UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Metadados para Apoio à Recuperação de Componentes de Software baseada em Agrupamento por Rede Neural

Claudia Alves de Souza Mello

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

M527ma

Mello, Claudia Alves de Souza.
Metadados para apoio à recuperação de componentes de software baseada em agrupamento por rede neural / Claudia Alves de Souza Mello. -- São Carlos : UFSCar, 2005.
69 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2005.

1. Engenharia de software. 2. Componentes de software. 3. Classificação. 4. Repositório. 5. Recuperação da informação. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Metadados para Apoio à Recuperação de Componentes de Software baseada em Agrupamento por Rede Neural"

CLAUDIA ALVES DE SOUZA MELLO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Profa. Dra. Marina Teresa Pires Vieira
(Orientadora – PPG-CC/UFSCar)

Profa. Dra. Marilde Terezinha Prado Santos
(DC/UFSCar)

Prof. Dr. João Eduardo Ferreira
(IME/USP)

São Carlos
Agosto/2005

Ao meu amado marido

Agradecimentos

A Deus, sem o qual nada é possível.

Às minhas orientadoras, pela orientação e amizade.

Ao meu marido Rodrigo, pelo carinho, paciência e apoio no desenvolvimento deste trabalho.

À minha família, pelo incentivo e apoio constantes.

À Cristiane e Clarissa pelas contribuições, amizade e motivação.

Ao Luciano Senger pela disponibilização e ajuda com a rede neural ART-2A.

Aos colegas da pós-graduação, especialmente à Luciene e Fabiana, pelo bons momentos de amizade.

A todos, que de alguma forma, contribuíram para a realização deste trabalho.

Resumo

Novas abordagens de desenvolvimento de software têm sido propostas para possibilitar o aumento da produtividade e qualidade das aplicações. Entre essas abordagens pode-se destacar o desenvolvimento de software baseado em componentes (DBC). Essa abordagem é caracterizada pelo reuso de componentes de software. Para possibilitar esse reuso é necessário um repositório que armazene componentes e forneça mecanismos eficazes para a sua localização. Nesse contexto, esta pesquisa tem como objetivo identificar e modelar metadados de componentes em um banco de dados possibilitando a definição de estratégias de busca eficazes para sua recuperação. Para alcançar tais objetivos são utilizados conceitos da área de recuperação de informação e da arquitetura de rede neural artificial auto-organizável ART-2A. São apresentados resultados de experimentos que comprovam a viabilidade da proposta.

Abstract

New software development approaches have been proposed in order to permit the increasing of productivity and application quality. Among these approaches, the software development based on components can be evidenced. This approach is characterized by the software component reuse. In order to permit this reuse, it is necessary a repository which is responsible for storing, searching and retrieving software components. In this context, this work aims to identify and model component metadata on databases, allowing the definition of efficient search strategies to retrieve such components. For reaching these goals, information retrieval concepts and the ART-2A self-organizing artificial neural network architecture were adopted. Experiments were conduced to confirm the viability of this proposal.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO

1

Introdução

1.1 Considerações Iniciais

O crescente desenvolvimento tecnológico impulsionou a produção de software tornando o mercado competitivo. Empresas de diversas áreas de domínio exigem que seus sistemas sejam desenvolvidos em períodos curtos, com baixo custo e com qualidade. Para atender essas exigências faz-se necessária a busca de técnicas mais eficientes para o desenvolvimento de software. Uma técnica aplicada, desde o princípio da computação, por muitos programadores, é o reuso de idéias, abstrações e trechos de códigos as quais permitem otimizar o processo de desenvolvimento de software. Atualmente, o conceito de reuso é base de um dos paradigmas emergentes, o desenvolvimento de software baseado em componentes (DBC).

O DBC consiste na implementação de aplicações através do reuso de partes existentes, denominadas de componentes, disponíveis em repositórios (Werner & Braga, 2000). As principais contribuições desse paradigma são a melhoria da qualidade do produto, o aumento da produtividade e redução de custos tanto no desenvolvimento como na manutenção de sistemas (Pressman, 2002).

Nesse contexto está sendo desenvolvido o projeto entitulado Desenvolvimento de Software Baseado em Componentes Multimídia (DBCM) (Vieira, 2003), onde está inserida a pesquisa relatada nesta dissertação.

1.2 Objetivo da Pesquisa

O objetivo deste trabalho é definir um repositório que manipule metadados de componentes de software proporcionando mecanismos eficazes para a sua localização e reuso.

Para alcançar tal objetivo foram realizados estudos sobre trabalhos na literatura que abordam a classificação e recuperação de componentes armazenados em repositório (Prieto-Diáz, 1991; Maarek *et al.*, 1991; Zaremski & Wing, 1995; Seacord *et al.*, 1998; Sugumaran & Storey, 2003; Vitharana *et al.*, 2003). Essas abordagens se distiguem pelos metadados adotados para representar os componentes no repositório. Algumas propõem a definição de categorias baseadas em análise de dominío (Prieto-Diáz, 1991), outras utilizam técnicas mais refinadas como o uso de modelos de dominíos e ontologias (Sugumaran & Storey, 2003; Vitharana *et al.*, 2003). Há também propostas que utilizam informações da própria estrutura do componente e de sua documentação (Maarek *et al.*, 1991; Zaremski & Wing, 1995; Seacord *et al.*, 1998). Aspectos observados nessas propostas foram: a forma como as informações são obtidas e como os componentes são classificados de acordo com tais informações. As propostas que utilizam modelos de domínios e ontologias necessitam de especialistas para que as informações sejam consistentes. Além disso, os componentes não podem ser automaticamente relacionados a essas informações, sendo necessário um administrador para o repositório. As abordagens que se baseiam em informações extraídas da estrutura ou documentação dos componentes podem ser realizadas automaticamente, contudo essas informações podem não ser suficientes para caracterizá-los no repositório, sendo necessários mecanismos mais elaborados na recuperação.

Após uma analise dessas abordagens foi escolhida a metodologia de extrair informações automaticamente da documentação dos componentes, pois facilita a criação do repositório e, para suprir os problemas que surgem nessa abordagem, é proposto o método de classificação e recuperação de componentes utilizando agrupamentos obtidos a partir de uma rede neural. Esses agrupamentos podem ser analisados por especialistas, possibilitando que informações semânticas sejam incluídas na classificação dos componentes.

As informações extraídas dos componentes e dos agrupamentos constituem os metadados de componentes de software, que foram modelados para serem armazenados em banco de dados. Para o gerenciamento desses metadados foram implementados módulos responsáveis pela extração e armazenamento em banco de dados. No desenvolvimento desses módulos foram utilizados conceitos da área de recuperação de informação e rede neural. As estratégias de busca utilizadas nos experimentos exploram a organização dos metadados de componentes no banco de dados para prover mecanismos eficazes para a sua localização e reuso.

1.3 Organização do Trabalho

Esta monografia está organizada da seguinte forma: no capítulo 2 é apresentado um levantamento bibliográfico sobre conceitos que envolvem componentes de software, bem como algumas abordagens existentes para a classificação e recuperação de componentes de software em repositórios; no capítulo 3 são apresentados conceitos da área de recuperação de informação que são úteis no contexto deste trabalho incluindo ainda uma descrição da arquitetura de rede neural

ART-2A, a qual é utilizada na tarefa de classificação dos componentes para seu agrupamento; no capítulo 4 são fornecidos detalhes sobre o desenvolvimento do servidor de componentes do projeto DBCM, apresentando o conjunto de metadados escolhidos para representar os componentes de software, a organização desses metadados para serem armazenados em um banco de dados, os mecanismos de extração e armazenamento desses metadados e as técnicas de recuperação que exploram os agrupamentos de componentes; no capítulo 5 são apresentados os resultados de experimentos realizados para avaliar a abordagem proposta; no capítulo 6 são apresentadas as conclusões e trabalhos futuros.

Foi adicionado um apêndice que apresenta uma lista de *stopwords* utilizada no processo de extração de metadados.

CAPÍTULO

2

Repositórios de Componentes de Software

2.1 Considerações Iniciais

Este capítulo apresenta conceitos sobre componentes de software. Além disso são apresentados trabalhos que abordam métodos de classificação e recuperação de componentes em repositórios ou bibliotecas.

2.2 Conceitos Fundamentais de Componentes

Werner e Braga (Werner & Braga, 2000) apresentam um estudo sobre definições dadas para componente de software. Segundo esse estudo um componente ou artefato é uma parte identificável de um sistema, a qual pode ser reusada. As características de um componente são:

1. Facilmente identificáveis - o componente deve ter características que facilitem a sua localização para reuso.
2. Função específica - a funcionalidade deve ser clara em relação à sua utilidade em um dado contexto.
3. Autocontidos - o componente deve desempenhar sua funcionalidade de forma completa, sem a necessidade de incluir ou depender de outros componentes.
4. Documentados - a documentação deve fornecer informações suficientes que permitam avaliar se o componente é adequado para o contexto de reutilização e quais adaptações são necessárias para integrá-lo ao novo ambiente.

5. Condições de reuso - as condições de reuso compreendem diferentes informações tais como a identificação do proprietário, o responsável em caso de problemas, as características de qualidade e versão, entre outras.

6. Interfaces bem definidas - as interfaces escondem detalhes de implementação e fornecem informações de como um componente pode interagir com os demais.

Para simplificar o desenvolvimento de softwares baseados em componentes foram propostos alguns modelos de componentes. Um modelo identifica um conjunto de padrões e convenções com os quais os componentes devem estar em conformidade. Esses padrões incluem a descrição da funcionalidade de cada componente e como eles interagem entre si (Bachmann *et al.*, 2000). Atualmente, há três organizações que se destacam no mercado com seus modelos de componentes: a OMG (Object Management Group), com o Corba Component Model (CCM) (Group, 2002; Nardi, 2003), a Microsoft, com o Component Object Model (COM) (Corporation, 2003) e o Distributed Component Object Model (DCOM) (Corporation, 1996) e a Sun Microsystems, com os JavaBeans (MicroSystems, 1997) e os Enterprise JavaBeans (EJB) (MicroSystems, 1999).

O modelo de componentes CORBA (CCM) define um componente como uma extensão ou especialização do tipo objeto CORBA, que pode ser especificado em IDL (*Interface Definition Language*) e representado no Repositório de Interfaces. Esse modelo especifica que uma ou mais implementações de componentes podem ser agrupadas em pacotes, cujas propriedades são definidas em um arquivo descritor (com vocabulário XML), visando futura implementação. Além dos esquemas para descrever pacotes, também é possível identificar no CCM referências a ferramentas para implantação de componentes individuais e montagem de aplicação, e uma linguagem para definição de implementação de componentes, a CIDL (*Component Implementation Definition Language*). Tal conjunto de características visa facilitar a composição e a distribuição de aplicações baseadas em componentes CORBA.

O modelo de objetos da microsoft (COM) especifica uma série de interfaces que devem ser implementadas pelos componentes para que eles possam se integrar com facilidade em qualquer tipo de aplicação. Um objeto COM pode, por exemplo, ser incorporado dinamicamente a uma aplicação como o Microsoft Word ou Internet Explorer, que são os contêineres desses componentes. O DCOM (Distributed Component Object Model) é uma extensão da tecnologia COM utilizado para a construção de aplicações distribuídas.

A Sun Microsystems desenvolveu padrões de desenvolvimento de software utilizando a linguagem Java. Nessa linguagem, os componentes são chamados de Beans e existem dois tipos: os JavaBeans e os Enterprise JavaBeans (EJB). Os JavaBeans são componentes utilizados em aplicações locais enquanto que o EJB é utilizado para construção de aplicações distribuídas. Ambos seguem algumas regras durante o desenvolvimento. Os JavsBeans são classes Java simples, que têm regras na definição de atributos, métodos e eventos. Esses componentes são

executados na máquina virtual java. Os Enterprise JavaBeans implementam interfaces específicas e são executados em um contêiner. O contêiner é um software que provê serviços para o desenvolvimento de aplicações distribuídas tais como de transação, persistência e segurança. A finalização do desenvolvimento de um bean ocorre com o empacotamento do conjunto de arquivos implementados em um arquivo JAR (*Java Archive*).

2.3 Abordagens para Repositórios de Componentes

Para promover o reuso de componentes é necessário definir metadados que classifiquem cada componente armazenado em um repositório com mecanismos eficazes para a busca.

Diversos trabalhos abordam métodos de classificação e recuperação de componentes em repositórios (Frakes & Nejmeh, 1987; Burton *et al.*, 1987; Prieto-Diáz, 1991; Maarek *et al.*, 1991; Chen *et al.*, 1993; Zaremski & Wing, 1995; Mili *et al.*, 1997; Zaremski & Wing, 1997; Seacord *et al.*, 1998; Sugumaran & Storey, 2003; Vitharana *et al.*, 2003; Meling *et al.*, 2000). As seções seguintes descrevem as abordagens que contribuiram para o desenvolvimento deste trabalho.

2.3.1 Classificação facetada

Prieto-Diáz (Prieto-Diáz, 1991) propõe um esquema de classificação facetada que consiste na representação de componentes através de descritores constituídos por termos pertencentes a categorias pré-definidas de um domínio particular. Cada categoria ou faceta é identificada por um nome e um conjunto de termos, definido como vocabulário. O vocabulário descreve aspectos relevantes da faceta. A implementação desse esquema requer uma análise de domínio, por algum especialista, permitindo identificar e agrupar corretamente os termos representativos da área.

Como ilustração do uso de facetas na classificação de componentes, Frakes e Pole (Frakes & Pole, 1994) apresentam um esquema de facetas para o domínio de comandos do Unix. Esse esquema é composto pelas seguintes facetas: ação executada pelo comando, contexto da funcionalidade e o tipo do objeto manipulado. A tabela 2.1 apresenta alguns termos definidos para essa facetas.

Por exemplo, considere os comandos $mkdir$, utilizado para criar diretórios no sistema de arquivos, e vi, que é um editor de texto. Seguindo o esquema facetado, esses comandos são classificados pelo descritor facetado da seguinte forma:

d(mkdir) = <criar,sistema de arquivos,diretório>

d(vi) = <(criar,modificar),editor,arquivo>

A consulta nesse esquema é formulada através da seleção de termos de cada uma das

Tabela 2.1: Exemplo de esquema de classificação facetada

Ação Executada	**Área Funcional**	**Objeto Manipulado**
criar	sistema de arquivos	diretório
remover	compilador	arquivo
modificar	sistema de comunicação	programa
compilar	sistema de banco de dados	processo
enviar	editor	:
traduzir	:	:

facetas, criando-se um descritor para a consulta. Esse descritor é comparado aos descritores dos componentes, sendo retornados os componentes que têm um maior número de termos em comum.

2.3.2 Classificação por palavras-chave

Há várias abordagens que adotam mecanismos de indexação na representação de componentes em repositórios (Frakes & Nejmeh, 1987; Burton *et al.*, 1987; Maarek *et al.*, 1991). A indexação, ou seja, o processo de identificar as palavras-chave e armazená-las em uma estrutura de índice, pode ser realizada manualmente ou de forma automática (Baeza-Yates & Ribeiro-Neto, 1999). Na técnica manual, um usuário deve indicar palavras-chave que caracterizem os componentes. Na indexação automática, o sistema é responsável por identificar as palavras-chave através de uma análise na documentação ou nos comentários dos códigos-fonte (quando disponíveis).

Nesse contexto, Maarek *et al.* (Maarek *et al.*, 1991) apresentam uma proposta para construção automática de bibliotecas de software a partir de um conjunto de componentes, utilizando mecanismos de indexação automática. Os componentes são classificados por pares de palavras-chave extraídas automaticamente da documentação, escrita em linguagem natural. Os autores utilizaram um método de indexação baseado no conceito de afinidade léxica, que é um conceito da linguística que descreve a correlação entre unidades da linguagem representando um conceito. Termos que têm afinidade léxica tendem a aparecer separados por, no máximo, cinco palavras dentro de uma sentença. Baseados nesse conceito os autores identificam pares de palavras analisando os termos e seus vizinhos na sentença. Para selecionar as afinidades léxicas mais relevantes conceitualmente os autores consideram a frequência de ocorrência de cada um dos termos da afinidade léxica no documento e, também, a frequência de ocorrência da afinidade léxica no documento.

As consultas são descritas em linguagem natural e dessa descrição são extraídos termos e afinidades léxicas utilizadas no processo de busca. Para cada termo e afinidade léxica são localizados componentes que são indexados por eles, sendo calculada a similaridade entre a consulta e os componentes.

2.3.3 Classificação através de informações semânticas

Sugumaran e Storey (Sugumaran & Storey, 2003) propõem uma abordagem semântica para a classificação e recuperação de componentes. Os autores utilizam ontologias e modelos de domínios para melhorar a eficácia de buscas por componentes. Os modelos de domínio estão organizados em objetivos, processos e ações. Os objetivos descrevem as principais atividades executadas no domínio. Essas atividades podem ser decompostas em processos que também são divididos em ações. Para complementar as informações do modelo de domínio é utilizada uma ontologia que apresenta sinônimos e relaciona os termos do domínio. Os componentes são classificados de acordo com essas informações.

Conceitos e termos relevantes, identificados nas consultas, são comparados com os objetivos especificados no modelo de domínio. Para assegurar o uso de termos corretos e permitir a expansão da consulta por meio de termos relacionados é utilizada a ontologia. Recuperam-se os componentes cujos métodos satisfazem a uma determinada porcentagem dos processos e ações identificados.

2.3.4 Esquema de Classificação baseado em Conhecimento

O esquema de classificação proposto por Vitharana (Vitharana *et al.*, 2003) é formado por identificadores estruturados e um descritor facetado semi-estruturado. Os identificadores estruturados representam informações gerais do componente, tais como nome, proprietário, versão e linguagem de programação. O descritor facetado semi-estruturado descreve características dos componentes e de suas partes (interfaces, métodos, atributos, exceções e tipos de dados) em vários níveis de abstração.

O modelo para implementação de repositórios baseados nesse esquema de classificação é composto por duas partes: uma para armazenar as informações estruturadas e outra para o descritor facetado. Para armazenar as informações estruturadas dos componentes é proposto o uso de um banco de dados. As informações do descritor facetado são codificadas utilizando a linguagem de marcação XML.

A busca é realizada inicialmente pelas informações estruturadas. Por exemplo, apresentando uma lista com os modelos de componentes (JavaBean, EJB) ou com domínios de aplicação. Através desses parâmetros o espaço de busca é reduzido e, em seguida, o usuário pode descrever com maiores detalhes sua requisição, sendo realizada uma busca na base dos descritores.

2.3.5 Abordagens Baseadas na Estrutura do Componente

A principal característica das abordagens dessa categoria é a representação dos componentes através de informações extraídas de sua estrutura, isto é, nome de variáveis, assinaturas de

métodos, etc. Nesse contexto estão as propostas de Zaremski e Wing (Zaremski & Wing, 1995) e Seacord *et al.* (Seacord *et al.*, 1998).

Zaremski e Wing (Zaremski & Wing, 1995) propuseram um método de recuperação de componentes baseado na busca (*matching*) das assinaturas das funções. Esse método descreve o comportamento de uma operação usando uma linguagem de especificação formal, Larch/ML. Esse método descreve o comportamento através dos parâmetros nas assinaturas das operações e não leva em consideração a semântica completa do componente.

O sistema Agora apresentado por Seacord *et al.* (Seacord *et al.*, 1998), adota que os componentes sejam representados pelas seguintes informações: modelo, nome, propriedades e métodos. Essas informações são obtidas através de mecanismos de introspecção, que permite o programa obter informações sobre sua própria estrutura. Uma característica importante desse trabalho é que ele não armazena propriamente os componentes, mas somente suas informações de representação e sua localização, funcionando como uma espécie de páginas amarelas. Para obter as informações dos componentes, o sistema é composto por agentes específicos do modelo de componente e pelo Alta Vista Internet Service, um serviço de busca de informação pela Internet.

2.4 Considerações Finais

Neste capítulo foram apresentados os conceitos fundamentais sobre componentes de software, que neste trabalho são vistos como artefatos executáveis disponibilizados em pacotes juntamente com suas documentações. Este capítulo apresentou, também, algumas abordagens para classificação e recuperaçao de componentes. Essas abordagens podem ser divididas em duas categorias: abordagens que extraem automaticamente informações que representem os componentes e outras que exigem uma estrutura semântica pré-definida para classificá-los, tais como esquemas de categorias ou algo mais refinado como ontologias e modelos de domínio. A tabela 2.2 exibe um resumo das vantagens e desvantagens dessas duas categorias de abordagens para classificação de componentes.

Analisando os benefícios e limitações dos trabalhos apresentados nesta seção, nessa pesquisa foi adotada uma abordagem de extração semi-automática de informações de componentes de software. Tal abordagem permite que a alimentação dos dados no banco de dados seja otimizada em relação ao tempo e possibilita que especialistas atribuam semântica aos dados armazenados.

Tabela 2.2: Resumo comparativo das abordagens

Abordagem	**Benefícios**	**Limitações**
Extração automática	Simplicidade na representação através de palavras-chave; Obtenção das palavras-chave pode ser automatizado, para isso há diversas técnicas de indexação.	Buscas exatas por palavras-chave podem recuperar componentes inadequados; Há necessidade de mecanismos mais refinados para obter bons resultados nas buscas.
Estruturas semânticas pré-definidas	Apresenta informações mais completas na representação dos componentes	Há necessidade de especialistas para definição das informações; Exige maior esforço na implementação e gerenciamento do repositório.

CAPÍTULO

3

Técnicas de Recuperação de Informação e Agrupamento por Rede Neural

3.1 Considerações Iniciais

Na área de recuperação de informação são investigadas formas de representação, armazenamento, organização e acesso a coleção de documentos, permitindo ao usuário fácil acesso à informação na qual está interessado (Baeza-Yates & Ribeiro-Neto, 1999). As técnicas dessa área podem ser utilizadas na documentação de componentes de software simplificando sua recuperação para reuso. Neste capítulo são apresentados alguns dos modelos de representação de informação e a arquitetura de rede neural artificial auto-organizável ART-2A utilizada para o agrupamento (*clustering*) de informações.

3.2 Recuperação de Informação

Um sistema de recuperação de informação é responsável por catalogar e recuperar documentos relevantes às consultas de usuários. Esses sistemas realizam a indexação dos documentos. As principais etapas que compõem a indexação são: a identificação de termos (simples ou compostos), a remoção de *stopwords* (palavras irrelevantes), a normalização morfológica (*stemming*) e a seleção dos termos mais relevantes, os quais constituem o índice. Para cada uma dessas etapas existem diversas técnicas (Baeza-Yates & Ribeiro-Neto, 1999).

Para recuperar os documentos relevantes deve-se utilizar um mecanismo de comparação entre a consulta do usuário e os documentos armazenados. Para isso existem os modelos de recuperação. Os modelos de recuperação definem formalmente a representação dos documentos

e consultas e, também, como são executadas as comparações. Esses modelos podem ser descritos utilizando a seguinte terminologia (Baeza-Yates & Ribeiro-Neto, 1999; Macedo, 2005):

- d_j representa um documento de uma coleção de documentos;
- q representa uma consulta;
- t representa a quantidade de termos de índice da coleção de documentos;
- k_i representa um termo do índice;
- $K = k_1, ..., k_t$ representa o conjunto de todos os termos de índice da coleção;
- $w_{i,j} >= 0$ representa o peso associado ao par termo de índice e documento (k_i, d_j) e $w_{i,j} = 0$ indica que o termo k_i não pertence ao documento d_j. O peso representa a importância do termo na descrição do documento.
- $sim(d_j, q)$ representa a função utilizada para comparar a consulta com os documentos da coleção. Essa função retorna um valor indicando a similaridade entre a consulta q e o documento d_j.

Há diversos modelos de recuperação na literatura tais como: booleano(Baeza-Yates & Ribeiro-Neto, 1999), booleano estendido (Salton *et al.*, 1983), probabilístico (Baeza-Yates & Ribeiro-Neto, 1999) e vetorial (Salton & McGill, 1983). As seções seguintes apresentam os modelos booleano, booleano estendido e vetorial, os quais foram utilizados para a validação das técnicas de recuperação por agrupamento.

3.2.1 Modelo Booleano

Esse modelo tem como base a teoria de conjuntos e a álgebra booleana. Os documentos são representados por um conjunto de termos de índice. As consultas são formuladas através de uma expressão booleana composta por termos ligados através dos operadores lógicos *AND*, *OR* e *NOT*. Esse modelo apresenta como resultado documentos cuja representação satisfazem às restrições lógicas da expressão de busca.

Considere a expressão de busca q = t_1 *AND* t_2 na qual são recuperados documentos indexados por ambos os termos (t_1 e t_2). Essa operação equivale à intersecção do conjunto dos documentos indexados pelo termo t_1 com o conjunto dos documentos indexados pelo termo t_2.

Considere a expressão de busca q = t_1 *OR* t_2 na qual são recuperados documentos indexados pelo termo t_1 ou pelo termo t_2. Essa operação equivale à união entre o conjunto dos documentos indexados pelo termo t_1 e o conjunto dos documentos indexados pelo termo t_2.

Uma expressão que utiliza apenas o termo $t1$ tem como resultado o conjunto de documentos indexados por t_1. A expressão *NOT* t_1 recupera todos os documentos que não são indexados pelo termo $t1$.

Termos e operadores booleanos podem ser combinados para especificar buscas mais detalhadas ou restritivas. A ordem de execução das operações lógicas de uma expressão influencia no resultado da busca sendo necessário explicitar essa ordem, delimitando partes da expressão através de parênteses. Na ausência de parênteses, a expressão booleana é interpretada de acordo com o padrão adotado pelo sistema, que pode ser a execução da expressão da esquerda para a direita ou em uma ordem pré-estabelecida, por exemplo *NOT - AND - OR*.

A utilização dos operadores permite que o usuário especifique consultas complexas, detalhadas e bem definidas. Contudo, estudos mostram que muitos usuários têm dificuldades em expressar suas necessidades de informação através de operadores booleanos (Macedo, 2005).

3.2.2 Modelo Booleano Estendido

Uma desvantagem encontrada no modelo booleano clássico é que os resultados não podem ser ordenados devido a utilização de pesos binários que indicam somente a presença dos termos em documentos. Motivado por esse problema, foi proposto um modelo booleano estendido que considera o peso de termos nos documentos e, também, permite que sejam especificadas pelo usuário as relevâncias dos termos para a consulta(Salton *et al.*, 1983). A proposta desse modelo está baseada na interpretação dos operadores de consulta conjuntivas e disjuntivas em termos de distâncias euclidianas em um espaço t-dimensional.

Para expressões conjuntivas o ponto (1,1) é o mais desejável, já que representa a situação na qual ambos os termos da expressão estão presentes na representação de um documento. Quanto menor à distância de um documento em relação a este ponto maior sua similaridade em relação à expressão de busca.

Para expressões disjuntivas o ponto (0,0) deve ser evitado, pois representa a situação na qual nenhum dos termos está presente no documento. Quanto maior à distância de um documento em relação a este ponto maior sua similaridade em relação à expressão de busca.

Para uma breve ilustração do modelo booleano estendido, considere a utilização de dois termos t_1 e t_2 para representar as consultas e documentos. Nesse caso é definido um espaço de busca bidimensional onde cada termo é associado a um eixo, como apresentado na figura 3.1. Um documento é representado por um vetor com dois elementos contendo pesos dos respectivos termos. Esses pesos definem o posicionamento do documento no espaço euclidiano.

A similaridade entre um documento $d_i = (w_{1i}, w_{2i})$ e uma consulta $q = t_1$ or t_2 é calculada através da equação 3.1, onde w_{1i} e w_{2i} representam os pesos de cada um dos termos de indexação do documento.

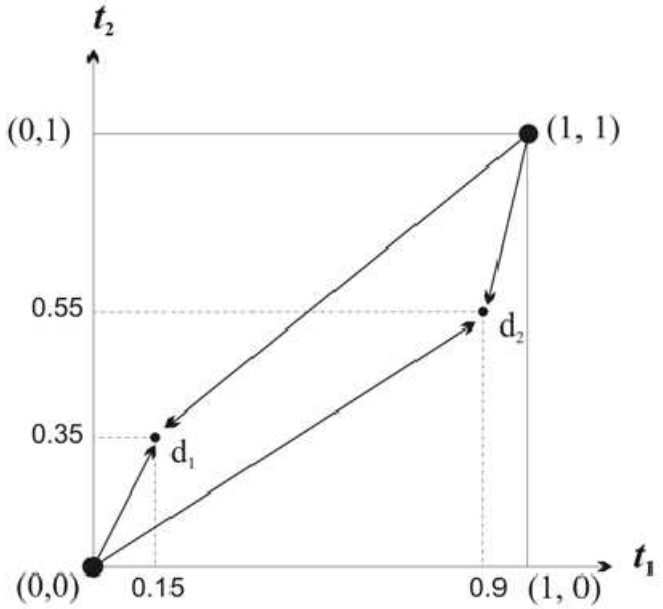

Figura 3.1: Exemplo de representação de documentos em um espaço bidimensional para o modelo booleano estendido

$$sim(q, d_i) = \sqrt{\frac{w_{1i}^2 + w_{2i}^2}{2}} \tag{3.1}$$

Para o exemplo apresentado na figura 3.1, a similaridade entre a consulta q e os documentos d_1 e d_2 é calculada da seguinte forma:

$$sim(q, d_1) = \sqrt{\frac{0,15^2 + 0,35^2}{2}} = 0,2692$$

$$sim(q, d_2) = \sqrt{\frac{0,90^2 + 0,55^2}{2}} = 0,7458$$

A similaridade entre um documento $d_i = (w_{1i}, w_{2i})$ e uma consulta $q1$ = t_1 and t_2 é calculada através da equação 3.2, onde w_{1i} e w_{2i} representam os pesos de cada um dos termos de indexação do documento.

$$sim(q, d_i) = 1 - \sqrt{\frac{(1 - w_{1i})^2 + (1 - w_{2i})^2}{2}} \tag{3.2}$$

Para o exemplo apresentado na figura 3.1, a similaridade entre a consulta $q1$ e os documentos d_1 e d_2 é calculada da seguinte forma:

$$sim(q, d_1) = 1 - \sqrt{\frac{(1 - 0,15)^2 + (1 - 0,35)^2}{2}} = 0,2434$$

$$sim(q, d_2) = 1 - \sqrt{\frac{(1-0,90)^2 + (1-0,55)^2}{2}} = 0,6740$$

Além das distâncias euclidianas, os autores generalizam o modelo definindo um parâmetro ρ que determina a interpretação dos operadores booleanos (Salton *et al.*, 1983; Baeza-Yates & Ribeiro-Neto, 1999).

3.2.3 Modelo Vetorial

Nesse modelo, cada documento é representado por um vetor de termos e cada termo tem um valor associado que indica o grau de importância (peso) dele no documento. Existem vários métodos para calcular o peso, mas geralmente esses cálculos se baseiam no número de ocorrências (frequência) do termo no documento (Salton & Buckley, 1988; Baeza-Yates & Ribeiro-Neto, 1999). Cada elemento do vetor é considerado como uma coordenada dimensional. Assim, os documentos podem ser representados em um espaço euclidiano de n dimensões (onde n é o número de termos) e a posição do documentos em cada dimensão é dada pelo seu peso.

A consulta do usuário também é representada por um vetor. Dessa forma, os vetores dos documentos podem ser comparados com o vetor da consulta e o grau de similaridade entre cada um deles pode ser identificado. Esse grau de similaridade é o cálculo da distância entre o vetor do documento e o da consulta. Esse cálculo é realizado utilizando a equação 3.3 de correlação de cossenos (*cosine correlation*), proposta por Salton (Salton & McGill, 1983), onde: w_{iq} é o peso do i-ésimo termo da consulta e w_{ij} é o peso do i-ésimo termo no documento (d_j). O resultado da busca é um conjunto de documentos ordenados pelo grau de similaridade entre cada documento e a consulta.

$$sim(q, d_j) = \frac{\sum_{i=1}^{n} w_{iq} * w_{ij}}{\sqrt{\sum_{i=1}^{n} w_{iq}^2} * \sqrt{\sum_{i=1}^{n} w_{ij}^2}} \tag{3.3}$$

A figura 3.2 exemplifica a representação de documentos e consultas em um espaço dimensional. Nesse caso, há três termos de índice formando um espaço tri-dimensional.

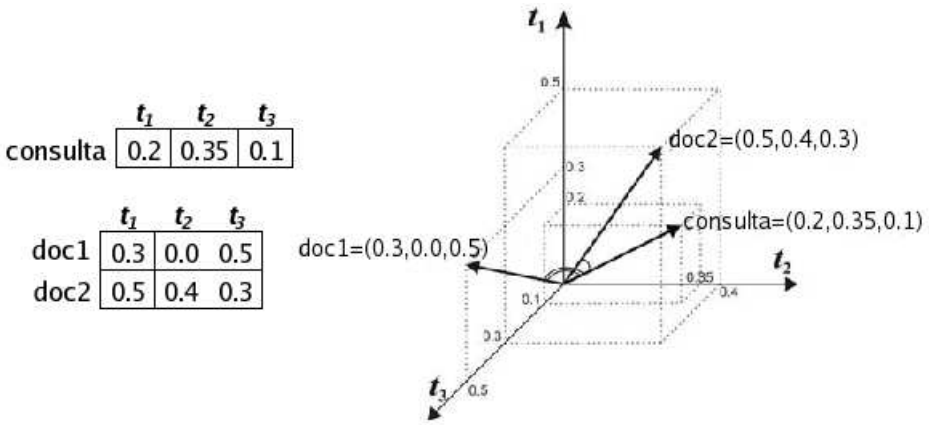

Figura 3.2: Exemplo de representação de documentos em um espaço dimensional para o modelo vetorial

Para determinar o conjunto resposta, resultante da busca vetorial, pode-se aplicar duas técnicas:

- Definir um limite para o grau de similaridade. Nesse caso são recuperados os documentos que têm o valor sim acima do limite definido. Supondo que l é o limite definido, então o conjunto resposta b é definido como $b = \{d_i | (q, d_i) > l\}$;

- Definir a quantidade de elementos que devem ser retornados. Nesse caso os documentos são classificados em ordem decrescente de acordo com o grau de similaridade. Supondo que p é a quantidade definida de elementos a serem retornados, então o conjunto resposta b é definido como $b = \{d_i | r(i) > p\}$, onde $r(i)$ é a posição de classificação do documento d_i.

A principal vantagem do modelo vetorial é a recuperação de documentos que satisfazem parcialmente os requisitos de uma consulta. No próximo capítulo esse modelo é tratado com mais detalhes. Informações adicionais podem ser encontradas em (Corrêa, 2003) que refere-se a um trabalho de mestrado desenvolvido no contexto dos projetos do grupo de banco de dados do departamento de computação da Universidade Federal de São Carlos.

3.3 Agrupamentos de Documentos

Para melhorar os resultados das buscas efetuadas em coleções de documentos tem-se utilizado conceitos de agrupamento. O agrupamento (*clustering*) é considerado uma técnica de aprendizado de máquina não supervisionada e seu principal objetivo é agrupar um conjunto de dados em diferentes grupos (Jain *et al.*, 1999). As principais abordagens para agrupamentos de documentos são o agrupamento por partição (*flat partition*) e por partição hierárquica.

O agrupamento por partição consiste em alocar os documentos em agrupamentos disjuntos. Documentos são agrupados de forma que todos os elementos de um mesmo agrupamento têm um grau mínimo de semelhança, o qual é indicado pelo número de características em comum. Alguns algoritmos de agrupamentos por partição são *k-means* e *bisecting k-means* (Jain *et al.*, 1999).

O agrupamento por partição hierárquica é representado por uma árvore que iterativamente divide os objetos em subconjuntos menores até que cada subconjunto consista de somente um objeto. Em tais hierarquias, cada nó da árvore representa um agrupamento. Esses agrupamentos hierárquicos podem ser criados através de duas abordagens: aglomerativa (*bottom-up*) e divisiva (*top-down*). Alguns algoritmos de agrupamento hierárquico são *single-link* e *complete-link* (Jain *et al.*, 1999)

Trabalhos relacionados ao agrupamento de documentos abordam principalmente os algoritmos de clusterização adotados para realizar essa tarefa, não fornecendo detalhes sobre as estratégias de busca (Jain *et al.*, 1999; Steinbach *et al.*, 2000; Fazli *et al.*, 2004; Dobrynin *et al.*, 2005; Conrad *et al.*, 2005).

Neste trabalho foi adotada uma arquitetura de rede neural para o agrupamento de componentes de software. Essa rede foi adotada por apresentar características tais como: a não necessidade de intervenção humana durante o processo de classificação; por ser não-supervisionada, fato que não a obriga a aprender a partir de exemplos certos e errados; pelo aprendizado da rede ser competitivo e pela sua capacidade de adaptação dos neurônios de suas camadas.

3.3.1 Arquitetura de Rede Neural ART-2A

Neste trabalho foi adotada uma arquitetura de rede neural auto-organizável denominada ART-2A, proposta por Carpenter *et al.*(Carpenter *et al.*, 1991) para realizar o agrupamento. Os agrupamentos são do tipo partição. A família de redes neurais ART (*Adaptive Ressonance Theory*) consiste em arquiteturas que realizam agrupamento de uma sequência arbitrária de padrões de entrada (vetores de características, dados, etc). Nas arquiteturas ART, o aprendizado é tratado como uma ação dinâmica, de forma que a rede possa, continuamente, adaptar-se aos novos padrões de entrada. O aprendizado nas redes ART é não-supervisionado e por competição.

No aprendizado não-supervisionado a rede neural aprende a partir de padrões de entrada não rotulados, ou seja, sem empregar um mecanismo supervisor externo, ao contrário de redes com aprendizado supervisionado, em que a rede recebe um conjunto de treinamento previamente classificado e rotulado. No aprendizado não-supervisionado, a rede tem a habilidade de formar representações internas para codificar as entradas através de um conjunto de unidades de saída de representação. É comum, nessa forma de aprendizado, arquiteturas formadas por uma camada de entrada, uma camada de saída e um conjunto de conexões ou vetores de pesos entre essas camadas. A camada de entrada realiza o processamento inicial do conjunto de entrada, fazendo,

em geral, uma normalização, e a camada de saída é responsável pela representação dos dados, através da criação de grupos. Cada grupo descreve um subconjunto dos padrões de entrada. O aprendizado consiste na busca por uma classe, ou agrupamento, que represente o padrão de entrada e na modificação recorrente dos vetores de pesos em resposta à entrada.

O aprendizado por competição consiste em um dos métodos para a implementação do aprendizado não-supervisionado, no qual as unidades da camada de saída realizam uma disputam para tornarem-se ativas em resposta ao padrão de entrada. Essa disputa descreve uma competição entre as unidades de representação para decidir entre elas a vencedora. Apenas a unidade vencedora é escolhida para representar o padrão de entrada, tendo seus respectivos vetores de pesos modificados.

A organização básica da arquitetura ART envolve dois componentes principais: o subsistema de atenção (*attentional sub-system*) e o subsistema de orientação (*orienting sub-system*) como apresentado na figura 3.3.

O subsistema de orientação é formado por um mecanismo que controla o nível de similaridade entre os padrões armazenados no mesmo neurônio de saída, chamado de *reset*. O subsistema de atenção é composto por uma camada de pré-processamento das entradas F_0, de uma camada de representação das entradas F_1 e de uma camada de representação das categorias ou classes F_2. A função principal das camadas F_0 e F_1 é o processamento inicial do vetor de entrada. Esse processamento pode ser simples ou envolver uma série de operações de normalização e filtragem de ruído. A quantidade de unidades de processamento das camadas F_0 e F_1 também depende da arquitetura da rede. A camada F_2 tem como função principal a representação ou agrupamento das entradas. As unidades de processamento dessa camada são dinâmicas, de forma que novas unidades de processamento possam ser criadas à medida que forem necessárias. No final da execução do aprendizado, a quantidade de unidades de processamento da camada (F_2) é igual à quantidade de classes criadas para representar os dados de entrada.

As camadas F_1 e F_2 são ligadas através de dois conjuntos de conexões direcionadas. A camada (F_1) envia seus sinais de ativação para a camada (F_2) através de pesos *bottom-up*. O peso *bottom-up* que conecta o i-ésimo neurônio da camada (F_1) até o j-ésimo neurônio da camada (F_2) é chamado de b_{ji}. Tais conexões são também chamadas de pesos *top-down*, chamados de t_{ji} ou conexões *feedback*. As conexões *bottom-up* e *top-down* são adaptativas e têm um papel importante no aprendizado da rede e são chamados de memória de longa duração.

Quando um padrão de entrada é apresentado à rede, é realizado um pré-processamento desse padrão pela camada F_0. A seguir, a camada F_1 recebe o padrão tratado e é calculada a ativação dos neurônios da camada F_2. O neurônio j com maior atividade da camada F_2 torna-se um candidato para codificar o padrão de entrada. Neste ponto, os outros neurônios tornam-se inativos e a camada F_2 combina a informação entre o padrão de entrada e o neurônio candidato resultando no vetor de pesos t_{ji}. A unidade de *reset* verifica a similaridade entre esse vetor

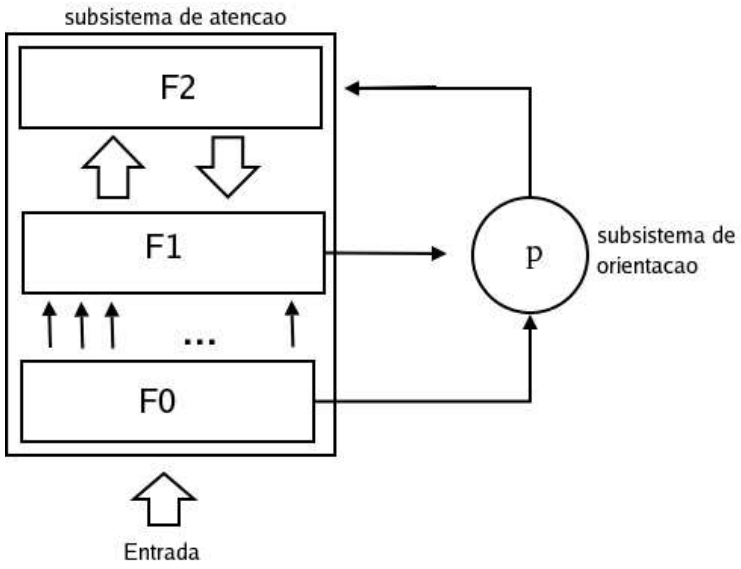

Figura 3.3: Arquitetura básica das redes ART

e o vetor de entrada. Se a similaridade for menor que um determinado limiar (parâmetro de vigilância ρ), o neurônio candidato é marcado como inibido e um novo candidato é escolhido. Tal sequência é mantida até que se encontre um neurônio que possa representar o padrão ou até que todos os neurônios da camada de saída estejam inibidos. Nesse caso, a rede pode criar um novo neurônio para armazenar o padrão ou informar que o padrão não pode ser representado pela rede.

O parâmetro de vigilância, denominado ρ, é controlado pelo usuário e permite modificar a sensibilidade da rede aos padrões de entrada, gerando mais ou menos agrupamentos. Utilizando $\rho = 0$, todos os padrões de entrada são agrupados no mesmo grupo. Com $\rho = 1$, é criado um grupo para cada padrão de entrada (Senger, 2005). Mello *et al.* (Mello *et al.*, 2004) apresentam experimentos indicando que o valor ideal do ρ está entre $[0.70, 0.75]$. Esse valor foi obtido da análise de duas medidas de performance: a distância *intra-cluster*, que define a distância entre os padrões contidos dentro do agrupamento, e a distância *inter-cluster*, que define a distância entre os centróides de cada agrupamento. O valor do ρ ideal define que a distância *intra-cluster* é pequena o suficiente para obter padrões similares no mesmo agrupamento e que a distância *inter-cluster* é suficientemente grande para separar os agrupamentos criados pela rede neural.

Nesta pesquisa foi utilizada a implementação da rede ART-2A desenvolvida por Senger (Senger, 2005) [1].

[1]Essa implementação é distribuída livremente através da licença GPL

3.4 Considerações Finais

Este capítulo apresentou os modelos de recuperação de informação booleano, booleano estendido e vetorial. O modelo vetorial é utilizado para validar a proposta desta pesquisa, cujos resultados são apresentados no capítulo 5. Estão sendo encaminhados testes para validar a proposta utilizando o modelo booleano e booleano estendido.

Neste capítulo também foi apresentada a arquitetura de rede neural auto-organizável ART-2A que desempenhou um papel fundamental para o estabelecimento da estratégia de busca utilizando os agrupamentos, proposta nesta pesquisa.

CAPÍTULO

4

Metadados para apoio à Recuperação de Componentes de Software

4.1 Considerações Iniciais

O objetivo desta pesquisa é definir um repositório que manipule metadados de componentes de software proporcionando mecanismos eficazes para a sua localização e reuso. O estudo de trabalhos relacionados à classificação e recuperação de componentes de software, apresentado no capítulo 2, possibilitou a identificação de um conjunto de metadados. Desse conjunto foram selecionados os metadados mais expressivos e novos metadados foram definidos. Neste capítulo são apresentados o conjunto de metadados, os mecanismos para a extração e armazenamento desses metadados em banco de dados e as técnicas utilizadas para recuperação a partir desses metadados.

4.2 Contextualização da Pesquisa

Esta pesquisa de mestrado está inserida no projeto intitulado Desenvolvimento de Software Baseado em Componentes Multimídia (DBCM), desenvolvido pelo grupo de Engenharia de Software e Banco de Dados do Departamento de Computação da Universidade Federal de São Carlos (Vieira, 2002). O projeto DBCM tem como objetivo definir estratégias para o processo de desenvolvimento de software baseado em componentes multimídia. Essas estratégias se dividiram em duas etapas. A primeira consistiu no desenvolvimento de componentes para um domínio, a segunda, no desenvolvimento de aplicações que reutilizem componentes desse domínio. Na primeira etapa foi realizada uma análise de domínio, que identifica objetos e operações de uma classe de sistemas em um domínio de problema particular. Como resultado dessa análise foram

modelados componentes que especificam as principais funcionalidades do domínio. Esses componentes foram implementados de acordo com um modelo de componentes e, posteriormente, armazenados em um banco de dados. Na segunda etapa estão sendo desenvolvidas aplicações ($Ap_1, Ap_2, ..., Ap_n$) nesse domínio, fazendo a reutilização de componentes(Vieira, 2003; Vieira *et al.*, 2005). A figura 4.1 apresenta as atividades dessas etapas e o servidor de componentes de software responsável pelo armazenamento e recuperação dos componentes.

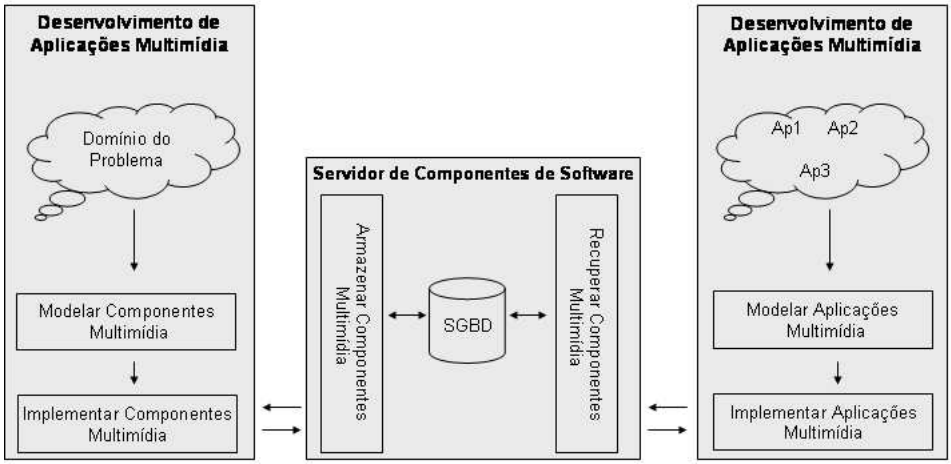

Figura 4.1: Atividades do Projeto DBCM

O desafio inicial para o desenvolvimento desse servidor, parcialmente implementado neste mestrado, foi a identificação de um conjunto de metadados para representar informações relevantes dos artefatos, ou componentes, de software. Analisando as abordagens de classificação e recuperação, apresentadas no capítulo 2, foram identificadas as seguintes categorias de metadados: estruturais, baseados em conteúdo, semânticos baseados em ontologias e de padrões (figura 4.2).

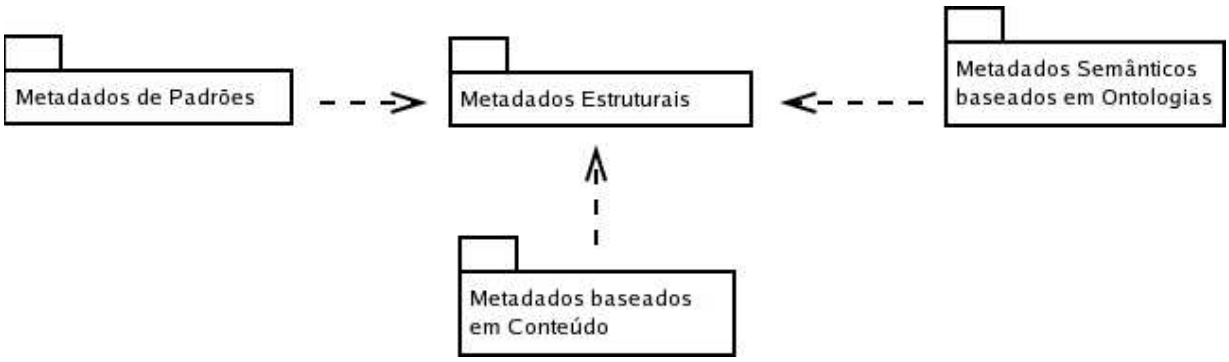

Figura 4.2: Categorias dos metadados de componentes

Os metadados estruturais correspondem às informações sobre a estrutura do componente, descrevendo quais são seus atributos, interfaces e métodos. Esses metadados servem para que o usuário tenha uma visão geral da composição do componente.

Os metadados baseados em conteúdo representam as informações extraídas automaticamente da documentação dos componentes.

Os metadados semânticos baseados em ontologias apresentam informações sobre o domínio de aplicação ao qual os artefatos de software pertencem (Yaguinuma *et al.*, 2005b).

Os metadados de padrões consideram informações sobre possíveis padrões de software identificados a partir da reutilização dos componentes. Esses metadados auxiliam na identificação e no reuso de padrões relacionados aos componentes armazenados no repositório além de facilitar a compreensão das funcionalidades e usos dos artefatos de software recuperados (Yaguinuma *et al.*, 2005a).

Através desses metadados são fornecidos mecanismos flexíveis para a realização de buscas no servidor. A pesquisa desenvolvida neste trabalho está relacionada ao projeto e implementação do servidor de componentes com suporte à categorização de metadados estruturais e baseados em conteúdo.

4.3 O Servidor de Componentes

O servidor de componentes é constituído por módulos responsáveis pelo gerenciamento de cada categoria de metadados e, também, pelos mecanismos de recuperação dos componentes. Na figura 4.3 estão destacados os módulos de gerenciamento dos metadados estruturais e baseados em conteúdo que foram implementados neste trabalho de mestrado. Esses módulos executam tarefas para obtenção e armazenamento de metadados sobre um banco de dados.

O módulo de recuperação, também destacado na figura 4.3, é responsável por avaliar as consultas submetidas ao servidor e recuperar os componentes mais adequados às necessidades expressas na consulta.

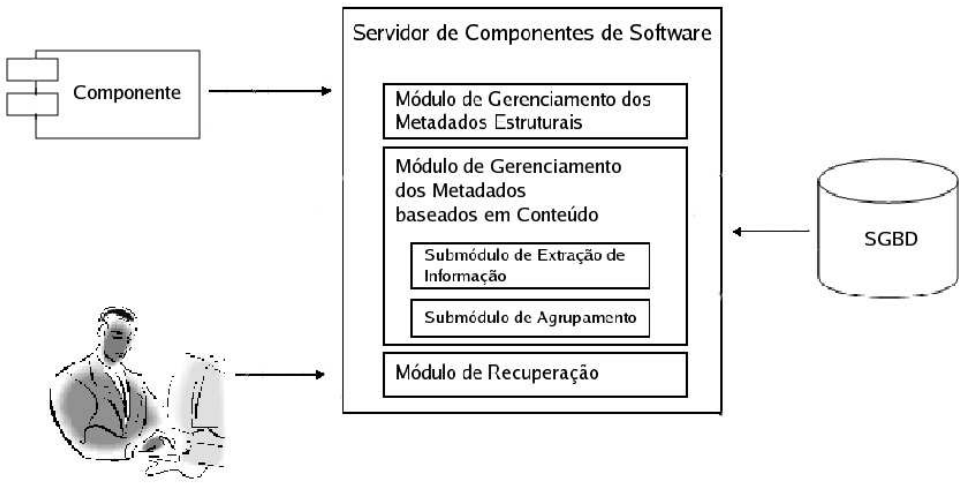

Figura 4.3: Módulos do Servidor de Componentes

Uma visão geral dos metadados estruturais e baseados em conteúdo é apresentada na figura 4.4. Nessa figura também é mostrada a ligação desses metadados com os metadados

semânticos baseados em ontologias (Yaguinuma *et al.*, 2005b).

Detalhes de cada categoria de metadados são apresentados nas seções seguintes.

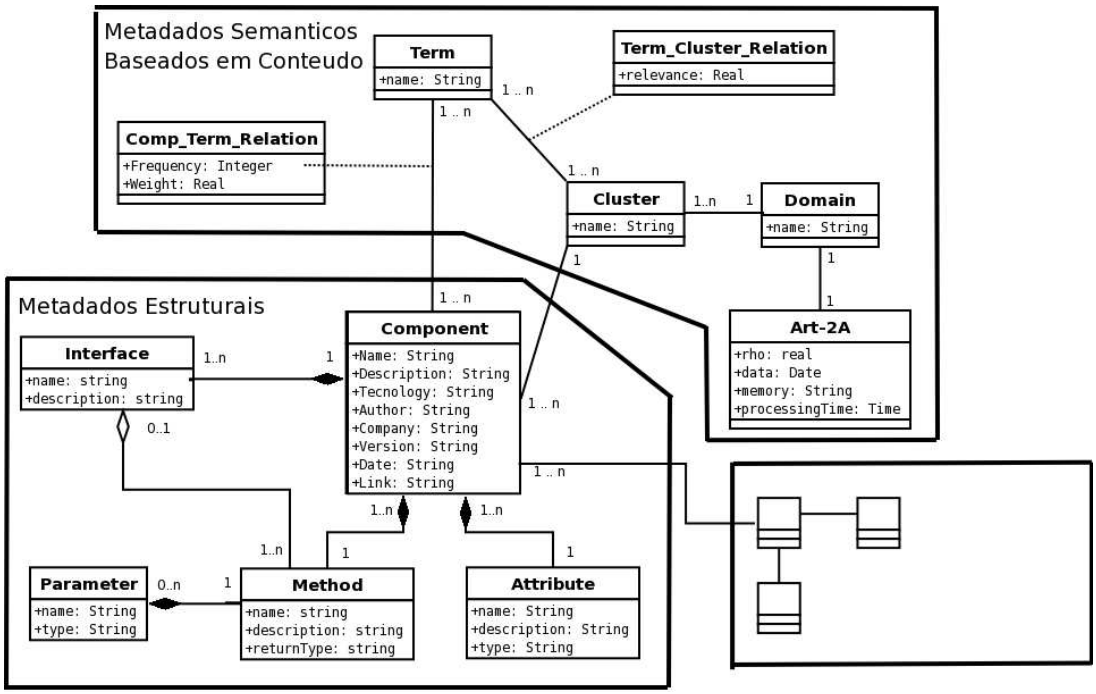

Figura 4.4: Metadados de Componentes

4.3.1 Módulo de gerenciamento de metadados estruturais

O diagrama de classes, apresentado na figura 4.5, mostra como esses metadados foram modelados no banco de dados do Servidor de Componentes do DBCM. Detalhes de cada classe são apresentados em seguida.

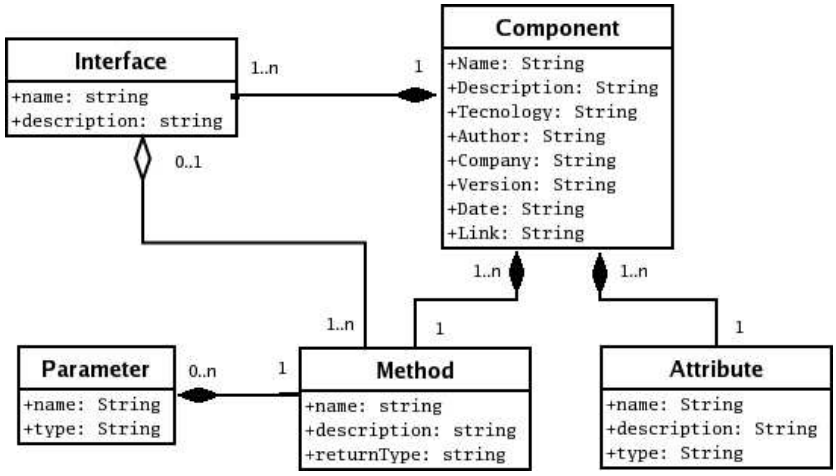

Figura 4.5: Modelagem dos metadados estruturais

A classe Component representa o componente propriamente dito, incluindo o nome, uma breve descrição de sua funcionalidade, a tecnologia utilizada para seu desenvolvimento,

nome do desenvolvedor, nome da empresa ou instituição na qual o componente foi desenvolvido, versão, data da versão e referência para o local onde o componente está disponível. Além dessas informações gerais, um componente é composto por atributos, métodos e interfaces.

Para cada atributo do componente são armazenados o seu nome e tipo de dados. Essas informações são mantidas na classe Attribute.

Os métodos são mantidos na classe Method. Para cada método é armazenado seu nome, uma breve descrição da ação executada por ele e tipo de retorno. Os métodos podem conter parâmetros. Para cada parâmetro são mantidos seu nome e tipo, sendo essas informações armazenadas na classe Parameter.

As interfaces representam os pontos de acesso que os componentes disponibilizam para o reuso. Para cada interface são armazenados seu nome e uma breve descrição das funcionalidades disponíveis, sendo essas informações armazenadas na classe Interface.

Neste trabalho foi implementado um módulo para extração semi-automática de metadados estruturais de componentes desenvolvidos na linguagem Java. Para isso foi utilizada a Reflection API, um pacote para obter informações de objetos disponíveis na máquina virtual java. Por meio desse mecanismo são obtidos: o nome do componente, o nome e tipo de atributos, o nome e tipo do retorno dos métodos, o tipo de dado dos parâmetros e o nome das interfaces. Os demais metadados estruturais são obtidos manualmente tais como as informações gerais dos componentes (descrição, tecnologia, autor, instituiçao, versão, data e link) e as descrições de atributos, métodos e interfaces.

4.3.2 Módulo de gerenciamento de metadados baseados em conteúdo

Este projeto parte do pressuposto que componentes construídos para reuso disponibilizam uma documentação que especifique suas principais funcionalidades. Essa documentação contém informações relevantes para classificá-los e armazená-los em repositórios. Independente do tipo de documentação, a menor estrutura de informação que pode ser extraída automaticamente são termos. Estes constituem os metadados baseados em conteúdo, os quais representam componentes no banco de dados. Tais metadados são utilizados no agrupamento dos componentes. Esses agrupamentos são a base da estratégia de busca proposta nesta pesquisa e, também, compõem a informação semântica. Essa informação semântica descreve o contexto no qual os componentes foram desenvolvidos e onde podem ser reutilizados. O módulo de gerenciamento de metadados é composto pelos submódulos de extração de informação e agrupamento.

4.3.2.1 Submódulo de extração de informação

O submódulo de extração de informação consiste em identificar um conjunto de termos para representar, suscintamente, as funcionalidades e propriedades dos componentes. Para obter

esse conjunto de termos é aplicada uma variação do mecanismo de indexação automática proposto por Salton e McGill (Salton & McGill, 1983), onde são considerados aspectos inerentes à documentação do componente.

A implementação utilizada para validar esse submódulo foi baseada no padrão de desenvolvimento de software da Sun Microsystems (MicroSystems, 2005). Esse padrão apresenta uma nomenclatura que define nomes de classes, componentes, métodos, atributos e comentários. Esses comentários podem ser utilizados na geração da documentação de programas (Microsystems, 2000a).

Para ilustrar as tarefas executadas nesse submódulo é considerada a documentação de um componente do modelo JavaBean, denominado de $CompExample$, que segue o padrão da Sun Microsystems.

A primeira tarefa executada por esse submódulo é a análise léxica, que consiste na substituição de múltiplos espaços e tabulações por um único espaço e remoção de símbolos e caracteres de controle de arquivo ou formatação. Além disso, nessa tarefa são analisados os nomes de métodos e atributos visando identificar termos que possam constituir a informação dos componentes. Normalmente, nomes de métodos e atributos são formados pela junção de termos. Esses termos são decompostos em termos distintos. Por exemplo, considere que o componente $CompExample$ tenha o método $getActionObject$. Esse método é decomposto em três termos: *get*, *Action* e *Object*.

A próxima tarefa a ser executada é a eliminação de uma lista de *stopwords*, que são termos com alta frequência em documentos e não são relevantes para identificá-los. Alguns exemplos de *stopwords* são artigos, preposições e conjunções. Neste trabalho a lista de *stopwords* foi obtida do sistema KEA 2.0, desenvolvido por Frank *et al.* (Frank, 2000). Essa lista foi complementada com termos do modelo de componentes da Sun Microsystems que são igualmente irrelevantes para a representação de um componente em particular (a lista de *stopwords* é apresentada no Apêndice A). Cada termo obtido da tarefa anterior é comparado com os termos contidos na lista de *stopwords*; caso o termo em questão seja encontrado nessa lista, este deve ser retirado da documentação.

Todos os termos obtidos na tarefa anterior passam por uma validação, confirmando sua pertinência ao vocábulario da língua inglesa[1]. Para isso é utilizado o *thesaurus* WordNet (University, 2004). Essa etapa elimina palavras inexistentes na língua inglesa, tais como abreviações utilizadas pelo desenvolvedor.

Após a validação, ocorre a normalização linguística, na qual as formas variantes de um termo são reduzidas a uma forma comum denominada *stem*. Os algoritmos de *stemming* fazem a remoção de prefixos ou sufixos de um termo. Neste trabalho é utilizado o algoritmo de Porter

[1] A língua inglesa foi adotada nos experimentos realizados neste trabalho, pois grande parte dos componentes disponibilizados na internet estão documentados nessa língua.

(Porter, 1980), que remove sufixos de termos da língua inglesa. Por exemplo as palavras *delete*, *deletes*, *deleted* e *deleting* são reduzidas ao radical *delet*. A tarefa de *stemming* permite uma redução significativa no número de termos que compõem a documentação.

Uma vez encontrados os termos normalizados, são obtidas suas frequências de ocorrência na documentação e calculados seus pesos. Os pesos indicam a relevância de cada termo na documentação do componente e na coleção de componentes armazenados. Para calcular o peso (w_i) é utilizada a equação 4.1 proposta por Salton e McGill (Salton & McGill, 1983), onde f_i é a frequência do termo i na documentação do componente, n é a quantidade total de componentes armazenados e n_i é a quantidade de componentes que possuem o termo i.

$$w_i = f_i * \log \frac{n}{n_i} + 1 \tag{4.1}$$

A técnica de truncagem foi aplicada para selecionar os 50 termos mais relevantes. Essa técnica consiste em estabelecer um número máximo de características a serem adotadas para representar um documento. Schitze e Silverstein (Schitze & Silverstein, 1997) indicam que, em geral, cinquenta termos são suficientes para representar um documento. Testes foram realizados na tentativa de reduzir esse número para a documentação de componentes de software, contudo os resultados não foram satisfatórios para obtenção dos agrupamentos.

A última tarefa executada pelo submódulo de extração de informação é de armazenar os termos suas freqüências e pesos no banco de dados. O diagrama de classes, apresentado na figura 4.6, mostra como essas informações estão organizadas e as relações existentes entre elas. Nesta tarefa são alimentados os dados nas classes $Term$ e $Comp\_Term\_Relation$.

A classe $Term$ mantém os radicais de até 50 termos extraídos da documentação de cada componente. Note-se que um componente estará associado a n termos ($n <= 50$) e cada termo também pode estar associado a n componentes.

A classe $Comp\_Term\_Relation$ mantém a frequência e os pesos do termo para um determinado componente. A frequência de ocorrência entre um termo e um componente é armazenada para facilitar a atualização dos pesos quando novos componentes são submetidos ao servidor.

Cada uma das etapas da indexação automática, descritas nos parágrafos anteriores, foram desenvolvidas para o padrão de documentação da Sun Microsystems e podem ser complementadas para outros tipos de documentação de componentes. O objetivo principal do submódulo de extração de informação é obter termos e seus pesos para representar o componente no banco de dados. Essas informações são essenciais para o agrupamento dos componentes.

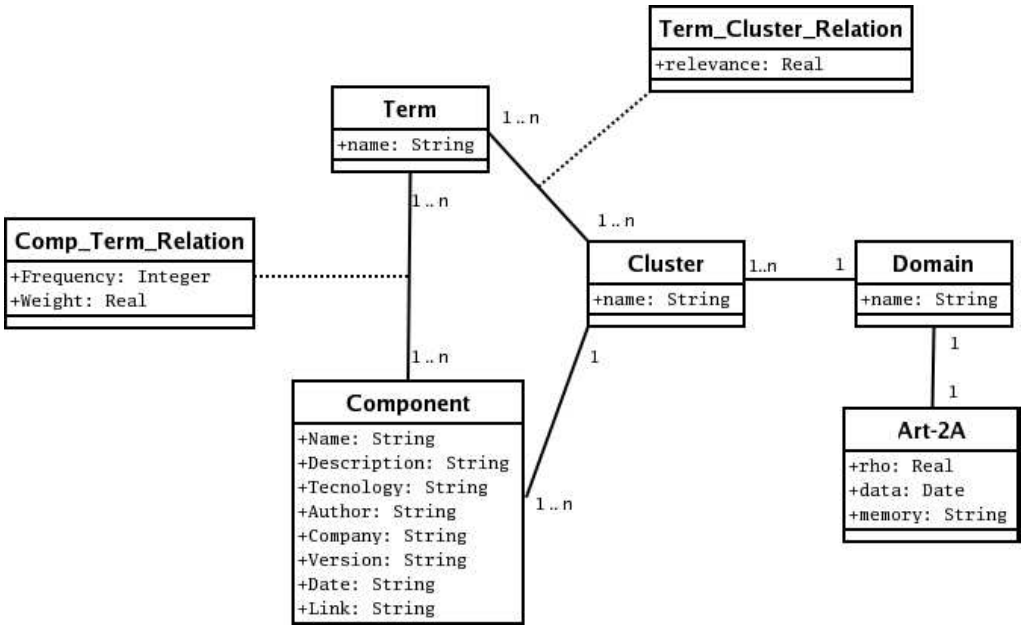

Figura 4.6: Metadados baseados em conteúdo

4.3.2.2 Submódulo de agrupamento

O submódulo de agrupamento é responsável por identificar conjuntos de componentes similares, ou seja, que pertencem a um mesmo contexto. Para o processo de agrupamento proposto neste trabalho, adotou-se a arquitetura de rede neural artificial auto-organizável Art-2A (Carpenter *et al.*, 1991). Os parâmetros de entrada dessa rede neural são: a matriz de componentes, o número de colunas dessa matriz e um parâmetro de vigilância ρ. A seguir é apresentada a abordagem proposta neste trabalho para possibilitar a recuperação de componentes com base em agrupamentos.

As linhas da matriz de componentes representam os componentes e as colunas os termos de índice. Os valores da matriz representam os pesos dos termos para cada componente. A matriz 4.2 representa uma coleção de componentes armazenados em banco de dados. Nesse exemplo, há cinco componentes armazenados (c_1, c_2, c_3, c_4 e c_5) e cinco termos de índice (t_1, t_2, t_3, t_4 e t_5).

$$M_{comp} = \left| \begin{array}{cccccc} & t_1 & t_2 & t_3 & t_4 & t_5 \\ c_1 & 0,67 & 0,79 & 0,0 & 0,80 & 0,85 \\ c_2 & 0,62 & 0,86 & 0,95 & 0,0 & 0,65 \\ c_3 & 0,85 & 0,60 & 0,67 & 0,86 & 0,75 \\ c_4 & 0,90 & 0,65 & 0,61 & 0,95 & 0,83 \\ c_5 & 0,0 & 0,94 & 0,89 & 0,60 & 0,76 \end{array} \right| \tag{4.2}$$

O parâmetro de vigilância ρ controla a sensibilidade da rede aos padrões de entrada, gerando mais ou menos agrupamentos. Nesta pesquisa, após experimentos, o valor adotado para ρ foi de $0,085$.

A tabela 4.1 exemplifica o resultado do processamento da rede neural para a matriz 4.2.

Tabela 4.1: Agrupamentos formados pela rede neural

Agrupamento	**Componentes**
0	c_2 e c_5
1	c_1, c_3 e c_4

Essas informações são mantidas pelo relacionamento das classes *Cluster* e *Component* (figura 4.6). Cada componente pertence a apenas um agrupamento, pois os conjuntos formados por esses rede neural são do tipo *flat partition*, ou seja, são conjuntos disjuntos.

Nesse submódulo também é executada a tarefa de rotulação dos agrupamentos. Essa rotulação consiste em identificar um conjunto de termos e suas relevâncias para representar cada um dos agrupamentos, possibilitanto a recuperação baseada em agrupamentos. Os termos e suas relevâncias são obtidos dos componentes que pertencem aos agrupamentos. Para isso é realizada a união entre os conjuntos de termos que representam cada componente. Para obter a relevância do termo para o agrupamento é adotado o maior valor de peso de termo entre os termos do conjunto.

Para representar o resultado do processo de rotulação para um agrupamento i, adotou-se a seguinte notação:

$Agr_i = \{(t_k; maiorpeso_k), k = 1, .., n\}$, onde n é o número total de termos e $maiorpeso_k$ é o maior peso do termo k entre os componentes do agrupamento. O $maiorpeso_k$ que representa a relevância do termo t_k para o agrupamento.

Para ilustrar o processo de rotulação considere os agrupamentos formados na tarefa anterior (tabela 4.1). Para cada agrupamento é obtida uma matriz, onde as linhas representam os componentes pertencentes ao agrupamento, as colunas os termos de índice e os valores representam os pesos dos termos para os componentes.

A matriz 4.3 representa o agrupamento 0.

$$MAGR_0 = \begin{vmatrix} & t_1 & t_2 & t_3 & t_4 & t_5 \\ c_2 & 0,62 & 0,86 & 0,95 & 0,0 & 0,65 \\ c_5 & 0,0 & 0,94 & 0,89 & 0,60 & 0,76 \end{vmatrix} \tag{4.3}$$

O resultado do processo de rotulação para o agrupamento 0 é dado a seguir:

$Agr_0 = \{(t_1; 0,62), (t_2; 0,94), (t_3; 0,95), (t_4; 0,60), (t_5; 0,76)\}$

A matriz 4.4 representa o agrupamento 1.

$$MAGR_1 = \begin{vmatrix} & t_1 & t_2 & t_3 & t_4 & t_5 \\ c_1 & 0,67 & 0,79 & 0,0 & 0,80 & 0,85 \\ c_3 & 0,85 & 0,60 & 0,67 & 0,86 & 0,75 \\ c_4 & 0,90 & 0,65 & 0,61 & 0,95 & 0,83 \end{vmatrix} \tag{4.4}$$

O resultado do processo de rotulação para o agrupamento 1 é representado da seguinte maneira:

$$Agr_1 = \{(t_1; 0,90), (t_2; 0,79), (t_3; 0,67), (t_4; 0,95), (t_5; 0,85)\}$$

Após a rotulação, essas informações são armazenadas no banco de dados conforme o diagrama de classes da figura 4.6. A classe *Term_Cluster_Relation* mantém a rotulação de cada agrupamento, armazenando para cada par ($term$,$cluster$) sua relevância (maior peso) no agrupamento.

Realizada a rotulação dos agrupamentos, um especialista é responsável por analisar essas informações para definir um nome para cada agrupamento que expresse as funcionalidades dos componentes que o compõem e o domínio ao qual ele pertence. O nome do agrupamento é mantido na classe Cluster e o nome do domínio é mantido na classe Domain da figura 4.6.

Além dessas informações semânticas são armazenadas no banco de dados informações da rede neural tais como a data do último processamento, o tempo do processamento, o valor do parâmetro de vigilância ρ e a memória que registra as informações dos agrupamentos formados pela rede neural. Essa informações são mantidas na classe Art-2A e são utilizadas quando novos componentes são submetidos ao servidor. Caso um novo componente seja submetido e seu termos já pertencem aos termos de índice é possível utilizar a memória da rede neural para reduzir o tempo gasto no processamento do processo de agrupamento. Entretanto se os termos não estiverem presentes não é possível utilizar a memória, sendo necessário reprocessar todo o conjunto de componentes armazenados. Nesta última situação é apresentado ao usuário o tempo gasto no último processamento para que ele possa decidir em qual momento deverá ser realizado o reagrupamento dos componentes.

4.3.3 Módulo de Recuperação

O módulo de recuperação é responsável por avaliar as consultas submetidas ao servidor e retornar os componentes mais adequados às necessidades expressas na consulta. Para validar a estratégia de busca baseada em agrupamentos, proposta nesta pesquisa, são utilizados dois modelos de recuperação clássicos: vetorial e booleano. A escolha por esses modelos deve-se ao fato de sua ampla adoção e consolidação na literatura (Baeza-Yates & Ribeiro-Neto, 1999; Salton & McGill, 1983).

No modelo vetorial as consultas são formuladas através de palavras-chave e suas respectivas relevâncias para a consulta. No modelo booleano as consultas são formuladas través de palavras-chave e operadores booleanos.

A recuperação de componentes pode ser baseada em termos fornecidos pelo usuário (seções 4.3.3.1 e 4.3.3.2) ou baseada em domínios (seção 4.3.3.3).

4.3.3.1 Estratégia de busca vetorial utilizando os agrupamentos

Para a estratégia de busca vetorial foram definidas duas abordagens, a primeira, considerando o agrupamento mais relevante e, a segunda, considerando um limite de similaridade aplicado a agrupamentos e aos componentes pertencentes a eles. Tais abordagens são detalhadas a seguir.

a) Abordagem pelo agrupamento mais relevante

A primeira abordagem adotada neste trabalho de mestrado para a utilização de agrupamentos, na estratégia de busca do modelo vetorial, é apresentada no algoritmo 4.1.

Dada uma determinada consulta $q = \{(t_1; r_1), t_2; r_2), ..., (t_k; r_k)\}$, onde $j = 1..k$, t_j representa um termo, r_j representa a relevância do termo t_j para a consulta e k é o número de termos da consulta. A estratégia de busca, inicialmente, aplica a cada termo da consulta o processo de normalização morfológica (linhas 3,4 e 5). Para cada termo normalizado são localizados no banco de dados os agrupamentos indexados por ele . O resultado dessa busca é um conjunto de agrupamentos que contêm ao menos um dos termos da consulta (linhas 7,8,9 e 10). Para cada agrupamento recuperado é calculado seu grau de similaridade com a consulta (linhas 11, 12 e 13). Esse grau é calculado através da equação 4.5, onde w_{iq} é o peso do i-ésimo termo da consulta e r_{ih} é a relevância do i-ésimo termo no agrupamento (a_h) obtida no processo de rotulação. Essa equação, proposta originalmente por Salton (Salton & McGill, 1983), foi aqui adaptada para ser aplicada em agrupamentos.

$$sim(q, a_h) = \frac{\sum_{i=1}^{n} w_{iq} * r_{ih}}{\sqrt{\sum_{i=1}^{n} w_{iq}^2} * \sqrt{\sum_{i=1}^{n} r_{ih}^2}} \tag{4.5}$$

Após a identificação do agrupamento mais similar à consulta é realizada uma busca no banco de dados para recuperar os componentes pertencentes a esse agrupamento (linhas 14 e 15). Para cada componente são localizadas e apresentadas suas informações estruturais ao usuário (linhas 16, 17, 18 e 19).

Para exemplificar a técnica de recuperação apresentada no algoritmo 4.1, considere a consulta $q = \{(t_1; 90\%), (t_5; 80\%)\}$, a coleção de componentes apresentada na tabela 4.2 e os agrupamentos apresentados na tabela 4.3.

Após os termos da consulta serem normalizados é realizada uma busca no banco de dados pelos agrupamentos indexados por pelo menos um desses termos. No exemplo, seriam retornados os dois agrupamentos (0 e 1). Em seguida é calculado o grau de similaridade entre a consulta e os agrupamentos (equação 4.5).

Algoritmo 4.1 Estratégia de busca vetorial utilizando agrupamentos *flat partition*

entrada: $q = \{(t_1; r_1), t_2; r_2), ..., (t_k; r_k)\}$
saída: conjunto de componentes
para todo termo t_k pertencente a q **faça**
    submeter t_k ao processo de normalização morfológica
fim para
$AgrCand = \{\}$
para todo termo t_k normalizado pertencente a q **faça**
    Agr = resultado da busca no banco de dados por agrupamentos indexados por t_k
    $AgrCand = AgrCand \cup Agr$
fim para
para todo agrupamento a_i pertencente a $AgrCand$ **faça**
    calcular o grau de similaridade utilizando a fórmula $sim(q, a_i)$
fim para
R = agrupamento mais similar à consulta
CR = resultado da busca por componentes pertencentes ao agrupamento R
para todo componente c_j de CR **faça**
    localizar no banco de dados as informações gerais estruturais.
    apresentar informações do componente para o usuário.
fim para

Tabela 4.2: Exemplo de uma coleção de componentes armazenados no banco de dados

Agrupamento	**Componentes**	**Termos de índice**				
		t_1	t_2	t_3	t_4	t_5
1	c_1	0,67	0,79	0,0	0,80	0,85
0	c_2	0,62	0,86	0,95	0,0	0,65
1	c_3	0,85	0,60	0,67	0,86	0,75
1	c_4	0,90	0,65	0,61	0,95	0,83
0	c_5	0,0	0,94	0,89	0,60	0,76

Tabela 4.3: Exemplo de rotulação dos agrupamentos

Agrupamentos	**Termo de índice**				
	t_1	t_2	t_3	t_4	t_5
0	0,62	0,94	0,95	0,60	0,76
1	0,90	0,79	0,67	0,95	0,85

$$sim(q, a_h) = \frac{(r_{1q} * r_{1a}) + (r_{2q} * r_{2a})}{\sqrt{w_{1q}^2 + w_{2q}^2} * \sqrt{w_{1a}^2 + w_{2a}^2}}$$

$$sim(q, Agr_0) = \frac{(0,90 * 0,62) + (0,80 * 0,76)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,62^2 + 0,76^2}} = 0,9872$$

$$sim(q, Agr_1) = \frac{(0,90 * 0,90) + (0,80 * 0,85)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,90^2 + 0,85^2}} = 0,9995$$

Nesse exemplo, o agrupamento 1 é 99,9% similar à consulta, sendo considerado parte do

conjunto resposta. Portanto seus componentes c_1, c_3 e c_4 são apresentados como resultado ao usuário.

b) Abordagem pelo limite de similaridade

Analisando os resultados de testes realizados utilizando a primeira estratégia de busca vetorial (algoritmo 4.1), apresentados no próximo capítulo, percebeu-se que esses resultados poderiam ser melhorados. Para isso foi estabelecido um critério de similaridade entre a consulta e os agrupamentos e, também, para o componentes.

A estratégia de busca vetorial utilizando agrupamentos e considerando o grau de similaridade é apresentada no algoritmo 4.2.

Os passos iniciais desse algoritmo são idênticos aos apresentados no algoritmo 4.1 (linhas 1 a 10). O diferencial é que para cada agrupamento recuperado é calculado seu grau de similaridade, conforme a equação 4.5, e, os agrupamentos que têm o grau de similaridade maior ou igual que 60% são incluídos no conjunto de agrupamentos relevantes (linhas 11, 12, 13, 14, 15 e 16). Neste trabalho admitiu-se um limite para a similaridade $s = 0,60$, pois outros trabalhos relatados na literatura (Salton & McGill, 1983; Baeza-Yates & Ribeiro-Neto, 1999) também adotam esse valor. Trabalhos futuros devem ser realizados para analisar valores ideais para s. Para cada agrupamento, pertencente ao conjunto de agrupamentos relevantes para a consulta, é localizado no banco de dados os componentes pertencentes a ele. Esses componentes são incluídos no conjunto de componentes candidatos (linhas 18, 19, 20 e 21). Para cada componente candidato é calculado seu grau de similaridade com a consulta. Esse grau é calculado através da equação 4.6 (Salton & McGill, 1983), onde w_{iq} é o peso do i-ésimo termo da consulta e w_{ij} é a relevância do i-ésimo termo no componente (c_j).

$$sim(q, c_j) = \frac{\sum_{i=1}^{n} w_{iq} * w_{ij}}{\sqrt{\sum_{i=1}^{n} w_{iq}^2} * \sqrt{\sum_{i=1}^{n} w_{ij}^2}} \tag{4.6}$$

Os componentes que têm o grau de similaridade maior que 60% são incluídos no conjunto de componentes relevantes (linhas 22, 23, 24, 25, 26 e 27). Esse conjunto é ordenado de acordo com o grau de similaridade, constituindo o conjunto resposta (linha 28). Para cada componente pertencente ao conjunto resposta são localizadas e apresentadas suas informações estruturais ao usuário (linhas 29,30,31 e 32).

Para exemplificar a técnica de recuperação apresentada no algoritmo 4.2, considere a consulta $q = \{(t_1; 90\%), (t_5; 80\%)\}$, a coleção de componentes apresentada na tabela 4.2 e os agrupamentos apresentados na tabela 4.3.

Após os termos da consulta serem normalizados é realizada uma busca no banco de dados pelos agrupamentos indexados por pelo menos um desses termos. No exemplo, seriam

Algoritmo 4.2 Estratégia de busca vetorial utilizando agrupamentos e considerando o grau de similaridade

1: entrada: $q = \{(t_1; r_1), t_2; r_2), ..., (t_k; r_k)\}$
2: saída: conjunto de componentes ordenados pelo grau de similaridade
3: **para todo** termo t_k pertencente a q **faça**
4: submeter o t t_k ao processo de normalização morfológica
5: **fim para**
6: $AgrCand = \{\}$
7: **para todo** termo t_k normalizado pertencente a q **faça**
8: Agr = resultado da busca no banco de dados por agrupamentos indexados por t_k
9: $AgrCand = AgrCand$ U Agr
10: **fim para**
11: **para todo** agrupamento a_i pertencente a $AgrCand$ **faça**
12: calcular o grau de similaridade utilizando a fórmula $sim(q, a_i)$
13: **se** $sim(q, a_i) > 0{,}60$ **então**
14: $AgrRel = AgrRel$ U $\{a_i\}$
15: **fim se**
16: **fim para**
17: $CompCand = \{\}$
18: **para todo** agrupamento a_i pertencente a $AgrRel$ **faça**
19: $Comp$ = resultado da busca no banco de dados por componentes pertencentes a a_i
20: $CompCand = CompCand$ U $Comp$
21: **fim para**
22: **para todo** componente c_j pertencente a $CompCand$ **faça**
23: calcular o grau de similaridade utilizando a fórmula $sim(q, c_j)$
24: **se** $sim(q, c_j) > 0{,}60$ **então**
25: $CompRel = c_j$
26: **fim se**
27: **fim para**
28: CR = conjunto de componentes pertencentes a $CompRel$, ordenados pelo grau de similaridade
29: **para todo** componente c_j de CR **faça**
30: localizar no banco de dados as informações gerais estruturais.
31: apresentar informações do componente para o usuário.
32: **fim para**

retornados os dois agrupamentos (0 e 1). Em seguida é calculado o grau de similaridade entre a consulta e os agrupamentos (equação 4.5).

$$sim(q, Agr_0) = \frac{(0,90 * 0,62) + (0,80 * 0,76)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,62^2 + 0,76^2}} = 0,9872$$

$$sim(q, Agr_1) = \frac{(0,90 * 0,90) + (0,80 * 0,85)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,90^2 + 0,85^2}} = 0,9995$$

Ambos os agrupamentos são considerados relevantes para a consulta pois o grau de similaridade é maior que o limite $s = 0,60$. Para cada um desses agrupamentos são localizados componentes, formando o conjunto de componentes candidatos. Para cada componente desse conjunto é calculado o grau de similaridade utilizando a equação 4.6.

Cálculo de similaridade para os componentes pertencentes ao agrupamento 1:

$$sim(q, c_1) = \frac{(0,90 * 0,67) + (0,80 * 0,85)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,67^2 + 0,85^2}} = 0,9844$$

$$sim(q, c_3) = \frac{(0,90 * 0,62) + (0,80 * 0,65)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,62^2 + 0,65^2}} = 0,9966$$

$$sim(q, c_4) = \frac{(0,90 * 0,85) + (0,80 * 0,75)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,85^2 + 0,75^2}} = 1$$

Cálculo de similaridade para os componentes pertencentes ao agrupamento 0:

$$sim(q, c_2) = \frac{(0,90 * 0,90) + (0,80 * 0,83)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,90^2 + 0,83^2}} = 1$$

$$sim(q, c_5) = \frac{(0,90 * 0,0) + (0,80 * 0,76)}{\sqrt{(0,90^2 + 0,80^2)} * \sqrt{0,0^2 + 0,76^2}} = 0,6644$$

O conjunto resposta é formado pelos cinco componentes pois todos têm o grau de similaridade acima do limite $s = 0,60$. Esse conjunto é ordenado de acordo com o grau de similaridade. Para cada componente pertencente a esse conjunto são localizadas e apresentadas suas informações estruturais ao usuário

4.3.3.2 Estratégia de busca booleana utilizando os agrupamentos

O algoritmo 4.3 apresenta a técnica adotada para recuperar componentes através da utilização de agrupamentos na estratégia de busca do modelo booleano. Dada uma determinada consulta q = $\{t_1\ \theta_1\ t_2\ \theta_2\ ...\ \theta_i\ t_k\}$, onde i representa um termo e θ_i representa um operador

booleano (*and* ou *or*). Como a ordem das operações lógicas de uma expressão booleana influencia no resultado da busca, nesta pesquisa foi adotada a ordem de execução da esquerda para a direita.

Para cada termo localizado na consulta é aplicado o processo de normalização morfológica e, para cada operador booleano é realizada a substituição pela ação que deve ser executada no banco de dados (linhas 4 a 16).

O resultado da análise da consulta é a expressão de busca que é avaliada contra a rotulação de cada agrupamento no banco de dados (linha 18). O resultado dessa busca é um conjunto de agrupamentos que satisfazem tal expressão de busca.

Para cada um dos agrupamentos desse conjunto são recuperados os componentes formando o conjunto resposta CR (linhas 20-23). Para cada componente pertencente ao conjunto resposta são recuperadas suas informações estruturais e apresentadas ao usuário.

Algoritmo 4.3 Estratégia de busca booleana utilizando agrupamentos

1: entrada: $q = \{t_1\ \theta_1\ t_2\ \theta_2\ ...\ \theta_i\ t_k\}$
2: saída: conjunto de componentes
3: inicializar $eb = \{\}$
4: **para todo** item it_j pertencente a q **faça**
5: **se** it_j é um termo t_k **então**
6: submeter o termo t_k ao processo de normalização morfológica
7: $eb = eb$ + termo t_k normalizado
8: **fim se**
9: **se** it_j é um operador booleano θ_i **então**
10: **se** $\theta_i = and$ **então**
11: $eb = eb$ + "intersect"
12: **senão**
13: $eb = eb$ + "union"
14: **fim se**
15: **fim se**
16: **fim para**
17: $Agr = \{\}$
18: Agr = resultado da busca no banco de dados por agrupamentos que satisfazem a expressão de busca eb
19: $CR = \{\}$
20: **para todo** agrupamento a_i pertencente a Agr **faça**
21: C = resultado da busca por componentes pertencentes ao agrupamento a_i
22: $CR = CR \cup C$
23: **fim para**
24: **para todo** componente c_j de CR **faça**
25: localizar no banco de dados as informações estruturais do componente c_j
26: apresentar informações do componente c_j para o usuário
27: **fim para**

Para exemplificar a busca booleana, suponha que a mesma consulta q, utilizada nos exemplos anteriores, tenha sido formulada utilizando operadores booleanos $q = t_1$ *AND* t_5. Ini-

cialmente é realizada a leitura da consulta, onde os termos são normalizados e os operadores booleanos substituídos pelas ações. O resultado desse processo para a consulta q é a expressão de busca eb = t_1 *intersect* t_5. Essa expressão é encaminhada para o banco de dados onde são localizados os agrupamentos que satisfazem essa expressão. O resultado dessa busca são os agrupamentos 0 e 1 ambos indexados pelos termos t_1 e t_5 conforme apresentado na tabela 4.4.

Tabela 4.4: Exemplo de rotulação dos agrupamentos

height**Agrupamentos**	**Termo de índice**				
	t_1	t_2	t_3	t_4	t_5
0	0,62	0,94	0,95	0,60	0,76
1	0,90	0,79	0,67	0,95	0,85

Então é exibido para o usuário, como conjunto resposta, os componentes c_2, c_5, c_1, c_3, c_4. Observe que o componente c_5 não é indexado pelo termo t_1, conforme a tabela 4.2, porém aparece no conjunto resposta pois pertence ao agrupamento 1 que possui esse termo em sua rotulação.

4.3.3.3 Recuperação baseada em Domínios

A utilização de domínios na recuperação de componentes possibilita um refinamento nas estratégias de busca, pois o usuário ao invés de especificar diretamente os termos, pode selecionar os domínios que ele tenha interesse.

Para exemplificar esse mecanismo de recuperação considere que os componentes c_1, c_2, c_3, c_4 e c_5, apresentados na tabela 4.2, pertencem ao domínio de aplicação Multimídia. Dentro desse domínio, o agrupamento 0 identifica os componentes que manipulam gráficos e o agrupamento 1 identifica os componentes que manipulam som e audio.

Neste caso, essa informação semântica, que está armazenada nas classes Domain (Aplicação Multimídia) e Cluster (Manipulação de gráficos, Manipulação de som e audio) da figura 4.6, é apresentada ao usuário para que ele possa selecionar o domínio de interesse. Em seguida são apresentados os nomes dos agrupamentos do domínio. Após o usuário selecionar um agrupamento são apresentados os componentes.

4.4 Comparação com trabalhos relacionados

As abordagens para classificação e recuperação de componentes de software em repositórios, apresentadas no capítulo 2, possibilitou a identificação dos metadados estruturais. A base para os metadados estruturais foi o trabalho apresentado por (Seacord *et al.*, 1998). Nesse trabalho, o autores utilizam nomes de métodos e atributos para recuperar os componentes. Neste trabalho de mestrado, esses dados foram utilizados e complementados para simplificar a apresentação da estrutura dos componentes recuperados.

A parte dos metadados baseados em conteúdo é a contribuição desta pesquisa, pois uniu os conceitos das abordagens de extração automática de informação de componentes com as abordagens que utilizam conceitos semânticos. Esses metadados possibilitam a utilização de diversas estratégias de busca tais como a busca vetorial utilizando palavras-chave fornecidas na consulta pelo usuário ou a navegação pelo repositório utilizando informações semânticas.

4.5 Considerações Finais

Neste capítulo foi apresentado o conjunto de metadados, proposto nesta pesquisa, para representar componentes de software em repositórios. Esses metadados são compostos por informações da estrutura do componente, denominados metadados estruturais, e por informações extraídas de sua documentação, denominados metadados baseados em conteúdo.

Os metadados estruturais descrevem informações gerais dos componentes tais como o nome, uma uma breve descrição de sua funcionalidade, a tecnologia utilizada para seu desenvolvimento, nome do desenvolvedor, nome da empresa ou instituição na qual o componente foi desenvolvido, versão, data da versão e referência para o local onde o componente está disponível. Além dessas informações gerais, são mantidas informações de seus atributos atributos, métodos e interfaces.

Os metadados baseados em conteúdo são termos extraídos da documentação do componente através de mecanismos de indexação automática da área de recuperação de informação. Esses termos foram utilizados para automaticamente agrupar os componentes no banco de dados. Para essa tarefa foi adotada uma arquitetura de rede neural auto-organizável ART-2A. Esses agrupamentos são a base das estratégias de busca para recuperação dos componentes.

Para validar a estratégia de busca baseada em agrupamentos foi utilizado o modelo de recuperação vetorial. Para a estratégia de busca vetorial foram propostas duas abordagens para recuperação: pelo agrupamento mais relevante e pelo limite de similaridade. Além disso foi proposta uma estratégia de busca utilizando domínios dos componentes. No próximo capítulo são apresentados os resultados realizados para avaliar essas estratégias de busca.

CAPÍTULO

5

Experimentos

5.1 Considerações Iniciais

Este capítulo apresenta experimentos que visam avaliar a estratégia de busca de componentes proposta neste trabalho. Foram realizados experimentos com classes da Java API e gerados resultados seguindo as métricas de precisão (*precision*) e revocação (*recall*). Além dessas métricas foram analisados os tempos despendidos nas buscas. As seções seguintes apresentam detalhes sobre os experimentos e seus resultados.

5.2 Métricas de Avaliação

Para avaliar os resultados são utilizadas as métricas de precisão (*precision*) e revocação (*recall*) apresentadas por Salton e McGill (Salton & McGill, 1983). Essas medidas são utilizadas em sistemas de recuperação de informação para verificar quão satisfatória foi a resposta para uma determinada consulta.

A precisão, equação 5.1 (onde: tir representa o total de itens relevantes na consulta e tr o total de itens recuperados do banco de dados), indica a quantidade de documentos relevantes para o usuário dentre os itens que foram retornados como resposta a uma busca.

$$P = \frac{tir}{tr} \tag{5.1}$$

A revocação, equação 5.2 (onde: $tirr$ representa o total de itens relevantes recuperados e ta representa o total de itens relevantes armazenados no banco de dados), indica a quantidade de itens relevantes recuperados, dentre os itens relevantes existentes na base de dados (definidos

por um especialista).

$$R = \frac{tirr}{ta} \tag{5.2}$$

Além dessas métricas, o tempo consumido em cada busca foi analisado. Para isso foi adotada a métrica de *speedup*, apresentada na equação 5.3, que reflete o ganho de desempenho da solução proposta sobre a estratégia de busca convencional. O *speedup* avalia em quantas vezes houve aumento de desempenho de um sistema otimizado (T_o) em relação a um sistema sem otimizações (T_{so}) (Almasi & Gottlieb, 1994).

$$S_p = \frac{T_{so}}{T_o} \tag{5.3}$$

5.3 Configuração do Ambiente

Os experimentos foram realizados sobre um conjunto de classes da Java API apresentado na tabela 5.1. A Java API (*Java Application Programming Interface*) é uma biblioteca de classes Java predefinidas que são agrupadas por categorias (pacotes). A documentação dos pacotes e classes estão disponíveis em (Microsystems, 2000b).

Esse conjunto foi adotado, pois a autora desta pesquisa de mestrado possui conhecimento sobre essas classes e suas funcionalidades o que permitiu a definição de um conjunto de consultas e, para cada consulta, a determinação das classes consideradas relevantes como resposta.

Foram definidas 30 consultas, apresentadas na tabela 5.7, e a distribuição destas em relação aos pacotes da Java Api foi: para o pacote java.net foram elaboradas 6 consultas (5.2); para o pacote java.util foram definidas 7 consultas (5.3); para o pacote java.io foram elaboradas 8 consultas (5.3) e, para o pacote java.awt foram definidas 9 consultas (5.5).

A primeira etapa para a realização dos experimentos foi a submissão das classes ao módulo de extração de informação. Como resultado dessa etapa foram obtidos 1553 termos representativos que foram armazenados em banco de dados. Para cada termo foi obtida a sua frequência e calculado seu peso, sendo essas informações armazenadas no banco de dados.

A etapa seguinte foi a execução do agrupamento dos componentes (classes Java). Para isso foi construída uma matriz, onde as linhas representam as classes Java e as colunas os termos, sendo que em cada posição $(classe_i, termo_j)$ tem-se o peso do $termo_j$ na $classe_i$. Essa matriz foi submetida à rede neural Art-2A.

Nesta etapa de agrupamento foram realizados alguns testes variando o parâmetro de vigilância ρ visando avaliar o número de classes contidas nos agrupamentos e sua capacidade de classificação. O valor inicial adotado para avaliar o parâmetro de vigilância foi de $0,75$, pois é considerado o valor ideal conforme (Mello *et al.*, 2004). Esse valor não gerou resultados

Tabela 5.1: Pacotes utilizados para o conjunto de teste

Pacote	Descrição	Classes
java.net	Disponibiliza classes para suporte de rede.	ContentHandler, DatagramPacket, DatagramSocket, InetAddress, ServerSocket, Socket, URL, URLConnection, URLEncoder, URLDecoder, URLStreamHandler, HttpURLConnection, URLClassLoader, Authenticator, PasswordAuthentication
java.util	Disponibiliza classes utilitárias para manipulação de estrutura de dados, conjuntos de bits, dados, *string*, etc.	Collections, Dictionary, Hashtable, LinkedList, TreeSet, Map, Stack, StringTokenizer, Vector, Calendar, GregorianCalendar, Locale, TimeZone, Timer, TimerTask
java.io	Disponibiliza classes para manipulação de arquivos.	BufferedInputStream, BufferedOutputStream, ByteArrayInputStream, ByteArrayOutputStream, DataInputStream, DataOutputStream, File, FileInputStream, FileOutputStream, InputStream, OutputStream, PipedInputStream, PipedOutputStream, PrintStream, StreamTokenizer, BufferedReader, BufferedWriter, CharArrayReader, CharArrayWriter, DataInput, DataOutput, FileReader, FileWriter, PipedReader, PipedWriter, PrintWriter, Writer, Reader, StringWriter, StringReader, PushbackInputStream, PushbackReader, ObjectInputStream, ObjectOutputStream, FilterReader, FilterWriter
java.awt	Disponibiliza classes para criação de interfaces gráficas.	BorderLayout, Button, Canvas, CardLayout, Checkbox, CheckboxGroup, CheckboxMenuItem, Choice, Color, Container, Cursor, Dialog, FlowLayout, GridBagLayout, GridLayout, Insets, Label, List, Menu, MenuBar, MenuItem, MenuShortcut, MediaTracker, Polygon, Rectangle, Point, PopupMenu, Scrollbar, TextArea, TextField, Panel, PageAttributes, JobAttributes, Frame

Tabela 5.2: Conjunto de consultas elaboradas para o pacote java.net

Consulta	Classes Relevantes
1	DatagramPacket e DatagramSocket
2	ServerSocket e Socket
10	InetAddress
17	Authenticator e PasswordAuthentication
18	ContentHandler, URL, URLConnection, URLStreamHandler e HttpURLConnection
28	HttpURLConnection

satisfatórios, sendo realizados testes com os valores de $\rho = 0,085$ e $\rho = 0,2$, que apresentaram uma classificação mais condizente com a funcionalidade das classes. Por exemplo, classes relacionadas com funcionalidade de entrada e saída foram colocadas em um mesmo agrupamento.

Tabela 5.3: Conjunto de consultas elaboradas para o pacote java.util

Consulta	Classes Relevantes
4	Stack, Vector, TreeSet e LinkedList
6	StringTokenizer
9	Timer e TimerTask
11	Dictionary
16	Calendar, TimeZone e GregorianCalendar
19	Collections
20	TreeSet

Tabela 5.4: Conjunto de consultas elaboradas para o pacote java.io

Consulta	Classes Relevantes
7	BufferedInputStream, BufferedOutputStream, ByteArrayInputStream, ByteArrayOutputStream, DataInputStream, DataOutputStream, FileInputStream, FileOutputStream, InputStream, OutputStream, PipedInputStream, PipedOutputStream, PrintStream, PushbackInputStream, ObjectInputStream e ObjectOutputStream
8	File
15	BufferedReader, BufferedWriter, CharArrayReader, CharArrayWriter, FileReader, FileWriter, PipedReader, PipedWriter, PrintWriter, Writer, Reader, StringWriter, StringReader, PushbackReader, FilterReader e FilterWriter
23	CharArrayReader e CharArrayWriter
24	ObjectInputStream e ObjectOutputStream
26	PushbackReader e PushbackInputStream
27	StreamTokenizer

Tabela 5.5: Conjunto de consultas elaboradas para o pacote java.awt

Consulta	Classes Relevantes
3	BorderLayout, Container, CardLayout, FlowLayout, GridBagLayout e GridLayout
5	Point, Polygon e Rectangle
12	Button, Canvas, Checkbox, CheckboxGroup, CheckboxMenuItem, Choice, Cursor, Dialog, Label, List, Menu, MenuBar, MenuItem, PopupMenu, Scrollbar, TextArea e TextField
13	JobAttributes e PageAttributes
14	Dialog e Frame
21	Color
22	TextArea, TextField e Label
25	PipedReader, PipedWriter, PipedInputStream e PipedOutputStream
29	FlowLayout
30	Menu, MenuItem e PopupMenu

Para o parâmetro de vigilância igual a $0,085$ foram criados 17 grupos. Para o parâmetro de vigilância igual a $0,2$ foram criados 26 grupos. Melhores resultados foram observados com o valor de $0,085$, pois houve capacidade da rede em generalizar o conjunto de dados e melhor agrupar classes de acordo com suas funcionalidades. A tabela 5.6 apresenta os agrupamentos gerados para o conjunto de 100 classes.

Tabela 5.6: Agrupamentos formados para o conjunto de teste (100 classes)

Agr.	**Classes**	**Nome do Agrupamento**
0	Calendar, TimeZone, Locale, GregorianCalendar	Manipulação de Data e Hora
1	Rectangle, Polygon, Point	Manipulação de figuras geométricas
2	PasswordAuthentication, Authenticator	Autenticação de usuários
3	CharArrayWriter, FilterWriter, Reader, PipedWriter, DataOutputStream, ByteArrayOutputStream, OutputStream, CharArrayReader, BufferedWriter, ObjectInputStream, DataOutput, FileWriter, PipedInputStream, ObjectOutputStream, Writer, BufferedOutputStream, ByteArrayInputStream, PipedOutputStream, StringWriter, PrintWriter	Entrada e Saída
4	Hashtable, Collections, Dictionary, Map	Estrutura de Dados
5	StreamTokenizer, StringTokenizer	Identificação de tokens
6	URLStreamHandler, DatagramSocket, ContentHandler, InetAddress, URL, URLClassLoader, ServerSocket, DatagramPacket, Socket	Serviços de rede
7	HttpURLConnection, MediaTracker, URLConnection	Gerenciamento na execução de atividades
8	Container, GridBagLayout, CardLayout, MenuItem, Choice, Panel, Checkbox, Frame, Scrollbar, BorderLayout, Label, GridLayout, Color, TextArea, List, Button, Canvas, FlowLayout, TextField, Insets	Interface gráfica
9	Stack, Vector, TreeSet, LinkedList	Estrutura de dados
10	MenuBar, PopupMenu, Menu, CheckboxMenuItem, MenuShortcut	Componentes gráficos
11	Timer, TimerTask	Agendamento de tarefas
12	URLEncoder, URLDecoder	
13	JobAttributes, Dialog, PageAttributes	Gerenciamento de impressão
14	FileReader, FileInputStream, PipedReader, StringReader, PushbackInputStream, PushbackReader, DataInput, BufferedReader, DataInputStream, InputStream, FilterReader, BufferedInputStream, FileOutputStream, File	Entrada e saída
15	Cursor	Componente gráfico
16	CheckboxGroup	Componente gráfico

Analisando os agrupamentos obtidos observou-se que grande parte deles foi homogênea em termos dos pacotes aos quais as classes pertenciam. Somente dois agrupamentos apresentaram algum grau de heterogeneidade (agrupamentos 5 e 7), ou seja, reuniram componentes de pacotes distintos. Isso ocorreu pois os componentes pertencentes a esses agrupamentos apresentam um conjunto de termos muito similares, identificando uma outra funcionalidade diferente da

qual foi adotada na criação dos pacotes da Java API. Por exemplo, o agrupamento 5 representa classes que realizam o processo de $tokenizer$, ou seja, identificam $tokens$ em segmentos de texto ou fluxo de bytes. O agrupamento 7 representa classes que acompanham (gerenciam) a execução de uma determinada atividade. Nesse caso, a classe MediaTracker acompanha a execução de uma mídia e as classes HttpURLConnection e URLConnection acompanham a execução de uma conexão através do protocolo http.

Para cada agrupamento gerado foi definido um nome que serve para identificar a funcionalidade das classes do agrupamento. Essa atividade é realizada por um especialista do domínio das classes.

5.4 Aplicação da Estratégia de Busca Vetorial

Para a estratégia de busca vetorial foram definidas duas abordagens, a primeira, considerando o agrupamento mais relevante e, a segunda, considerando um limite de similaridade aplicado a agrupamentos e aos componentes pertencentes a eles, conforme descritos no capítulo 4.

Para avaliar essas abordagens foram utilizadas as consultas apresentadas nas tabelas 5.2, 5.3, 5.4 e 5.5. Essas consultas foram formuladas com os termos obedecendo a forma (t_k, r_k), onde t_k representa um termo e r_k a relevância do termo para a consulta (tabelas 5.7 e 5.8). Os resultados de tais abordagens são apresentadas na próximas seções.

5.4.1 Avaliação da abordagem pelo agrupamento mais relevante

Nesta abordagem o conjunto resposta é formado pelos componentes que pertencem ao agrupamento mais similar à consulta. Os resultados dessa abordagem são comparados aos resultados da busca vetorial convencional proposta por Salton (Salton & McGill, 1983). A tabela 5.9 apresenta os resultados em termos de precisão e revocação para cada consulta.

Os campos P_{vet} e R_{vet} indicam, respectivamente, a precisão e revocação da estratégia de busca vetorial convencional. Os campos P_{agr} e R_{agr} indicam, respectivamente, a precisão e revocação da estratégia de busca vetorial utilizando os agrupamentos obtidos por meio da rede neural Art-2A.

Analisando a tabela 5.9 observa-se que nas consultas $14, 16, 18, 21, 23, 25, 29, 30$ a busca vetorial convencional teve melhores resultados na precisão. Essa situação ocorre nas consultas relacionadas às classes pertencentes aos pacotes java.io e java.awt que são os mais populosos dentre os pacotes utilizados (vide tabela 5.1). Esses pacotes não foram subdivididos em muitos agrupamentos, pois suas classes apresentam um conjunto de termos muito similares.

Tabela 5.7: Consultas submetidas para a avaliação das estratégias de busca vetorial

Consulta	**Objetivo do usuário**
$q_1 = \{(sends, 80\%), (receives, 80\%), (packets, 95\%)\}$	obter classes que enviam e recebem pacotes pela rede
$q_2 = \{(port, 80\%), (host, 75\%), (socket, 95\%)\}$	obter classes que disponibilizem serviços na rede
$q_3 = \{(interface, 80\%), (window, 85\%), (layout, 90\%)\}$	obter classes para construção de interfaces gráficas
$q_4 = \{(vectors, 90\%), (arrays, 85\%)\}$	obter classes que manipulam estruturas de dados
$q_5 = \{(polygon, 90\%), (rectangle, 90\%)\}$	obter classes para criação de figuras geométricas
$q_6 = \{(break, 80\%), (string, 70\%), (tokens, 90\%)\}$	obter classes que manipulem sequência de caracteres
$q_7 = \{(read, 80\%), (write, 80\%), (file, 70\%), (bytes, 75\%), (streams, 90\%)\}$	obter classes que permitem a leitura e escrita de arquivos
$q_8 = \{(represent, 75\%), (file, 95\%), (directory, 95\%)\}$	obter classes que representam arquivos ou diretórios
$q_9 = \{(schedules, 90\%), (delay, 70\%), (task, 85\%)\}$	obter classes que permitem o agendamento de execução de tarefas
$q_{10} = \{(host, 100\%), (address, 100\%)\}$	obter classes que representem endereço IP
$q_{11} = \{(dictionary, 95\%), (key, 80\%), (value, 70\%)\}$	obter classes que manipulem estruturas de dados do tipo dicionário
$q_{12} = \{(widget, 90\%), (components, 85\%), (graphic, 80\%), (event, 80\%)\}$	obter classes que representam objetos gráficos
$q_{13} = \{(job, 85\%), (print, 95\%), (page, 80\%), (document, 75\%)\}$	obter classes que controlem impressão de arquivos
$q_{14} = \{(window, 95\%), (title, 80\%), (border, 80\%)\}$	obter classes que permitem o gerenciamento de janelas gráficas
$q_{15} = \{(read, 80\%), (write, 80\%), (character, 95\%), (files, 85\%)\}$	obter classes de leitura e arquivo de caracteres em arquivos
$q_{16} = \{(calendar, 95\%), (time, 80\%), (zone, 75\%)\}$	obter classes que manipulam datas
$q_{17} = \{(authentication, 95\%), (password, 85\%)\}$	obter classes que realizam autenticações em rede
$q_{18} = \{(connection, 85\%), (url, 90\%)\}$	obter classes que estabelecem conexões através de urls

Para ilustrar esse problema, considere a consulta $q_{21} = \{(rgb, 90\%), (color, 95\%), (red, 80\%), (green, 80\%), (blue, 80\%)\}$, cuja resposta esperada é apenas a classe Color , conforme a tabela 5.5.

Na busca vetorial convencional foram retornadas somente três classes, o que implica em P_{vet} = $1/3$ = $33,33\%$.

Na busca vetorial utilizando os agrupamentos foram retornadas 20 classes, pois elas pertencem ao agrupamento mais similar à consulta, implicando que sua precisão caísse para P_{agr} = $1/20$ = $5,00\%$.

Nas consultas 18 e 23 ocorreu outra situação problema referente à abordagem adotada.

Tabela 5.8: Consultas submetidas para a avaliação das estratégias de busca vetorial

Consulta	Objetivo do usuário
$q_{19} = \{(binary, 90\%), (search, 95\%), (sort, 85\%)\}$	obter classes que realizam buscam binária
$q_{20} = \{(sorted, 90\%), (set, 85\%), (elements, 80\%)\}$	obter classes que manipulam conjunto de dados ordenados
$q_{21} = \{(rgb, 90\%), (color, 95\%), (red, 80\%), (green, 80\%), (blue, 80\%)\}$	obter classes de gerenciamento de cores em interfaces gráficas
$q_{22} = \{(graphics, 85\%), (text, 90\%), (component, 80\%)\}$	obter classes que permitem manipulação de textos em interfaces gráficas
$q_{23} = \{(reads, 80\%), (writes, 80\%), (character, 95\%), (array, 90\%)\}$	obter classes de leitura e escrita de caracteres em arrays
$q_{24} = \{(storage, 85\%), (objects, 95\%), (file, 85\%), (stream, 70\%)\}$	obter classes que permitem a persistência de objetos em arquivos
$q_{25} = \{(thread, 85\%), (read, 80\%), (write, 80\%), (data, 75\%)\}$	obter classes de leitura e escrita de dados em threads
$q_{26} = \{(data, 75\%), (pushed, 85\%), (back, 80\%), (stream, 80\%)\}$	obter classes que enviam dados para um buffer de escrita
$q_{27} = \{(stream, 95\%), (tokenizer, 90\%)\}$	obter classes que permitem ler partes de uma sequencia de dados
$q_{28} = \{(request, 80\%), (http, 90\%), (server, 75\%)\}$	obter classes que estabelecem conexões com protocolo http.
$q_{29} = \{(components, 80\%), (left, 85\%), (right, 85\%), (flow, 85\%)\}$	obter classes que disponibilizem componentes gráficos da esquerda para a direita em uma interface gráfica.
$q_{30} = \{(menu, 80\%), (popup, 95\%)\}$	obter classes para criação de menus gráficos

A precisão e revocação nessas consultas foi de $0,0\%$, pois a resposta não contemplou qualquer componente relevante esperado. Esses componentes pertenciam a outros agrupamentos que não foram considerados na resposta.

Em relação à revocação observa-se que nas consultas $7, 14, 15, 25$ e 30 a busca vetorial convencional teve melhores resultados na revocação. Isso ocorreu, pois as classes relevantes para a consulta estavam dispersas nos agrupamentos, consequentemente, algumas delas não foram consideradas na resposta devido à abordagem adotada.

Analisando o conjunto total desses experimentos é possível comprovar que há uma melhoria média de $15,21\%$ na precisão obtida aplicando a estratégia de busca vetorial utilizando os agrupamentos sobre a estratégia de busca vetorial convencional. Os agrupamentos permitem retornar componentes mais relacionados e relevantes a uma determinada consulta. Contudo houve perda de 13,17% na revocação.

Os experimentos mencionados acima foram realizados sobre os agrupamentos obtidos usando o parâmetro de vigilância $\rho = 0,085$ para a rede neural.

Foram realizados os mesmos experimentos utilizando parâmetro de relevância $\rho = 0,2$.

Tabela 5.9: Resultados da abordagem vetorial pelo agrupamento mais relevante

Consulta	P_{vet}	P_{agr}	**Melhoria**	R_{vet}	R_{agr}	**Melhoria**
1	25,00%	25,00%	0,00%	100,00%	100,00%	0,00
2	20,00%	25,00%	5,00%	100,00%	100,00%	0,00
3	28,57%	30,00%	1,43%	100,00%	100,00%	0,00
4	6,90%	100,00%	93,10%	100,00%	100,00%	0,00
5	20,00%	66,67%	46,67%	100,00%	100,00%	0,00
6	1,03%	50,00%	48,97%	100,00%	100,00%	0,00
7	44,12%	61,11%	16,99%	100,00%	73,33%	-26,67%
8	2,44%	100,00%	97,56%	100,00%	100,00%	0,00
9	100,00%	100,00%	0,00%	100,00%	100,00%	0,00
10	6,25%	10,00%	3,75%	100,00%	100,00%	0,00
11	5,56%	25,00%	19,44%	100,00%	100,00%	0,00
12	26,32%	50,00%	23,68%	29,41%	58,82%	29,41%
13	50,00%	66,67%	16,67%	100,00%	100,00%	0,00
14	25,00%	5,00%	-20,00%	100,00%	50,00%	-50,00%
15	41,67%	62,50%	20,83%	93,75%	62,50%	-31,25%
16	100,00%	75,00%	-25,00%	100,00%	100,00%	0,00
17	33,33%	100,00%	66,67%	100,00%	100,00%	0,00
18	22,73%	0,00%	-22,73%	100,00%	0,00%	-100,00%
19	11,11%	25,00%	13,89%	100,00%	100,00%	0,00
20	11,11%	25,00%	13,89%	100,00%	100,00%	0,00
21	33,33%	5,00%	-28,33%	100,00%	100,00%	0,00
22	12,00%	15,00%	3,00%	100,00%	100,00%	0,00
23	6,06%	0,00%	-6,06%	100,00%	0,00%	-100,00%
24	5,88%	11,11%	5,23%	100,00%	100,00%	0,00
25	17,39%	11,11%	-6,28%	100,00%	50,00%	-50,00%
26	6,25%	11,11%	4,86%	100,00%	100,00%	0,00
27	2,33%	50,00%	47,67%	100,00%	100,00%	0,00
28	3,03%	33,33%	30,30%	100,00%	100,00%	0,00
29	9,09%	5,00%	-4,09%	100,00%	100,00%	0,00
30	15,79%	5,00%	-10,79%	100,00%	33,33%	-66,67%
		Média	15,21%		**Média**	-13,17%

Nesse caso houve uma pequena melhora na precisão, contudo houve perda na revocação. A precisão melhorou pois os agrupamentos criados foram mais específicos, contendo classes muito similares entre si. Consequentemente, o número de classes por agrupamento foi menor, reduzindo a quantidade de classes recuperadas e piorando a revocação.

Além dos resultados de precisão e revocação foram avaliados os tempos consumidos nas buscas, incluindo a busca no banco de dados pelos agrupamentos e componentes e os cálculos de similaridade (ver figura 5.1).

Foi observado que o tempo médio consumido na busca vetorial convencional foi de $904,8$ milisegundos, enquanto que na busca baseada em agrupamentos, proposta nesta pesquisa, foi de $192,9$ milisegundos. Esse aumento de desempenho ocorre dado que há uma redução no espaço

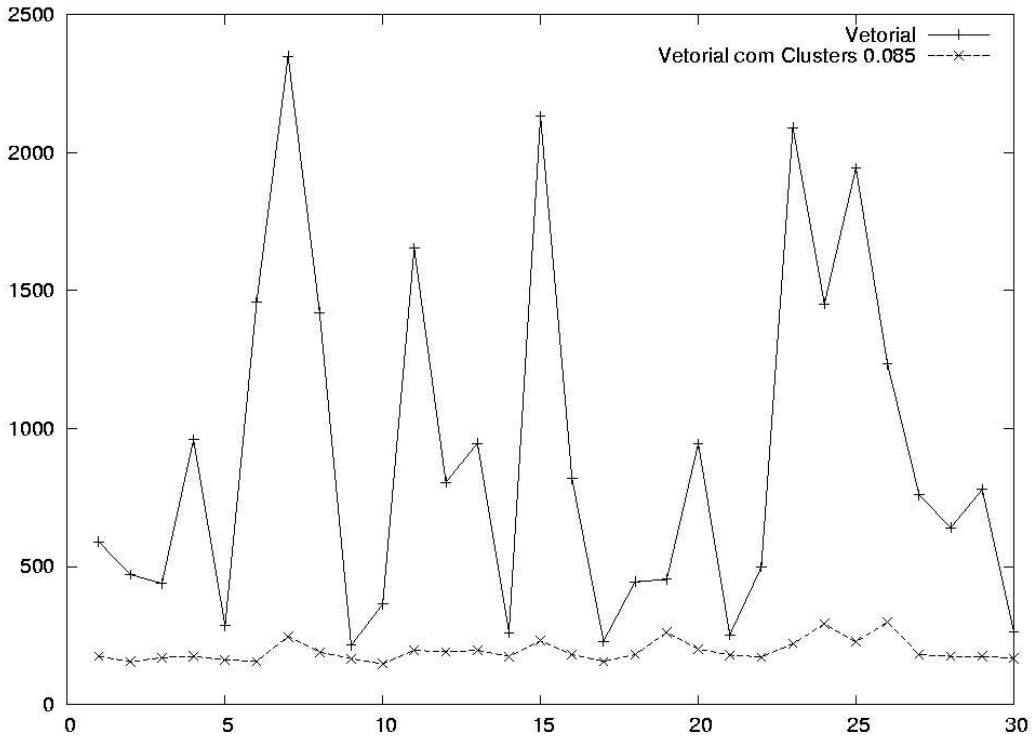

Figura 5.1: Tempo consumido na busca vetorial

de busca e, além disso, na abordagem adotada, o cálculo de similaridade é realizado somente para os agrupamentos. A abordagem adotada atinge um *speedup* de 4,69 sobre a estratégia de busca vetorial convencional.

5.4.2 Avaliação da abordagem pelo limite de similaridade

Nesta abordagem é definido um limite de similaridade entre a consulta e os agrupamentos. O valor adotado foi de 60%. Além de aplicar esse limite de similaridade aos agrupamentos, também é realizado o cálculo de similaridade nos componentes pertencentes aos agrupamentos. Esse limite também é utilizado para apresentar os resultados do modelo vetorial convencional.

A tabela 5.10 mostra os resultados em termos de precisão e revocação para cada consulta. Os campos P_{vet} e R_{vet} indicam, respectivamente, a precisão e revocação da estratégia de busca vetorial convencional. Os campos P_{agr} e R_{agr} indicam, respectivamente, a precisão e revocação da estratégia de busca vetorial utilizando os agrupamentos com grau de similaridade de 60%.

Analisando a tabela 5.10 observa-se que a utilização de limites de similaridade tanto para os agrupamentos como para os componentes pertencentes a eles melhorou os resultados da busca vetorial utilizando os agrupamentos. Nesse caso, é possível comprovar que há uma melhoria média de $9,55\%$ na precisão obtida aplicando estratégia de busca vetorial utilizando os agrupamentos com similaridade. Embora esta média seja menor do que aquela obtida com a abordagem vetorial pelo agrupamento mais relevante (tabela 5.9), observa-se que os problemas

Tabela 5.10: Resultados da abordagem vetorial pelo limite de similaridade

Consulta	P_{vet}	P_{agr}	**Melhoria**	R_{vet}	R_{agr}	**Melhoria**
1	25,00%	40,00%	15,00%	100,00%	100,00%	0,00%
2	20,00%	25,00%	5,00%	100,00%	100,00%	0,00%
3	28,57%	31,58%	3,01%	100,00%	100,00%	0,00%
4	6,90%	8,89%	1,99%	100,00%	100,00%	0,00%
5	20,00%	25,00%	5,00%	100,00%	100,00%	0,00%
6	1,03%	50,00%	48,97%	100,00%	100,00%	0,00%
7	44,12%	44,12%	0,00%	100,00%	100,00%	0,00%
8	2,44%	4,00%	1,56%	100,00%	100,00%	0,00%
9	100,00%	100,00%	0,00%	100,00%	100,00%	0,00%
10	6,25%	12,50%	6,25%	100,00%	100,00%	0,00%
11	5,56%	6,25%	0,69%	100,00%	100,00%	0,00%
12	26,32%	28,57%	2,26%	29,41%	23,53%	-5,88%
13	50,00%	100,00%	50,00%	100,00%	100,00%	0,00%
14	25,00%	28,57%	3,57%	100,00%	100,00%	0,00%
15	41,67%	63,64%	21,97%	93,75%	87,50%	-6,25%
16	100,00%	100,00%	0,00%	100,00%	100,00%	0,00%
17	33,33%	100,00%	66,67%	100,00%	100,00%	0,00%
18	22,73%	29,41%	6,68%	100,00%	100,00%	0,00%
19	11,11%	14,29%	3,17%	100,00%	100,00%	0,00%
20	11,11%	33,33%	22,22%	100,00%	100,00%	0,00%
21	33,33%	33,33%	0,00%	100,00%	100,00%	0,00%
22	12,00%	15,79%	3,79%	100,00%	100,00%	0,00%
23	6,06%	7,14%	1,08%	100,00%	100,00%	0,00%
24	5,88%	10,53%	4,64%	100,00%	100,00%	0,00%
25	17,39%	20,00%	2,61%	100,00%	100,00%	0,00%
26	6,25%	7,69%	1,44%	100,00%	100,00%	0,00%
27	2,33%	2,38%	0,06%	100,00%	100,00%	0,00%
28	3,03%	10,00%	6,97%	100,00%	100,00%	0,00%
29	9,09%	10,00%	0,91%	100,00%	100,00%	0,00%
30	15,79%	16,67%	0,88%	100,00%	100,00%	0,00%
		Média	9,55%		**Média**	-0,40%

reportados na primeira abordagem foram sanados.

Em relação a revocação o resultado foi muito próximo ao modelo vetorial convencional, o que não ocorreu na primeira abordagem.

Além dos resultados de precisão e revocação foram avaliados os tempos consumidos nas buscas, incluindo a busca no banco de dados pelos componentes e os cálculos de similaridade (ver figura 5.2).

Foi observado que o tempo médio consumido na busca vetorial convencional foi de $904,8$ milisegundos, enquanto que na busca baseada em agrupamentos, proposta nesta pesquisa, foi de $734,97$ milisegundos. A estratégia proposta atinge um *speedup* de $1,2310$, que reflete o ganho de desempenho da solução proposta sobre a estratégia de busca vetorial convencional. Em relação

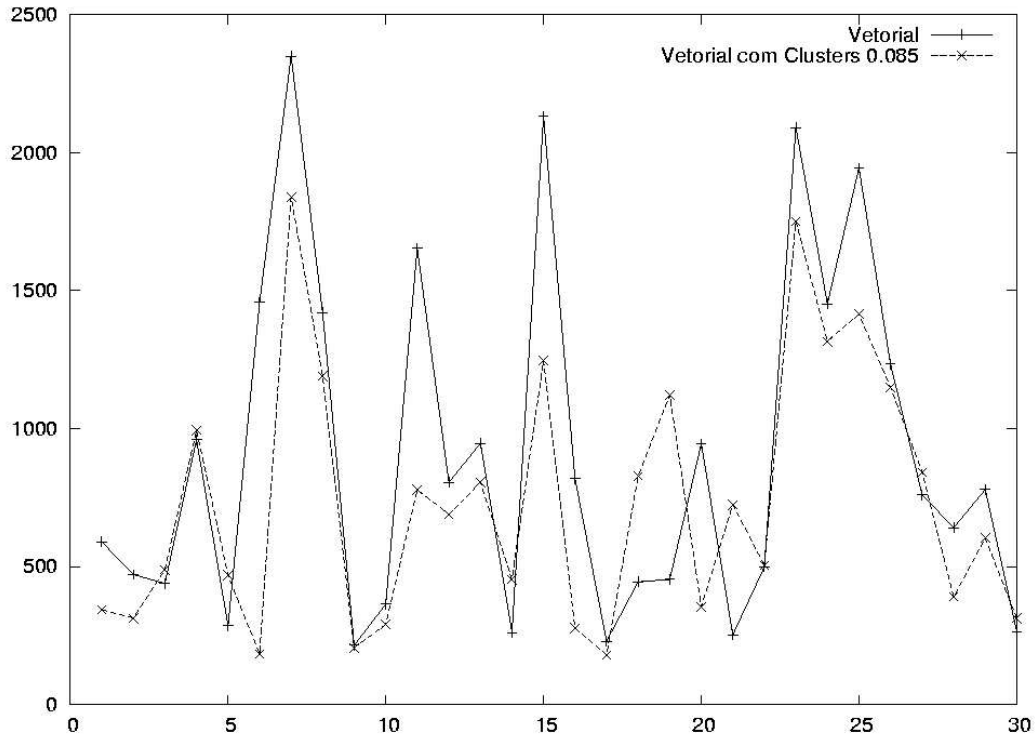

Figura 5.2: Tempo consumido na busca vetorial utilizando limite de similaridade

ao *speedup* obtido na estratégia anterior houve uma queda devido à necessidade de cálculo de similaridade para cada componente nos agrupamentos obtidos.

5.5 Considerações Finais

Este capítulo apresentou os resultados dos experimentos realizados para a estratégia de busca vetorial apresentada no capítulo 4. Para realizar esses experimentos foi selecionado um conjunto de classes da Java API e definido um conjunto de consultas, onde para cada consulta foram determinadas as classes consideradas relevantes como resposta.

Para a estratégia de busca vetorial foram definidas duas abordagens de recuperação: pelo agrupamento mais relevante e pelo limite de similaridade.

A abordagem pelo agrupamento mais relevante apresentou bons resultados nas medidas de precisão e tempo de execução de cada consulta. Contudo em algumas situações apresentou resultados inferiores aos resultados da estratégia de busca vetorial convencional. Além disso, houve uma perda em relação à revocação. Esse problemas ocorreram pois nessa abordagem foram considerados na resposta somente os componentes pertencentes ao agrupamento mais similar a consulta.

A abordagem pelo limite de similaridade apresentou bons resultados nas medidas de precisão e revocação, solucionando os problemas encontrados na primeira abordagem. Nessa

abordagem foram considerados os agrupamentos e, posteriormente, os componentes que apresentavam um grau de similaridade miníma com a consulta.

CAPÍTULO

6

Conclusões

6.1 Considerações Finais

Esta pesquisa de mestrado está inserida no projeto intitulado Desenvolvimento de Software Baseado em Componentes Multimídia (DBCM), desenvolvido pelo grupo de Engenharia de Software e Banco de Dados do Departamento de Computação da Universidade Federal de São Carlos (Vieira, 2003). Esse projeto tem por objetivo a pesquisa de técnicas, métodos e ferramentas que visam a definição de estratégias para auxiliar no desenvolvimento de software baseado em componentes multimídia. Dentro desse contexto, esta pesquisa de mestrado teve como objetivo contribuir para o desenvolvimento de um servidor de componentes, responsável por gerenciar metadados extraídos de componentes de software e fornecer mecanismos eficientes para localizá-los em um banco de dados.

6.2 Contribuições

O estudo das abordagens de classificação e recuperação de componentes, apresentado no capítulo 2, possibilitou a definição de um novo conjunto de metadados. Esses metadados foram divididos em categorias: metadados estruturais e metadados baseados em conteúdo. Os metadados estruturais representam informações da estrutura do componente e foram baseados no trabalho de (Seacord *et al.*, 1998). Os metadados baseados em conteúdo foram a contribuição desta pesquisa, pois uniu os conceitos das abordagens de extração automática de informação de componentes com as abordagens que utilizam conceitos semânticos.

Esses metadados são informações extraídas automaticamente da documentação e que são utilizadas no processo de classificação formando os agrupamentos de componentes. Cada

agrupamento é rotulado com os termos dos componentes que pertencem a ele. As estratégias de busca propostas nesta pesquisa consideram inicialmente as informações dos agrupamentos para posteriormente realizar a busca nos componentes. Essas estratégias promoveram uma recuperação eficaz dos componentes armazenados no banco de dados. Os resultados experimentais, apresentados no capítulo 5, comprovam a eficácia dessas estratégias.

A pesquisa desenvolvida extrapola os limites do projeto DBCM podendo ser aplicada na construção de repositórios de quaisquer componentes de software, conforme foi comprovado com os experimentos apresentados no capítulo 5.

6.3 Trabalhos Futuros

Dentre os trabalhos futuros estão:

- definir uma interface gráfica para que especialistas possam analisar os agrupamentos formados pela rede neural, definindo semanticamente os domínios de aplicação.
- comparar resultados dos experimentos realizados nesta pesquisa com o trabalho desenvolvido para representação de componentes através de metadados semânticos baseados em ontologias.
- verificar a viabilidade do uso de agrupamentos com os modelos de recuperação booleano e booleano estendido.
- realizar testes envolvendo um número maior de componentes de software.

APÊNDICE

A

Apêndice A

Neste apêndice é apresentada uma lista de *stopwords* criada especialmente para componentes de software que foram implementados utilizando a linguagem Java.

Tabela A.1: Lista de *stopwords para a linguagem Java*

abstract	index	public
api	independent	return
argument	indistinguishable	returns
assert	indistinct	real
awt	indistinctly	set
bean	implements	sets
beans	inherited	serializable
boolean	interface	serialized
byte	interfaces	static
break	integer	synchronized
char	instanceof	switch
class	import	short
classes	inner	subclasse
com	int	subclasses
constructor	io	super
case	java	summary
continue	javax	sun
component	lang	set
catch	long	tree
deprecated	method	util
default	meaning	try
double	malform	throw
detail	nbsp	throws
else	null	void
extends	overview	current
false	object	exception
frames	private	except
field	protected	
fields	public	
final	package	
finally	panel	
float	previous	
get	prev	
gets	property	
help	properties	

Referências

Almasi, G.; Gottlieb, A. (1994). *Highly Parallel Computing*. Benjamin/Cummings Publishing Company Inc.

Bachmann, F.; Bass, L.; Buhman, C.; Comella-Dorda, S.; Long, F.; Robert, J.; Seacord, R.; Wallnau, K. (2000). Volume ii: Technical concepts of component-based software engineering, 2nd edition. Relatório Técnico CMU/SEI-2000-TR-008, Software Engineering Institute.

Baeza-Yates, R.; Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison Wesley.

Burton, B.; Aragon, S.; Bailey, S.; Koehlerand, S.; Mayes, L. (1987). The reusable software library. *IEEE Software*, v.4, n.4, p.25–33.

Carpenter, G.; Grossberg, S.; Rosen, D. (1991). Art 2a: An adaptive resonance algorithm for rapid category learning and recognition. *Neural Networks*, v.4, n.4, p.34–504.

Chen, P. S.; Hennicker, R.; Jarke, M. (1993). On the retrieval of reusable software components. *2nd Int. Workshop on Software Reusability*, p. 99–107, Lucca,Italy. IEEE Computer Society.

Conrad, G.; Al-Kofahi, K.; Zhao, Y.; Karypis, G. (2005). Document clustering for large heterogeneous law firm collections. *ICAIL*, v.29, p.697–717.

Corporation, M. (1996). Distributed component object model protocol-dcom - technical overview. *URL: http://msdn.microsoft.com/library/default.asp – Consultado em: 06/04/2005*.

Corporation, M. (2003). Microsof component object model home. *URL: http://www.microsoft.com/com/default.asp – Consultado em: 06/04/2005*.

Corrêa, A. C. G. (2003). Recuperação de documentos baseada em informação semântica no ambiente ammo. Dissertação de mestrado, Universidade Federal de São Carlos.

Dobrynin, V.; Patterson, D.; Galushka, M.; Rooney, N. (2005). Sophia: An interactive cluster-based retrieved system for the ohsumed collection. *IEEE Transactions on Information Technology in Biomedicine*, v.9, n.2, p.256–265.

Fazli, C.; Altingovde, S.; Demir, E. (2004). Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems*, v.29, p.697–717.

Frakes, W.; Nejmeh, B. (1987). An information system for software reuse. *Proceedings of the Tenth Minnorbrook Workshop on Software Reuse*.

Frakes, W. B.; Pole, P. T. (1994). An empirical study of representation methods for reusable software components. *IEEE Trans. Softw. Eng.*, v.20, n.8, p.617–630.

Frank, E. (2000). Kea: Practical automatic keyphrase extraction. Relatório Técnico Working Paper 00/5, Departament of Computer Science, The University of Waikato.

Group, O. M. (2002). Corba components 3.0 - component model. *URL: http://www.omg.org/cgi-bin/doc?formal/02-06-69 – Consultado em: 12/04/2005*.

Jain, A.; Murty, M.; Flynn, P. (1999). Data clustering: A review. *ACM Computing Surveys*, v.31, n.3, p.264–323.

Maarek, Y. S.; Berry, D. M.; Kaiser, G. (1991). An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*, v.17, n.8, p.800–813.

Macedo, A. A. (2005). *Care: Um arcabouço de apoio à criação automática de relacionamentos entre repositórios homogêneos de informações.* Tese (Doutorado), Universidade de São Paulo, São Carlos.

Meling, R.; Montgomery, J.; Ponnusamy, S. P.; Wong, E. B.; Mehandjiska, D. (2000). Storing and retrieving software components: A component description manager. *Proceedings of the 2000 Australian Software Engineering Conference*, p. 107–118. IEEE Computer Society.

Mello, R. F.; Senger, L. J.; Yang, L. T. (2004). Automatic classification using artificial neural network. *Workshop on High Performance Computacional Science Engineering – HPCSE04*, v.29, n.7, p.649–664.

MicroSystems, S. (1997). Javabeans - specification. *URL: http://java.sun.com/products/javabeans – Consultado em: 05/04/2005*.

MicroSystems, S. (1999). Enterprise javabeans technology. *URL: http://java.sun.com/products/ejb – Consultado em: 05/04/2005*.

Microsystems, S. (2000a). How to write comments for the javadoc tool. *URL: http://java.sun.com/j2se/javadoc/writingdoccomments/index.html – Consultado em: 25/07/2003*.

Microsystems, S. (2000b). Reference api specifications. *URL: http://java.sun.com/reference/api/index.html – Consultado em: 25/07/2003*.

MicroSystems, S. (2005). Naming conventions. *URL: http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html – Consultado em: 21/06/2005*.

Mili, R.; Mili, A.; Mittermeir, R. T. (1997). Storing and retrieving software components: A refinement based system. *IEEE Trans. Softw. Eng.*, v.23, n.7, p.445–460.

Nardi, R. A. (2003). Componentes corba. Dissertação (Mestrado), Universidade de São Paulo, São Paulo.

Porter, M. (1980). An algorithm for suffix stripping. *Program 14*, v.3, p.130–137.

Pressman, R. S. (2002). *Engenharia de Software*. McGraw-Hill.

Prieto-Diáz, R. (1991). Implementing faceted classification for software reuse. *Commun. ACM*, v.34, n.5, p.88–97.

Salton, G.; Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, v.24, n.5, p.513–523.

Salton, G.; Fox, A.; Harry, W. (1983). Extended booleano information retrieval. *Communications pf the ACM*, v.26, n.12, p.1022–1036.

Salton, G.; McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill.

Schitze, H.; Silverstein, C. (1997). Projection for efficient document clustering. *Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, p. 60–66.

Seacord, R. C.; Hissam, S. A.; Wallnau, K. C. (1998). Agora: A search engine for software components. Relatório Técnico CMU/SEI-98-TR-011, Software Engineering Institute.

Senger, L. J. (2005). *Escalonamento de processos: uma abrodagem dinâmica e incremental para a exploração de características de aplicações paralelas*. Tese (Doutorado), Universidade de São Paulo, São Carlos.

Steinbach, M.; Karypis, G.; Kumar, V. (2000). A comparison of document clustering techniques. *In Notes from KDD Workshop on Text Mining, held at the Sixt Conference on Knowledge Discovery and Data Mining (KDD'00)*.

Sugumaran, V.; Storey, V. C. (2003). A semantic-based approach to component retrieval. *SIGMIS Database*, v.34, n.3, p.8–24.

University, C. S. L. P. (2004). Wordnet - a lexical database for the english language. *URL: http://wordnet.princeton.edu – Consultado em: 07/07/2005*.

Vieira, M. T. P. (2002). Projeto DBCM: Desenvolvimento de software baseado em componentes multimídia. *Programa de Apoio à Pesquisa e Desenvolvimento e Inovação em Tecnologia da Informação - PDI-TI - Programa CT-INFO - CNPq*, Processo nro. 552080/2002-0.

Vieira, M. T. P. (2003). Projeto DBCM: Desenvolvimento de software baseado em componentes multimídia. *Programa de Apoio à Pesquisa e Desenvolvimento e Inovação em Tecnologia da Informação - PDI-TI - Programa CT-INFO - CNPq*, Processo nro. 552080/2002-0. Relatório Técnico I.

Vieira, M. T. P.; Santos, F. G.; Silva, E. A.; Prado, A. F. (2005). Reuse of multimedia components in the development of distance learning applications. *In V IEEE International Conference on Advanced Learning Technologies*, Kaohsiung - Taiwan.

Vitharana, P.; Zahedi, F. M.; Jain, H. (2003). Knowledge-based repository scheme for storing and retrieving business components: A theoretical design and an empirical analysis. *IEEE Trans. Softw. Eng.*, v.29, n.7, p.649–664.

Werner, C.; Braga, R. M. M. (2000). Desenvolvimento baseado em componentes. *XIV Simpósio Brasileiro de Engenharia de Software - Minicursos/Tutoriais*, v. 14, p. 297–329, João Pessoa,Paraíba.

Yaguinuma, C. A.; Santos, M. T. P.; Biajiz, M.; Prado, A. F. (2005a). Repositório de componentes auxiliando a identificação e a reutilização de padrões de software. *V Workshop de Desenvolvimento Baseado em Componentes*, Juiz de Fora - MG.

Yaguinuma, C. A.; Santos, M. T. P.; Vieira, M. T. P. (2005b). Ontology-based meta-model for storage and retrieval of software components. *VLDB Workshop on Ontologies-based techniques for DataBases and Information Systems – ODBIS 2005*, Norway.

Zaremski, A. M.; Wing, J. M. (1995). Signature matching: a tool for using software libraries. *ACM Trans. Softw. Eng. Methodol.*, v.4, n.2, p.146–170.

Zaremski, A. M.; Wing, J. M. (1997). Specification matching of software components. *ACM Trans. Softw. Eng. Methodol.*, v.6, n.4, p.333–369.