adobe Acrobat, etc. A capacidade de organizar os documentos de diferentes maneiras é algo pouco explorado atualmente, mas que possui grande potencial (SHADBOLT, Nigel; BERNERS-LEE, Tim; HALL, Wendy), esse recurso é explorado pelas aplicações das seções 5.3.1 e 5.3.2.

3.3 Listagem completa da hierarquia

A listagem abaixo apresenta o conteúdo de um dos arquivos utilizados na construção da hierarquia, contendo os tipos classificadores e os relacionamentos entre tipos. Vale ressaltar que há outro arquivo com a listagem de todos os tipos de arquivos e suas respectivas extensões.

```
# mime-types in Gnome usually are in /usr/share/mime-info/gnome-vfs.mime


#======= COMPACT ORGANIZATION ======#

:compact_organization :can_be
  :office
  :multimedia
  :archive
  :program

#======= EXPANDED ORGANIZATION ======#

:expanded_organization :can_be
  :spreadsheet
  :presentation
  :text_document
  :database
  :image
  :audio
  :video
  :multimedia
```

```
  :program
  :archive
  :backup

#============== BACKUP ===============#

:backup :can_be
  application/x-backup

#============= DATABASE =============#

:database :can_be
  application/x-palm-database
  application/x-xbase
  text/tab-separated-values
  text/x-comma-separated-values

#============== ARCHIVE ===============#

:archive :can_be
  application/x-arj
  application/x-compress
  application/x-compressed-tar
  application/x-gtar
  application/x-gzip
  application/x-iso-image
  application/x-java-archive
  application/x-rar
  application/x-rar-compressed
  application/x-lha
  application/x-tar
  application/zip
  application/x-bzip
  application/x-bzip-compressed-tar
  application/x-unix-archive
  application/x-deb
  application/x-lhz
  application/x-rpm
  application/x-ustar
  application/x-zoo


#************************************************************************#
#                                                                        #
#              S T R U C T U R A L     O R G A N I Z A T I O N           #
#                                                                        #
#************************************************************************#

#====== STRUCTURAL ORGANIZATION =====#

:structural_organization :can_be
  :text
  :binary

#============= TEXT ===============#

text/plain :can_be
  text/xml
  text/x-vcard
  text/x-yacc
  text/x-comma-separated-values
  text/tab-separated-values
  text/calendar
  text/spreadsheet
  application/x-oleo            # OLEO spreadsheet
  :language

#============== XML ===============#
```

```
text/xml :can_be
  image/svg+xml
  application/x-glade
  application/x-smil
  application/x-gnumeric
  text/mathml
  :staroffice6
  text/abiword
  application/x-kword
  application/x-xbel            # XML Bookmark Exchange Language
  application/rdf+xml           # not in gnome config

#============== RDF ===============#

application/rdf+xml :can_be
  application/x-owl             # not in gnome config
  application/x-rdfs            # not in gnome config
  application/x-foaf            # not in gnome config
  application/rss+xml           # not in gnome config

#============= BINARY ==============#

:binary :can_be
  :executable
  :video
  # others...

#*************************************************************************#
#                                                                         #
#                             P R O G R A M                               #
#                                                                         #
#*************************************************************************#


#============= PROGRAM ==============#

:program :can_be
  :executable
  :language

#============ EXECUTABLE ============#

:executable :can_be
  application/x-shared-library
  application/x-shared-library-la
  application/x-python-byte-code
  application/x-java-byte-code
  application/x-ms-dos-executable
  application/x-msx-rom
  application/x-n64-rom
  application/x-nes-rom
  application/x-object-file

#============ LANGUAGE ==============#

:language :can_be
  :source_code
  :descriptive_language

#=========== SOURCE CODE ============#
:source_code :can_be
  text/x-c
  text/x-c++
  text/x-c-header
  text/x-emacs-lisp
  text/x-fortran
  text/x-haskell
```

```
  text/x-java
  text/x-javascript
  text/x-lua
  text/x-objc
  text/x-pascal
  text/x-perl
  text/x-python
  text/x-sh
  application/x-php
  application/x-asp
  text/x-asm
  text/x-tcl

#======= DESCRIPTIVE LANGUAGE =======#

:descriptive_language :can_be
  text/html
  text/sgml
  application/postscript
  text/x-sql
  text/x-dtd
  text/x-lyx
  text/css

#============= OFFICE ==============#

:office :can_be
  :staroffice5
  :openoffice6
  :koffice
  :msoffice
  application/rtf
  text/richtext
  application/vnd.lotus-1-2-3
  application/x-gnumeric
  application/x-mrproject
  application/pdf
  text/abiword
  text/spreadsheet
  application/x-oleo
  text/x-tex                    # ext: tex cls sty ltx

:koffice :can_be
  application/x-killustrator
  application/x-kpresenter
  application/x-kspread
  application/x-kword

:staroffice5 :can_be
  application/vnd.stardivision.calc
  application/vnd.stardivision.chart
  application/vnd.stardivision.draw
  application/vnd.stardivision.impress
  application/vnd.stardivision.mail
  application/vnd.stardivision.math
  application/vnd.stardivision.writer

:openoffice6 :can_be
  application/vnd.sun.xml.calc
  application/vnd.sun.xml.calc.template
  application/vnd.sun.xml.draw
  application/vnd.sun.xml.draw.template
  application/vnd.sun.xml.impress
  application/vnd.sun.xml.impress.template
  application/vnd.sun.xml.math
  application/vnd.sun.xml.writer
  application/vnd.sun.xml.writer.global
  application/vnd.sun.xml.writer.template
```

```
:msoffice :can_be
  application/vnd.ms-excel
  application/vnd.ms-powerpoint
  application/msword


# BY FUNCTION

:text_document :can_be
  application/pdf
  application/rtf
  text/richtext
  text/abiword
  text/x-tex
  application/x-kword
  application/vnd.stardivision.writer
  application/vnd.sun.xml.writer
  application/vnd.sun.xml.writer.global
  application/vnd.sun.xml.writer.template
  application/msword

:spreadsheet :can_be
  application/vnd.ms-excel
  application/vnd.lotus-1-2-3
  application/x-gnumeric
  text/spreadsheet
  application/x-oleo
  application/x-kspread
  application/vnd.stardivision.calc
  application/vnd.sun.xml.calc.template

:presentation :can_be
  application/x-kpresenter
  application/vnd.ms-powerpoint
  application/vnd.stardivision.impress
  application/vnd.sun.xml.impress
  application/vnd.sun.xml.impress.template


#*****************************************************************************#
#                                                                             #
#                                 M E D I A                                   #
#                                                                             #
#*****************************************************************************#

#============== MEDIA ===============#
:media :can_be
  :audio
  :image
  :video
  :multimedia

#============ MULTIMEDIA =============#

:multimedia :can_be
  application/x-smil
  application/x-blender
  model/vrml
  application/x-shockwave-flash

#============== AUDIO ===============#
:audio :can_be
  audio/ac3
  audio/basic
  audio/mpeg
  audio/prs.sid
  audio/x-aiff
```

```
  audio/x-flac
  audio/x-it
  audio/x-m4a
  audio/x-midi
  audio/x-mod
  audio/x-mpegurl
  audio/x-ms-asx
  audio/x-pn-realaudio
  audio/x-real-audio
  audio/x-s3m
  audio/x-scpls
  audio/x-stm
  audio/x-ulaw
  audio/x-voc
  audio/x-wav
  audio/x-xi
  audio/x-xm
  application/x-ape      # http://www.monkeysaudio.com/

#============== IMAGE ===============#

:image :can_be
  image/bmp
  image/gif
  image/ief
  image/jpeg
  image/png
  image/tiff
  image/vnd.djvu
  image/x-applix-graphic
  image/x-bmp
  image/x-cmu-raster
  image/x-compressed-xcf
  image/x-dcm
  image/x-emf
  image/x-fits
  image/x-gray
  image/x-icb
  image/x-ico
  image/x-iff
  image/x-ilbm
  image/x-lwo
  image/x-lws
  image/x-miff
  image/x-palm
  image/x-pcx
  image/x-photo-cd
  image/x-pict
  image/x-portable-anymap
  image/x-portable-bitmap
  image/x-portable-graymap
  image/x-portable-pixmap
  image/x-psd
  image/x-rgb
  image/x-sgi
  image/x-sun-raster
  image/x-tga
  image/x-xbitmap
  image/x-xcf
  image/x-xfig
  image/x-xpixmap
  image/x-xwindowdump
  application/x-dia-diagram
  :vectorial

:vectorial :can_be
  image/svg+xml
  image/vnd.dwg
```

```
  image/vnd.dxf                 # Autocad
  image/x-3ds                   # 3D Studio
  image/x-wmf
  image/cgm                     # http://www.w3.org/TR/NOTE-cgm-970618
  application/vnd.corel-draw

#============== VIDEO ===============#

:video :can_be
  video/mpeg
  video/quicktime
  video/vnd.vivo
  video/x-anim
  video/x-flc
  video/x-fli
  video/x-ms-asf
  video/x-ms-wmv
  video/x-ms-wvx
  video/x-msvideo
  video/x-nsv
  video/x-sgi-movie
  application/ogg
```

4 ARQUITETURA

Uma possível estratégia para se acrescentar informações hierárquicas aos tipos de arquivo é uma implementação nativa nos próprios sistemas operacionais ou ambientes gráficos, visto que estes são os responsáveis pela identificação de documentos e pela manutenção do catálogo de tipos de arquivo. Os aplicativos poderiam, apenas consultando o sistema, tirar proveito das vantagens do novo modelo de tipos sem a necessidade de intermediários, a exemplo do sistema operacional MacOS X e dos *Uniform Type Identifiers*.

Entretanto, não é possível predizer quando tais mecanismos serão incorporados: as limitações de um modelo tabular são significativas e existem há décadas, e apenas recentemente um sistema de tipos de arquivo hierárquico foi adotado na prática e possui apelo comercial suficiente para influenciar os responsáveis pelas demais soluções. Além disso, seria altamente desejável que as várias plataformas utilizassem o mesmo sistema, evitando discrepâncias nas classificações, o que é ainda mais difícil ocorrer.

A implementação da hierarquia de tipos de arquivo como um *middleware* resolve esses problemas disponibilizando um sistema portável e consistente entre os diversos sistemas operacionais. Ele pode ser visto como um prestador de serviços que, além de identificar documentos por meio de sua hierarquia interna, oferece serviços de classificação através da indexação de arquivos.

4.1 Visão geral

A arquitetura envolvendo o *middleware* pode ser analisada em camadas. O sistema operacional, compondo a camada de baixo nível, comporta o sistema de arquivos, que é acessado pelo mecanismo de indexação do *middleware*. Este, por sua vez, compõe a camada intermediária que fornece uma interface para consulta aos tipos e aos arquivos indexados. As aplicações, formando a camada de mais alto nível, utilizam os serviços do *middleware* por meio da interface (figura 4.1).

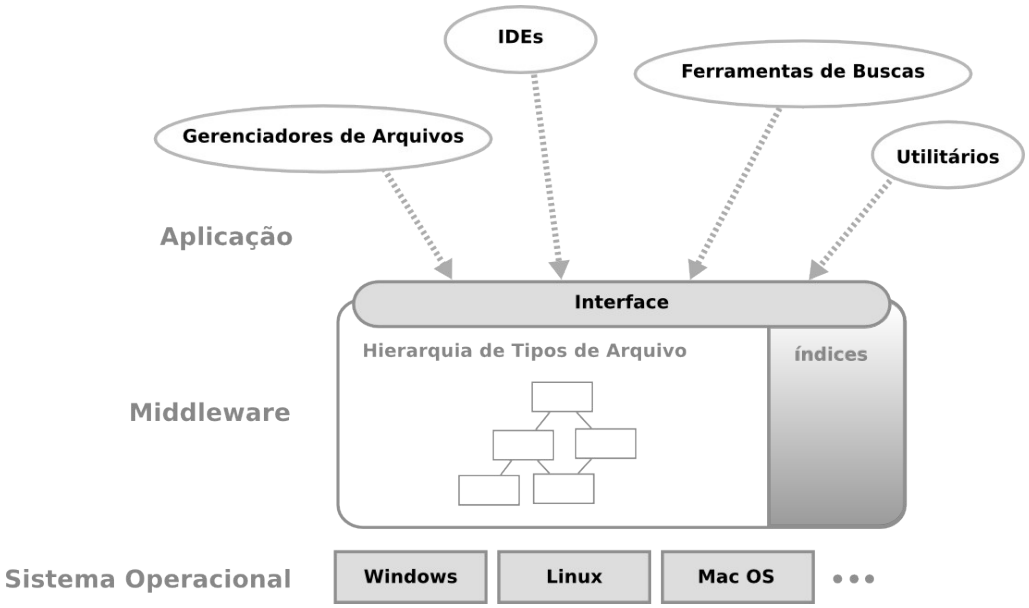

Figura 4.1: Visão geral da arquitetura do *middleware* de tipos de arquivos

Para que a implementação do *middleware* seja realmente portável, basta que a linguagem utilizada para desenvolvimento seja multiplataforma, ou que o *software* seja devidamente compilado para cada uma deles. Ou seja, não existem dependências de arquitetura de um sistema operacional específico. Quanto à hierarquia de tipos, esta é independente do sistema de tipos de arquivo do sistema operacional sendo utilizado. Ou seja, os mesmos tipos e relacionamentos contidos no *middleware* estarão disponíveis para todas as plataformas, sem alterações. Assim, quando o *middleware* é utilizado, o sistema de tipos nativo não é consultado: os serviços de tipos de arquivos são oferecidos exclusivamente pelo *middleware*.

Vale ressaltar que a sua utilização não impede que o sistema de tipos do sistema operacional seja também acessado. Por exemplo, um aplicativo poderia, quando não encontrasse um tipo de arquivo no *middleware*, acessar o sistema de tipos da plataforma subjacente, na esperança de obter alguma informação acerca do tipo (todavia deixando de ser multiplataforma, por amarrar chamadas ao sistema operacional), conforme ilustrado na figura 4.2.

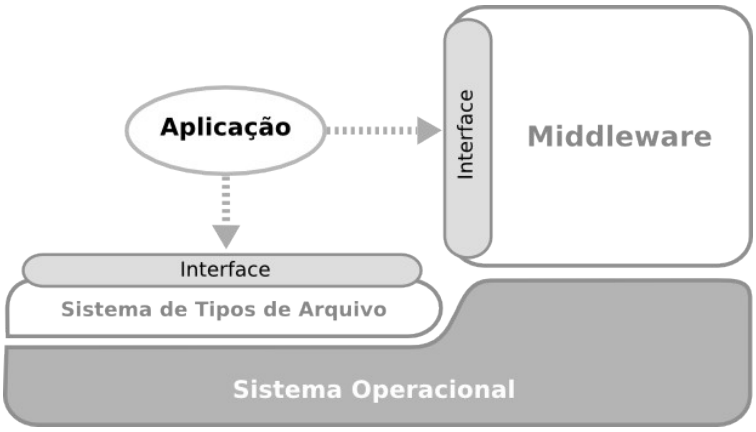

Figura 4.2: Uma aplicação pode acessar o *middleware* e o sistema de tipos nativo simultaneamente

A portabilidade do *middleware* pode torná-lo em uma alternativa interessante para IDEs ou aplicativos que implementam o seu próprio mecanismo de identificação de arquivos. Muitos desses softwares não podem depender do sistema operacional, seja porque o sistema de tipos varia em cada plataforma ou porque as APIs para acesso ao sistema de tipos não possuem as funcionalidades desejadas (como hierarquia de tipos e indexação de arquivos). Assim, é importante que a implementação do *middleware* ofereça um conjunto rico de funcionalidades, assim como um bom desempenho, procurando atender ao maior número de aplicações possíveis.

Para as aplicações voltadas ao usuário, o *middleware* é disponibilizado como uma biblioteca contendo uma API simples, da mesma forma que muitos outros middlewares, como sistemas de distribuição de mensagens ou bibliotecas gráficas. Através de sua interface, aplicações podem customizar o repositório de tipos de arquivos adicionando e excluindo tipos e associações, de modo a garantir que tipos de arquivos essenciais à aplicação estejam de fato presentes, ou a expandir os tipos classificadores no intuito de atender uma classificação alternativa.

4.2 Sincronização com os sistemas subjacentes

É importante que o *middleware* comporte todos os tipos do sistema subjacente para evitar que as aplicações necessitem consultá-los em dois locais diferentes. Para tanto, ele deve primeiramente adicionar todos os tipos de arquivos do sistema e suas associações com aplicativos para dentro de sua estrutura. Se o tipo sendo importado já existir na hierarquia, nada é alterado (exceto se o *middleware* suportar associações entre aplicativos e tipos de arquivos; nesse caso, as associações são importadas). Porém, se não o contiver, o tipo e eventualmente suas associações são adicionadas, mas destacados da hierarquia principal por não se conhecer sua natureza, conforme ilustrado na figura 4.3. Posteriormente, é possível que os tipos isolados sejam conectados a outros tipos quando, por exemplo, for feita uma atualização do *middleware*.

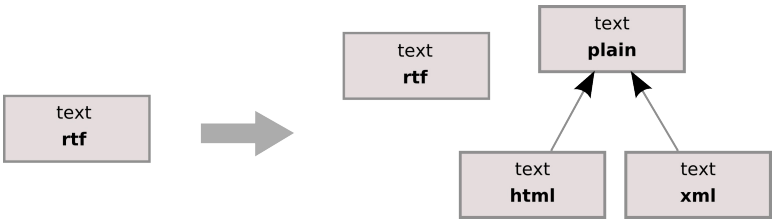

Figura 4.3: Importação de um tipo inexistente no *middleware*

Outra questão importante é a sincronização dos tipos presentes no sistema de arquivos subjacente com aqueles existentes no *middleware*. Muitos aplicativos, na instalação, registram novos tipos de arquivos ou se associam aos já existentes. Outras vezes, tipos são desassociados ou mesmo excluídos do sistema de tipos. Isto ocorre, por exemplo, quando programas são desinstalados. É preciso, portanto, manter o *middleware* em sincronia com o sistema, interceptando as modificações no sistema de tipos e atualizando também o *middleware*. A implementação de tal mecanismo difere bastante entre as plataformas, pois envolve APIs e arquivos específicos. No Linux, por exemplo, cada ambiente possui um sistema de tipos diferente. Nesta etapa de sincronização os tipos e aplicativos são agregados à hierarquia principal ou adicionados como nodos isolados da mesma forma que na importação inicial.

Além da sincronização de tipos de arquivos, é necessário também que os índices do *middleware* estejam atualizados com os metadados dos arquivos do sistema. A indexação de metadados não é demorada, conforme indicam os testes realizados com o protótipo, mas mesmo que leve alguns segundos, o processo se torna inviável se a cada utilização dos serviços seja necessário uma nova indexação. É necessário, portanto, interceptar sempre que um arquivo for criado, renomeado, movido ou excluído.

Porém a implementação varia significativamente de acordo com a plataforma. No Linux, por exemplo, essas operações são implementadas na biblioteca *glibc*. O mecanismo exigido pelo *middleware* é semelhante ao da biblioteca *libtrash*, que modifica a função `unlink()` da biblioteca *glibc*, responsável pela exclusão de arquivos. Em vez de excluir completamente os metadados associados ao arquivo, ele é movido para um diretório provisório, que serve como lixeira. No caso do *middleware*, o índice deve ser atualizado assim que qualquer uma das operações sobre arquivos citadas anteriormente seja executada interceptando as funções `link()`, `unlink()`, `rename()` (TANENBAUM, 2001).

No Windows o princípio para manter os arquivos indexados sincronizados é o mesmo. Um serviço residente intercepta chamadas de função aos serviços básicos da Windows API, mantendo os índices atualizados. Tanto no Windows quanto no Linux, se algumas operações sobre arquivos não forem capturadas e refletidas no *middleware*, o conteúdo deste permanecerá desatualizado. Embora isso não comprometa o funcionamento do *middleware*, pode se tornar inconveniente para o usuário quando for solicitada uma operação sobre um arquivo desatualizado. O efeito é o mesmo do

aplicativo Google Desktop, que indexa o conteúdo de arquivos para que o usuário possa fazer buscas rápidas. Se o serviço não estiver acionado no momento em que um arquivo é excluído, por exemplo, ele ainda será listado nas buscas. Da mesma forma, se um arquivo for criado ou copiado e o serviço de indexação não estiver em funcionamento, seu conteúdo não estará indexado, e portanto não será incluído nas buscas.

4.3 Interface

Embora a interface oferecida pelo *middleware* possa variar em cada implementação, existe um conjunto de operações que deverá estar presente em todas elas, incluindo a adição e exclusão de tipos de arquivo e consultas a respeito da hierarquia (como a obtenção dos ascendentes ou descendentes de um determinado tipo). Afinal, essas são as operações básicas necessárias para a realização de pesquisas sobre os tipos.

Além disso, se o *middleware* implementar o mecanismo de indexação de arquivos, os métodos de acesso apropriados deverão ser disponibilizados. Também seria desejável que as funcionalidades oferecidas pela interface fossem independentes do sistema operacional para facilitar o desenvolvimento de aplicações baseadas no *middleware* e conseqüentemente ampliando a adoção da solução entre os desenvolvedores.

4.4 Tipos de arquivo como tags

Existe um tipo de aplicação que necessita de uma arquitetura mais complexa. São aquelas que organizam os documentos de uma maneira diferente da árvore de diretórios, e por isso necessitam indexá-los para um rápido acesso. Uma das principais aplicações desse grupo é a classificação de documentos segundo seu tipo, sendo cada um deles representado por uma pasta contendo seus tipos descendentes (essa aplicação é descrita na seção 5.3.1). Porém, existem outros exemplos de aplicações baseadas nessa arquitetura, como contagem rápida de arquivos, classificação de arquivos por tamanho ou data (independentemente de sua localização física) e localização de arquivos temporários residuais.

O mapeamento de metadados de arquivos, como localização, nome, tamanho e data para entidades lógicas de mais alto nível requer uma estrutura propícia de armazenamento que funcione como um índice, por motivos de desempenho. Caso contrário, na busca por documentos de determinado tipo seria necessário fazer uma varredura percorrendo todos os arquivos, identificando cada um deles e selecionando-o se fosse o caso, o que certamente inviabilizaria o processo. A arquitetura de um mecanismo de recuperação segundo os critérios mencionados, portanto, é muito similar à arquitetura de *tags* para *desktop*, como aquela adotada pelo projeto Leaftag descrito na subseção 2.3.1. Tal semelhança permite que um tipo de arquivo seja visto como uma *tag*. Essa abordagem é explorada no *middleware* a fim de mesclar o conceito de tipo de arquivo com *tags*, potencializando as *tags* para *desktop*.

Nessa arquitetura cada tipo de arquivo possui uma *tag* associada, com exceção dos nodos classificadores. Isso ocorre porque os nodos classificadores não possuem arquivos

diretamente associados. Eles podem, entretanto, ser totalmente inferidos a partir dos tipos descendentes. Para o usuário final, é como se os próprios nodos classificadores estivessem associados aos arquivos. Uma diferença existente entre as *tags* comuns e *tags* de tipos de arquivo é que um arquivo deve pertencer a apenas uma única *tag* de tipo de arquivo. Sendo assim, a arquitetura de *tags* comportando tipos de arquivos se assemelha mais à figura 4.4 do que a figura 2.6, contendo *tags* comuns.

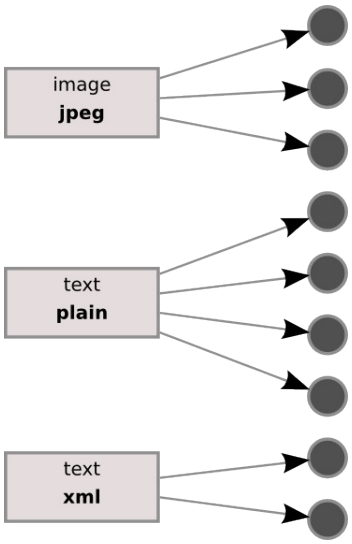

Figura 4.4: Cada tipo de arquivo não-classificador é um índice

Para fazer uso da estrutura hierárquica dos tipos de arquivo é preciso que os relacionamentos de herança entre os tipos sejam armazenados. Como as *tags* definidas pelo usuário não possuem suporte à herança (não há qualquer relacionamento entre *tags*, apenas entre *tags* e arquivos), o grafo de tipos de arquivo precisaria ser implementado como um serviço paralelo. Por outro lado, se a implementação das *tags* suportassem grupos de outras *tags*, não haveria essa necessidade. Grupos adicionam um pouco de compexidade às *tags*, mas são um recurso bastante útil. Por exemplo, os usuários poderiam desejar juntar as *tags gramado*, *canela*, *caracol* e *natal luz* em uma outra *tag* de maior abrangência, chamada *viagem para a serra*. Esse recurso seria suficiente para suportar a hierarquia de tipos de arquivo conforme apresentada na seção 3.2, pois funcionam de forma análoga: os tipos de arquivo são grupos contendo seus descententes (figura 4.5). É essa a estrutura que é empregada pelo *middleware*. Em vez do usuário definir as *tags* de cada grupo, porém, o próprio sistema faria isso automaticamente de acordo com a hierarquia, a exemplo do que já ocorre com alguns emblemas no ambiente Gnome (GNOME, 2006).

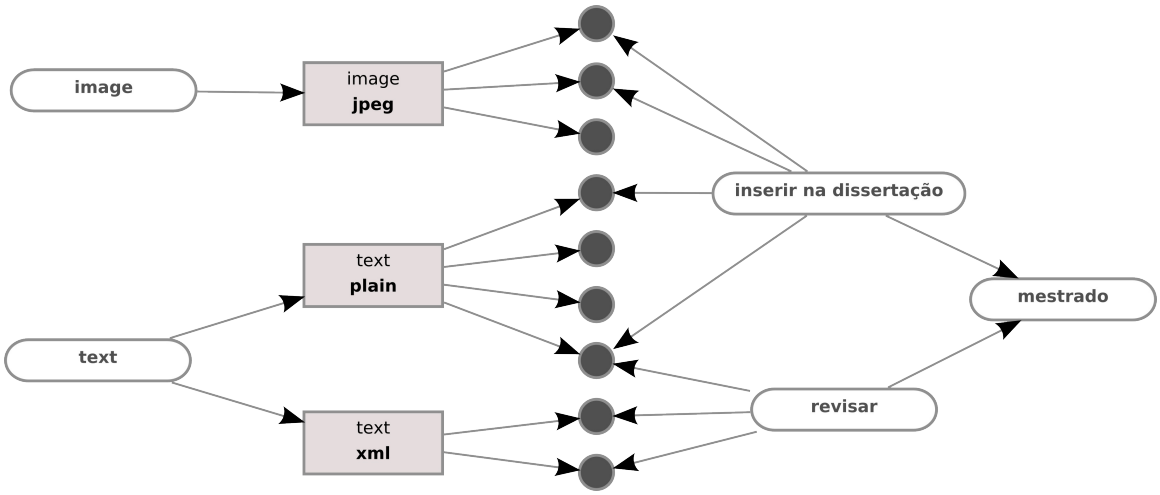

Figura 4.5: tipos classificadores e *tags* possuem arquitetura muito semelhante

A combinação de *tags* definidas pelo usuário com *tags* de tipo de arquivo habilita novas opções de organização de arquivos, como visualizar os arquivos pertencentes a determinada *tag* em tipos de arquivos. Por exemplo, em um cenário fictício, com a finalidade de fazer uma cópia de reserva (*backup*) de alguns documentos importantes, um usuário marca com *tags* imagens JPEG, documentos do Microsoft Word e do Microsoft Excel, algumas anotações em texto puro e algumas músicas em formato MP3. Utilizando um ambiente que suporta *tags*, o usuário rapidamente encontra os arquivos marcados informando a *tag* desejada. O resultado da busca é apresentada na forma de uma lista contendo todos os documentos marcados com aquela *tag*. Para obter uma visão mais organizada dos arquivos marcados, o usuário pode, neste momento, optar por visualizar os documentos de acordo com seu tipo. Assim, seriam exibidas as pastas *images*, *audio*, *office* e *text* juntamente com o número de arquivos em cada uma. Um protótipo com esses recursos, o Nautic, foi implementado para este trabalho e é apresentado na seção 6.2.

5 APLICAÇÕES

Uma hierarquia de tipos de arquivos implementada como um *middleware* possui muitas aplicações, pois qualquer *software* voltado para a manipulação de arquivos, como ambientes gráficos, IDEs e gerenciadores de arquivos podem tirar proveito das relações de herança existentes. Este capítulo apresenta as principais aplicações, divididas em dois grupos: aquelas dependentes apenas do *middleware* e outras que necessitam também de um índice contendo os arquivos envolvidos. Vale ressaltar, que, por fornecer serviços genéricos, certamente as aplicações não se restringem àquelas descritas neste capítulo.

5.1 Ações sobre arquivos

Os ambientes gráficos disponibilizam um conjunto de ações que podem ser realizadas sobre os documentos, como *abrir*, *visualizar*, *enviar*, *imprimir*, sendo a operação *abrir* geralmente a ação padrão. O suporte à múltiplas ações permite detalhar a operação que será realizada com o documento. Por exemplo, enquanto que a ação padrão para um arquivo no Windows é *abrir*, se quisermos imprimir o documento `relatório.doc`, criado com o aplicativo Microsoft Word, podemos clicar com o botão direito do mouse sobre seu ícone para chamar o menu de contexto e selecionar *imprimir*. Esta facilidade é possível porque documentos do tipo *Documento do Microsoft Word* possuem a ação associada *imprimir*.

Infelizmente, apenas um pequeno conjunto das possíveis ações entre aplicativos e tipos de arquivos são conhecidas. Na ausência de uma estrutura de mapeamanto mais rica entre tipos de arquivos e aplicativos, muitas ações são perdidas, deixado ao usuário a tarefa de indicar manualmente um aplicativo adequado para tratar o documento.

Para exemplificar essa limitação, considere que se queira imprimir o arquivo texto `lista.txt` no sistema operacional Microsoft Windows XP. Para isso, deve-se chamar o menu de contexto do ícone e selecionar *imprimir*. O sistema então se encarrega de executar o aplicativo adequado, como o Bloco de Notas, passando a imprimir o documento. Para um documento XML, porém, a ação de impressão não está disponível, embora todo documento XML seja também um documento texto. Essa restrição é uma característica da ausência de hierarquia entre tipos de arquivos.

As relações de herança entre tipos de arquivos propagam as ações de um tipo aos seus descendentes, aumentando significativamente o número de ações disponíveis. Atualmente, em todos os sistemas de arquivos, com exceção dos UTIs, cada ação é

associada diretamente a um conjunto de tipos de arquivo. Supondo que um editor de textos simples associe as ações *abrir* e *imprimir* ao tipo de arquivo correspondente às extensões `.txt` e `.text`, quaisquer outras extensões, incluindo aquelas cujos tipos também são texto, não são afetadas. Documentos HTML, XML ou o código-fonte de qualquer linguagem de programação não tirarão proveito das novas ações, obrigando os aplicativos a listarem uma grande quantidade de extensões ou tipos de arquivos em configurações de mapeamento.

É claro que existe a alternativa de abrir ou imprimir um arquivo texto a partir de um editor, mas o objetivo das ações é justamente fornecer uma forma mais simples de se trabalhar com documentos, evitando o trabalho, neste caso, de executar o editor de textos e selecionar o arquivo a ser aberto. A figura 5.1 apresenta as associações, assim que o sistema Windows XP é instalado, entre dois aplicativos e os tipos de arquivo Text, XML, HTML e XHTML. Os demais tipos apresentados na figura não possuem associações, de forma que seriam representados no sistema operacional pelo ícone de documentos desconhecidos. Além disso, não possuiriam nenhuma ação disponível, nem mesmo a ação *abrir* (um aplicativo deveria ser manualmente indicado para abrir o documento).

Já em um sistema contendo os mesmos tipos, mas comportando informação de herança entre recursos, todos os tipos de arquivos poderiam ser editados ou impressos pelo aplicativo Notepad, e o Internet Explorer poderia ser usado para visualizar o conteúdo de arquivos SVG, ou qualquer outro tipo baseado em XML (figura 5.2). Porém, em vez de exibir uma imagem, o conteúdo de documentos SVG seriam apresentados no Internet Explorer como um texto estruturado, da mesma forma que documentos do tipo XML. Ou seja, quando um tipo de arquivo não possui associações, aplicativos associados aos tipos ascendentes poderiam ser empregados, dando preferência aos mais especializados. No exemplo, o Internet Explorer é escolhido para visualizar documentos XML, e não o Notepad.

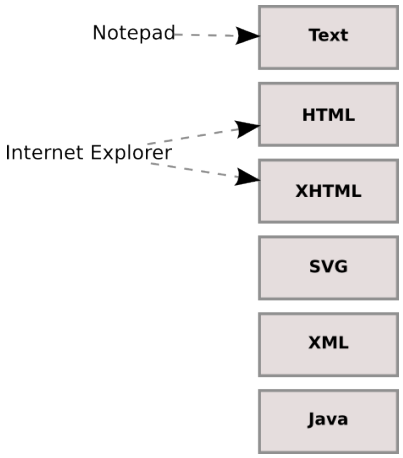

Figura 5.1: Associações entre aplicativos e tipos de arquivo no Windows

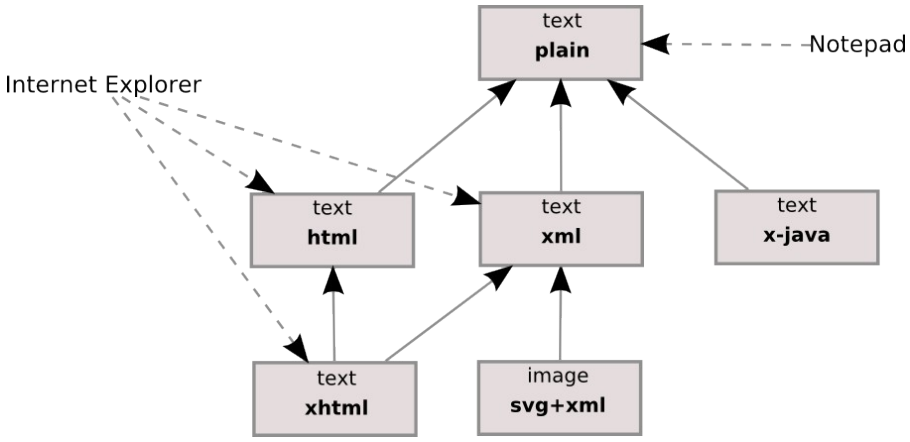

Figura 5.2: Associações entre aplicativos e tipos de arquivo utilizando uma estrutura hierárquica

5.2 Extensão em sistemas de busca

É indiscutível a importância que motores de busca de palavras-chave possuem hoje em dia, como aqueles oferecidos pelo Google e o Yahoo. Eles são utilizados diariamente por milhões de pessoas para os mais variados fins, se tornando em ferramenta indispensável para inúmeras tarefas. Muitas vezes, porém, é difícil encontrar o que se procura, principalmente devido à ambigüidade das línguas naturais e à forma como operam esses mecanismos de busca. Palavras com vários significados ou aplicáveis a muitos contextos dificultam a pesquisa. Essas limitações são bem conhecidas, e fazem parte de uma área bastante ativa de estudos.

A Web Semântica (SHADBOLT, Nigel; BERNERS-LEE, Tim; HALL, Wendy; 2006), um projeto ambicioso e de longo prazo conduzido pelo W3C, procura resolver ambigüidades e definir significados precisos através de instrumentos formais e utilização de tecnologias fortemente baseadas em metadados, com identificações únicas para os recursos. É uma abordagem radicalmente diferente quanto a localização de recursos, relegando a um segundo plano a descoberta de informações por heurística ou análise de conteúdo. Mas exige que aplicações e sistemas geradores de informação sejam responsáveis por anexar metadados à informação gerada. Infelizmente, deve demorar anos, segundo os próprios coordenadores do projeto, para que uma grande parte das informações disponíveis na Web sejam devidamente identificadas com metadados. Enquanto isso, a expansão dos atuais sistemas de busca com uma hierarquia de tipos de documentos pode auxiliar na restrição de contextos e desambigüação de termos, ao custo de um esforço pequeno.

Além de palavras-chave, os motores de busca interpretam comandos especiais que podem auxiliar na restrição de escopo, apresentando apenas os resultados mais interessantes para o usuário. No Google, alguns desses comandos, embora desconhecidos para a maioria das pessoas, são bastante úteis e eficazes. A expressão `filetype:` seguida de uma extensão de arquivo, por exemplo, nos retorna apenas documentos com

aquela extensão. Assim, ao se fazer uma busca por `java filetype:pdf`, será apresentada uma lista de documentos PDF contendo a palavra *Java*.

Se os sistemas de busca empregassem hierarquia de extensões, seria possível fazer pesquisas restritas por grupos de tipos. Ao executar uma busca com `type:text java`, por exemplo, seriam retornados apenas resultados de documentos do tipo texto (excluindo formatos como PDF e Microsoft Word) contendo a palavra *java*. Para demonstrar a utilidade dessa funcionalidade, considere o seguinte exemplo. Em um estudo comparativo entre diversas linguagens de programação, se deseja anexar o código fonte dos algoritmos que calculam os números de Fibonacci implementados em cada uma das linguagens. Para isso, seria possível fazer buscas no Google pelas expressões `fibonacci java`, `fibonacci c++`, `fibonacci php`, e assim por diante. Porém, existem dois inconvenientes nessa busca. Primeiro, não seriam retornadas apenas listagens de programas, mas também páginas HTML com comentários, mensagens de fóruns e outros textos. Esse problema poderia ser resolvido utilizando a palavra `filetype`. Mas nesse caso seriam necessárias várias pesquisas. Além disso, a extensão de arquivos de cada linguagem de programação teria de ser conhecida: `java` para Java, `cs` para C#, `rb` para Ruby, etc.

Se todas as extensões das principais linguagens de programação pudessem ser expressas em uma única palavra, assim como a hierarquia de tipos de arquivos apresentada no capítulo 3, a busca poderia ser resumida a algo do tipo: `fibonacci type:code`. Pesquisas sobre documentos XML também se beneficiariam bastante com esse tipo de busca, pois muitos documentos XML possuem outras extensões além de `.xml`, como `.xhtml`, `rdf`, `rdfs`, `foaf`, `rss`, incorrendo no mesmo problema do exemplo anterior se realizado com os sistemas de buscas atuais.

Adicionalmente, em buscas por arquivos locais em sistemas *desktops*, seria de grande utilidade que fosse possível simplesmente dizer que se procura por imagens, vídeos ou músicas, preferencialmente por meio de uma interface gráfica, como simples botões. Essa funcionalidade também poderia ser implementada com uma hierarquia de tipos de arquivos. Em vez disso, atualmente é necessário procurar pelos arquivos utilizando caracteres coringa e formatações que muitas vezes mesmo usuários experientes desconhecem.

5.3 Aplicações envolvendo indexação de arquivos

Algumas aplicações necessitam de serviços de indexação e cache de metadados, que, em conjunto com o grafo de tipos, habilitam uma nova categoria de serviços.

5.3.1 Gerenciadores de arquivos

Uma das principais aplicações do *middleware* é a potencialização dos gerenciadores de arquivos através da classificação de documentos em grupos semânticos. Além da visualização de arquivos e diretórios segundo sua organização física, os

gerenciadores de arquivos, utilizando a hierarquia de tipos, poderiam apresentar grupos de imagens, documentos de escritório, código-fonte, músicas e filmes, conforme definidos pelos tipos classificadores ou mesmo pelos tipos comuns.

É claro que os arquivos visualizados devem ser mapeados para sua posição física, realizada pelo mecanismo de indexação do *middleware*. Afinal, em última instância, eles devem ser localizados para ser acessados. Mas o acréscimo de processamanto e memória exigido pelo *middleware* é pequeno se comparado com os benefícios envolvidos.

O conceito de *pasta* empregada pelos gerenciadores de arquivos atuais pode ser reaproveitada nos grupos de tipos. Cada pasta pode representar um tipo classificador ou um tipo de arquivo comum, e as subastas, os tipos descendentes. Como os tipos de arquivos do *middleware* são praticamente os mesmos em vários computadores (ao contrário das pastas e arquivos em geral), cada tipo pode ser identificado por um ícone diferente. Ícones são identificados pelo usuário muito mais facilmente do que palavras, de forma que, além do ganho de tempo, a utilização de ícones diferenciados traz maior conforto ao usuário.

Outro recurso interessante que pode ser adicionado aos gerenciadores de arquivos é a informação de quantidade de arquivos nas pastas. Atualmente, não há como saber quantos arquivos existem nas pastas, exceto pela contagem recursiva, geralmente oferecida por um menu de contexto. Os índices possibilitam uma contagem de arquivos muito mais rápida, que pode ser empregada na visualização das pastas. Por exemplo, o número de arquivos pode ser exibido abaixo do ícone de cada tipo, ou ainda o tamanho da pasta pode variar de acordo com a quantidade de arquivos.

O Nautic, um gerenciador de arquivos experimental, implementa muitos dos recursos apresentados, incluindo variação do tamanho do ícone conforme a quantidade de arquivos e ícones individuais para cada tipo. Maiores detalhes sobre as funcionalidades e implementação do aplicativo são apresentados na seção 6.2.

Enfim, vale ressaltar que a classificação por tipos não procura substituir a classificação física, mas sim complementá-la. Em gerenciadores de arquivos, o usuário deve sempre ter a opção de organizar seus documentos com a estrutura que desejar. A classificação semântica, portanto, deve ser vista como mais uma opção de visualização de arquivos a disposição do usuário.

5.3.2 Repositório de documentos

Muitas vezes a localização dos arquivos sendo manipulados não importa ao usuário. Ocorre então que a escolha de um local para armazenamento só é efetuada porque é necessária. Alguns aplicativos procuram evitar esse trabalho, como o navegador Mozilla Firefox, permitindo que se indique um local padrão no qual os arquivos baixados da Internet sejam armazenados.

Porém, conforme o número de documentos cresce e vão sendo armazenados em diferentes lugares, se torna cada vez mais difícil a localização dos arquivos pelo usuário com o passar do tempo. Adicionalmente, quando muitos arquivos são armazenados em

um só local, como no *desktop*, eles acabam sendo copiados para outros lugares de qualquer forma.

Esse problema pode ser minimizado utilizando a hierarquia de tipos e índices de metadados de arquivos. Como a localização dos arquivos segundo a classificação em tipos independe da organização física, o usuário não precisaria, na maior parte dos casos, distribuir manualmente seus arquivos nas pastas, pois estes já seriam classificados na própria visualização. Para ser melhor avaliada o impacto dessa mudança, porém, são necessários testes com uma quantidade razoável de usuários reais.

6 IMPLEMENTAÇÃO DO SOFTWARE EXPERIMENTAL

O desenvolvimento de um *software* experimental permite a análise de fatores propositalmente não considerados ou não previstos no plano teórico. Neste caso, onde se propõe também uma alternativa na forma como documentos são manipulados e visualizados pelos usuários, é importante que a proposta seja de fato implementada e validada, embora testes rigorosos e estudos aprofundados de usabilidade e desempenho estejam fora do escopo deste trabalho.

A implementação experimental do *middleware* não comporta todas as funcionalidades apresentadas, como, por exemplo, a sincronização de tipos e descritores de arquivos com o sistema. Mas os principais componentes, como a hierarquia de tipos e índices de metadados de arquivos foram desenvolvidos, com o objetivo de demonstrar a utilidade da proposta.

Foi implementado também um gerenciador de arquivos que utiliza os recursos oferecidos pelo *middleware*, o Nautic. Suportando diferentes formas de visualização e contendo diversas configurações, o Nautic agrupa os arquivos em tipos, visualizados como pastas ou ícones individuais. Entre outras funcionalidades, é possível visualizar pastas com tamanhos diferentes conforme a quantidade de arquivos e navegar pela hierarquia de tipos.

A linguagem escolhida para o desenvolvimento dos softwares foi Ruby (http://www.ruby-lang.org), ideal para prototipação por sua produtividade e expressividade. Ruby é uma linguagem dinamicamente tipada, orientada a objetos e de aprendizado relativamente rápido. Para a construção da interface gráfica do Nautic foi utilizada a biblioteca GTK2 (http://www.gtk.org) através de extensões para a linguagem Ruby presentes na biblioteca Ruby/GTK2. Tanto a linguagem Ruby quanto a biblioteca GTK são multiplataforma, o que facilita o porte do protótipo para os principais sistemas operacionais, necessitando de pouca alteração no código fonte (as alterações são aquelas que dizem respeito ao sistema de tipos e sistema de arquivos subjacente).

6.1 Middleware

Apesar de ser um protótipo, o *software* desenvolvido possui uma arquitetura de implementação de nível de produção, distribuindo suas funcionalidades em classes e módulos apropriados. A hierarquia de tipos de arquivo com suas respectivas funções compõem um módulo, enquanto que a estrutura que comporta os índices de metadados de arquivos e métodos relacionados fazem parte de outro.

A hierarquia de tipos de arquivo é armazenada em uma estrutura genérica, um grafo direcionado. Embora seja utilizado atualmente apenas herança, é possível criar qualquer tipo de relacionamento entre os tipos de arquivo. No entanto, como o *middleware* não possui conhecimento sobre a semântica dos relacionamentos, eventuais funções de mais alto nível devem ser implementadas para os novos tipos. Por exemplo, se poderia associar aplicativos aos tipos de arquivos, representando a ação *abrir* através de um novo tipo de relacionamento. Mas para se obter uma funcionalidade como saber se um aplicativo pode abrir determinado tipo de arquivo considerando também a hieraquia de tipos, seria necessário implementar as funções de mais alto nível, que se baseariam nos dois tipos de relacionamentos.

Sobre a implementação da hieraquia de tipos de arquivo, foi desenvolvido um mecanismo que comporta os índices de metadados de arquivos. Inicialmente, os arquivos são indexados através da função `collect(`*caminho*`)`, onde *caminho* é um parâmetro informando o diretório que deve ser indexado (todos os arquivos em todos os subdiretórios são recursivamente indexados). No Nautic, esse parâmetro pode ser alterado a qualquer momento, disparando uma nova indexação.

Nessa indexação os arquivos são analisados recursivamente na árvore de diretórios e seus nomes e localizações são adicionados a um repositório, classificados pela extensão de seus nomes. O repositório de nomes de arquivos é um *hash* (um índice, portanto) onde as chaves são os tipos e os valores são vetores (*arrays*) de *strings* contendo os nomes e as localizações dos arquivos (figura 6.1).

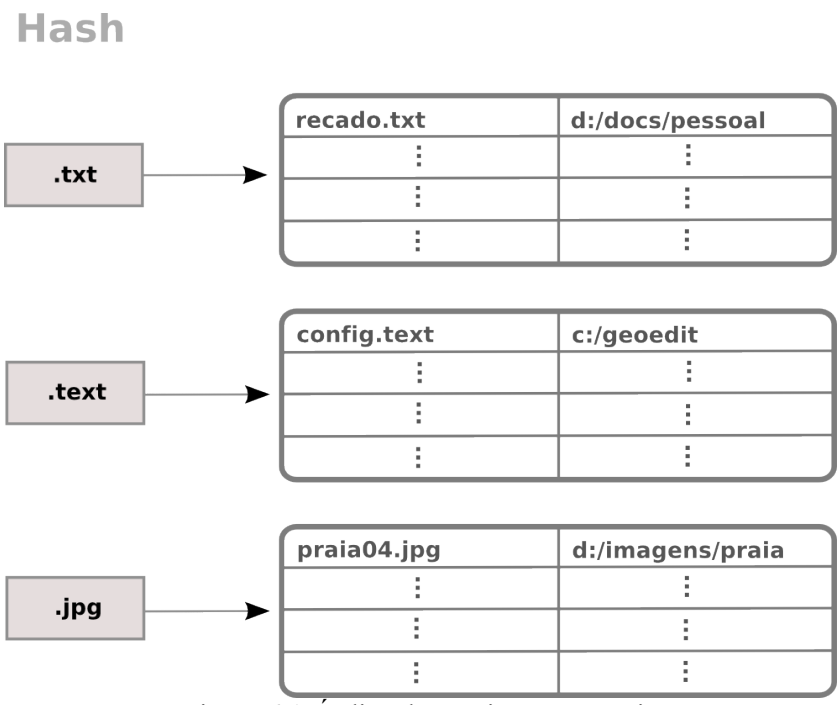

Figura 6.1: Índice de arquivos no Nautic

As extensões do *hash* de tipos são relacionados no Nautic com as extensões dos tipos conhecidos (proveniente do arquivo com os tipos MIME), obtendo-se então o tipo MIME de cada arquivo. Se houver arquivos com as extensões `.txt` e `.text`, por exemplo, eles serão adicionados em *arrays* diferentes, mas serão relacionados ao mesmo tipo de arquivo `text/plain`. É possível, portanto, localizar diretamente todos os arquivos de um determinado tipo do grafo. Como apresentado na seção 3.2, os tipos de arquivo e relacionamentos são lidos de dois arquivos em formato de texto simples.

6.2 Nautic

O Nautic é um aplicativo que explora a hierarquia de tipos de arquivos com o objetivo de oferecer uma organização alternativa de documentos para o usuário. Trata-se de um gerenciador de arquivos (assim como o Windows Explorer e o Nautilus, do ambiente gráfico Gnome) que classifica os arquivos segundo a hierarquia de tipos, e não de acordo com a usual árvore de diretórios. As pastas, em vez de diretórios, representam tipos de arquivos, ou seja, tipos MIME e tipos classificadores disponibilizados pelo *middleware*. Procurando ser intuitivo e prático, no Nautic pode-se navegar pelas pastas como se estivesse navegando pela árvore de diretórios. Quanto maior a profundidade da árvore, mais específico é o tipo de arquivo. Nessa estrutura são mesclados nodos classificadores, contendo apenas outras pastas, e tipos finais, contendo principalmente arquivos.

Vale ressaltar que trata-se de um protótipo, um *software* experimental. Dessa forma, são oferecidas talvez mais funcionalidades e opções do que em um *software* comercial voltado para o usuário final. A principal preocupação durante a implementação foi a possibilidade de experimentar e testar a visualização alternativa mencionada. Da mesma forma, alguns recursos, como sincronização de operações em arquivos com o sistema, não foram implementados.

Embora fosse razoável em um produto final definir um diretório inicial padrão a ser apresentado assim que o gerenciador abrisse, para facilitiar os testes o Nautic espera que seja digitado um caminho para a exibição das pastas. A figura 6.1 apresenta a tela inicial do Nautic assim que o programa é executado, e a figura 6.2 depois de ter sido especificado um diretório no campo *Collect from*.

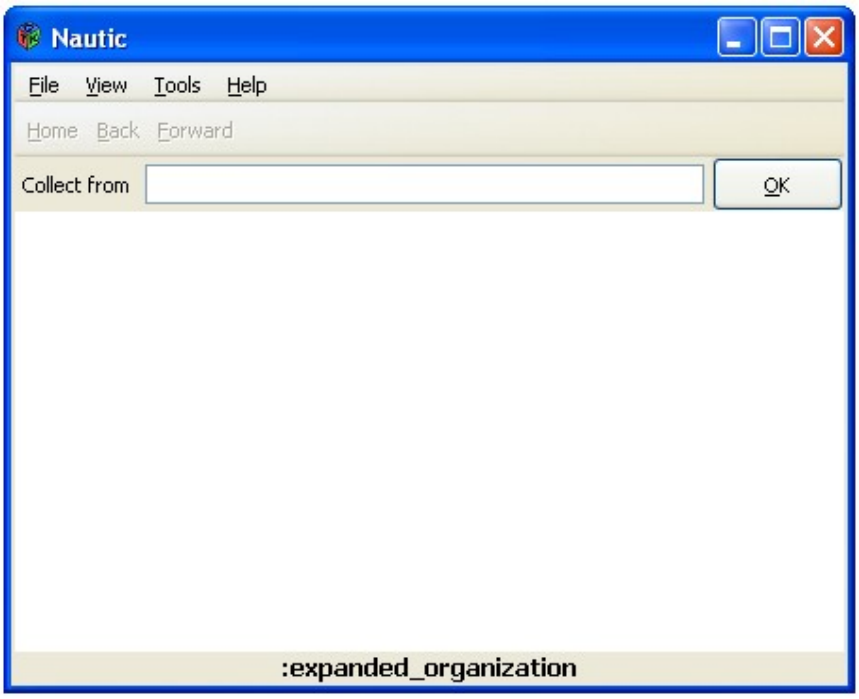

Figura 6.2: Tela inicial do Nautic

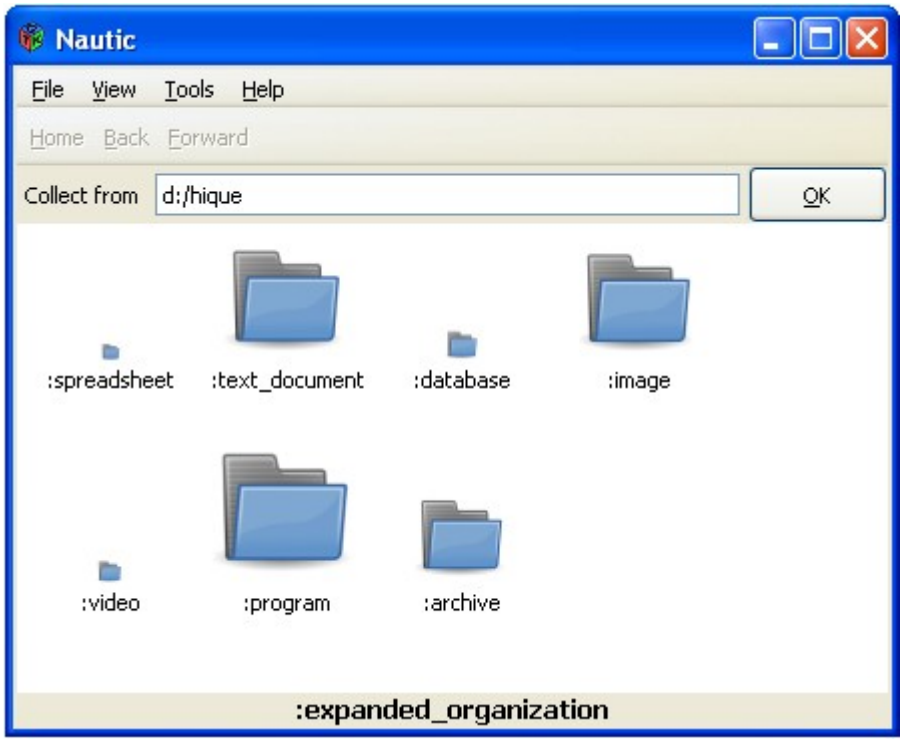

Figura 6.3: Exemplo da distribuição de tipos de um diretório de dados típico

Aqui apenas são apresentadas as pastas que contém arquivos. Mas além daquelas apresentadas na figura 6.2, tipos classificadores como *audio* e *backup* também fazem parte dos tipos iniciais do Nautic, que emprega como nodo raiz *expanded_organization*. A figura 6.3 mostra todos os tipos de *expanded_organization*, inclusive aqueles que não contém arquivos. Essa visualização é obtida por meio de uma opção de exibição do Nautic.

É importante mencionar que o grafo de tipos de arquivos não possui uma raiz. Existem dois tipos classificadores principais, que não estão contidos em nenhum outro tipo e que podem ser considerados como os tipos mais abrangentes: *compact_organization* e *expanded_organization*, este último utilizado como raiz no Nautic por possuir menos tipos classificadores intermediários, como *office*. Outros tipos classificadores principais poderiam ser facilmente adicionados.

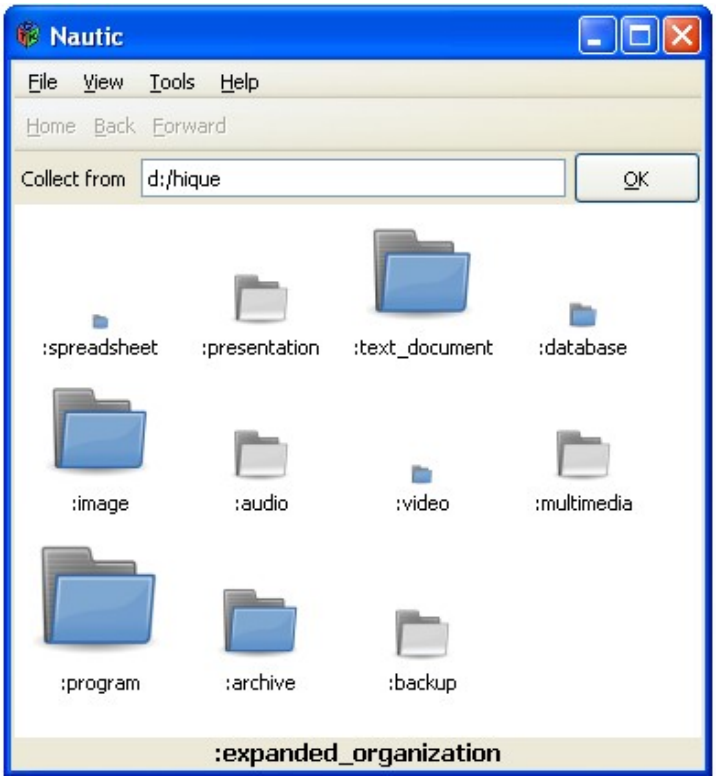

Figura 6.4: Nautic exibindo todos os tipos de arquivos de `expanded_organization`, inclusive aqueles que não possuem arquivos

Outras opções de visualização estão disponíveis, como pastas com um mesmo tamanho (exibição padrão dos gerenciadores usuais), contagem do número de arquivos do tipo (figura 6.4) e abreviação nos nomes dos tipos MIME. Em vez de seguir uma escala linear conforme a quantidade de arquivos, o tamanho das pastas segue uma escala

logarítmica. Assim, para poucos arquivos, a variação de tamanho é maior. A idéia é passar ao usuário uma aproximação do número de arquivos sem sacrificar a visualização.

Já a abreviação de nomes dos tipos omite o tipo na dupla *tipo/subtipo* nos nomes, já que o tipo em questão é a própria pasta atual. Também são omitidos os prefixos `x-` e `vnd.`, relativos respectivamente a extensões (tipos que não foram definidos pelo IETF) e a tipos específicos de um aplicativo, informações que não são relevantes para o usuário final. O objetivo desse recurso é facilitar a rápida identificação dos tipos exibidos.

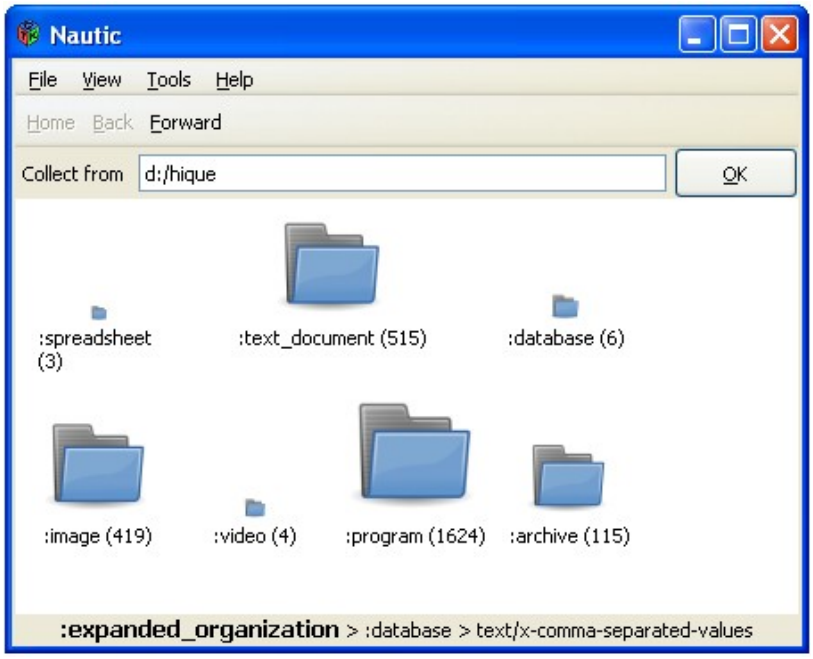

Figura 6.5: Exibição da contagem do número de arquivos de cada tipo. O tamanho da pasta não possui uma relação proporcional com a quantidade de arquivos, mas sim logarítmica.

Os arquivos são exibidos conforme ilustrado na figura 6.5. Neste caso, os arquivos são do tipo `text/x-perl`, como pode ser observado no caminho presente no rodapé da janela. Se o usuário clicar em um arquivo, o aplicativo associado é disparado ou o arquivo é executado, caso seja executável.

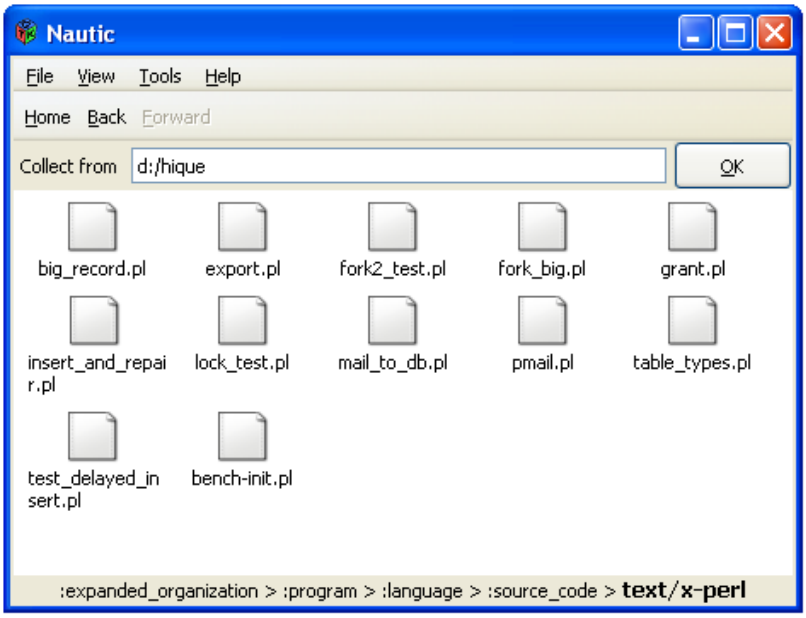

Figura 6.6: Arquivos do tipo `text/x-perl`

Como mencionado na seção 4.2, idealmente o processo de coleta de arquivos deveria ocorrer apenas uma vez, indexando todos os arquivos do sistema. As posteriores operações de criação, alteração de nome e exclusões de arquivo do sistema necessitariam ser interceptadas para também modificar o índice gerado na coleta, mantendo-o sempre atualizado. Isso evita que o usuário passe um tempo considerável aguardando a coleta de arquivos toda vez que o gerenciador de arquivos baseado em tipos for disparado. No Nautic, porém, o índice de arquivos é apenas modificado quando efetuada uma coleta.

A duração do processo de indexação depende principalmente do sistema operacional, configuração de *hardware* do computador, conteúdo da *cache* de metadados do sistema de arquivos e a quantidade de arquivos. No geral, o Linux (utilizando *raiserfs*) levou menos tempo para a atividade do que o Windows (empregando NTFS) nos testes simples realizados. Adicionalmente, as coletas subseqüentes no Linux se demonstraram muito mais rápidas, em razão da eficiente *cache* do sistema. No Windows o Nautic demora aproximadamente um segundo para indexar mil arquivos (em um computador equipado com um processador Sempron 2800+), enquanto que no Linux, no mesmo computador, quatro mil arquivos são indexados por segundo em um primeiro momento, e vinte mil arquivos nas vezes subseqüentes. É claro que para se conhecer o tempo de indexação mais concretamente são necessários testes mais detalhados, mas esses valores dão uma idéia do tempo necessário para o processo.

7 CONCLUSÕES

As limitações existentes nos atuais sistemas de tipos de arquivos são visíveis, pois estes ignoram uma quantidade significativa de informações por não implementarem uma hierarquia de tipos de arquivo (com exceção dos UTIs). Isso acarreta em uma experiência do usuário final nos ambientes gráficos inferior àquela que poderia existir, e favorece a criação de soluções próprias pelos aplicativos. Nos ambientes gráficos, os reflexos de um modelo limitado podem ser percebidos na aparência dos ícones, na quantidade de ações, na quantidade de aplicativos associados, na ausência de visualizações alternativas em gerenciadores de arquivos, etc.

Além disso, como diferentes sistemas operacionais possuem métodos de identificação de tipos de arquivo e catálogo de tipos distintos, não há consistência entre as plataformas: um mesmo arquivo pode não ser classificado com o mesmo tipo em plataformas diferentes. Considerando a crescente importância da integração de sistemas e intercâmbio de informações, aplicações multiplataforma que fazem uso de tipos de arquivo são forçadas ainda mais a desenvolver uma solução própria.

Com o objetivo de oferecer às aplicações uma alternativa com mais recursos aos atuais sistemas de tipos de arquivos, neste trabalho foi criada uma hierarquia de tipos de arquivos e proposta uma arquitetura para sua implementação, baseada em *tags* para *desktop*. Para facilitar sua utilização e possibilitar seu porte para diversas plataformas, o *software* é disponibilizado como um *middleware*, oferecendo serviços relacionados a tipos de arquivo e à indexação de arquivos.

Para a construção da hierarquia de tipos de arquivos, foi necessária a investigação do formato e significado de cada tipo, sendo que para muitos deles foi difícil localizar informações relacionadas, visto que muitos formatos se tornaram obsoletos (mas nem por isso deixaram de possuir importância, dada a grande quantidade de arquivos antigos ainda sendo utilizados). Para outros tipos, surgiram problemas de ambigüidade: dois formatos diferentes eram identificados pela mesma extensão de nome de arquivo. Nesses casos, foi dada prioridade aos formatos de maior relevância (empregando-se como critério a quantidade de ocorrências em testes realizados com ferramentas de buscas na web). Para formatos com aproximadamente a mesma relevância, se optou pelo formato que provavelmente possuirá maior apelo no futuro.

Apesar das dificuldades, o resultado do trabalho se mostrou bastante satisfatório, tanto em relação à hierarquia de tipos de arquivos quanto à validação da arquitetura por meio de um *software* experimental. Embora seja desejável a melhor precisão possível na identificação de tipos, ocorre que para as finalidades que as aplicações empregam sistemas de tipos, classificações incorretas isoladas não inviabilizam os serviços. Por exemplo, em uma IDE, se um arquivo não for identificado ou for identificado

incorretamente, a conseqüência disso será provavelmente a apresentação de uma mensagem informando que aquele formato não é válido.

Em outro cenário, se em um gerenciador de arquivos apresentando documentos classificados em tipos houver uma identificação incorreta, o documento problemático não comprometerá a classificação dos demais. E para arquivos não identificados, basta separá-los em uma pasta (assim como faz o Nautic). As conseqüências não são graves porque as aplicações construídas sobre o sistema de tipos levam em conta o fato de que a identificação de tipos de arquivo é imprecisa. De qualquer forma, a grande maioria dos arquivos possui uma extensão conhecida e que corresponde ao seu formato, e sempre que necessário, as aplicações podem expandir a hierarquia de tipos através da API do *middleware*.

7.1 Conclusões do software experimental

Não foram encontradas maiores dificuldades na implementação, e tendo sido atingidos os objetivos do *software* experimental é razoável dizer que a escolha das tecnologias, como linguagem de programação e bibliotecas, foram acertadas para a prototipação

O Nautic, que na verdade apenas é capaz de listar e apresentar arquivos, também atendeu as expectativas de demonstrar na prática a visualização de arquivos organizada em tipos. Um aspecto interessante é o fato de que sendo a organização dos tipos sempre a mesma, a tendência é que o usuário localize cada vez mais rapidamente seus arquivos, apesar dos tipos terem nomes explicativos. Por exemplo, o usuário pode ficar em dúvida se documentos PDF pertencem a `:text_document` ou a `:image`. Porém, a partir do momento em que descobrir a correta localização, não terá mais dificuldades em encontrar qualquer documento do tipo.

Para a maioria dos tipos, é fácil a localização de um determindado arquivo segundo essa visualização. Porém, para tipos contendo muitos arquivos, como JPEG de usuários que armazenam fotos digitais ou C++ de quem possui um projeto muito grande, se torna difícil a localização de um determinado documento, pois a lista de arquivos do tipo pode chegar a milhares. Uma maneira de contornar esse problema é distribuir esses arquivos segundo as pastas físicas em que se encontram. Por exemplo, se algumas das imagens encontram-se no diretório *praia* e outras em *casa*, em vez de listar esses arquivos, poder-se-ia apresentar apenas as duas pastas. Além disso, é possível diminuir a quantidade de arquivos reduzindo a abrangência para um determinado diretório. Uma forma de realizar isso é adicionar uma entrada ao menu de contexto que dispara a distribuição em tipos sobre um diretório. Se o usuário está interessado em examinar apenas os arquivos de sua pasta pessoal, por exemplo, ele poderia acionar a distribuição de tipos apenas para essa pasta. Seria o equivalente a preencher o caminho desse diretório no campo *collect* do Nautic.

7.2 Trabalhos futuros

Boa parte do potencial da organização hierárquica de tipos é usufruida apenas quando existe integração no nível de código-fonte com o ambiente gráfico, assim, o ideal seria integrar o *middleware* e o gerenciador de arquivos apresentado ao ambiente, simplificando a coexistência dos projetos e facilitando seu desenvolvimento. Isso seria factível apenas em plataformas gráficas de código aberto, como Gnome (GNOME, 2006) e KDE (KDE, 2006). Como, no caso do Gnome, o aproveitamento das *tags* para *desktop* originadas no projeto Leaftag seria o caminho natural, o próximo passo será modificar esse ambiente gráfico para comportar a nova visualização baseada em tipos.

REFERÊNCIAS

APPLE. **Simplifying Data Handling with Uniform Type Identifiers**. 2006. Disponível em <http://developer.apple.com/macosx/uniformtypeidentifiers.html/>. Acesso em: out. de 2006.

APPLE. **Uniform Type Identifiers Overview**. 2006. Disponível em: <http://developer.apple.com/documentation/Carbon/Conceptual/understanding_utis/>. Acesso em: out. de 2006.

BECKET, D. New Syntaxes for RDF. In: WORLD WIDE WEB CONFERENCE, WWW, 13., 2004, New York. **Proceedings...** New York: IW3C2, 2004.

BERNERS-LEE, T. **Notation3 (N3)**. 1998. Disponível em <http://www.w3.org/DesignIssues/Notation3>. Acesso em: mar. 2006.

BERNERS-LEE, T. **Primer**: Getting into RDF & Semantic Web using N3. 2005. Disponível: <http://www.w3.org/2000/10/swap/Primer>. Acesso em: mar. 2006.

BOLLOBAS, B. **Modern Graph Theory**. [S.l]: Springer, 2002.

BRICKLEY, D.; MILLER, L. **FOAF Vocabulary Specification**. 2005. Disponível em: <http://xmlns.com/foaf/0.1/>. Acesso em: mar. 2006.

ECLIPSE FOUNDATION. **Eclipse Website**. 2006. Disponível em <http://eclipse.org>. Acesso em: fev. 2006.

GIFFORD, D. K. et al. Semantic File Systems. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, SOSP, 13, 1991, Pacific Grove. **Proceedings...** New York: ACM Press, 1991.

GNOME. **Gnome Website**. 2006. Disponível em <http://www.gnome.org>. Acesso em: fev. 2006.

INTERNET ASSIGNED NUMBERS AUTHORITY (IANA). **MIME Media Types**. 2006. Disponível em: <http://www.iana.org/assignments/media-types/>. Acesso em: jun. 2006.

KDE. **KDE Website**. 2006. Disponível em: <http://www.kde.org>. Acesso em: maio 2006.

KLANG, M. **The o:XML Programming Language**. Disponível em: <http://www.o-xml.org/spec/langspec.html>. Acesso em: maio 2006.

KOSSINETS, G.; WATTS, D. J. Empirical Analysis of an Evolving Social Network. **Science**, [S.l.], v. 311, n. 5757, p. 88-90, Jan. 2006.

KROSKY, E. **The Hive Mind**: Folksonomies and User-Based Tagging. 2005. Disponível em <http://infotangle.blogsome.com/2005/12/07/the-hive-mind-folksonomies-and-user-based-tagging/>. Acesso em: mar 2006.

LESKOVEC, J.; KLEINBERG, J.; FALOUTSOS, Christos. Graph Evolution: Densification and Shrinking Diameters. **ACM Transactions on Knowledge Discovery from Data (TKDD),** New York, 2007.

LI, W.-J.. et al. Fileprints: Identifying File Types by n-gram Analysis. In: WORKSHOP ON INFORMATION ASSURANCE, 6., 2005, West Point, NY. **Proceedings...** [S.l.]: IEEE, 2005.

MCDANIEL, M.; HEYADRI, M. H. Content Based File Type Detection Algorithms. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 36., 2003, Hawaii. **Proceedings...** [S.l.]: IEEE, 2003.

MUNZNER, T. **Interactive Visualization of Large Graphs and Networks**. 2000. Ph.D. (Dissertation) - Department of Computer Science, Stanford University, Stanford.

OBJECTWEB CONSORTIUM (OW2). **OW2 Website**. Disponível em: <http://www.objectweb.org/>. Acesso em: maio 2006.

OLIVEIRA, R. S.; CARÍSSIMI, A. da S.; TOSCANI, S. S. **Sistemas Operacionais**. 3. ed. Porto Alegre: Sagra-Luzzato, 2004.

PÉDAUQUE, R. T. **Document: Form, Sign and Medium, As Reformulated for Electronic Documents**. Paris: CNRS, 2003.

SHADBOLT, N.; BERNERS-LEE, T.; HALL, W.. The Semantic Web Revisited. **IEEE Intelligent Systems**, Washington, v. 3, n. 21, p. 96-101, may/june. 2006.

SINHA, R. 2005. **A cognitive analysis of tagging**. Disponível em <http://www.rashmisinha.com/archives/05_09/tagging-cognitive.html>. Acesso em: mar 2006.

REIMER, J. **A History of GUI.** 2005. Disponível em <http://arstechnica.com/articles/paedia/gui.ars>. Acesso em: nov 2006.

SINGH, A. **Mac OS X Internals**: A Systems Approach. [S.l.] Addison-Wesley Professional, 2006.

SHIRKY, C. **Ontology is Overrated**: Categories, Links and Tags. 2005. Disponível em: <http://www.shirky.com/writings/ontology_overrated.html>. Acesso em: mar 2006.

TANENBAUM, A. S. **Modern Operating Systems**. 2nd ed. Upper Saddle River: Prentice Hall, 2001.

TANENBAUM, Andrew S.; WOODHULL, Albert S. **Operating Systems Design and Implementation**. Upper Saddle River, 3.ed. Prentice Hall, 2006.

TROWBRIDGE, D. **Leaftag Website**. 2005. Disponível em <http://www.chipx86.com/wiki/Leaftag>. Acesso em: ago 2006.

WIKIPEDIA. 2006. **List of File Formats**. Disponível em <http://en.wikipedia.org/wiki/List_of_file_formats>. Acesso em: jul 2006.

WEST, D. B. **Introduction to Graph Theory**. 2nd ed. Upper Saddle River: Prentice Hall, 2001.

WINTER, D. 2005. **RSS 2.0 Specification**. Disponível em <http://blogs.law.harvard.edu/tech/rss>. Acesso em: mar 2006.

WORLD WIDE WEB CONSORTIUM (W3C). 1998. **Document Object Model (DOM) Level 1 Specification.** Disponível em: <http://www.w3.org/TR/REC-DOM-Level-1/>. Acesso em: mar 2006.

WORLD WIDE WEB CONSORTIUM (W3C). 2006. **Extensible Markup Language (XML) 1.0**. Disponível em: <http://www.w3.org/TR/xml/>. Acesso em: mar 2006.

WORLD WIDE WEB CONSORTIUM (W3C). 2004. **RDF/XML Syntax Specification.** Disponível em: <http://www.w3.org/TR/rdf-syntax-grammar/>. Acesso em: mar 2006.

WORLD WIDE WEB CONSORTIUM (W3C). 2004. **OWL Web Ontology Language**. Disponível em <http://www.w3.org/TR/owl-guide/>. Acesso em: mar 2006.