Classificação semi-automática de componentes java

Claudia de Oliveira Melo

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Área de Concentração: Ciência da Computação
Orientador: Profa. Dra. Ana Cristina Vieira de Melo

São Paulo, maio de 2007

Classificação semi-automática
de componentes java

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Claudia de Oliveira Melo
e aprovada pela Comissão Julgadora.

Banca Examinadora:

- Profa. Dra. Ana Cristina Vieira de Melo (Orientadora) - IME-USP.
- Prof. Dr. Jorge Luis Risco Becerra - Poli-USP.
- Profa. Dra. Cecília Mary Fischer Rubira - Unicamp.

aos meus pais Jesus e Luzia *(in memorian)*
e à minha irmã Luciana

Agradecimentos

Agradecer no fim de um trabalho de tantos anos é uma tarefa complicada. Preciso voltar ao início de tudo, às primeiras pessoas que me apoiaram. Ao grande amigo Otávio, que sempre me incentivou na mudança para São Paulo, devo um muito obrigada! Ao meu pai Jesus e minha irmã Luciana que sempre estiveram ao meu lado, mesmo de longe. Ao meu amigo Marcos, companheiro de muitas batalhas no início da carreira e de vida. Ao professor Fabio Kon, que me deu a oportunidade de cursar a primeira matéria no IME-USP. Ao primeiro amigo de mestrado, Rogério Seiji, que tanto me ensinou nas várias madrugadas de estudo e trabalho. À minha orientadora Ana Cristina Vieira de Melo, que acreditou no meu potencial, me incentivou e, principalmente, soube compreender os percalços do meu trajeto. Às empresas Open Communications Security e Diebold Procomp, que possibilitaram grande crescimento profissional e intelectual ao apoiar meu mestrado. Dos grandes amigos que nelas fiz, preciso agradecer especialmente ao Thiago Martins, Paulo Oliveira, Maria Fernanda Vaz e José Rubens Curtt.

Aos tantos amigos que fiz, meu muito obrigada. Vocês estiveram comigo sempre, abrindo portas, trazendo alegria, aprendizado, descobertas ou conforto. Agradeço à Ana Lúcia, Karen Sandhof e ao Elói pela primeira oportunidade de lecionar no Ensino Superior. Aos amigos que me receberam André Sanches, Eduardo Takeo, Marcos Couto e Carlos Queiroz. Aos amigos que recebi ;) Christian Paz, Giulian Luz, Braga Jr., Celina Takemura, Raphael Camargo, Rosianni Cruz, Leandro César e todos os AABalados (principalmente os do LCPD). Ao amigo de pesquisa Andre "Oberon". Ao Clodis pela grande e verdadeira amizade. Ao Vidal pela compreensão e pelos conselhos. Ao Marcelo Palin, pelo exemplo de caráter e superação constante.

A todos os funcionários do IME, em especial ao Pinho. Agradeço a todos aqueles que nunca acreditaram em mim (roubando a frase do André Sanches). Aos meus amigos de Brasília, que tanto me apoiaram nesta fase final. Ao meu namorado Rodrigo, por todo o amor.

"O som da viola bateu no meu peito doeu meu irmão
Assim eu me fiz cantador sem nenhum professor, aprendi a lição
São coisas divinas do mundo que vem num segundo a sorte mudar
Trazendo pra dentro da gente as coisas que a mente vai longe buscar
Trazendo pra dentro da gente as coisas que a mente vai longe buscar
Em versos se fala e canta o mal se espanta e a gente é feliz
No mundo da rima e provas eu sempre dei provas das coisas que fiz
Por muitos lugares passei, mas nunca pisei em falso no chão
Cantando interpreto a poesia levando alegria onde há solidão
Cantando interpreto a poesia levando alegria onde há solidão
O destino é o meu calendário o meu dicionário é a inspiração
A porta do mundo é aberta minha alma desperta buscando a canção
Com minha viola no peito meus versos são feitos pro mundo cantar
É a luta de um velho talento menino por dentro sem nunca cansar
É a luta de um velho talento menino por dentro sem nunca cansar"

Resumo

As recentes tecnologias de desenvolvimento e distribuição de componentes possibilitaram o aumento do número de componentes disponíveis no mercado. No entanto, eles muitas vezes estão dispersos e não publicados adequadamente para a comunidade de pesquisa e desenvolvimento de software. Encontrar componentes apropriados para solucionar um problema particular não é uma tarefa simples e novas técnicas devem ser desenvolvidas para o reuso efetivo de componentes. Um dos maiores desafios em reusar componentes consiste em classificá-los corretamente para futuras consultas. Classificar componentes para possibilitar uma busca eficaz depende da qualidade das informações adquiridas, que viabilizam melhor precisão e cobertura das consultas ao encontrar componentes reutilizáveis em potencial. Ao mesmo tempo, mecanismos de classificação e busca devem ser fáceis o suficiente para convencer os desenvolvedores a reusar componentes.

Este trabalho estuda as técnicas de classificação de componentes de software, repositórios e métodos de busca. É apresentada uma proposta de modelo de classificação de componentes que considera não apenas sua função, mas o negócio onde ele está inserido e seus atributos de qualidade. Um método de preenchimento semi-automático das informações é proposto, de modo a diminuir os custos de classificação. O protótipo REUSE+ foi construído para exemplificar o uso do modelo e do método de classificação semi-automática, de forma a validar a proposta, destacando, por fim, as principais contribuições do trabalho.

Palavras-chave: classificação semi-automática, repositório, componentes.

Abstract

The recent developments on components technologies have increased the number of components available to the market. These components are, however, distributed overall the world and not properly advertised to the research and development communities. Finding the appropriate components to solve a particular problem is not very straightforward and new techniques must be developed to effectively reuse components. One of the great challenges in reusing components is concerned with how to actually classify components "properly" in order to further retrieve them. Classifying components for effective retrieval depends on acquiring the appropriate information in classification to improve the precision and recall rates in retrieval; finding only the potentially reusable components and not missing potential solutions. At the same time, the classification and retrieval mechanisms must be easy enough to persuade developers to reuse components. This work studies the classification techniques of software components, repository and retrieval methods. Hereafter is presented a proposal of components classification model that considers not just its function, but business and quality attributes. It is proposed a semi-automatic classification mechanism of software information, allowing a cheaper classification. REUSE+ prototype was built to exemplify the use of model and method of semi-automatic classification, allowing the described proposal validation, highlighting at the end the mainly contributions of the work.

Keywords: semi-automatic classification, repository, components.

Sumário

Lista de Figuras **xiii**

Lista de Tabelas **xv**

1 Introdução **1**

1.1 Motivação 1

1.2 Objetivos 2

1.3 Organização da dissertação 3

2 Reutilização de Software **5**

2.1 Definições 5

2.2 Histórico 6

2.3 Principais Objetivos 8

2.4 Abordagens de Reutilização de Software 10

2.5 Técnicas para Reuso 11

2.5.1 Bibliotecas de Classes 11

2.5.2 Padrões de Software 11

2.5.3 *Frameworks* 12

2.5.4 Componentes de Software 12

2.5.5 Componentes de Software e Orientação a Objetos 13
2.6 Desenvolvimento dirigido por reuso 14
2.7 Principais Barreiras à Reutilização 18
2.8 Suporte à Reutilização 19
2.9 Considerações Finais 20

3 Classificação e busca de componentes de software 21
3.1 Palavras-chave em texto livre 22
3.1.1 Processamento de Linguagem Natural (PLN) 24
3.1.2 Reuso com bibliotecas de documentos de propósito geral 24
3.1.3 Reuso com bibliotecas especialistas 25
3.1.4 Reuso com acesso guiado por menu 26
3.1.5 Outros tipos de reuso 26
3.2 Facetas 27
3.2.1 Facetas para Componentes de Negócio 28
3.3 Método Enumerativo 30
3.3.1 Relacionamentos de artefatos em redes semânticas 32
3.3.2 Taxonomia de componentes em categorias 33
3.4 Recuperação de Componentes de Software 33
3.4.1 Recuperação de Informação 34
3.5 Análise dos Métodos 40
3.6 Considerações finais 42

4 Modelo Proposto de Classificação 43
4.1 Visão Geral 43

4.2 Um Novo Modelo de Classificação de Componentes 44
4.2.1 Atributos e Facetas 45
4.3 Método para preenchimento semi-automático de atributos e facetas 49
4.3.1 Armazenamento das informações 53
4.4 Considerações finais 53

5 O Sistema REUSE+ 55

5.1 Visão Geral 55
5.2 Mecanismos de classificação 58
5.2.1 Análise Estática 59
5.2.2 *Parser* de Comentários 60
5.2.3 Remoção de *Stopwords*, *Stemming* e Indexação 62
5.3 Mecanismos de Armazenamento 63
5.4 Mecanismos de Busca e Recuperação 65
5.5 Arquitetura do sistema 65
5.5.1 Projeto Arquitetônico 66
5.6 Exemplo de uso - Componente Cinema 68
5.6.1 Perfil Administrador 69
5.6.2 Perfil Classificador 70
5.6.3 Perfil Usuário Final 70
5.7 Exemplo de uso 72
5.7.1 A coleção de testes 72
5.7.2 Tipos de Avaliação 74
5.7.3 Resultados 77
5.8 Considerações finais 80

6 Conclusões **81**

6.1 Considerações Gerais 81

6.2 Contribuições e Limitações do Trabalho 82

6.3 Trabalhos futuros 83

A Questionário de usabilidade **85**

A.1 Questionário de legibilidade 85

A.2 Questionário de Experiência do Usuário 88

A.3 Questionário de Consistência do REUSE+ 89

A.4 Questionário de Densidade informacional do REUSE+ 91

A.5 Questionário de Feedback do REUSE+ 92

Referências Bibliográficas **95**

Lista de Figuras

2.1 Desenvolvimento COM e PARA reuso de software 15

2.2 IDEF0 - Nível 0 - do processo de produção de entidades para reutilização (Adaptado de [50]). 17

3.1 Relação entre a hierarquia e o nível de abstração de um componente (adaptada de Vitharana et al [91]).

3.2 Processo típico de classificação e recuperação de informação. Adaptado de Baeza-Yates [3] 35

3.3 Representação dos vetores de consulta Q e dois documentos D1 e D2 39

4.1 Exemplo de hierarquia do Componente Cinema. 50

4.2 Abstração do componente *versus* Preenchimento automático de facetas. 51

4.3 Facetas e atributos que podem ser preenchidos automaticamente por origem 52

5.1 Funcionamento do REUSE+ 56

5.2 Processo de classificação automática. 58

5.3 Parse do Código-Fonte: extração de informações para as Facetas *Function*, *Element/Part* e *Action* 62

5.4 Visão lógica do sistema REUSE+ 64

5.5 Arquitetura MVC do sistema REUSE+ 68

5.6 Ambientes do administrador do REUSE+ 69

5.7 Ambientes do classificador de componentes no REUSE+ 71

5.8 Ambientes do usuário final no REUSE+ 72

5.9 Informações detalhadas sobre o componente 73
5.10 Termo Cinema Expandido 75
5.11 Termo Ticket Expandido 75
5.12 Curva de Precisão x Cobertura - Consultas **SE** 78
5.13 Curva de Precisão x Cobertura - Consultas **CE** *versus* **SE** 79
5.14 Resultados do Estudo de Usabilidade 80

Lista de Tabelas

3.1 Atributos do Modelo de Classificação de Vitharana *et al* para Componentes de Negócio 30

3.2 Facetas do Modelo de Classificação de Vitharana *et al* para Componentes de Negócio . 30

3.3 Exemplo de preenchimento dos atributos do Componente Cinema 31

3.4 Exemplo de preenchimento das facetas do Componente Cinema 31

4.1 Exemplo de preenchimento dos Atributos do componente Cinema seguindo o modelo proposto 48

4.2 Exemplo de preenchimento das Facetas do componente Cinema seguindo o modelo proposto 48

Capítulo 1

Introdução

A produtividade de software aumentou nas últimas décadas, mas não o suficiente para acompanhar a exigente demanda da indústria. Esse fato, comumente chamado de "crise de software" [60], continua a existir, mesmo após muitos estudos na área de Engenharia de Software e áreas correlatas - como Inteligência Artificial.

A reutilização apareceu como uma das poucas abordagens reais para tratar o problema [56], aumentando a produtividade (menor esforço na construção) e a qualidade (menor esforço de manutenção e satisfação do usuário) que a indústria necessitava. Vários avanços foram conseguidos em bibliotecas de software, técnicas de classificação, criação e distribuição de componentes reutilizáveis, ambientes de apoio à reutilização, entre outros. Mesmo assim, o grau de reutilização manteve-se muito aquém das expectativas.

A classificação de software é uma das etapas mais importantes da construção de uma biblioteca, sendo determinante no sucesso de futuras buscas. Pesquisas nessa área têm sido feitas há mais de 2 décadas e, embora muitas soluções sofisticadas tenham sido propostas, a prática da reutilização ainda é caracterizada pelo uso de métodos ad hoc, de baixa tecnologia, como as classificações manuais. Por outro lado, a aplicação de soluções de alta tecnologia (automatizadas), como as de métodos formais, ainda são pouco adotadas devido ao custo proibitivo [70].

1.1 Motivação

A reutilização de software é um processo relativamente caro - tempo e custos adicionais são gerados - e sua adoção necessita de justificativas claras que compensem tal esforço. Um dos maiores problemas em reusar software já existente é localizar aquele que execute as funções desejadas pelo

reutilizador. A situação ideal seria encontrar um componente que implementasse exatamente os requisitos de software desejados. Um mecanismo de classificação ou de busca ruins diminuem a probabilidade de reuso, pois o desenvolvedor terá dificuldades em fazer pesquisas e quando encontrar algum componente, a curva de aprendizado necessária será mais alta - e pode ainda revelar que o componente não é de fato adequado.

A comunidade de pesquisa em Engenharia de Software trabalhou fortemente durante os anos 80 e 90 em busca de novas idéias para apoiar o reuso de software. Muitas soluções foram propostas, mas nenhuma delas oferece a combinação certa de eficiência, eficácia, usabilidade e generalidade que aponte uma real otimização em Reutilização de Software [58]. Depois de alguns anos de intervalo, voltou-se a pesquisar ferramentas e técnicas de auxílio ao reuso, considerando agora novas tecnologias e arquiteturas.

Analisando os métodos já propostos para classificação e recuperação de componentes, é possível identificar oportunidades de melhoria tanto no tempo gasto para classificar software, quanto na qualidade dos resultados recuperados pelo usuário em suas buscas ao repositório. Tais oportunidades são reflexo da evolução do software, cuja complexidade é crescente, prejudicando modelos de classificação clássicos que não foram criados nesse contexto. Exemplo disso são os componentes de negócio, desenvolvidos com a intenção de lidar apenas com as regras de negócio de certa empresa ou ramo. Eles tratam um nível de abstração mais complexo para a computação, ligado ao domínio da aplicação e às regras e papéis que exercem nesse domínio.

Além disso, questões como a larga acessibilidade dos repositórios de componentes e a qualidade do software nele armazenado ganharam força, visto que influenciam diretamente a probabilidade de reuso. É necessário, então, incorporar esses requisitos no desenvolvimento de uma solução de apoio à reutilização.

1.2 Objetivos

Levando em consideração as motivações citadas anteriormente, os principais objetivos desse trabalho são:

1. Propor um novo modelo de classificação mais abrangente e coerente com as demandas existentes no mercado;

2. Desenvolver um método que permita automatizar parte da classificação proposta;

3. Criar um protótipo que permita classificar componentes utilizando o modelo e métodos propostos e também processar consultas de usuário interessados em reutilizar software;

4. Avaliar a viabilidade do modelo e método propostos para classificação e recuperação de componentes por meio de métricas conhecidas na área de Recuperação de Informação (RI)- precisão (*precision*) e cobertura (*recall*). Adicionalmente, avaliar a usabilidade da ferramenta por meio de *checklists*.

1.3 Organização da dissertação

Este trabalho está dividido em 5 capítulos descritos brevemente a seguir.

Capítulo 2: Reutilização de software
Este capítulo introduz os principais conceitos sobre Reutilização de Software, seu histórico e objetivos. São mostradas as abordagens de reuso existentes e quais as principais barreiras que impedem a reutilização. Por último, são resumidas as principais maneiras de dar suporte ao reuso.

Capítulo 3: Métodos de classificação e busca de componentes de software
Este capítulo aprofunda a visão iniciada do capítulo anterior sobre suporte ao reuso. Nele são descritos os principais métodos de classificação e recuperação de componentes de software, além de possíveis métodos de avaliação dos mesmos.

Capítulo 4: Modelo de Classificação
Este capítulo apresenta um novo modelo de classificação baseado nas constatações de limitação de propostas vistas no capítulo anterior, fazendo as extensões apropriadas. Também é proposto um método semi-automático para a realização de classificações que usarem o modelo.

Capítulo 5: O Sistema REUSE+
Este capítulo apresenta o protótipo REUSE+, construído para possibilitar a classificação e recuperação de componentes usando o modelo e métodos propostos. Com ele é possível comprovar a viabilidade do que este trabalho propõe, por meio de métricas de Recuperação de Informação e de Usabilidade da ferramenta.

Capítulo 6: Conclusão
Este capítulo conclui o trabalho e lista as principais contribuições desta dissertação. Além disso, suas limitações são discutidas e, por fim, listadas possibilidades de trabalho futuro.

Capítulo 2

Reutilização de Software

> "Façamos como os engenheiros de hardware! Não está certo começar cada novo desenvolvimento sempre do zero. Devia haver catálogos de software, tal como há catálogos de circuitos integrados: para construir um novo sistema, começaremos encomendando esses componentes e depois combinaremo-os, em vez de reinventar a roda. Dessa maneira, escreveríamos menos software, mas talvez escrevêssemos melhor. Não desapareciam então os problemas de que todos se queixam - custos elevados, atrasos e falta de viabilidade? Por que não é assim?" Mass Produced Software Component, Doug McIlroy, 1968

Este capítulo tem como objetivo apresentar os principais conceitos relacionados ao reuso de software para que seja possível compreender sua motivação, objetivos, métodos, ferramentas e desafios.

2.1 Definições

Para evitar problemas com as diversas, e semelhantes, nomenclaturas acerca de reutilização de software é interessante definir alguns termos:

1. *Software:* seqüência de instruções a serem seguidas e/ou executadas, na manipulação, redirecionamento ou modificação de um dado/informação ou acontecimento. Tecnicamente, software também é o nome dado ao conjunto de produtos desenvolvidos durante o Processo de Soft-

ware, o que inclui não só o programa de computador propriamente dito, mas também manuais, especificações, planos de teste etc.

2. *Reuso ou Reutilização:* uso de conceitos ou produtos previamente adquiridos ou construídos em uma nova situação. Isso envolve a representação desses produtos em vários níveis de abstração, o armazenamento dos mesmos para futuras referências, a identificação de similaridades entre situações novas e antigas, a recuperação de produtos já desenvolvidos (ou parte deles) e sua adaptação à nova situação [64].

3. *Reusabilidade:* medida da facilidade/potencialidade em se utilizar os conceitos e produtos existentes em novas situações. Em termos de projeto e codificação de software, a idéia é analisar se existe alta coesão e baixo acoplamento entre os módulos do sistema. Existem várias métricas para calcular a reusabilidade [33], mas não é escopo deste trabalho aplicá-las diretamente.

4. *Componente:* Componentes reutilizáveis são artefatos autocontidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras em conformidade com um dado modelo de arquitetura de software, documentação apropriada e um grau de reutilização definido [77]. Na Seção 2.5.4 esse assunto é explorado com mais detalhes.

5. *Processo de software:* Um processo de software é formado por um conjunto de passos de processo parcialmente ordenados, relacionados com conjuntos de artefatos, pessoas, recursos, estruturas organizacionais e restrições e tem como objetivo produzir e manter os produtos de software finais requeridos [49].

6. *Produtividade:* no contexto desta pesquisa, é a relação entre o esforço (em termos de custo, tempo gasto e trabalho executado) para se produzir algo, e o resultado obtido com esse esforço. Quanto menor é o esforço e maior o resultado, maior é a produtividade.

7. *Qualidade:* conjunto de atributos que têm impacto na capacidade do software de manter o seu nível de desempenho dentro de condições estabelecidas por um dado período de tempo [45].

2.2 Histórico

Do ponto de vista histórico, o desenvolvimento de software pode ser dividido em três períodos [87]. O primeiro deles decorreu até 1968 e pode ser classificado como pré-histórico. O segundo surge com a crise de software e o nascimento da Engenharia de Software como ciência. E o terceiro período

começa em meados de 1983 com o aparecimento dos computadores pessoais e o domínio do software sobre o hardware.

Durante o primeiro período, o principal objetivo do desenvolvimento de software consistia no aproveitamento dos limitados recursos de hardware. Grande parte do desenvolvimento era voltado para a otimização de código, escrito principalmente em linguagem de máquina. Os primeiros compiladores provaram que, para a maioria das aplicações, era possível gerar código com bom desempenho a partir de linguagens de alto nível. Surgiu então a compilação incremental, a alteração do código-fonte e a depuração de programas. Verificou-se que era necessário fazer manutenção dos programas e que o desenvolvimento de software exigia a atenção dos projetistas em outros domínios além da codificação.

O segundo período iniciou-se com a preocupação com os insucessos nos projetos de software e as suas causas. A compreensão de que a qualidade não pode ser assegurada exclusivamente por uma análise do produto final deu origem ao primeiro modelo de processo de desenvolvimento: cascata. A qualidade do software deve compreender todo o processo de desenvolvimento, em especial as fases iniciais.

Os métodos formais foram uma das formas utilizadas para aumentar a confiabilidade do software e aumentar a produtividade. Obteve-se sucesso na especificação e verificação de sistemas críticos, de pequenas dimensões. Para projetos maiores, verificou-se que nem os requisitos, nem o projeto podiam ser expressos usando um modelo matemático sucinto, como desempenho, compatibilidade, relacionamentos com dados exteriores e outras formas de requisitos não-funcionais - restrições e qualidades do software [52]. Houve ainda a dificuldade de comunicação com leigos devido à linguagem e notação utilizados.

O terceiro período começou com o aparecimento das estações de trabalho e terminais gráficos. O subseqüente aparecimento das interfaces e aplicações gráficas permitiu o surgimento de ferramentas CASE (*Computer Aided Software Engineering*), que auxiliam o processo de desenvolvimento de software. No entanto, a tecnologia CASE teve sucesso apenas em uma porção limitada de aplicações [78]. A necessidade de gerar e modificar código obrigava a propagar tais alterações para o projeto. Por outro lado, a separação entre a modelagem dos dados e dos processos dificultava a manutenção e a descrição de suas inter-relações.

No fim do primeiro período, McIlroy [54] propôs a criação de uma indústria de componentes normalizados com os quais pretendia construir aplicações complexas por meio de pequenos blo-

cos disponíveis em catálogo. Esperava-se que os programadores reutilizassem tudo o que estivesse disponível, desde pedaços de código a experiências anteriores, mas a reutilização não fazia parte explicitamente do processo de desenvolvimento. Com a criação de aplicações de maiores dimensões, a partir do início da década de 80, a reutilização de software sofreu um impulso decisivo.

Surgiu então o paradigma de orientação a objetos (OO), onde o sistema deveria ser escrito baseado em um modelo mental dos objetos reais ou imaginários que o representam, otimizando a compreensão do problema e a comunicação entre analistas, projetistas e programadores. Além disso, os principais elementos de OO, como o encapsulamento, herança e polimorfismo, resultaram de uma preocupação direta com a reutilização e manutenção de software. Na programação, a vantagem do uso de objetos é que eles permitem utilizar funções e atributos conhecidos (interface) sem necessidade de conhecimento sobre sua estrutura interna (encapsulamento). Para ilustrar: utilizamos o objeto 'liquidificador' pressionando seus botões, mas não precisamos saber como funciona seu 'motor'. Se for necessário trocar o 'motor', não será necessário alterar o modo de usá-lo [93]. Se um objeto é uma unidade auto-gerenciável, ou seja, capaz de gerir sua complexidade interna e ainda oferecer os serviços esperados, então o reuso desse objeto torna-se mais fácil.

Objetivando o aumento de produtividade e qualidade do software, o estudo sobre processos de software foi intensificado. A idéia era tornar o reuso parte do processo de desenvolvimento de software. O recente interesse em caracterizar o reuso usando modelos de maturidade e processos de adoção são um claro sinal de progresso [67]. Nesse contexto, há uma necessidade contínua por novos processos de software que sejam capazes de minimizar os custos e melhorar os resultados do desenvolvimento de sistemas computacionais.

2.3 Principais Objetivos

Reutilização de software é uma idéia antiga (formalmente proposta em 1969) e consiste em utilizar software existente para construir novos sistemas [11]. Apesar de inicialmente voltado para código-fonte, o reuso é aplicável também a todo trabalho gerado durante o processo de desenvolvimento de software, como dados, arquitetura e projeto. Portanto, a informação suscetível à reutilização inclui desde a análise de requisitos, especificações do sistema, estruturas de projeto, até qualquer informação necessária ao processo de desenvolvimento. Esses produtos do desenvolvimento são chamados *artefatos de software* [16].

Os principais objetivos da reutilização são claros e bem definidos, além de aceitos pela maioria dos autores [6, 77]. Dentre eles, podem ser destacados:

1. *Aumento da Produtividade:* reutilizando-se artefatos, o esforço no desenvolvimento (tempo de análise, projeto, codificação e testes) diminui.

2. *Aumento da Qualidade:* correções de erro acumulam-se de reuso em reuso. Isso significa maior qualidade em um componente reusado do que naquele construído e utilizado somente uma vez. Entretanto, isso requer a administração e manutenção de componentes, que não são atingidos com a simples reutilização de software.

3. *Redução dos custos:* o custo de construção de artefatos reutilizáveis é diluído entre os vários projetos que o reutilizarão. A longo prazo existe uma sensível redução de custos nos projetos de desenvolvimento, pois grande parte deles podem ser construídos a partir de software pronto, com esforço mínimo ou reduzido.

4. *Redução no tempo de entrega:* com a diminuição de trabalho redundante, o tempo de entrega de um software também diminui.

5. *Padronização:* a busca por padronização objetiva diminuir a curva de aprendizado de um software, o que aumenta a produtividade e facilita a manutenção. Com o reuso, as boas práticas de desenvolvimento tornam-se padrões, que por sua vez são incorporadas continuamente aos módulos reusáveis. Construir sistemas baseados nesses módulos padronizados aumenta, em consequência, a padronização existente no sistema.

6. *Interoperabilidade:* vários sistemas podem trabalhar melhor em conjunto se suas interfaces[1] forem implementadas consistentemente. O reuso obriga o desenvolvimento de interfaces padronizadas, para que seja possível manter o baixo acoplamento e a alta coesão entre os módulos do sistema. Sendo assim, módulos reusáveis tem maior chance de serem interoperáveis.

7. *Previsibilidade/Confiabilidade/Redução de Riscos:* o uso de componentes bem testados aumenta a confiabilidade de um sistema. Além disso, o uso de componentes em muitos sistemas aumenta a chance de identificação de erros e fragilidades, o que viabiliza correções e melhoria da qualidade do componente.

[1] Em Ciência da Computação, uma interface é um 'contrato' que determina a forma de comunicação entre componentes de software.

2.4 Abordagens de Reutilização de Software

Considerando apenas o processo de reuso dos artefatos de software, podemos identificar duas abordagens principais de reutilização [7, 72]: generativa e composicional.

Reutilização Generativa

A reutilização generativa baseia-se no reuso do nível de especificação do sistema. A partir de uma descrição/especificação de alto nível, como uma especificação formal[2], o programa é gerado. Em geral, ferramentas de apoio ao reuso generativo trabalham apenas com um domínio de aplicação e o processo de reutilização é transparente ao usuário, pois na geração do código se dá o reuso. O domínio do problema representa um conjunto de itens de informação presentes em um certo contexto do mundo real, inter-relacionados de forma bastante coesa, e que desperta o interesse de uma certa comunidade [61], como a área Médica ou Financeira.

Reutilização Composicional

A reutilização composicional é o uso de software existente como blocos de construção para o desenvolvimento de novos sistemas de software. A tecnologia de reuso composicional é baseada em coleções bem estabelecidas, sistemas de repositório e interfaces padronizadas. Embora o reuso de artefatos em geral seja pregado, na prática o reuso composicional ocorre principalmente em código-fonte. Temas de pesquisa relacionados a esse tipo de reuso incluem a busca e seleção de módulos (componentes) reusáveis, entendimento do módulo selecionado, a adaptação do módulo ao contexto do problema e, finalmente, sua integração ao sistema. Bibliotecas de componentes ou repositórios preocupam-se essencialmente com as duas primeiras atividades: busca e seleção de componentes. As demais são atividades de análise de domínio ou de interface. Ferramentas que se propõe a auxiliar o processo de reutilização composicional serão apresentadas no Capítulo 3.

A reutilização composicional possibilita um tipo de reuso mais geral e menos acoplado ao domínio da aplicação. Optando por essa linha de reutilização, existem alguns requisitos mínimos que o software deve satisfazer: (i) ser bem definido; (ii) ter interface simples e clara e (iii)ser bem documentado. Todos eles refletem diretamente na curva de aprendizado do reutilizador, que provavelmente está interagindo com um repositório, buscando e selecionando módulos reusáveis e tentar adaptá-los

[2]Por Métodos Formais entende-se o uso de notações e métodos matemáticos na especificação de sistemas e propriedades de maneira precisa e sem ambigüidades.

e integrá-los ao seu novo sistema. Essa avaliação e incorporação deve ter o menor custo possível para que não onere o projeto, ou seja, para que de fato valha à pena reusar software existente.

Esses requisitos mínimos para um software reusável fazem parte da definição de componentes de software, que já preocupa-se com a possibilidade de reutilização. A próxima seção apresentará algumas técnicas para reuso de software, dentre elas o uso de componentes de software, que cada vez mais tornam-se populares na literatura e nas organizações.

2.5 Técnicas para Reuso

Existem diversas técnicas para reuso de software, em geral baseadas no tipo de produto que se deseja reusar. As seções seguintes descrevem algumas técnicas por tipo de produto de software.

2.5.1 Bibliotecas de Classes

As bibliotecas de classes são uma solução para reuso de software orientado a objetos. Elas permitem uma capacidade muito maior de compartilhamento e reutilização de código, pois é possível criar-se subclasses para atender novas necessidades, em função das classes já existentes. Muitas bilbiotecas são oferecidas juntamente com as ferramentas de desenvolvimento para reduzir o tempo e a complexidade de projetos de software, como a Microsoft Foundation Class (MFC) e a Visual Component Library (VCL) do Delphi.

2.5.2 Padrões de Software

Um padrão descreve uma solução para um problema que ocorre com freqüência durante o desenvolvimento de software, podendo ser considerado como um par *¡problema,solução¿* [30]. Projetistas familiarizados com certos padrões podem aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los [36]. Padrões de software podem se referir a diferentes níveis de abstração no desenvolvimento de sistemas orientados a objetos. Assim, existem padrões arquiteturais, em que o nível de abstração é bastante alto, padrões de análise, padrões de projeto, padrões de código, entre outros [8]. O uso de padrões proporciona um vocabulário comum para a comunicação entre projetistas, além de atuar como blocos construtivos a partir dos quais projetos mais complexos podem ser construídos [36].

2.5.3 *Frameworks*

Segundo [8], após a popularização das primeiras linguagens orientadas a objetos - como Smalltalk - paralelamente às bibliotecas de classes, começaram a ser construídos *frameworks* de aplicação, que acrescentam às bibliotecas de classes os relacionamentos e interação entre as diversas classes que o compõem. Com os *frameworks*, reutilizam-se não somente as linhas de código, como também o projeto abstrato envolvendo o domínio de aplicação. Essas classes podem fazer parte de uma biblioteca de classes ou podem ser específicas da aplicação. *Frameworks* possibilitam reutilizar não só componentes isolados, como também toda a arquitetura de um domínio específico.

2.5.4 Componentes de Software

A reutilização de componentes é a reutilização de um software específico, o qual denominamos componente. Dentre as várias definições existentes na literatura, a mais abrangente foi dada por Sametinger pp.68 [77]:

"Componentes reutilizáveis são artefatos autocontidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras em conformidade com um dado modelo de arquitetura de software, documentação apropriada e um grau de reutilização definido."

Sametinger também enfatiza alguns elementos necessários para que um artefato de software seja considerado componente:

1. *Autocontido:* diz respeito à capacidade de um componente ser reutilizável por si só, ou seja, não dependa de outros componentes para ser reusado. As dependências entre componentes, se existirem, devem ser vistas como um único componente reutilizável;

2. *Identificação:* componentes devem ser claramente identificáveis, ou seja, devem estar contidos em um único local ao invés de distribuídos e misturados com outros artefatos de software ou documentação;

3. *Funcionalidades:* diz respeito às funcionalidades de um componente, ou seja, componentes podem descrever funcionalidades e/ou executar funções. Assim, pode-se considerar toda a documentação do ciclo de vida do software como componente, embora ela não contenha codificação de funcionalidades;

4. *Interfaces:* componentes devem ter interfaces claras a fim de facilitar o reuso e a conexão com outros componentes, além de esconder detalhes que não são necessários ao reuso;

5. *Documentação:* uma documentação clara é indispensável para o reuso, pois a documentação irá indicar o tipo de componente e sua complexidade. A falta de uma documentação apropriada torna o componente menos útil para o reuso;

6. *Condição de reuso:* condições de reuso devem ser estabelecidas, contendo informações sobre quem é o proprietário do componente, quem mantém o componente, com quem se deve entrar em contato no caso de problema, e qual é o estado de qualidade do componente.

2.5.5 Componentes de Software e Orientação a Objetos

As definições de componente e orientação a objetos causam dúvidas e discussões, afinal possuem muitas similaridades. De fato existem distinções entre essas abordagens, mas também uma relação de complementaridade. O desenvolvimento baseado em componentes pode ser considerado uma evolução do desenvolvimento orientado a objetos. Porém, um componente não precisa necessariamente ser desenvolvido no paradigma OO, podendo ser criado utilizando outras abordagens, como o desenvolvimento estruturado. O que torna um software componente é a sua conformidade com a definição de componente, principalmente em relação às interfaces, arquitetura e funcionalidades [86].

A orientação a objetos traz vantagens para o reuso de software por meio dos conceitos de encapsulamento (classes e pacotes), herança e polimorfismo. Uma classe pode herdar a estrutura e comportamento de outra classe e, adicionalmente, ser estendida e modificada. Do ponto de vista do reuso, essa seria uma das principais vantagens, pois classes trabalham juntas sem ter o conhecimento da existência da outra, do mesmo modo que componentes conseguem grande independência entre si [77].

Componentes de software estendem os princípios da orientação a objetos, por exemplo, é possível especificar objetos por meio de interfaces que representam todas as capacidades de um componente de software. Assim, o desenvolvimento de software baseado em componentes diferencia-se de outras abordagens pela separação entre a especificação e a implementação de um componente de software, e também pela divisão da especificação em várias interfaces. Isso torna o componente independente, podendo ser alterado ou substituído, desde que este especifique uma interface comum [35].

As diferenças entre componentes e orientação a objetos estão concentradas principalmente quanto

à persistência e abstração. Por exemplo: um objeto tem estado, o qual pode ser um estado persistente; já componentes não tem persistência de estado, de modo que duas instalações do mesmo componente não teriam as mesmas propriedades [86]. Componentes geralmente têm granularidade maior do que objetos e classes. Porém, se uma classe e uma interface bem definida acerca do que ela implementa e do que ela requer formarem um pacote, esta classe pode ser considerada um componente [24].

Objetos são manipulados unicamente por linguagens orientadas a objetos, já os componentes podem ser desenvolvidos em qualquer linguagem, desde que satisfaçam suas interfaces. Classes representam abstrações lógicas, podem ser agrupadas formando um componente e suas operações podem ser acessadas diretamente. Já os componentes de software representam elementos físicos, os quais podem ocupar espaço num sistema de arquivo. São implementações físicas de um conjunto de elementos lógicos, como classes, suas operações são alcançáveis somente por meio de suas interfaces [10].

A tecnologia de desenvolvimento baseado em componentes têm realizado um forte trabalho de mercado [74], essencial para a sua adoção. Para Szyperski [86], uma tecnologia imperfeita com um mercado é sustentável, porém, uma tecnologia perfeita sem mercado desaparecerá. O principal apelo dos componentes está na facilidade de integração e substituição no sistema; independência; interface clara e bem-definida, levando ao oferecimento de um serviço bem delimitado e conhecido; facilidade de compra e venda (podem ser vendidos separadamente, de acordo com a necessidade do cliente). Além disso, a adoção do paradigma orientado a objetos nas organizações reforçou o conceito e a importância do desenvolvimento baseado em componentes.

Por esses motivos, este trabalho fará referência constante ao conceito de "componentes de software", com o objetivo de enfatizar a característica de reusabilidade e integrabilidade do software que deseja-se classificar e recuperar.

2.6 Desenvolvimento dirigido por reuso

A reutilização de conceitos, componentes e outros artefatos provenientes de um processo de desenvolvimento segue também um processo. Ele pode ser decomposto em dois subprocessos distintos [77]:

- O desenvolvimento de software *para* reutilização;
- O desenvolvimento de software *com* reutilização.

O desenvolvimento de software *para* reutilização não deve ser confundido com o desenvolvimento de software *com* reutilização. O primeiro pressupõe um ciclo de desenvolvimento de software e, principalmente, cuidados para que o âmbito de aplicação do resultado obtido possa ser aplicado em várias situações. Exemplos de paradigmas de desenvolvimento para reuso são a engenharia de domínio [65] e as linhas de produto de software [12]. O segundo limita-se a preparar software, que pode ou não ter sido desenvolvido com objetivos de reutilização, para que ele possa ser usado em certo projetos. A Figura 2.1 ilustra a associação entre o desenvolvimento *com* e *para* reuso de software, mostrando que são processos complementares.

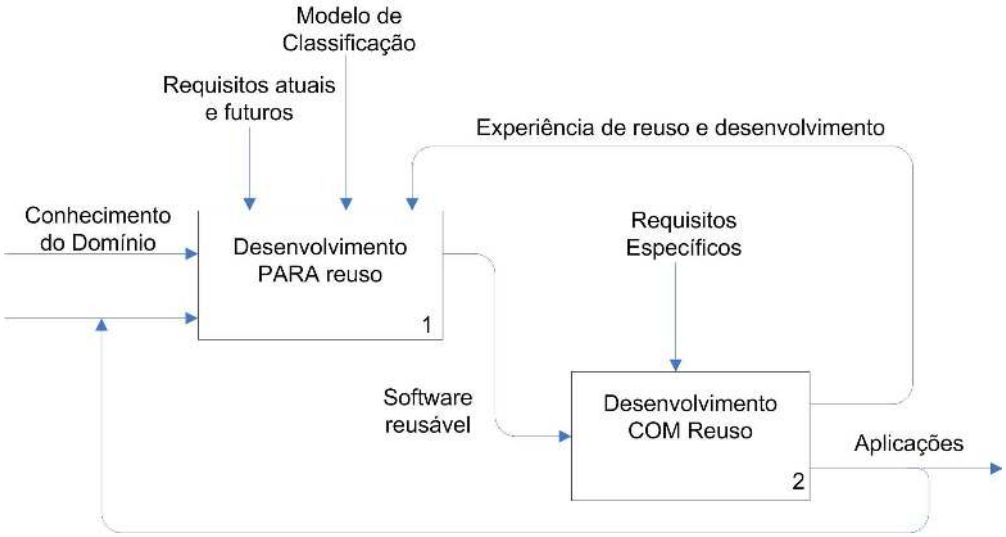

Figura 2.1: Desenvolvimento COM e PARA reuso de software

Considerando ambas as abordagens, pode-se reutilizar software seguindo uma abordagem de reengenharia. De acordo com [50], o processo de reengenharia de software [3] com vistas à produção de software reusável pode ser decomposto em cinco atividades seqüenciais:

- *Candidatura:* esta fase é composta pelas atividades de análise de código-fonte e criação de conjuntos de componentes de software. Cada um deles é um candidato a fazer parte de um módulo de software reutilizável quando desacoplado [4] apropriadamente e possivelmente generalizado.

[3] No contexto de software, reengenharia significa examinar e alterar um sistema para reconstituí-lo em uma nova forma.

[4] Acoplamento é uma conexão ou interação entre dois sistemas, mediante o que se transfere energia de um para

- *Eleição:* esta fase é composta pelas atividades de análise dos conjuntos de componentes de software criados na fase anterior e a produção de entidades reutilizáveis por meio de técnicas de reengenharia.
- *Qualificação:* esta fase é composta pelas atividades de produção de especificações para cada um dos módulos reusáveis obtidos na fase anterior. Exemplos disso são as especificações funcionais e de interface.
- *Classificação e Armazenamento:* esta fase é composta pelas atividades de classificação dos módulos reusáveis e respectivas especificações de acordo com uma taxonomia de referência. O objetivo é definir um sistema de repositório e populá-lo com os módulos reusáveis produzidos.
- *Recuperação e Exposição:* esta fase é composta pelas atividades de interação entre o usuário final e o repositório. O objetivo é tornar a tarefa de encontrar módulos que satisfaçam as necessidades de usuário o mais simples possível, dando todo o suporte visual para navegar através do repositório.

A Figura 2.2 apresenta um IDEF0 (*Integration Definition for Function Modeling*) [5] do processo de produção de entidades para reutilização segundo [50].

Na maioria dos casos, os componentes não foram gerados pensando em reutilização, devido a limitações financeiras ou temporais. A falta de uma cultura voltada para a reutilização pode ser, no entanto, determinante em situações em que as margens financeiras e temporais permitiam a sua aplicação, sendo ainda responsável pela ausência de um mercado para bibliotecas de componentes reutilizáveis.

Assim, quer os componentes tenham sido pensados para freqüente reutilização, quer tenham sido projetados para situações concretas, eles são passíveis de reutilização em determinado caso concreto. No processo de produção pretende-se identificar as características relevantes de cada componente para reutilização. Estas características são depois atribuídas a uma representação que permita a sua classificação, que traduz o resultado do processo de produção. Este é um dos mais complexos passos de todo o processo de reutilização.

Se a classificação for incorreta, o componente provavelmente não será encontrado, ou só será selecionado para situações impróprias. Por outro lado, se a classificação for muito geral, o componente

outro. Para computação, acoplamento é o nível de inter-dependência entre os módulos de um programa.

[5] IDEF0 é um método para modelar decisões, ações e atividades em uma organização ou sistema.

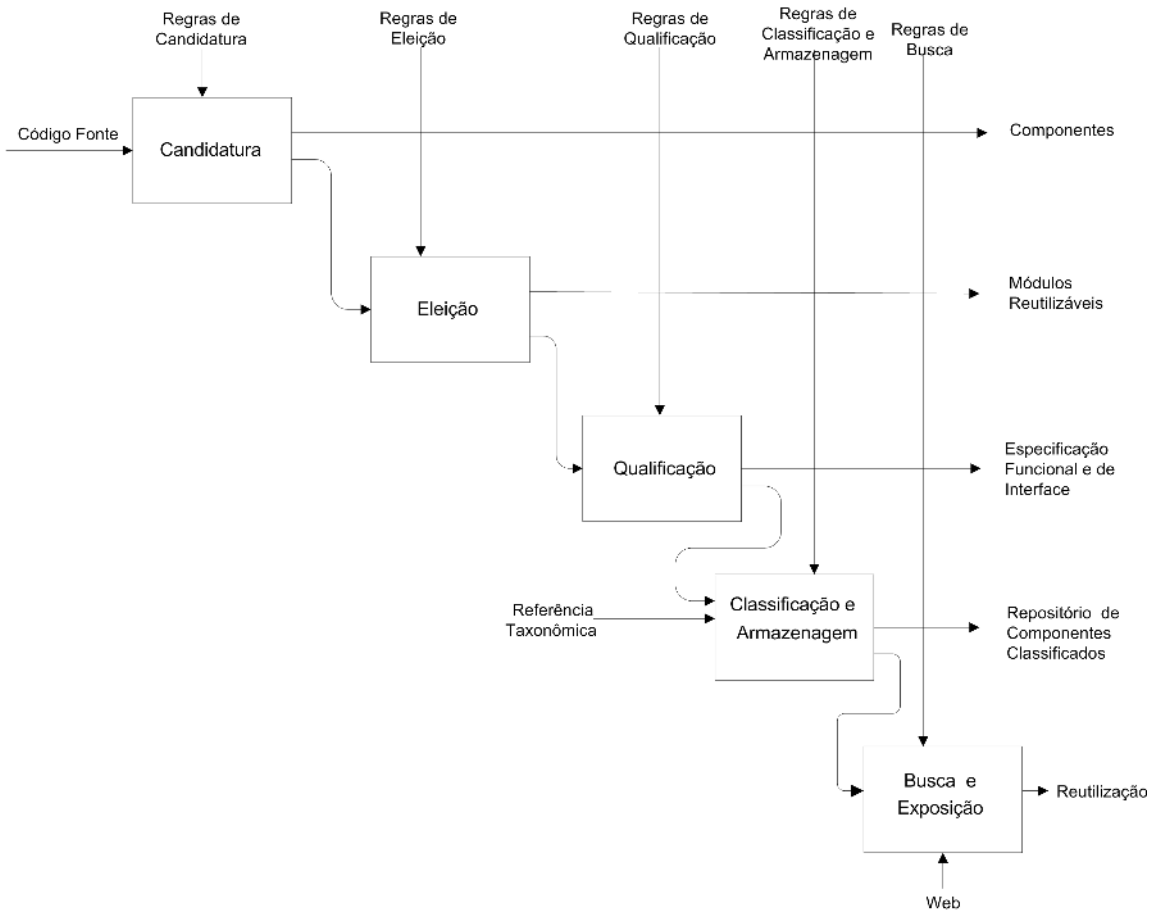

Figura 2.2: IDEF0 - Nível 0 - do processo de produção de entidades para reutilização (Adaptado de [50]).

será selecionado quase sempre, anulando as vantagens da classificação em si. No processo de reutilização propriamente dito, podem-se identificar várias fases que culminam na efetiva reutilização de uma parte de software. As quatro fases fundamentais são [78]: a construção da abstração, a seleção das entidades que satisfazem a abstração, a especialização do componente caso sejam necessárias modificações e, finalmente, a integração do componente no novo sistema.

A abstração é uma representação incompleta, ou visão parcial, de determinada entidade. A necessidade de escolher os componentes, com base num conjunto de características diretamente relacionadas com os requisitos, torna a abstração uma técnica essencial na reutilização. Para que a

operação de seleção, onde é obtida uma lista de possíveis candidatos, seja bem sucedida, é necessário que seja identificada previamente uma abstração.

Na operação de especialização pretende-se determinar o custo necessário para alterar o componente, obtido na seleção, de forma a desempenhar a funcionalidade desejada. Se o componente tiver sido gerado para reutilização, este processo pode se limitar à determinação do valor de certos parâmetros. Por outro lado, componentes gerados sem preocupações de reutilização podem obrigar alterações. No fim da operação de especialização deve ser possível avaliar o custo de integração dos vários componentes selecionados e determinar qual deles será reutilizado, ou se é necessário desenvolver do zero.

As alterações e parametrizações necessárias são efetuadas na operação de integração, em geral com o auxílio de uma linguagem. Essa linguagem pode ser a mesma utilizada para descrever o componente, quer este esteja codificado ou em fase de análise ou projeto, ou uma especialmente projetada para isso.

Finalmente, é conveniente destacar que a utilização de um componente numa aplicação, por exemplo uma rotina, é considerada um caso de reutilização. No entanto, as diversas invocações da mesma rotina durante a execução do programa, as diversas execuções do mesmo programa, ou a duplicação para distribuição sem alterações, não são considerados casos de reutilização.

2.7 Principais Barreiras à Reutilização

O desenvolvimento de aplicações com base em artefatos reutilizáveis deve observar algumas características que podem constituir tanto o sucesso, quanto uma barreira à reutilização [78]:

1. *Empenho:* a principal razão da baixa taxa de reutilização que se verifica deve-se à inexistência de qualquer esforço para reutilizar. A indiferença em reutilizar tem origem em razões como a falta de incentivo, limitações temporais, considerar os benefícios da reutilização duvidosos, falta de treinamento, falta de recursos técnicos, falta de suporte na organização ou falta de apoio por parte da gestão, entre outros.

2. *Existência:* o fato de não existirem partes isoladas, sob a forma de artefatos, quer tenham sido desenhados para reutilização ou não, representa o principal impedimento à reutilização. A falta de incentivo econômico para produzi-los ou o fato de exigir uma tecnologia mais recente para a sua produção também são razões para a inexistência de artefatos reutilizáveis.

3. *Disponibilidade:* a falta de repositórios e a limitação do seu uso são as principais razões da falta de disponibilidade do artefato. A falta de acesso ao código-fonte ou a dificuldade em analisar em profundidade o artefato são também razões que conduzem a que o artefato não seja considerado para reutilização.

4. *Acessibilidade:* uma má, ou insuficiente, representação e classificação do artefato ou a fraca qualidade das ferramentas de busca são as principais razões para não encontrá-lo. A dificuldade ou incapacidade para especificar o que procurar pode também fazer com que o artefato desejado nunca seja encontrado.

5. *Legibilidade:* a dificuldade em compreender o artefato, seja por documentação insuficiente ou devido à complexidade do mesmo, estão na origem da sua desconsideração para reutilização.

6. *Validação:* a falta de teste do artefato, o baixo desempenho, a não adesão aos padrões de normalização ou a falta de suporte por parte do produtor do artefato são consideradas as principais razões para sua rejeição na validação.

7. *Integrabilidade:* a existência de incompatibilidade de hardware e ambiente de integração são as razões mais freqüentes que impedem a integração do artefato. A necessidade de modificações muito extensas e trabalhosas, bem como a dependência de software exterior pode também conduzir à eliminação do artefato no desenvolvimento de determinada aplicação.

2.8 Suporte à Reutilização

De acordo com um relatório do Departamento de Defesa Americano (DoD/SRI) [23] sobre reutilização de software, as funções e atributos mais importantes de uma biblioteca de reutilização são quase que inteiramente determinadas pela escolha cuidadosa de algumas opções do repositório, tais como:

1. *Plataforma de Representação*, que pode incluir: sistemas baseados em papel, sistemas de gerenciamento de bases de dados, sistemas de armazenamento e busca de informações, sistemas baseados em conhecimento e hipertexto.

2. *Métodos de Indexação e Classificação*, dos quais os mais utilizados na prática são aqueles estudados pela comunidade científica biblioteconômica, que serão vistos na seção seguinte;

3. *Escalabilidade do Repositório* entre plataformas de diferentes tamanhos e complexidade, o que é de grande importância na incorporação de políticas de reuso em grandes empresas de desenvolvimento, como a NASA.

Algumas questões sobre Escalabilidade de Reuso também foram consideradas [23]: interoperabilidade das bibliotecas, projeto de interface das bibliotecas, bases de dados distribuídas, segurança da base de dados, garantia de qualidade, gerenciamento de mudança, suporte automático à indexação por vocabulário controlado, enfim, melhores representações de coleções de bibliotecas para ajudar os usuários a achar e entender as partes que procuram.

2.9 Considerações Finais

Neste capítulo foram abordados os principais conceitos e aspectos relacionados a reutilização de software. No próximo capítulo serão resumidos os principais métodos adotados pelos grupos de pesquisa e desenvolvimento para a construção de bibliotecas de software, concentrando-se na classificação e recuperação dos artefatos reutilizáveis.

Capítulo 3

Classificação e busca de componentes de software

> "A indústria de software não é industrializada" P. Naur and B. Randell, 1968

A classificação de software é uma etapa crucial na construção de uma biblioteca, sendo determinante no sucesso das buscas e, conseqüentemente, da adoção da ferramenta na prática de reutilização. A utilização de boas técnicas de classificação de componentes é essencial para que eles sejam descritos de forma completa e sintética, facilitando e melhorando o processo de seleção [58]. Embora as técnicas de classificação não façam parte do processo de reutilização, mas do processo de produção de componentes reutilizáveis, elas são normalmente abordadas em conjunto com as técnicas de seleção.

A qualidade de uma técnica de classificação só é mensurável pelo processo de seleção. Por exemplo: em uma biblioteca, o sucesso da recuperação depende, em geral, da qualidade da abstração escolhida para descrever os livros. Os bibliotecários têm formação especial para garantir uma classificação consistente dos livros arquivados. O treinamento permite descrever melhor as propriedades físicas do livro, como: nome, autor, páginas, ISBN etc. No entanto, como os utilizadores procuram a informação contida nos livros e não os livros em si, é preciso encontrar formas de descrever o conteúdo. Como nesse exemplo, o sucesso em descrever conteúdo de um componente de software tem sido limitado.

Na produção de software baseada em componentes reutilizáveis, a identificação correta de abstrações na fase de análise de requisitos é extremamente importante. É com base nas abstrações obtidas que será possível procurar um conjunto de componentes com características funcionais e não funcionais similares. Para recuperar um conjunto de componentes similares mediante a descrição de abstrações, é necessário dispor de um processo de seleção para efetuar as buscas e comparações pertinentes. Este conjunto de componentes deve ser o menor possível e conter os componentes que melhor

respondem às necessidades descritas. O processo de seleção ideal recuperará um só componente que corresponde à menor distância entre os requisitos apresentados e as características dos componentes disponíveis (cálculo de distância mínima).

Para obter componentes com características semelhantes aos requisitos é necessário criar condições para que tais requisitos sejam descritos de uma forma completa. Da mesma maneira, os componentes existentes também deve ser bem descritos, viabilizando o cálculo de distância mínima [78].

Os métodos de classificação estão divididos quanto a forma e quanto ao método. A classificação é automática se for efetuada por ferramentas e manual se exigir a intervenção humana. Existem também algumas situações híbridas, designadas por semi-automáticas ou assistidas, em que parte do processo de classificação é automático e parte é manual.

As técnicas existentes para classificação e seleção de componentes cobrem um largo espectro de métodos e algoritmos. Neste capítulo é apresentada uma visão geral sobre os principais métodos de classificação já desenvolvidos e também algumas técnicas de recuperação de informação utilizadas na seleção de componentes de software.

Os principais métodos de classificação podem ser divididos em três grupos [16]:

1. Classificação baseada em Palavras-chave em texto livre;
2. Classificação baseada em Facetas;
3. Classificação Enumerativa.

Existem modelos de classificação híbridos, que combinam duas ou mais abordagens de classificação. As seções seguintes apresentam resumidamente os três grupos acima citados.

3.1 Palavras-chave em texto livre

As primeiras bibliotecas de software forneciam pouca automatização para acessar e recuperar suas funções, módulos e estruturas de dados. Em bibliotecas matemáticas como o IMSL [1] [89], os fornecedores entregavam uma sólida documentação com descrição detalhada de cada componente reutilizável, um sumário e um índice analítico. Já no Unix ou no Microsoft Foundation Classes, a documentação estava acessível na forma de manuais on-line com busca. Somente nos últimos anos

[1] nome dado à coleção de bibliotecas matemáticas e estatísticas da empresa Visual Numerics

que as bibliotecas incorporaram métodos mais sofisticados de classificação baseados no processo de identificar e aglomerar strings de palavras-chave encontradas no texto do próprio artefato de software.

Salton [75] realizou um levantamento completo de técnicas baseadas em palavras-chave aplicadas à recuperação de texto e sugere o uso de:

- Índices invertidos para melhor acesso aos registros do texto;
- Restrições de distância para avaliar mais precisamente a proximidade de dois registros;
- Pesos e freqüências para distingüir a importância de palavras-chave;
- Listas de parada para eliminar palavras não importantes geralmente utilizadas;
- Listas de sinônimos e enciclopédias para ampliar as consultas de recuperação de texto;
- Similarização de palavras (*word stemming*) pelas raízes comuns;
- Procura por nível de quorum para controlar o tamanho da saída da recuperação;
- Técnicas de procura em listas parciais para considerar subconjuntos de termos da pesquisa;
- Técnicas de formação de frases para controlar o contexto de co-ocorrência de palavras-chave;
- Indexação estatística de texto para automatizar a classificação de documentos e
- Aglomeração de documentos para melhorar a recuperação de documentos relacionados.

Em um trabalho posterior, Salton e Smith [76] também investigou várias abordagens lingüísticas para indexação de documentos, como a geração de identificadores de conteúdo complexo, uso de termos semânticos obtidos de dicionários lidos por máquina e a utilização de bases de conhecimento especialmente construídas para essa tarefa. Destacam a imprecisão inaceitável dos métodos sintáticos utilizados em seus experimentos e indica o poder de métodos estatísticos mais simples.

Quase todas as técnicas aplicáveis na recuperação de texto foram testadas com sucesso variável na classificação e recuperação de artefatos de software textuais. Muitos dos métodos identificados também foram estendidos ou combinados com outros tipos de métodos de classificação do software, como Facetada ou Enumerativa [16], apresentados nas seções seguintes).

3.1.1 Processamento de Linguagem Natural (PLN)

Girardi e Ibrahim [37] desenvolveram uma nova abordagem de classificação e recuperação de software que resultou no sistema de apoio à reutilização denominado ROSA (*Reuse Of Software Artifacts*). O principal objetivo era criar um sistema viável em relação aos custos de classificação, independente de domínio e com boa efetividade na recuperação de software.

O esquema proposto pode ser visto como um avanço da classificação baseada em palavras-chave em texto livre em direção às técnicas de processamento de linguagem natural (PLN). A idéia central era permitir a classificação automática dos componentes de software por meio do processamento da linguagem natural existente nas descrições dos componentes. Informações léxicas, sintáticas e semânticas eram extraídas de orações verbais e nominais das descrições e, posteriormente, analisadas com o intuito de criar uma rede semântica dos componentes classificados. Utilizando regras lógicas, ROSA tenta inferir a semântica dos termos analisados léxica e sintaticamente, descobrindo importantes informações sobre os componentes, como ações e objetivos, dentre outros.

O mecanismo de expansão de consultas é automatizado com o uso do léxico Wordnet [27]. Originalmente construído para a língua inglesa, ele apóia-se em um tesauro (*thesaurus*) que define a semântica dos seus termos. Ele é utilizado na obtenção de informações morfológicas, categorias gramaticais dos termos e o relacionamento semântico entre eles. Portanto, para cada termo buscado pelo usuário, era possível obter termos semanticamente relacionados a ele (como sinônimos e hiperônimos), melhorando a eficácia do sistema de recuperação dos componentes de software.

O sistema possui limitações no mecanismo semântico, que trabalha essencialmente com orações nominais e verbais curtas - de uma frase. Além disso, enfrenta os desafios de construção de um sistema de processamento de linguagem natural.

3.1.2 Reuso com bibliotecas de documentos de propósito geral

Como a maioria dos artefatos de software são textuais, a abordagem mais óbvia para a classificação e busca de software é adotar sistemas de processamento de texto pré-existentes. Frakes e Nejmeh [31] empregaram o sistema baseado em palavras-chave denominado CATALOG para criar uma biblioteca pequena de módulos de software.

O sistema CATALOG contém um gerador de base de dados, uma ferramenta interativa para

criar, modificar, adicionar e remover registros, além de uma interface de busca. Essa interface permite combinações booleanas dos termos e conjuntos de registros recuperados e as consultas são resolvidas com técnicas de casamento parcial. A busca é realizada usando um índice invertido de termos significativos e uma lista de palavras que devem ser descartadas (lista de parada).

Para promover clareza e a exatidão da informação reusável, Frakes e Nejmeh a estruturaram usando moldes predefinidos (*templates*) que tratavam diferentes tipos de artefatos de software, como módulos e funções. A aproximação feita neste sistema experimental mostrou que a tecnologia de recuperação de informação (RI) padrão (não construída especificamente com o propósito de classificar e buscar artefatos de software) poderia ser usada eficazmente na organização de bibliotecas simples.

Mostraram também que a organização de grandes repositórios necessitam de tecnologias especializadas que vão além de RI. RI poderia ajudar na interpretação, indexação e estruturação dos artefatos de software, mas necessitaria de técnicas de processando de informação tais como: PLN, representação do conhecimento e uso de enciclopédias inteligentes.

3.1.3 Reuso com bibliotecas especialistas

Um sistema especializado em reuso de software, mais conhecido pelo seu mecanismo de recuperação baseado em palavras-chave, é o STARS (*Software Technology for Adaptable, Reliable Systems*), do Departamento de Defesa Americano (DoD) [22].

A experiência com a construção e o uso do sistema conduziu os desenvolvedores às seguintes questões sobre a adoção de padrões de recuperação de texto em reutilização de software:

- similarização de palavras às vezes reduz (ou similariza) palavras não correlatas;
- algumas palavras aparentemente sem importância são de fato palavras que têm um conteúdo em alguns contextos;
- o uso automático de sinônimos às vezes aumenta o espectro da consulta de maneira imprópria;
- determinadas combinações de palavras devem ser tratadas como frases ao invés de palavras de busca;
- a expansão de consultas sem acertos nem sempre é útil.

Em vista desses problemas, o DoD desenvolveu e incorporou ao sistema as seguintes facilidades:

- Similarização de palavras mais exata e baseada no uso pretendido de palavras;
- Controle maior sobre o uso de palavras de menor valor, como os sinônimos, e de expansão da consulta;
- Casamento (*matching*) de frases no refinamento de consultas;
- Ajuda no processo de reformulação da consulta;
- Maior poder no tratamento de expressões técnicas.

3.1.4 Reuso com acesso guiado por menu

O sistema REUSE (*REUsing Software Efficiently*) foi construído para classificar e recuperar eficazmente a informação existente do software [1], tais como: módulos, pacotes e programas executáveis. Similarmente ao STARS, ele usa o esquema de palavras-chave para classificar seus componentes da biblioteca. Entretanto, o acesso de usuário ao sistema é organizado não por meio das consultas, mas por uma interface adaptável, guiada por menu.

O software usa as palavras-chave, que refletem as necessidades de uma organização, para construir um sistema hierárquico de menus que refletem os padrões e as metodologias organizacionais. Tais menus e palavras-chave fornecem ferramentas para classificar e recuperar componentes de software reutilizáveis. O sistema REUSE mantém também uma enciclopédia para reduzir diferenças de terminologia dentro da comunidade de usuários.

3.1.5 Outros tipos de reuso

Sistemas mais recentes estendem a recuperação fundamental por palavras-chave, incluindo facilidades que realçam o acesso e a manutenção dos repositórios de reutilização. No sistema SIB (*Software Information Base*) [34], as palavras-chave que caracterizam os artefatos reutilizáveis são organizadas em descritores da palavra-chave. São determinados pesos para sua importância relativa e então relacioná-los pela similaridade semântica. Os descritores e seus relacionamentos formam uma rede elaborada de nós e ligações que podem ser buscados pela facilidade de busca do sistema.

No sistema CART (*Computer-Aided Reuse Tool*) [48], a classificação é executada usando palavras-chave geradas automaticamente de modelos de especificação. Os usuários podem então fazer buscas no sistema usando uma linguagem muito mais rica do que aquela usada no modelo. Isso é possível

com o auxílio de uma enciclopédia, normalização da terminologia, e alguma ajuda do usuário sempre que a resolução for muito complicada para o sistema.

No sistema CodeFinder [42], o sistema de recuperação baseado em palavras-chave é sustentado pela reformulação interativa das consultas e por uma aproximação da recuperação usando uma rede neural, com um algoritmo capaz de recuperar eficazmente os componentes de software relacionados à consulta, seja feita por palavras-chave, frases ou afinidades lexicais.

3.2 Facetas

A classificação baseada em facetas foi proposta em [73], como uma técnica eficaz para a gerência de informação da biblioteca. É a principal concorrente do popular sistema decimal de Dewey para classificação de coleções de biblioteca. Prieto-Diaz e Freeman foram os primeiros a sugerir a possibilidade de adotar o esquema de facetas na classificação e recuperação dos artefatos reutilizáveis de software [64, 68].

Tradicionalmente, a técnica de classificação por facetas exige a existência de um grande número de termos do domínio organizados em diversos conjuntos distintos, mutuamente exclusivos e ortogonais denominados **facetas**. Cada artefato é descrito nos termos de um vetor de descrição, onde cada valor do vetor é escolhido de uma das facetas predefinidas. Tais vetores de classificação são armazenados então em uma tabela relacional, onde as colunas representam facetas e as fileiras denotam descrições dos artefatos. A tabela de classificação pode mais tarde ser procurada para encontrar e recuperar as descrições necessárias do artefato. Procurá-los seria simples e eficaz utilizando uma linguagem de consulta relacional, tal como o SQL (*Structured Query Language*).

Um método alternativo de recuperação é baseado em uma métrica que avalia a distância conceitual entre uma consulta e as facetas de cada artefato. Outros mecanismos de busca podem utilizar métodos mais sofisticados, baseados nos perfis estatísticos das descrições do artefato, métricas de afinidade e similaridade, espaço vetorial ou lógica fuzzy [68, 75].

Sua principal vantagem é a facilidade de classificação do artefato, a simplicidade de representação e armazenamento das descrições, a uniformidade de atributos da classificação e a facilidade da automatização [66]. A principal deficiência é o custo elevado da manutenção do repositório, pois a classificação é essencialmente manual [57, 82].

Os seguintes aspectos são característicos nos sistemas de classificação baseados em facetas:

- as facetas especificam um vocabulário controlado usado na classificação do artefato;
- as facetas são geralmente combinadas para fazer a classificação de um assunto específico;
- os termos de classificação são geralmente ampliados com o uso de listas de sinônimos e uma enciclopédia;
- as facetas são ordenadas por sua relevância pelos usuários da coleção;
- a similaridade de termos da faceta é avaliada organizando-as em um grafo de distância conceitual;
- os artigos recuperados são classificados por sua proximidade semântica à consulta;
- a classificação baseada em facetas é extensível, porque os termos novos da classificação e as novas facetas podem ser adicionados a qualquer momento.

3.2.1 Facetas para Componentes de Negócio

Vitharana *et al* [91] propuseram um esquema efetivo de classificação para componentes de negócio considerando vários níveis de detalhe e abstração, além de um repositório para armazenar e recuperar tais componentes. Um componente de negócio pode ser visto como a implementação de um único conceito autônomo de negócio. Avanços recentes na área de Web Services e a adoção de padrões como SOAP (*Simple Object Access Protocol*) [13], WSDL (*Web Service Description Language*) [14] e UDDI (*Universal Description, Discovery Integration*) [62] deram suporte ao desenvolvimento de componentes de negócio e aplicações baseadas nesses componentes.

O diferencial da técnica é a preocupação em capturar informações que o componente de negócio tem, tornando a classificação mais completa e abrangente. O que diferencia um componente de negócio de um componente simples (como os de interface gráfica) é a sua capacidade de oferecer uma funcionalidade de negócio, como processamento de cartão de crédito, pagamento de contas, validação de dados etc. É possível compor vários componentes de negócio afim de oferecer uma funcionalidade mais complexa ao sistema, seguindo a mesma abordagem de construção de software baseada em componentes que Mcllroy propôs [54].

Como mostra a Figura 3.1, um componente de negócio pode ser descrito em vários níveis de abstração ao longo de uma hierarquia. Nela, o componente de negócio representa o mais alto nível

de abstração, enquanto as interfaces, tipos de dados, métodos, etc, representam os níveis de detalhe subseqüentes. Particularmente, em orientação a objetos, a hierarquia de abstração de um componente consiste em uma ou mais interfaces que representam as funcionalidades providas pelo componente.

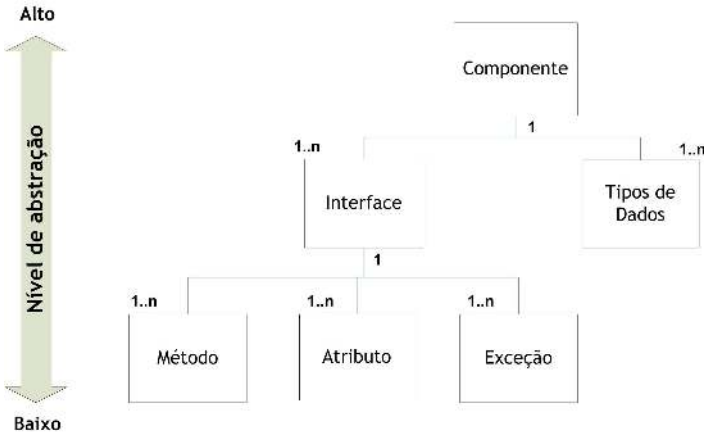

Figura 3.1: Relação entre a hierarquia e o nível de abstração de um componente (adaptada de Vitharana et al [91]).

Para prover informações suficientes sobre um componente de negócio levando em conta diferentes níveis de abstração, Vitharana *et al* [91] definiram dois tipos de informação: estruturadas e semi-estruturadas. As informações estruturadas são todas as características bem definidas que descrevem um componente (como versão e plataforma) e as semi-estruturadas são aquelas que podem assumir diversas formas (palavras-chave, texto ou mesmo nulo, quando não se aplicarem). Atributos são utilizados para armazenar os identificadores estruturados e facetas são utilizadas para descrever as características semi-estruturadas. Ambos são utilizados para classificar os diferentes níveis da hierarquia de abstração de um componente. O conjunto de atributos é descrito na Tabela 3.1 e o de facetas na Tabela 3.2.

Supondo o componente *Cinema* - responsável por gerenciar vendas de ingressos para um cinema e possíveis tarefas auxiliares (como gerar o ingresso, reservar lugares e cancelar reservas) - uma possível classificação seguindo o modelo de [91] é apresentada nas Tabelas 3.3 e 3.4.

A principal contribuição desse modelo é a possibilidade de classificar um componente de acordo com sua estrutura hierárquica e em vários níveis de abstração usando informações estruturadas e semi-estruturadas.

Atributo	**Descrição**
Identification (ID,name)	Identificador único do componente, Nome do componente
Type (system, algorithmic, application)	Tipo de sistema quanto ao escopo e nível de especialização
Management (owner, contact, version)	Informações sobre o autor do componente, forma de contato e a versão entregue
Industry-Specific Application	Tipo específico de Indústria onde o componente pode ser usado
Across-industry Application	Tipo geral de Indústria onde o componente pode ser usado
Operating System	Sistemas Operacionais onde o componente pode ser instalado
Implementation Language	Linguagem de Programação em que o componente foi construído

Tabela 3.1: Atributos do Modelo de Classificação de Vitharana *et al* para Componentes de Negócio

Faceta	**Descrição**
Synonym	Outros possíveis nomes para o componente
Role	Papéis que o componente poderia exercer em aplicações
Business Rules	Regras importantes de alto nível aplicáveis ao componente
Function/task	O que o componente faz
Element/part	Elementos ou partes associadas ao componente
Action/event	Ações e eventos relacionados ao componente
User	Pessoa/coisa que usa o componente

Tabela 3.2: Facetas do Modelo de Classificação de Vitharana *et al* para Componentes de Negócio

3.3 Método Enumerativo

Uma maneira completamente diferente de organizar artefatos de software pode ser obtida com o uso da classificação enumerativa [64], onde acredita-se que artefatos podem ser incluídos em uma hierarquia predefinida de categorias. Uma das aplicações bem sucedida deste método é a classificação de títulos de livro por assunto via método decimal de Dewey. Uma outra forma de classificação hierárquica é representada pelos sistemas de conceitos orientados a objetos [21, 39, 41].

Atributo	**Descrição**
Identification (ID,name)	21, Cinema
Type (system, algorithmic, application)	Aplicação
Management (owner, contact, version)	Xpto Desenvolvimento de Software, contato@xpto.com.br, 1.3
Industry-Specific Application	Indústria do Entretenimento
Across-industry Application	Gerenciamento de Venda de Tickets
Operating System	Windows, Linux, UNIX, Mac OS
Implementation Language	Java

Tabela 3.3: Exemplo de preenchimento dos atributos do Componente Cinema

Faceta	**Descrição**
Synonym	cinema, teatro, ópera
Role	venda de ingressos em um sistema de gerenciamento de Cinema
Business Rules	ingressos podem ser vendidos antecipadamente
Function/task	precificação, geração de ingressos
Element/part	ingresso
Action/event	reserva de ingresso, compra de ingresso
User	cliente, atendente

Tabela 3.4: Exemplo de preenchimento das facetas do Componente Cinema

Orientação a Objetos consiste tipicamente em muitos conceitos relacionados, geralmente referenciados como objetos. Eles são descritos nos termos de seus valores de propriedade que podem incluir referências a outros objetos. Os objetos podem também definir métodos, que produzem computações úteis no processamento de pedidos recebidos de outros objetos do sistema ou de interfaces do sistema [78]. Os objetos novos, com todas as suas propriedades e métodos, são criados (instanciados) de acordo com as descrições abstratas fornecidas pelas classes. As classes são organizadas em hierarquias de herança. Como resultado, as classes mais específicas herdam as propriedades e o comportamento de suas superclasses.

Diferentes abordagens de classificação podem ter de ser feitas para os diferentes tipos de objetos no sistema. Pode-se desejar construir uma taxonomia diferente para especificações e projetos da classe, alternativas de projeto, peculiaridades arquiteturais ou detalhes de implementação da classe [18].

Os defensores de orientação a objetos [55] pregam que as estruturas e os mecanismos utilizados em sistemas de software orientados a objetos facilitam extremamente a reutilização de componentes de software, pois:

- A programação orientada a objetos promove o desenvolvimento *bottom-up* usando classes já existentes;
- Um sistema orientado a objetos pode ser visto como uma coleção de classes inter-relacionadas e com comunicação, o que é melhor que um bloco monolítico de programação;
- As classes combinam dados e procedimentos, são modulares e, em geral, suas relações são claras e abstratas;
- As classes novas podem ser derivadas das classes existentes a partir de especialização e instanciamento;
- As características e o comportamento de classes existentes estão disponíveis nas classes derivadas através do mecanismo de herança e herança múltipla;
- É possível desenvolver especialmente para reuso, pois podem incluir um número de características "adiadas" (por exemplo, métodos virtuais), que serão implementadas na especialização da classe;
- As classes combinam elementos do projeto e implementação em uma única unidade reutilizável.

3.3.1 Relacionamentos de artefatos em redes semânticas

As redes semânticas são os sistemas mais recentes para descrição estruturada de objeto - uma coleção de conceitos e valores-propriedade, formando uma rede dos nós relacionados por suas propriedades [69].

Embora tais sistemas não sejam projetados para serem repositórios de componentes de software, muitas técnicas desenvolvidas inicialmente para a manipulação de conhecimento em redes semânticas são aplicáveis ao processamento de artefatos e suas descrições.

As redes semânticas são estruturadas em planos de nós relacionados a um único conceito. Cada plano, entretanto, não tem nenhuma estrutura interna, uma vez que um conceito semântico é representado como por uma rede de todos os nós alcançáveis naquele plano. Um conceito pode também

estar relacionado com os conceitos de outros planos. Tais ligações representam inferências e deslocamentos de atenção [3].

Na rede semântica não há nenhuma hierarquia predeterminada de classes e superclasses: cada nó define sua própria hierarquia de conceitos alcançáveis do seu ponto na rede.

3.3.2 Taxonomia de componentes em categorias

A organização taxonômica de artefatos de software é muito popular em sistemas que armazenam componentes OO reutilizáveis. Muitos deles foram parcialmente classificados na fase de projeto e implementação. Devanbu *et al* [20] descrevem o sistema de informação do software LaSSIE que integra visões arquiteturais, conceituais e de código de um grande sistema.

As facilidades de pesquisa e navegação do sistema permitem que os programadores procurem possíveis componentes reutillizáveis em um grande sistema de software. A base de conhecimento de LaSSIE armazena os componentes de software em uma estrutura taxonômica que representam ações (por exemplo, ações externas e internas), objetos (por exemplo: dispositivo, recurso, hardware e software), atores (por exemplo: usuários, processos, grupos e processadores) e estados (por exemplo: da linha, do recurso, dos dados e da rede).

Sistemas baseados em conhecimento como LaSSIE representam e classificam componentes de software na estrutura com lógica. Isso permite:

1. Agregação da infomação sobre programas e seus componentes;
2. Recuperação semântica de componentes de software;
3. Uso da classificação e da herança para dar suporte aos *updates* do software;
4. Uso de uma base de conhecimento como índice inteligente de artefatos do software.

3.4 Recuperação de Componentes de Software

Para aqueles que adotam políticas de reutilização, o crescente desenvolvimento de componentes de software implica no aumento das bibliotecas de software. É razoável imaginar que a dificuldade em localizar os componentes classificados também cresce na mesma proporção.

Encontrar o componente que se deseja reutilizar é o objetivo principal de qualquer sistema de apoio à reutilização de software. Sendo assim, a qualidade do mecanismo de seleção de componentes torna-se essencial para a efetividade do sistema como um todo. Nesse contexto, a área de Recuperação de Informação (RI)- apresentada a seguir - mostra-se uma ferramenta poderosa de apoio à seleção de componentes de software.

3.4.1 Recuperação de Informação

A disciplina de Recuperação de Informação (*"Information Retrieval"*) foi criada por Moores [59] em 1951 e definida para tratar "dos aspectos intelectuais da descrição da informação e sua especificação para busca, e também de qualquer sistema, técnicas ou máquinas que são empregadas para realizar essa operação".

O termo "recuperação de informação" atribuído a sistemas computacionais é ainda hoje bastante questionado, sendo que muitos autores preferem o termo "recuperação de documentos" (*document retrieval*) ou "recuperação de textos" (*text retrieval*). De fato, os sistemas não recuperam informação, mas sim documentos ou referências cujo conteúdo poderá ser relevante para a necessidade de informação do usuário. A diferença entre esses sistemas e os sistemas de Bancos de Dados está na possibilidade de recuperar informação potencial sobre um determinado assunto ao invés de simplesmente dados que satisfaçam precisamente uma expressão de consulta. No contexto de reutilização, o termo *documento*, freqüentemente utilizado em RI, pode ser entendido como um componente de software que será classificado e posteriormente buscado.

Um sistema de recuperação de informação deve representar o conteúdo dos documentos do *corpus* (conjunto de documentos do sistema), apresentando-os de maneira adequada a cada consulta do usuário, possibilitando uma rápida seleção dos itens de interesse. Tais consultas, em alguns sistemas de RI, podem ser expandidas, agregando novos termos aos originalmente inseridos. O objetivo desse procedimento é obter uma classificação, para os documentos recuperados, mais apurada quanto à relevância [40].

Nesse tipo de sistema existem duas fases:

1. *Processo de Indexação:* a coleção de documentos é indexada para melhorar o desempenho da recuperação e
2. *Processo de Recuperação:* dada uma consulta, os documentos que apresentarem maior similar-

idade serão recuperados.

A Figura 3.2 mostra um processo típico de classificação e recuperação de informação, onde o artefato sofre o processo de classificação (ou Indexação) e o usuário faz consultas à biblioteca de documentos.

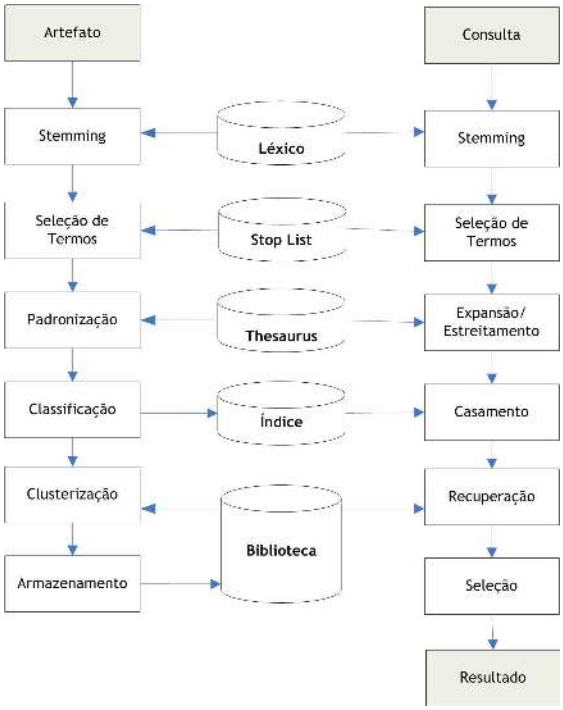

Figura 3.2: Processo típico de classificação e recuperação de informação. Adaptado de Baeza-Yates [3]

Resumidamente, um processo de indexação envolve: documentos, um tesauro e um índice invertido. Para cada conjunto de documentos, esse processo gera uma lista de termos existentes no conjunto - o tesauro - e informações sobre a ocorrência dos termos nos documentos. Nem todas as palavras de um documento são termos. Aquelas que possuem menor valor - *stopwords* - não são consideradas por aparecerem com muita freqüência no texto, perdendo sua utilidade no contexto da recuperação de informação. A maioria delas é composta por artigos e preposições.

Retirando-se as *stopwords*, as palavras restantes passam por um outro processo denominado Remoção de Afixos (*stemming*), que tem como objetivo extrair os radicais das palavras. Essa fase

garante que palavras diferentes originadas de um mesmo radical tenham o mesmo tratamento durante a classificação e a recuperação da informação.

O processo de recuperação envolve o processamento da consulta (função de busca), cálculo da relevância dos documentos e ordenação dos resultados. Dependendo do método de busca, o cálculo de pesos é realizado de maneira diferente. Muitas ferramentas de busca utilizam a técnica de recuperação ordenada, que utiliza a distribuição da freqüência dos termos da consulta na coleção de documentos.

O principal elemento do processo de recuperação de informação é a função de busca, que compara as representações internas dos documentos com a expressão de busca dos usuários e recupera os itens que supostamente fornecem a informação desejada. O fato de um termo utilizado na expressão de busca aparecer na representação de um documento não significa que ele seja relevante para a necessidade do usuário. Para resolver esse tipo de problema, foram propostos alguns modelos de recuperação que pudessem melhorar eficácia dos documentos encontrados para determinada expressão de busca. Alguns deles serão resumidos nas seções a seguir.

Modelo Booleano

Os sistemas de busca booleana foram os pioneiros RI e eram baseados em palavras-chave. Utilizando os operadores da Lógica Clássica (AND, OR e NOT), o mecanismo processava a consulta e devolvia os documentos que continham literalmente as palavras da expressão de consulta.

O aprimoramento desse modelo foi obtido com a utilização de operadores parciais, que permitem especificar condições relacionadas à distância e posição dos termos no texto. O formato genérico de um operador parcial é [3]:

$$t_1 \ \boldsymbol{n \ unidades \ de} \ t_2$$

onde n é um inteiro que indica a unidade de distância em palavras, sentenças ou parágrafos entre os termos da consulta t_1 e t_2. Combinando os operadores booleanos e de proximidade foi possível construir expressões de busca mais restritivas ou mais genéricas, melhorando a qualidade de recuperação de informação do método booleano.

Esse método apresenta desvantagens consideráveis: retorna resultados muito grandes e não é capaz de calcular a relevância dos documentos recuperados, ficando a cargo do usuário avaliar a relevância e ordenar manualmente os documentos.

Modelo Vetorial

O modelo vetorial propõe um esquema no qual é possível obter documentos que respondem parcialmente a uma expressão de busca. Isto é feito por meio da associação de pesos aos temos de indexação e aos termos da expressão de busca. Os pesos são utilizados para calcular o grau de similaridade entre a expressão de busca formulada pelo usuário e cada um dos documentos do *corpus*. O resultado é um conjunto de documentos ordenado pelo grau de similaridade à expressão de busca.

A representação de cada documento é feita com um vetor e cada elemento dele é usado para representar o peso (ou relevância) de um termo de indexação para o documento. Sendo assim, o vetor descreve a posição do documento em um espaço multidimensional, onde cada termo de indexação representa uma dimensão. Ocorre uma normalização dos valores de pesos entre 0 e 1, e quanto mais próximo de 1 for o peso, mais importante é o termo na descrição do documento.

Da mesma forma que os documentos do *corpus* são representados por vetores, as expressões de busca também são convertidas para esse mesmo modelo durante o processamento da consulta. Isso permite o cálculo do grau de similaridade entre a expressão de busca e cada um dos documentos.

A representação de consultas e documentos em vetor ocorre da seguinte forma: para uma coleção com n documentos D, onde cada documento d_i possui m diferentes termos t_j, um certo documento d_i é representado como:

$$d_i = (w_{1,d}, w_{2,d}, \ldots, w_{m,d}) \tag{3.1}$$

onde $w_{j,d}$ é o peso do j-ésimo termo do documento d_i. O peso mede a relação de importância entre o documento e o termo.

O modelo vetorial foi consagrado por definir um modelo conceitual de representação da informação, o que serviu como base para uma série de pesquisas na área de RI. O principal sistema resultante do modelo foi o SMART [76]. Além de usar o conceito de *"term frequency (TF)"*, que é o número de vezes que uma palavra aparece em certo documento, ele propõe um método automático para o cálculo de pesos de documentos e expressões de buscas, chamado *"inverse document frequency (idf)"*, que possibilita ver a distribuição de cada termo no *corpus*. Quanto menor o número de documentos que contêm determinado termo, maior o *idf* do termo. Se todos os documentos do *corpus* contiverem esse termo, seu idf será máximo (1). Essa medida auxilia o cálculo dos pesos dos termos

em relação a cada documento, onde os melhores termos (melhores discriminates) serão aqueles que ocorrem com uma grande freqüência em poucos documentos.

Para determinar o peso $w_{j,d}$ mencionado na Equação 3.1, ambas as medidas foram relacionadas na fórmula TF-IDF [46]:

$$w_{t,d} = tf_{t,d} \cdot idf_{t,d} \tag{3.2}$$

onde $tf_{i,d}$ (*term frequency*) é o número de aparições do termo t no documento d, $idf_{t,d} = log(\frac{n}{n_t}+1)$ (*inverse document frequency*) mede a freqüência relativa do termo na coleção e n_t é o número de documentos em que o termo t aparece.

Após representar consultas e documentos no espaço vetorial multidimensional, a similaridade entre dois vetores pode ser calculada pelo co-seno do ângulo formado pelos vetores da seguinte maneira:

$$sim(d,q) = \frac{\sum_{t=1}^{m} w_{t,d} \cdot w_{t,q}}{\sqrt{\sum_{t=1}^{m} w_{t,d}^2 \cdot \sum_{t=1}^{m} w_{t,q}^2}} \tag{3.3}$$

Quanto maior o valor do co-seno, maior o grau de similaridade. É possível também calcular a similaridade entre um par qualquer de documentos do *corpus*. Com os resultados de similaridade entre a expressão de busca e os documentos disponíveis, cria-se uma lista ordenada que normalmente restringe tais resultados a um número máximo de documentos ou mesmo um limite mínimo para o valor de similaridade.

A Figura 3.3 ilustra o exemplo de uma consulta Q feita ao *corpus* que contém os documentos *D1* e *D2*. O cálculo necessário para a representação das consultas e documentos em vetores já foi feito, como demonstrado na Figura 3.3.

Para calcular a similaridade entre os documentos e a consulta, aplica-se a equação 3.3 resultando em:

$$sim(D2,Q) = \frac{(0.4+0.2)+(0.8+0.7)}{\sqrt{[(0.4)^2+(0.8)^2]\cdot[(0.2)^2+(0.7)^2]}} = \frac{0.64}{\sqrt{0.42}} = 0.98 \tag{3.4}$$

$$sim(D1,Q) = \frac{0.56}{\sqrt{0.58}} = 0.74 \tag{3.5}$$

Portanto, o documento $D2$ é mais similar à consulta Q do que o documento $D1$, sendo o primeiro

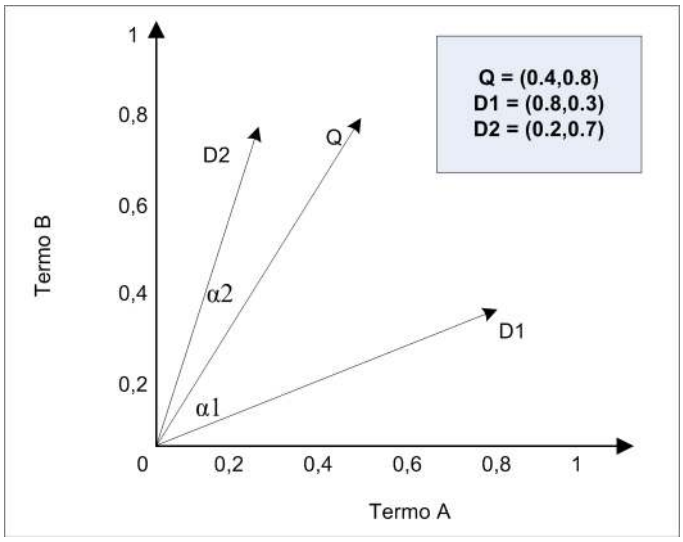

Figura 3.3: Representação dos vetores de consulta Q e dois documentos D1 e D2

da lista ordenada de resultados.

Uma característica desse modelo é que os termos de indexação são independentes, ou seja, não existe um relacionamento entre eles. Baeza-Yates e Ribeiro Neto [3] destacaram que não existem evidências conclusivas que apontem essa característica como desvantagem para o modelo. Uma limitação conhecida é a impossibilidade de formulação de buscas booleanas, o que o torna menos flexível.

Modelo Probabilístico

O modelo probabilístico tenta representar o processo de recuperação de informação sob um ponto de vista probabilístico. Os pesos dos termos de indexação e os de expressão de busca também são representados em vetores e existe a medida do grau de similaridade. O resultado do grau de similaridade é na verdade a probabilidade de similaridade entre eles. Apesar do forte embasamento teórico desse modelo, existem fortes questionamentos sobre as hipóteses assumidas em alguns cálculos probabilísticos que contribuem para as incertezas sobre a precisão do modelo.

3.5 Análise dos Métodos

Como visto anteriormente, para analisar um método de classificação de componentes é necessário um método de seleção. Os métodos de seleção são caracterizados quanto ao casamento e quanto à qualidade do processo. O casamento descreve a forma de comparação dos requisitos com as características dos componentes. O casamento exato só devolve os componentes que obedecem rigorosamente a todos os requisitos. Esta forma de casamento é utilizada em processos de integração automática, mas o seu sucesso só é aceitável sob condições controladas. Na quase totalidade dos casos de desenvolvimento de software o casamento é aproximado, com base na distância conceitual ou na ordem parcial [78].

A qualidade do processo de seleção é o parâmetro que permite determinar o sucesso não só do processo de seleção, como dos processos de classificação e de especialização com os quais interage. Os critérios de qualidade são:

1. *Cobertura:* mede o número de componentes relevantes obtidos no processo de seleção versus o número total de candidatos que obedecem aos requisitos.

$$C = \frac{\text{Número relevante que é recuperado}}{\text{Número total relevante}} \tag{3.6}$$

   Uma recuperação baixa significa que muitos componentes interessantes estão sendo ignorados.

2. *Precisão:* a precisão mede o número de componentes realmente utilizáveis frente ao número de componentes obtidos no processo de seleção.

$$P = \frac{\text{Número recuperado que é relevante}}{\text{Número total recuperado}} \tag{3.7}$$

   Sendo assim, se um sistema recupera x documentos para uma consulta e x/2 deles são relevantes, a precisão do sistema é de 50%. Uma baixa precisão significa que o método de seleção devolve muitos componentes que, na realidade, não obedecem às especificações informadas.

3. *Sobreposição:* a sobreposição oferece uma medida comparativa entre dois métodos, ou seja, não é uma característica intrínseca do método. A sobreposição mede o número de componentes recuperados por ambos os métodos versus o número de componentes diferentes que os dois métodos recuperaram em conjunto. Um valor baixo de sobreposição permite concluir que métodos difer-

entes recuperam componentes diferentes, mesmo que tenham valores de recuperação e precisão semelhantes.

4. *Tempo de busca:* o tempo de busca mede o tempo gasto para obter os resultados pretendidos. Notar que embora estes tempos possam diferir significativamente, o custo de adaptação dos componentes obtidos pode ser suficiente para contrabalançar as diferenças de tempo em várias ordens de grandeza.

5. *Custo de adaptação:* mede uma estimativa do esforço, em tempo e recursos utilizados, que se prevê necessário para adaptar o componente. Esta medida é muito subjetiva, pois lidar com métodos de trabalho diferentes pode facilmente alterar a ordem de valores. No entanto, o custo de adaptação permite verificar se o componente de adaptação mais fácil está entre os recuperados.

Além dos critérios acima apresentados, deve-se ainda levar em conta:

1. *Interface utilizada:* deve ser avaliada quanto ao seu poder descritivo, facilidade de manejo, curva de aprendizagem e adaptabilidade às preferências do utilizador.

2. *Esquema de classificação:* os esquemas de classificação devem ser avaliados quanto à correta descrição dos componentes, objetividade dos critérios de classificação etc.

3. *Processo de recuperação:* deve ser avaliado quanto à facilidade de utilização da linguagem de consulta, flexibilidade de acesso ao repositório, adaptabilidade aos requisitos do utilizador (número máximo de componentes, ordenação, etc), possibilidade de refinar buscas (por iteração ou interação) ou capacidade de combinação de métodos.

Experimentos conduzidos por Frakes e Pole [21] mostraram que não há uma vantagem clara de um método sobre outro, considerando-se a eficácia de recuperação e a precisão na busca do artefato. Também foi determinado que a classificação por palavras-chave é mais barata, pois não requer intervenção humana no processo de indexação de documentos. Sabe-se também que as classificações manuais são mais caras que as automáticas, podem conter erros humanos, mas também oferecem maior riqueza de descrição e redução de problemas de sinônimos [2] e polissemia [3].

[2] palavras utilizadas nas expressões de busca são diferentes das utilizadas na classificação, porém sinônimas
[3] uma mesma palavra tem diferentes significados dependendo do contexto

3.6 Considerações finais

Ao analisar todos os métodos já propostos para classificação e recuperação de componentes, Mili [58] destaca que mesmo após décadas de pesquisa nessa área, o problema continua em aberto. Mesmo com a variedade de soluções criadas, nenhuma delas oferece a combinação certa de eficiência, eficácia, usabilidade e generalidade que aponte uma real otimização em Reutilização de Software.

Existem então lacunas a serem investigadas, principalmente na relação custo *versus* benefício dos esquemas de classificação e sua contribuição para o sucesso da busca do usuário. A proposta deste trabalho é definir um esquema de classificação semi-automática que diminua o custo de classificação (por não ser inteiramente manual), mas que ofereça ao usuário informações em alto nível de abstração necessárias ao entendimento de um componente de software (via classificação manual). Além disso, a classificação será mais abrangente, capaz de representar uma grande variedade de componentes - dos básicos aos de negócios. Também é escopo do trabalho medir a qualidade do método proposto por meio da análise dos resultados obtidos durante a recuperação dos componentes. No próximo capítulo essa proposta é descrita e detalhada.

Capítulo 4

Modelo Proposto de Classificação

4.1 Visão Geral

A evolução do conceito de componentes de McIlroy [54] deu origem a CBSE (*Component-Based Software Engineering*), um conjunto de técnicas e métodos para a construção de software a partir de componentes. Mesmo com uma disciplina dedicada ao estudo do desenvolvimento baseado em componentes, pesquisas recentes [4] revelaram que CBSE não foi amplamente aceita e utilizada. Um dos principais motivos identificados foi a ausência de componentes catalogados, processo que os torna disponíveis para efetivo reuso. Isso leva ao questionamento da real contribuição de algumas ferramentas CASE existentes - como repositórios - na promoção do CBSE [51].

Repositórios são catálogos de componentes organizados de acordo com algum tipo de informação predefinida. Se comparados a uma biblioteca, onde os livros são organizados segundo assunto ou autor, um repositório pode armazenar componentes segundo sua função ou versão. O processo de organização de componentes com o objetivo de recuperá-los para reuso em um determinado contexto é denominado classificação. A classificação é usada para agrupar componentes similares, que compartilham características que os demais grupos não têm [77].

Além de auxiliar a classificação de componentes e armazená-los, os repositórios devem oferecer mecanismos de recuperação, onde o desenvolvedor possa procurar componentes que satisfaçam suas necessidades. Existem algumas barreiras ao bom aproveitamento das informações armazenadas em um repositório, como o aprendizado da linguagem de consulta ou a incapacidade de expressar consultas que recuperem uma quantidade maior de componentes relevantes [83]. Além disso, as informações utilizadas na maioria dos esquemas de classificação podem ser insuficientes para expressar o que o

componente de fato é ou provê, prejudicando o resultado da recuperação.

Analisando os métodos já propostos para classificação e recuperação de componentes, é possível identificar oportunidades de melhoria na classificação, tanto na qualidade das informações utilizadas, quanto no custo do processo. Tais melhorias afetam diretamente os resultados obtidos nas buscas, o que possibilita o aumento da recuperação de componentes relevantes. Essas oportunidades são reflexo da evolução do software, cuja complexidade crescente coloca em xeque antigos modelos de classificação criados fora desse contexto. Exemplos disso podem ser percebidos na variedade de software produzido atualmente, partindo do software básico até os sofisticados serviços de negócio em um domínio específico de aplicação.

O objetivo desse trabalho é apresentar um modelo de classificação de software mais rico em informações e condizente com as atuais tecnologias de componentes. O aumento de informações adequadas melhora a descrição do software, propicia melhores resultados na recuperação e aumenta as chances de reutilização. Além disso, será apresentado um mecanismo para a redução do custo de classificação com o modelo, o que viabiliza sua adoção.

4.2 Um Novo Modelo de Classificação de Componentes

A combinação da abordagem de classificação automática de ROSA (3.1.1) e a amplitude de informação sugerida por Vitharana *et al.* (3.2.1) gera uma alternativa interessante de evolução dos modelos de classificação tradicionais, pois melhora a relação entre quantidade e qualidade das informações sobre o componente sem que para isso, onere o tempo de classificação. Essa alternativa resulta no modelo proposto neste trabalho.

Além da melhor relação custo/benefício prevista, este modelo tem como objetivo manter-se atualizado em relação às tecnologias de componentes que preocupam-se hoje em promover o reuso e a qualidade de software por meio de padronização (ex: especificação de componentes) e popularização (ex: distribuição pela Internet - *Web Services* ou Serviços Web). É importante prover um modelo adaptável aos diferentes padrões, beneficiando diversos tipos de software. Dessa maneira, criam-se condições para que o classificador possa atribuir aos componentes (qualquer que seja o seu grau de formalização) uma correta e precisa descrição de suas características.

4.2.1 Atributos e Facetas

Assim como descrito em [91], o modelo de classificação aqui proposto utiliza atributos (informações estruturadas) e facetas (informações semi-estruturadas). As informações estruturadas servem para reduzir sensivelmente o conjunto de possíveis candidatos durante a recuperação e as semi-estruturadas para refinar esse conjunto, gerando o resultado da consulta. Além disso, as informações estruturadas podem ser selecionadas a partir de uma lista pré-definida de valores, o que facilita o preenchimento dos atributos.

Dentro desse modelo, o processo de classificação ocorre em vários níveis de abstração (negócio, função, qualidade, etc), desde os níveis mais altos (interface), até os níveis mais baixos (métodos, exceções, tipos de dados) da hierarquia de componente. Um conjunto de facetas e atributos serão utilizados para descrever cada um dos níveis, assim como em [91]. Isso permite aprofundar o nível de informação sobre o componente, uma vez que ele passa a ser classificado ao longo da sua hierarquia em vários níveis de abstração e não somente pelo todo.

O modelo possibilita a classificação de um amplo espectro de componentes, primando pela ortogonalidade das dimensões escolhidas - garantia de complementaridade entre os atributos e facetas e a não repetição de informações. Ele permite descrever software sob diversos pontos de vista (tanto técnico, quanto negocial), incorporando atributos e facetas sobre qualidade (maturidade e métricas), além de testes (programas e documentação). Para o reutilizador, essa descrição melhora seu poder de busca e seleção, uma vez que encontra informações mais detalhadas sobre diversos aspectos dos componentes, inclusive sobre seu grau de qualidade (confiabilidade e estabilidade, por exemplo).

O reutilizador pode também querer estender as funcionalidades de um componente que atenda parcialmente suas necessidades. Para isso, um atributo sobre acesso ao código-fonte foi incluído informando se existe ou não acesso ao código-fonte. Isso pode determinar o interesse no reuso do componente, uma vez que certas modificações necessitam desse acesso.

O atributo de versão foi refinado, incluindo agora a data da versão. O principal motivador vem da crescente necessidade de gestão de configuração de software, sendo que alguns repositórios já se integram com ferramentas de controle de versão [19]. Apesar da grande importância das ferramentas de gestão de configuração para a promoção do CBSE, não faz parte do escopo do modelo tratar essa integração.

O trabalho de Vitharana *et al* [91] sugere a faceta "Sinônimo"para atribuir os possíveis sinônimos

de um componente. Ela foi excluída do modelo proposto por ser substituível por um tesauro (sistema que guarda o significado dos termos de uma língua e o relacionamento semântico entre eles, como o Wordnet [27]), que indica sinônimos automaticamente. Dessa forma, os sinônimos já estão armazenados no tesauro e não precisam ser duplicados em uma faceta. A seção 4.3 contextualiza a utilização desse tesauro.

1. **Atributos**

   - *Identification (ID, name):* o ID é utilizado para garantir a unicidade do componente no repositório. Nome é um identificador aceito universalmente [91] e representa uma abstração curta sobre o componente. Um componente para agências de viagem que reservam passagens aéreas poderia chamar-se "Agência de Viagem", por exemplo.
   - *Management (owner, contact, version, version date):* identifica o responsável pelo desenvolvimento do componente, versão e forma de contato. O usuário, por exemplo, pode precisar de suporte, relatar erros ou sugerir melhorias. De acordo com [51], essas informações são úteis em um mercado de componentes competitivo.
   - *Environment (Operation System, plattform, programming language/tools):* informações sobre o ambiente são importantes para descrever aspectos não-funcionais, como portabilidade ou restrições de compilação.
   - *Source Code Access:* Indica se o código-fonte do componente classificado será disponibilizado. Esta informação é útil para a determinação do tipo de reuso que se pretende fazer (extensão, modificação, composição).
   - *Maturity (age, number of reuses):* A idade do componente e o número de reusos dão a noção de estabilidade ou maturidade do componente. Muitas vezes certos riscos de integração não podem ser assumidos.

2. **Facetas**

   - *Application Domain:* possíveis contextos de utilização/integração de um componente. Vitharana [91] considera o domínio de aplicação uma informação bem definida ou estruturada (um atributo dentro do modelo de classificação). Exemplos disso seriam os domínios "Financeiro"ou "Médico". Infelizmente, o termo é sobrecarregado de diferentes sentidos e interpretações, tornando difícil estabelecer uma terminologia que possibilite referir-se

aos domínios sem incorrência de erros. Dentre possíveis soluções para o problema, foram propostas taxonomias que classificam os domínios de aplicação, o que permite padronizar termos e estruturar os trabalhos de pesquisa e troca de informações sobre domínios particulares de aplicação [38]. Enquanto tais taxonomias são desenvolvidas, este modelo considera a descrição do domínio uma informação semi-estruturada (ou faceta), na qual o classificador tem a liberdade de determinar sua própria taxonomia. Ele poderá designar à faceta palavras simples ou estruturas mais complexas de classificação, como um XML (*Extensible Markup Language*) que determina a hierarquia que descreve o domínio em sub-domínios. O usuário pode preferir basear-se em taxonomias mais conhecidas, como NAICS [94], sistema de classificação norte-americano usado para classificar e mensurar sua atividade econômica.

- *Role:* papéis que o componente pode (potencialmente) assumir uma variedade de aplicações. O componente de Agência de Viagem poderia tanto atuar dentro de um sistema de reservas de passagens áreas, quanto em um de bilhetes de trens.
- *Rule:* Processos de negócio ou funções em um domínio particular que o componente oferece suporte. O usuário pode comparar as regras de negócio de sua aplicação com as do componente recuperado, assegurando se suas regras estão ou não cobertas.
- *Function:* Funções ou tarefas executadas pelo componente e seus métodos.
- *Element:* Representam parte do domínio de aplicação, sendo objetos manipulados nele. Um componente de software básico que manipula arquivos do sistema tem como elemento o próprio arquivo. Da mesma forma, um componente de reservas de passagem aérea manipula o elemento "bilhetes de passagem". Variáveis de um programa podem informar tais objetos.
- *Action:* Ações e eventos que o componente provê.
- *User:* Corresponde à faceta "Meio"de [68], que descreve onde a ação é executada. No contexto de negócio, o Usuário pode ser caracterizado como o usuário final de uma aplicação baseada em componentes.
- *Test:* Indica a presença de testes para componente (unitários ou de integração) por meio de um mecanismo de inclusão. Podem ser anexados documentos (casos de teste, descrição do ambiente, etc) e o próprio código-fonte.
- *Quality:* Como mecanismo de garantia/medição da qualidade do componente, essa faceta

permite a descrição de métricas de qualidade de software que estejam disponíveis. As características medidas podem ser as definidas pela norma ISO/IEC 9126, por exemplo.

O modelo possibilita tanto a classificação de componentes simples, quanto os de negócio, bastando preencher ou não certas informações nos atributos e facetas. Um exemplo de classificação do componente *Cinema* segundo o modelo proposto é dado nas Tabelas 4.1 e 4.2.

Atributo	**Descrição**
Identification (id,name)	21, Cinema
Management (owner, contact, version, version date))	Xpto Desenvolvimento de Software; contato@xpto.com.br; 1.3; 20/02/04
Environment (Operation System, plattform, programming language, tools)	Windows, Linux, UNIX, Mac OS; J2EE; Java; Eclipe 2.0
Source Code Access	Sim
Maturity	3 anos, 4 vezes

Tabela 4.1: Exemplo de preenchimento dos Atributos do componente Cinema seguindo o modelo proposto

Faceta	**Descrição**
Application Domain	Gestão de Venda de Ingressos.
Role	venda de ingressos em um sistema de gerenciamento de Cinema
Business Rules	ingressos podem ser vendidos antecipadamente
Function/task	precificação, geração de ingressos
Element/part	ingresso
Action/event	reserva de ingresso, compra de ingresso
User	cliente, atendente
Test	*link* para Testes Unitários do Componente e para documento que descreve os casos de teste cobertos
Quality	Disponibilidade = 99 Manutenibilidade (MI/SEI) = 65

Tabela 4.2: Exemplo de preenchimento das Facetas do componente Cinema seguindo o modelo proposto

Em relação ao trabalho de Vitharana *et al* [91], este modelo agrega informações importantes para qualquer ambiente de desenvolvimento baseado em componentes, tais como métricas de qualidade do software e plano de testes (documentação e programas). O reuso torna-se mais eficiente à medida que são disponibilizados meios para o desenvolvedor integrar componentes já existentes dentro de um

sistema maior. Os testes facilitam o processo de integração (com ou sem modificação do componente reutilizado)e as métricas auxiliam a mensuração da qualidade final do produto. Esta é uma das principais contribuições deste trabalho. Outros atributos foram incluídos (como data de versão e acesso ao código-fonte) para auxiliar a filtragem durante as futuras buscas, ou excluídos (faceta *Synonym*) por serem substituíveis por outros mecanismos.

Na criação de um modelo de classificação deve-se primar pelo custo/benefício de cada atributo ou faceta adicionado. Em relação ao trabalho de [91], este modelo acrescentou novos itens, o que acarreta maior tempo (custo) de classificação. Porém, além de manter a ortogonalidade (não sobreposição de informações entre as facetas e atributos), o modelo mantém-se atualizado com as tendências da Engenharia de Software que primam cada vez mais pelos métodos e tecnologias que dão suporte à qualidade. Ainda no contexto de custo de classificação, este trabalho propõe (Seção 4.3) um método de preenchimento automático de algumas facetas, diminuindo muito o tempo necessário para a catalogação dos componentes.

4.3 Método para preenchimento semi-automático de atributos e facetas

Apesar da grande aceitação das Facetas como modelo de classificação, em bibliotecas maiores o custo de um processo totalmente manual pode tornar-se proibitivo. Para viabilizar a prática do modelo é necessário criar um método que automatize o preenchimento de facetas, diminuindo o tempo de classificação. Vale ressaltar que deve-se primar pela qualidade das informações obtidas automaticamente sobre o componente.

O primeiro passo é identificar as facetas do modelo que podem ser preenchidas de maneira automatizada, baseado no estudo dos dados disponíveis sobre o componente. O código-fonte disponibiliza informação de um nível de abstração baixo, o que impede a extração de conceitos significativamente abstratos [96]. A documentação, interna ao código-fonte ou não, melhora o processo de entendimento de um programa. Certos aspectos da documentação referem-se ao domínio da aplicação, já outros, à linguagem de programação. Boas documentações são necessárias não somente para o bom entendimento do programa, mas também como fonte de informação para classificação de software em níveis mais altos de abstração.

As principais informações encontradas nos componentes estão na sua documentação e no código-fonte. No presente trabalho, o nosso principal objetivo é identificar quais dos elementos de classificação, sejam eles atributos ou facetas, podem ser automaticamente detectados a partir da informação

existente na maioria dos componentes disponíveis:

Código-fonte: Inoue *et al* [43] classificam um componente unicamente por meio de uma análise estática que extrai as relações de uso descritas no código-fonte. Em um componente orientado a objetos, informações como herança, associação e implementação podem ser extraídas. Elas seriam úteis para o preenchimento da faceta Elemento/Parte, uma vez que estabelecem a noção de relacionamento entre os componentes classificados, além do sujeito e objeto desse relacionamento. A Figura 4.1 mostra o exemplo do Componente *Cinema*, onde método *checkIfOpen* (da Interface *FrontOffice*) usa a classe *Date* para definir se determinada data já foi liberada para venda de tickets. Nesse momento, o elemento *Date* é classificado como parte, ou seja, objeto manipulado nesse sub-domínio. O preenchimento do atributo (*source code access*) é uma conseqüência da análise do código-fonte.

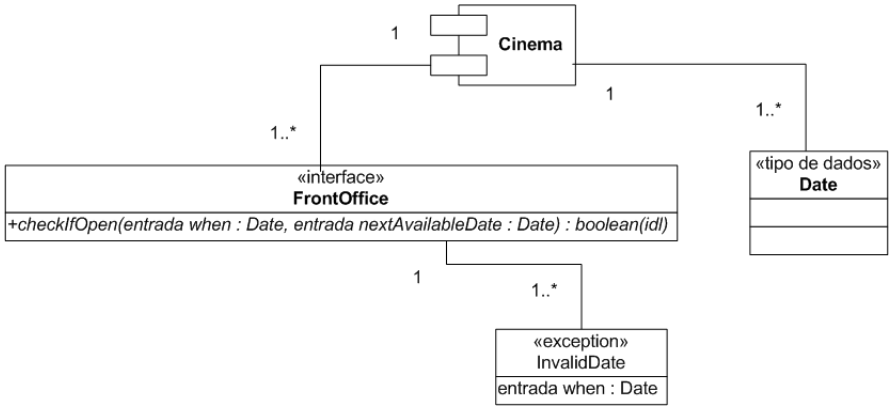

Figura 4.1: Exemplo de hierarquia do Componente Cinema.

Documentação: As ações e a funcionalidade do componente podem ser obtidas da documentação, principalmente se estiverem presentes no código-fonte, pois estão fisicamente próximos ao trecho de código a ser classificado (documentação das interfaces, métodos ou atributos, por exemplo). As orações contidas na documentação, os identificadores de atributos, métodos, classes, etc podem ser extraídas e tratadas de maneira a encontrar as palavras-chave que definem as funções e ações do componente.

O tratamento de funcionalidade, por exemplo, é feito com a análise sintática das orações da documentação referente a uma classe, método, etc; identificação de sujeitos e predicados; e

atribuição dos verbos e objetos (diretos e indiretos) à faceta Função. Essa técnica é derivada da área de Processamento de Linguagem Natural e já foi utilizada em outros sistemas de apoio ao reuso, como no já discutido ROSA 3.1.1 e em [26]. Os atributos de gerenciamento (*owner*, *version* e *version date*) também podem ser extraídos, bastando obedecer algum tipo de padronização básica (como a que existe no Javadoc [84]).

Apesar da documentação de alguns componentes conter informações sobre projeto, não existe um padrão específico que poderia ser identificado automaticamente. Sendo assim, este trabalho está concentrado na informação extraída da documentação do código-fonte.

A Figura 4.2 ilustra a relação entre os níveis de abstração do componente (discutidos na Seção 3.2.1) e quais informações podem ser obtidas deles automaticamente para o preenchimento das facetas.

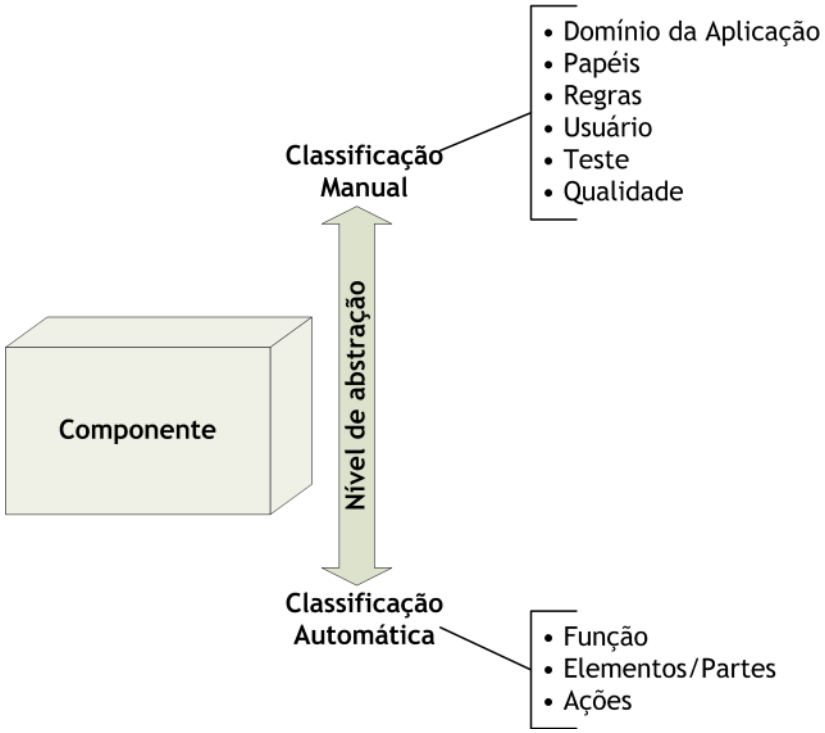

Figura 4.2: Abstração do componente *versus* Preenchimento automático de facetas.

Determinar sinônimos é uma técnica útil para expansão (ou relaxamento) das consultas de usuário,

recuperando-se também componentes com descrições similares às pesquisadas. Considerando a dificuldade dos usuários em formular consultas e a natural distância conceitual entre tais consultas e a classificação armazenada, a expansão é uma técnica fundamental [42]. Para suprir essa necessidade sem aumentar os custos de classificação, um tesauro como o Wordnet [27] pode ser utilizado. A cada busca, as palavras-chave informadas pelo usuário são tratadas a fim de obter seu radical (resolvendo assim os problemas de gênero, plural e tempo verbal). É possível identificar por meio do tesauro as relações desse radical com outros termos. Neste caso específico, o interesse está na relação de sinonímia e hiperonímia (*é-um-tipo-de*) entre eles. É possível, também, determinar o quão forte deve ser essa relação, de forma a configurar o nível de expansão da consulta desejado. Os termos sinônimos devem ser pesquisados no repositório de componentes catalogados como complemento à consulta original. O relaxamento pode ser automático ou manual, dependendo do interesse do usuário.

A Figura 4.3 sumariza as informações que podem ser obtidas automaticamente da documentação ou do código-fonte para o preenchimento das facetas e atributos do modelo.

		Documentação	**Código-fonte**
Facetas	Application Domain		
	Role		
	Rule		
	Function	X	
	Element		X
	Action	X	
	User		
	Test		
	Quality		
Atributos	Identification		
	Management	X	
	Environment		
	Source Code Access		X
	Maturity		

Figura 4.3: Facetas e atributos que podem ser preenchidos automaticamente por origem

4.3.1 Armazenamento das informações

Além de identificar os elementos dos atributos e das facetas que podem ser preenchidos automaticamente a partir de informação existente nos componentes, é preciso que tal informação esteja armazenada de forma a se ter uma recuperação eficiente dos componentes. De acordo com [91], uma linguagem de marcação como XML [9] é apropriada para codificar conhecimento baseado em facetas, pois possibilita a manutenção de esquemas para codificar uma base semi-estruturada de conhecimento do componente e dar suporte à busca de componentes.

O CKBR (*Component Knowlegde-Based Repository*), repositório proposto em [91], combina consulta estruturada por meio de banco de dados com a recuperação de conhecimento semi-estruturado codificado em XML. As palavras-chave especificadas pelo usuário são comparadas com informações do banco de dados e dos arquivos XML. Os motivos para isso são:

- desempenho de busca dos dados estruturados - quando colocado em um sistema gerenciador de banco de dados podem ser recuperados rapidamente;
- manutenibilidade das facetas - é característica da classificação facetada a possibilidade de acréscimo de novas facetas ao modelo, o que XML oferece suporte naturalmente. Basta usar um modelo de estrutura como um DTD (*document type definition*) para reformatar os arquivos XML e possibilitar a alteração da informação semi-estruturada.

Vitharana et al construíram o protótipo do CKBR e um estudo empírico foi realizado para validar a efetividade do seu modelo. O resultado foi positivo, garantindo a utilidade do repositório proposto para o armazenamento e busca de componentes. Essa conclusão aponta que a abordagem baseada em XML e um sistema de banco de dados pode ser uma boa opção para o armazenamento das informações que seguirão o modelo aqui proposto.

4.4 Considerações finais

Neste capítulo foi proposto um novo modelo de classificação de componentes que considera não só aspectos técnicos, mas também negociais e de qualidade. A aplicabilidade do modelo é alta se considerada a realidade do atual mercado de componentes de software, no qual as empresas buscam fornecedores de serviços que atendam suas necessidades negociais. Além disso, característica desse quadro é a alta competitividade entre as empresas de software, uma vez que a distribuição de serviços

pela Internet tornou-se popular com os *Web Services*. Por esse motivo, é importante também incluir aspectos mais formais sobre a qualidade do componente - um diferencial competitivo.

As principais vantagens do modelo são a inclusão de facetas de qualidade e testes, refinamento de atributos que facilitam o gerenciamento (*Management* e a identificação do tipo de reuso *Source Code Access*), como extensão, modificação ou composição de um componente. Um método de preenchimento automático das facetas foi proposto, o que reduz efetivamente os custos da classificação e viabiliza a construção de repositórios bem organizados. Em suma, aspectos gerenciais e técnicos estão cobertos nesta proposta, o que a torna condizente com a variedade de software existente em um mercado exigente.

No capítulo que se segue será apresentado o protótipo de sistema construído para validar o modelo e o método aqui propostos. Serão também demonstrados os resultados da avaliação inicial baseada em aspectos de recuperação de informação e usabilidade da ferramenta.

Capítulo 5

O Sistema REUSE+

No capítulo anterior foi apresentado um novo modelo de classificação de componentes e um método para preenchimento semi-automático das facetas e atributos a partir de documentação e do código-fonte. Neste capítulo será apresentado o protótipo de sistema (o REUSE+) desenvolvido para classificar e recuperar componentes utilizando o modelo e método propostos. Serão estudados os mecanismos de classificação, armazenamento e recuperação, bem como a arquitetura do sistema e um estudo de caso. A execução do REUSE+ servirá de base para a avaliação da viabilidade do modelo proposto em relação às demais técnicas revisadas neste trabalho, demonstrando seu diferencial.

5.1 Visão Geral

Como visto anteriormente, classificar componentes de software com informações mais adequadas melhora a descrição, propicia melhores resultados na recuperação e aumenta as chances de reutilização. Utilizando o novo modelo de classificação proposto no Capítulo 4 é possível descrever componentes de maneira mais abrangente, pois, além da hierarquia, suas características de negócio e de qualidade são consideradas. Para reduzir o ônus que o tempo de classificação causa, o método de classificação semi-automática também deve ser utilizado, de forma a reduzir o custo final da criação de um repositório de software. Neste contexto, é objetivo deste trabalho experimentar as idéias propostas por meio da construção de um protótipo de sistema. Com essa finalidade, o REUSE+ foi criado e populado com componentes de diversos domínios de aplicação.

Resumidamente, O REUSE+ faz uma análise de informações do código fonte e sua documentação e gera automaticamente uma representação do conhecimento extraído. Adicionalmente, o usuário descreve o domínio da aplicação em classificação utilizando um conjunto de palavras-chave. As

informações obtidas sobre o componente serão armazenadas em uma base de conhecimento. A busca é assistida pela ferramenta e as consultas são descritas em um vocabulário não controlado. Um tesauro foi utilizado para a expansão das consultas, melhorando a efetividade da recuperação dos componentes. A Figura 5.1 ilustra o funcionamento do protótipo construído.

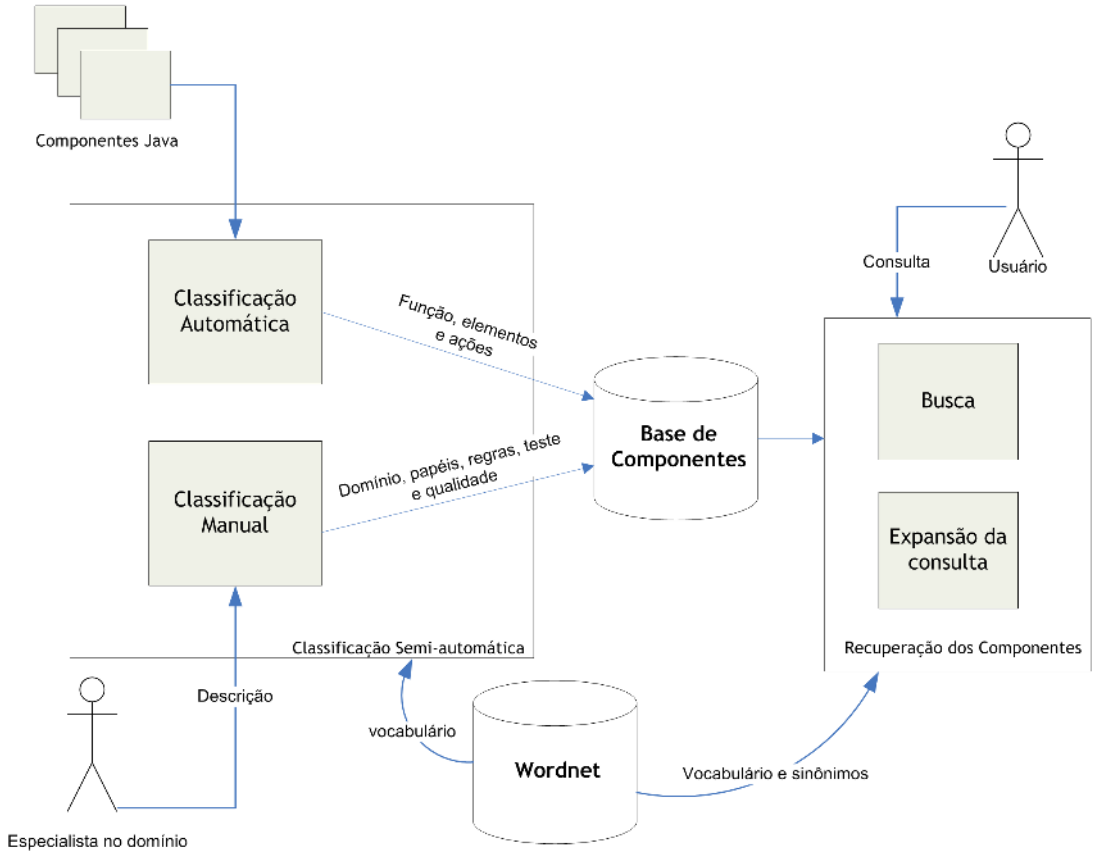

Figura 5.1: Funcionamento do REUSE+

O escopo do protótipo foi limitado a componentes de software orientado a objetos escritos em Java com documentação em inglês. Java é uma linguagem orientada a objetos de grande popularidade e com uma gramática relativamente simples, o que facilita a criação de uma ferramenta de maior alcance - um dos objetivos deste trabalho. O inglês é uma das línguas mais faladas no mundo, e firma-se cada vez mais como padrão na documentação de projetos de software, principalmente em

instituições multinacionais. Além disso, a maior parte das boas ferramentas de processamento de linguagem natural foram desenvolvidas para essa língua, contribuindo para a sua escolha.

No REUSE+, as informações extraídas do componente são indexadas seguindo o modelo dos sistemas de Recuperação de Informação (Seção 3.4.1) e associadas às facetas correspondentes. A escolha dessa técnica está ligada à natureza das informações extraídas, ou seja, os comentários específicos do javadoc existente no componente.

Javadoc é uma ferramenta que faz parte do kit de desenvolvimento para Java [84]. Ela possibilita a criação de documentação de pacotes, classes, atributos e métodos Java a partir do processamento do código fonte com comentários em um formato pré-definido. Além do formato, boas práticas de documentação foram sugeridas pela Sun (criadora da tecnologia) e são seguidas de maneira geral pelos desenvolvedores. Esse conjunto de diretivas guiam o processo de extração de informação do código fonte, uma vez que a forma de documentar não é totalmente livre. A extração de informação de código-fonte Java já foi objetivo de pesquisa em alguns trabalhos [43, 80, 92], porém nenhum deles beneficia-se da semântica oferecida pelo javadoc. A técnica utilizada para extrair essas informações é detalhada na Seção 5.2.2.

Para efeito de avaliação do modelo proposto, assumiu-se que os componentes de software são classes Java, não sendo verificadas nenhuma das condições necessárias a um componente como a existência de interfaces providas e requeridas. Isso não limita o poder de classificação de componentes formais do REUSE+, apenas simplifica a experimentação da ferramenta. Além disso, não existe ainda um consenso sobre métodos e técnicas de projeto de componentes, o que reforça pesquisas que retratam coerentemente o cenário atual do desenvolvimento baseado em componentes [90]. O experimento se propõe, portanto, a analisar a classificação de software sem rigores de componente, o que reflete a realidade da maioria das instituições desenvolvedoras de sistemas.

Seguindo o método proposto no Capítulo 4, foi implementado um mecanismo que preenche automaticamente as facetas *Function*, *Element/Part* e *Action*. Para preencher a faceta *Function* é feito um parse do código-fonte para retirar-se os comentários, que por sua vez são indexados pelo Apache Lucene [29]. Nessa indexação (ver Seção 3.4.1), os termos mais relevantes são descobertos e, então, usados para preencher a faceta *Function*. O mesmo parse de código-fonte extrai informações sobre hierarquia, relacionamentos e parâmetros (e sua documentação) para preencher a faceta *Element/Part*. Por último, a faceta *Action* é preenchida com os nomes gerados na análise dos identificadores encontrados no fonte.

Assim, parte da representação da informação sobre o componente foi gerada automaticamente e pode ser armazenada no repositório do REUSE+. A Figura 5.2 ilustra o processo automático.

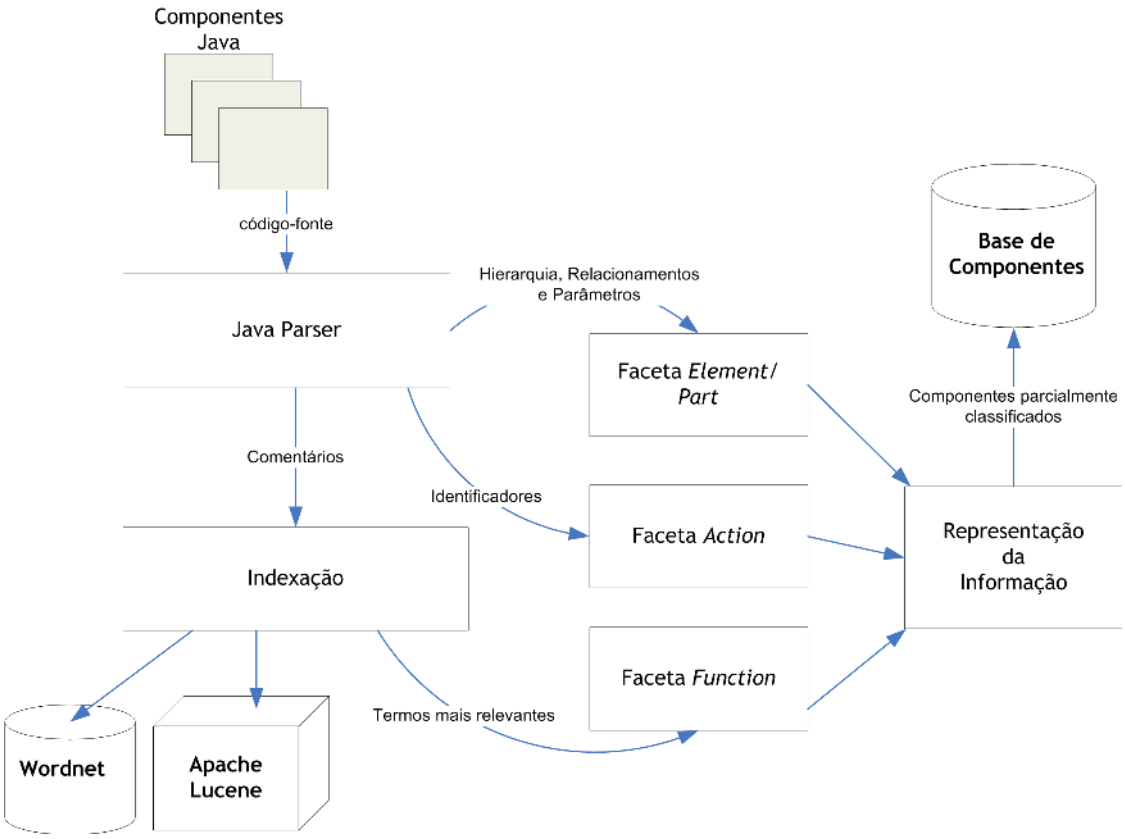

Figura 5.2: Processo de classificação automática.

Ao longo desse Capítulo serão apresentados detalhadamente os mecanismos que compõem o REUSE+ de forma que ele possa experimentar o modelo e o método de classificação propostos anteriormente.

5.2 Mecanismos de classificação

O modelo proposto na Seção 4.2 baseia-se no uso de facetas e atributos para descrever componentes de software em diversos níveis de abstração, aumentando suas chances de reuso. Na Seção 4.3 foi proposto um método de preenchimento semi-automático que diminui os custos de classificação de

componentes utilizando esse modelo. O grande desafio reside na automatização do processo, uma vez que a classificação manual e suas implicações (armazenamento e recuperação) já foram objeto de estudos passados [64] [57] [82] [66] e tornaram-se bem compreendidos.

Como as principais informações encontradas nos componentes estão na sua documentação e no código-fonte, um mecanismo de automatização deve estudar as possibilidades de extração de informação útil para preencher a descrição do software. Esse trabalho já foi mencionado na Seção 4.3, que descreveu as principais origens de informação sobre o qual um mecanismo de extração deve agir. Sua implementação será apresentada nas próximas seções, levando em conta os seguintes passos de classificação:

1. *Análise Estática:* fase que identifica a hierarquia de classes, seleciona uma classe para classificação e extrai informações para o preenchimento da faceta *Element/Part*;
2. *Parser de Comentários do Código-Fonte:* fase que extrai os comentários e identificadores do código-fonte para futura associação com as Facetas *Function* e *Action*;
3. *Remoção de Stopwords, Stemming e Indexação:* responsável por remover termos de pouco valor semântico, encontrar os radicais dos termos e analisá-los estatisticamente de forma a identificar bons discriminantes para Faceta *Function*.

As próximas seções detalham cada um desses passos, explicando, em detalhe, o processo de classificação semi-automática.

5.2.1 Análise Estática

A análise estática permite extrair as informações mais básicas de um sistema, incluindo seus componentes básicos, relações de definição e de referência. Ferramentas para fazer essa análise são chamadas de "*parsers*". Essa técnica é muito usada em compiladores para detectar problemas na sintaxe da linguagem. Esse tipo de "*parser*" pode ser definido usando ferramentas como Lex e Yacc [53]. Elas geram árvores de sintaxe abstrata (*AST - Abstract Syntax Tree*), que representam o código-fonte em forma de grafo.

Para necessidades mais específicas, pode-se desenvolver *parsers* que extraem um subconjunto da AST de programa. O modelo implementado pelo REUSE+ definiu que informações seriam interessantes para a classificação de um componente. Entre elas estariam o nome da classe, sua hierarquia

de classes e relacionamentos, além dos métodos e atributos. Essas informações são úteis não apenas para a classificação, mas também para a gestão de ligações, responsável por gerenciar esses relacionamentos.

A informação sobre a hierarquia das classes auxilia na detecção de responsabilidades, dependências e tipos de dados necessários para o uso de certo componente de software. A partir dessa informação é possível recuperar componentes sem correr o risco de erros de compilação, ou ainda pior, de execução por falta das suas dependências.

O processo de análise estática tem como objetivos: identificar a hierarquia de classes, selecionar uma classe da hierarquia, extrair informações sobre a mesma e criar uma rede de relacionamentos entre ela e as demais representações de informação existentes. Uma representação de informação nada mais é que a abstração obtida pela classificação de um componente. Nessa fase, é obtida parte da representação de informação, armazenada em uma estrutura com os seguintes itens:

1. Primeiro campo: nome da classe declarada no programa;
2. Segundo campo: ascendência, ou seja, a superclasse da classe em questão;
3. Terceiro campo: os métodos definidos nesta classe;
4. Quarto campo: identifica o tipo desses métodos (parâmetros de entrada e saída);
5. Quinto campo: relacionamentos do tipo "usa"da classe.

Dentro do modelo de classificação proposto, o segundo e o quinto campos serão utilizados para preencher a faceta *Element/Part*. Os demais campos são para a gerência de ligações e para a construção dos resultados de consultas.

5.2.2 *Parser* de Comentários

Estudos sobre extração de comentários e identificadores de código-fonte foram realizados em [25] e algumas conclusões importantes foram apontadas. Uma delas é de que os comentários feitos dentro do código-fonte podem ser considerados uma sub-linguagem, um subconjunto da linguagem natural com seus termos e gramática. Em virtude do reduzido tamanho desse subconjunto, torna-se viável realizar parse de comentários em busca de informação relevante. Além disso, nessa pesquisa demonstrou que 83% dos comentários escritos em forma de sentença estavam no tempo verbal presente. O mesmo

trabalho levantou as freqüências de tipos sintáticos em identificadores (variáveis e funções), chegando à conclusão de que a maior parte deles estão relacionados a verbos. Essas informações encorajam a técnica de extração de informações de comentários dos componentes e auxiliam na montagem da estratégia de uso das mesmas.

Outro fator importante para o *parser* de comentários é que existe uma padronização de escrita imposta pelo javadoc, seja pela ordem das descrições, seja pelo uso de *tags* específicas. Isso facilita o cruzamento entre trecho de código e comentário extraído, pois, em princípio, a documentação feita em código-fonte não segue padrão algum. As informações extraídas pelo *parser* são utilizados, posteriormente, durante a fase de análise sintática e indexação.

Por enquanto, interessa extrair todos os comentários que possam indicar as funcionalidades e ações do componente. As *tags* do javadoc que descrevem a funcionalidade de classes e métodos e os próprios identificadores foram usados como fonte desse tipo de informação. Além disso, os atributos de gerenciamento (*owner*, *version* e *version date*) também podem ser extraídos pelo *parser* do javadoc.

Dentre as *tags* existentes, foram analisadas: *@param* (parâmetros de entrada), *@return* (parâmetros de saída), @author (autor do código) e *@version* (versão do software que pode ou não estar acompanhada de data). As funcionalidades de interfaces, classes, métodos e construtores são escritas *antes* das *tags*. Os identificadores, conforme [25], são descritos em grande parte por verbos e seus objetos (adjetivos + nomes), o que traz pistas sobre as ações que o componente provê. Para tirar proveito disso, é necessário quebrar o nome do identificador para extrair palavras que façam sentido para a linguagem natural, uma vez que elas unidas formam o nome do identificador.

A Figura 5.3 resume o processo, mostrando um exemplo de extração de comentários e identificadores que apontam as funções, elementos e ações do componente Cinema.

A primeira seção de comentários javadoc é destinada ao registro do propósito da interface/classe/método que está sendo declarado. No exemplo dado, tem-se a declaração do propósito do método `reserveSeats`, responsável por reservar assentos em um teatro. Essa descrição é extraída para posterior indexação (Seção 5.2.3), onde apenas os termos relevantes serão selecionados para compor a Faceta *Function*. Os parâmetros de entrada e saída são considerados elementos do método e são manipulados por ele. No exemplo dado, o parâmetro de entrada `seat` é extraído como elemento manipulado e é usado para descrever a Faceta *Element/Part*. Por último, o próprio nome do método (`reserveSeats`) é extraído para compor a Faceta *Action*, que representa ações relacionadas ao componente. Antes dessa

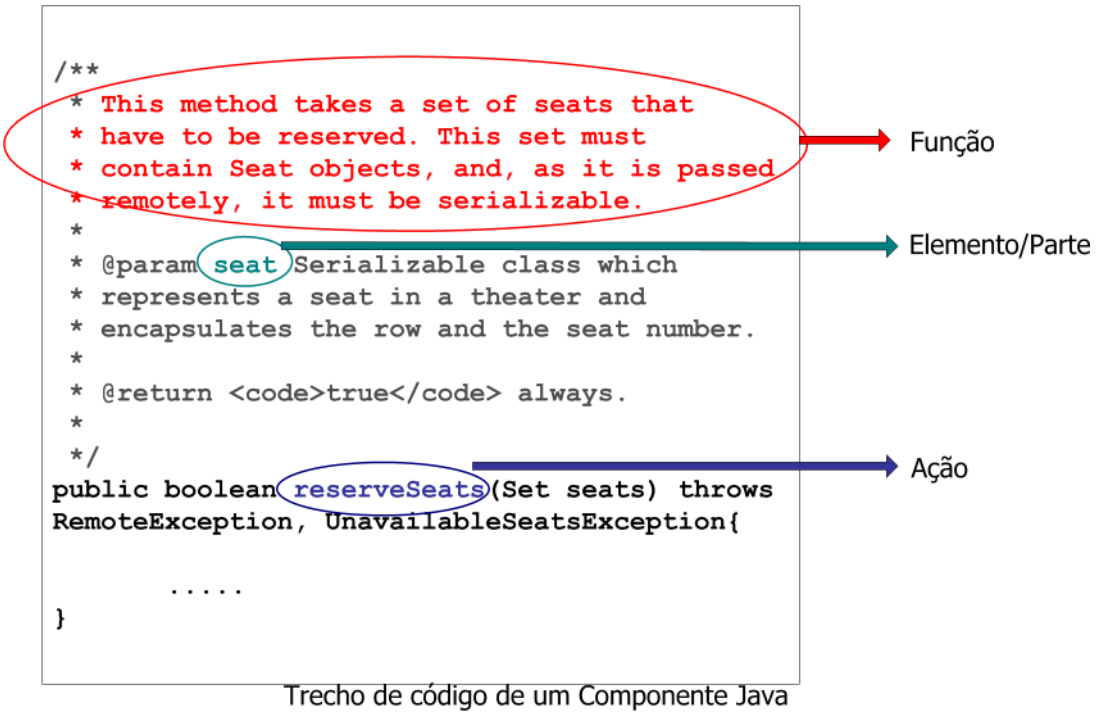

Figura 5.3: Parse do Código-Fonte: extração de informações para as Facetas *Function*, *Element/Part* e *Action*

associação, o identificador é quebrado para que faça sentido em linguagem natural, o que origina a ação *"reserve seats"*.

É importante lembrar que, devido à característica hierárquica do mecanismo de classificação, para chegar-se ao nível de classificação de método é necessário que os níveis superiores já tenham sido classificados. Sendo assim, para o exemplo do Componente Cinema, tem-se que a classe detentora do método `reserveSeats`, assim como um possível pai na hierarquia de classes, já foram classificados e estão armazenados no repositório de componentes.

5.2.3 Remoção de *Stopwords*, *Stemming* e Indexação

O processo de indexação é realizado por meio da ferramenta Apache Lucene para Java [29], uma biblioteca de acesso público. Ela oferece mecanismos de Indexação, Busca, Armazenamento de documentos, inclusive implementando algumas otimizações como compressão de índice e busca

disjuntiva [15].

Dentre os modelos de Recuperação de Informação vistos na Seção 3.4.1, o Lucene implementa o Modelo Vetorial 3.4.1, possibilitando a ordenação dos resultados de uma consulta pelo grau de similaridade. Durante o processo de classificação, o indexador do Lucene é utilizado para calcular o *"inverse document frequency (IDF)"*, que permite concluir: (1) quanto mais vezes um termo aparece em um documento, mais relevante ele é no contexto do documento e (2) quanto mais vezes um termo aparece no conjunto de todos os documentos, menos ele é capaz de discriminar os documentos desse conjunto.

Sendo assim, todos os termos retirados da documentação do componente ou provenientes da classificação manual passam pelo processo de remoção de *stopwords*, *stemming* e indexação. Portanto, as facetas *Application Domain*, *Role*, *Rule*, *Function*, *Element/Part*, *Action*, *User* e *Quality* são analisadas pelo Indexador. Somente a faceta *Test* não é analisada, pois seu conteúdo é o produto da inclusão de artefatos de teste.

1. Remoção de palavras com menor valor de busca (lista de parada ou *stopwords*). Essa lista para palavras da língua inglesa já está disponível [81];

2. Remoção de afixos (*stemming*) das palavras restantes - algoritmo já está disponível [81];

5.3 Mecanismos de Armazenamento

Conforme já descrito, o repositório do REUSE+ armazena o conhecimento obtido sobre os componentes durante a fase de classificação em banco de dados e XML. O conhecimento estruturado, obtido por meio dos atributos, foi armazenado em um banco de dados relacional por meio do gerenciador MySQL [2]. As informações semi-estruturadas, no caso contidas nas facetas, foram codificadas em XML. Essa estrutura de armazenamento favorece o desempenho de busca dos dados sem prejudicar a manutenibilidade das facetas, uma vez que o XML oferece suporte natural a esse tipo de tarefa.

As questões relacionadas ao desenvolvimento de um repositório de software útil e efetivo estão distribuídas em uma série de áreas da tecnologia [79]. Para a simplificação do REUSE+, o repositório é centralizado, o que facilita o gerenciamento da consistência dos componentes classificados. A Figura 5.4 apresenta uma visão geral da arquitetura do sistema com repositório centralizado, muito

semelhante a outros repositórios de software conhecidos, como o SVN [1] e o CVS [2].

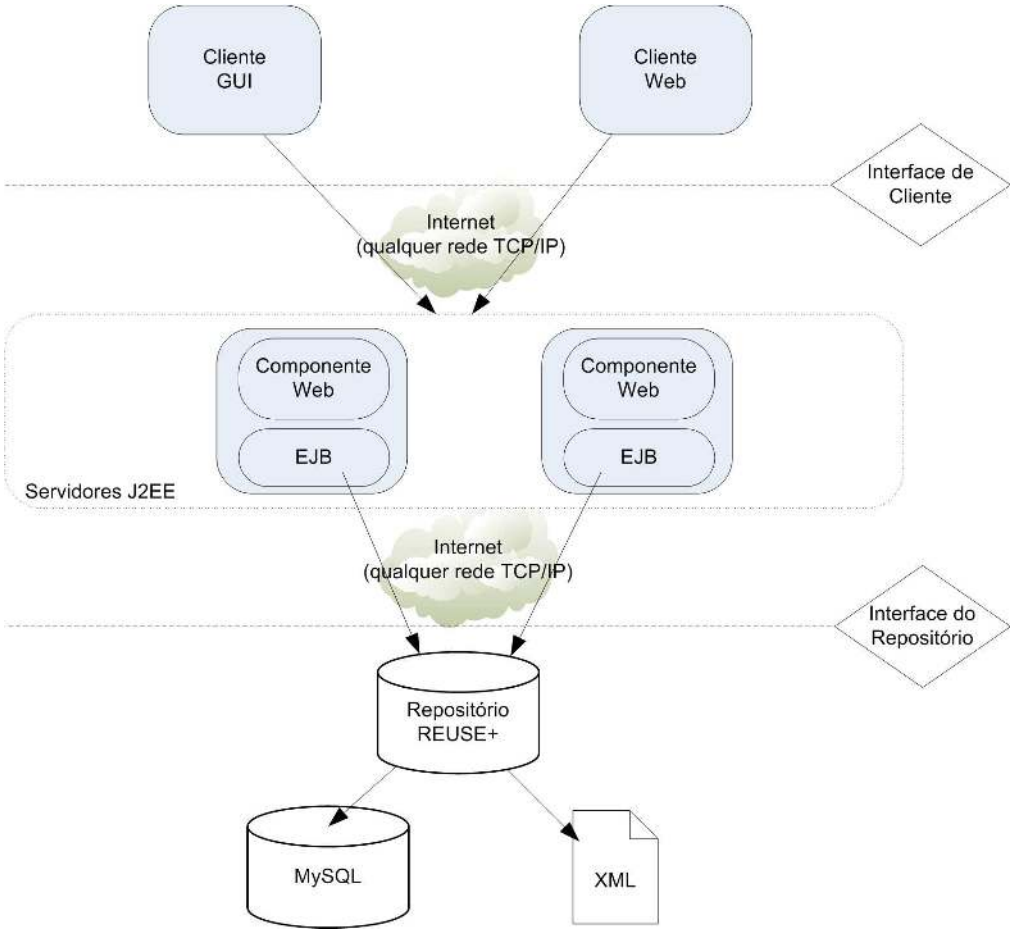

Figura 5.4: Visão lógica do sistema REUSE+

Embora essa solução não seja a ideal, uma vez que deseja-se alta escalabilidade, parte do processamento é realizado na camada de negócio, essa sim totalmente distribuída e escalável. É uma solução intermediária que pode ser evoluída quando o repositório crescer demais.

[1] http://subversion.tigris.org
[2] http://www.nongnu.org/cvs

5.4 Mecanismos de Busca e Recuperação

Durante o processo de recuperação, o sistema guia o usuário no correto preenchimento dos campos de busca, que correspondem a cada faceta existente no sistema. O usuário é instruído a realizar consultas com palavras-chave, mas caso ele coloque uma oração inteira, o mecanismo de recuperação retira as palavras menos importantes (conforme visto na seção anterior).

O processo de recuperação dos componentes é baseado nos sistemas de Recuperação de Informação (Seção 3.4.1), onde a consulta do usuário é processada e os documentos relevantes são apresentados ordenadamente. Nessa abordagem, o usuário cria sua expressão de consulta por meio de palavras-chave associadas a cada faceta disponível no sistema.

Para calcular a relevância dos documentos (componentes classificados) em relação à consulta realizada são utilizados os mesmos mecanismos utilizados na indexação, mais os seguintes passos:

1. Cálculo da similaridade (Modelo Vetorial) entre a consulta e cada componente classificado (soma dos termos das facetas), obtendo assim a relevância do mesmo em relação à consulta;
2. Ordenação dos resultados.

Além disso, o REUSE+ possibilita a expansão - ampliação de escopo - de consultas automaticamente, por meio do sistema léxico Wordnet [27] (ver Seção 3.1.1). Com a ajuda do Wordnet, o sistema buscará sinônimos e hiperônimos (relação *é-um-tipo-de*) para cada termo consultado, realizando com eles novas consultas ao repositório de componentes. Além disso, o Wordnet fornecerá o vocabulário que o sistema disponibilizará para a classificação manual. As consultas não utilizarão vocabulário controlado pela natureza não controlada da informação extraída dos comentários dos componentes.

5.5 Arquitetura do sistema

Seacord [79] aponta que as empresas não adotaram os repositórios de componentes devido à sua característica centralizada. Sistemas centralizados em geral têm baixa escalabilidade e acessibilidade, além de trabalharem com excessivo grau de controle e burocracia, o que inibe a reutilização de software em larga escala.

O sistema Agora [80] do SEI/CMU tem como objetivo criar automaticamente um repositório de componentes por meio da indexação de softwares disponíveis na Internet. Ele busca, realiza

introspecção e coleta informações sobre o componente no código-fonte. Dessa forma, um repositório livre, distribuído e inclusivo é criado, o que encoraja as práticas de reutilização.

Além do Agora, trabalhos recentes [17] [95] têm dado maior enfoque a repositórios distribuídos globalmente. Por esse motivo, o projeto de arquitetura do REUSE+ foi construído usando uma arquitetura para sistemas distribuídos e multicamadas: a plataforma J2EE [85].

5.5.1 Projeto Arquitetônico

O projeto arquitetônico resolve os aspectos estratégicos de desenho externo e interno, definindo a divisão do produto em subsistemas e escolhendo as tecnologias mais adequadas [63]. É dividido em sub-atividades [47], a saber:

- *Definição de Plataforma Arquitetural:* escolha da plataforma;
- *Escolha de Fornecedores:* escolha do fornecedor dos produtos que compõem a plataforma;
- *Escolha de Ambiente de Desenvolvimento:* definição das ferramentas de desenvolvimento;
- *Divisão em Camadas Lógicas:* desenho das camadas lógicas da aplicação e aplicação de alguns padrões de projeto [36];
- *Definição dos Pacotes Lógicos:* agrupamento das classes em pacotes segundo critérios de acoplamento e coesão;
- *Definição de Distribuição Física:* alocação dos componentes físicos da aplicação aos nós de processamento (servidores) adequados;
- *Definição de Integração entre Aplicações:* no caso da necessidade de comunicação com outras aplicações (novas ou já existentes), as estratégias para o atendimento deste objetivo devem ser traçadas.

As próximas seções detalham cada aspecto do desenho arquitetônico do sistema REUSE+ e mostram quais tecnologias foram utilizadas para a construção de um sistema de classificação semi-automático que armazena, busca e recupera componentes.

Definição da Plataforma Arquitetural

Foi escolhida a plataforma J2EE, que trata-se de um conjunto de especificações implementáveis por diversos fornecedores. Essa característica aberta possibilita a adoção de ferramentas de desenvolvimento não-pagas. Além disso, com ela é possível desenvolver sistemas distribuídos de maneira elegante e com menor esforço, devido à grande quantidade de bibliotecas de apoio que oferece (segurança, transações, transparência de serviços, entre outras).

Divisão em camadas lógicas

O sistema foi implementado em três camadas, onde cada uma agrupa classes que cooperam para a realização de uma mesma responsabilidade de alto nível. O padrão arquitetural que reforça esse conceito é o MVC (Modelo - Visão - Controle) [30], que nomeia as três camadas e sua forma de interação. A divisão será mapeada para a plataforma J2EE e suas tecnologias que são suporte ao desenvolvimento em camadas.

Definição dos pacotes lógicos

De acordo com as funcionalidades do sistema REUSE+, foram definidos os pacotes lógicos que consideram a divisão de trabalho entre as classes e suas afinidades. Foram identificados três tipos de trabalho: Classificação, Busca e Recuperação de componentes. Eles compõem a divisão lógica de pacotes que fazem parte das diferentes camadas;

Definição de Integração entre Aplicações

foram usadas aplicações já existentes para a construção do protótipo REUSE+: Apache Lucene [29] e sua sandbox, Wordnet [27] e XML Digester [28]. O Apache Lucene cuida da indexação e recuperação das informações contidas nas facetas. A sandbox do Lucene oferece integrações com outras ferramentas importantes para a indexação, tais como removedores de stop words e de afixos. O Wordnet é a base de conhecimento genérica utilizada para sugestão de correções em consultas e expansão automática por sinônimos. O Digester é a biblioteca utilizada para manipular arquivos XML de acordo com regras pré-determinadas. Seu uso deve-se à adoção de XML para a codificação das facetas, o que dá ao sistema maior portabilidade. Conforme recomendado no modelo de classificação, Seção 4.3.1, um banco de dados foi utilizado para armazenar os atributos. Os bancos de dados não são considerados sistemas de integração, pois espera-se que todos os sistemas manipulem dados,

sejam eles gerenciados ou não por algum sistema.

O desenho arquitetônico final, baseado nos itens anteriores, é apresentado na Figura 5.5.

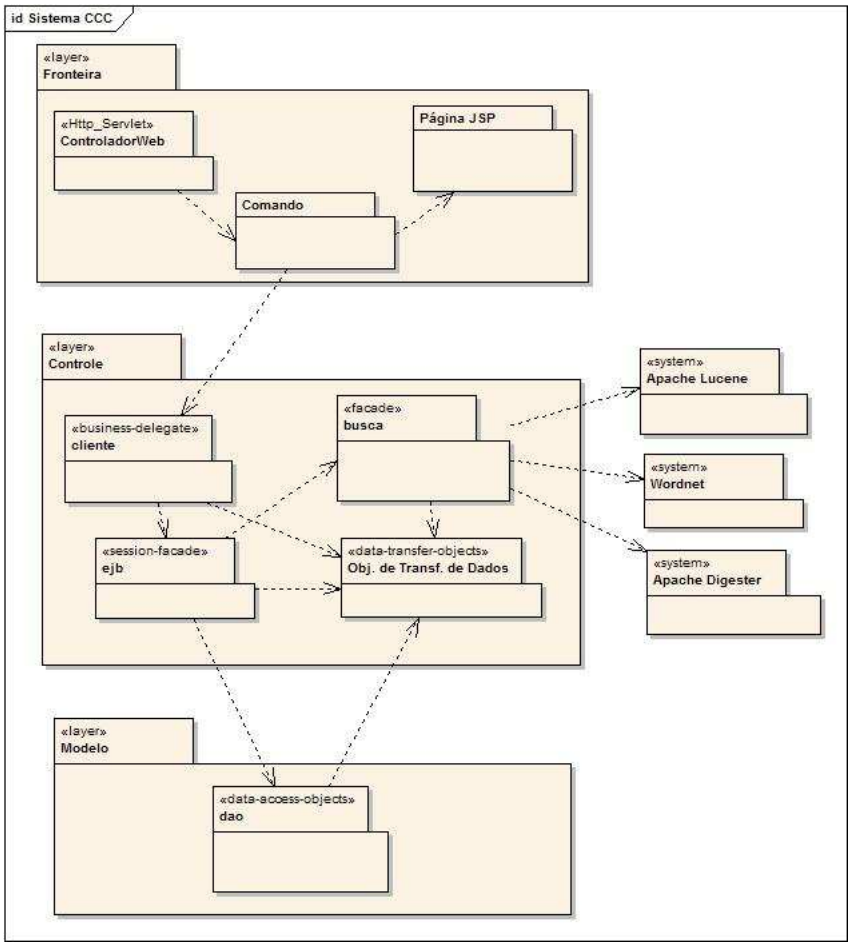

Figura 5.5: Arquitetura MVC do sistema REUSE+

5.6 Exemplo de uso - Componente Cinema

REUSE+ é um sistema Web, acessado por qualquer cliente via navegador. Quando acessa o sistema, o usuário pode ter três diferentes perfis: **classificador**, **usuário final** e o **administrador**. O classificador tem como objetivo classificar e armazenar componentes identificados como reusáveis

no repositório. O usuário final tem como objetivo usar a ferramenta como apoio seu processo de reuso e o administrador o de manter o repositório organizado.

5.6.1 Perfil Administrador

Para o usuário administrador do REUSE+ interessa manter as facetas e atributos existentes no modelo de classificação. Para isso, o usuário precisa de permissão para acessar essa área restrita. A Figura 5.6 mostra o usuário *Claudia* logado no REUSE, exercendo o papel de administrador.

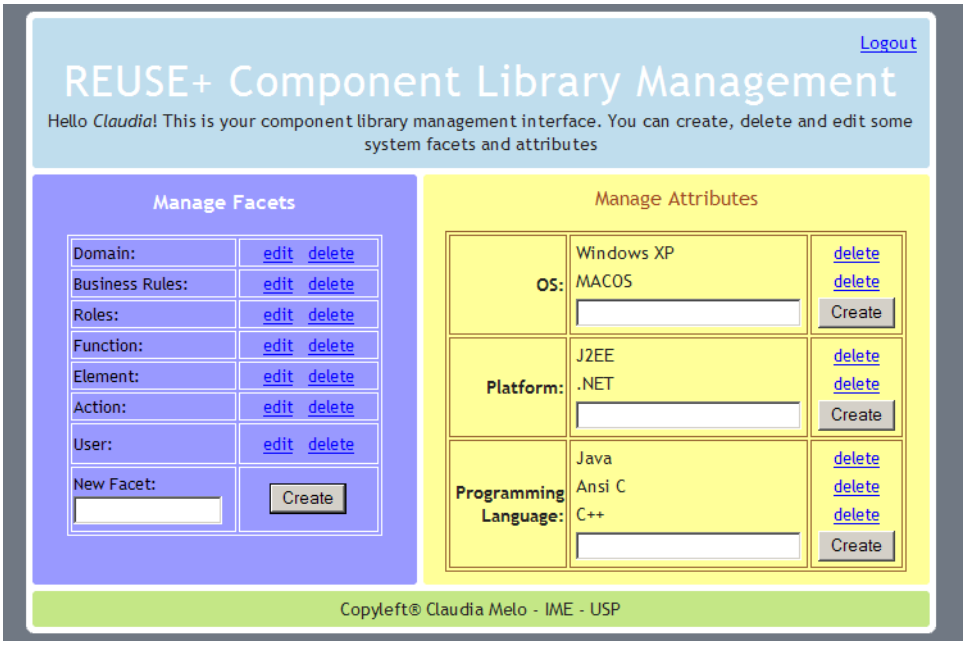

Figura 5.6: Ambientes do administrador do REUSE+

Se uma nova faceta for criada, sua descrição no momento da classificação deverá ser manual, pois o sistema não dá suporte de automatização para novas facetas. Se uma faceta for removida, todas as informações que a descrevem armazenadas no repositório serão removidas. Durante a edição, apenas o nome da faceta é alterado em todas as ocorrências do repositório.

O mesmo processo pode ser aplicado aos atributos. Quando um novo atributo é criado, agrega-se ao sistema a capacidade de filtrar resultados de acordo com a necessidade do usuário final. Faz parte do papel do administrador estudar que atributos são interessantes adicionar dentro das famílias de

atributos existentes (*OS, Plattform, Programmin Language*).

5.6.2 Perfil Classificador

O usuário classificador tem o papel de alimentar o sistema de componentes reutilizáveis. Para se beneficiar dos mecanismos que o REUSE+ oferece, o usuário deve submeter o código-fonte do componente. Caso contrário, as funcionalidades de automatização que dependem da documentação do código-fonte são perdidas (ver Seção 5.2). É importante que o classificador conheça bem o domínio da aplicação do componente que está classificando para não criar descrições incompatíveis ou mesmo não esperadas pelos usuários finais. O REUSE+ utiliza expansão automática de consulta justamente para suavizar os efeitos desse problema.

A Figura 5.7 mostra a interface disponibiliza para este usuário. Ela é dividida em dois ambientes:

1. *Seção de Upload e Classificação das Facetas:* o usuário final utiliza esse ambiente para submeter o componente que deseja classificar e descrever as facetas que faltarem. No exemplo da Figura 5.7, o usuário submete o código-fonte do componente `SmartTicketFacadeLocal` escrito em Java. O REUSE+ informa que recebeu o componente e confirma o acesso ao código-fonte. A partir desse momento, o mecanismo de classificação começa a agir, analisando as informações disponíveis para o preenchimento automático das Facetas *Function, Element/Part* e *Action*. Por fim, o sistema deixa a cargo do usuário o preenchimento das demais facetas ainda em branco;

2. *Seção de Descrição dos Atributos:* nessa seção o classificador pode atribuir algumas características pré-determinadas ao componente, como o Sistema Operacional em que opera. Alguns atributos podem ser obtidos automaticamente do componente. No caso da Figura 5.7, a Identificação, Versão e Data da Versão foram obtidos do código-fonte e preenchidos automaticamente.

Em qualquer momento o usuário poderá submeter a classificação do componente, sendo que nenhuma das facetas ou atributos são obrigatórios.

5.6.3 Perfil Usuário Final

Para o usuário final, o ambiente do REUSE+ é dividido em três partes (como mostra a Figura 5.8):

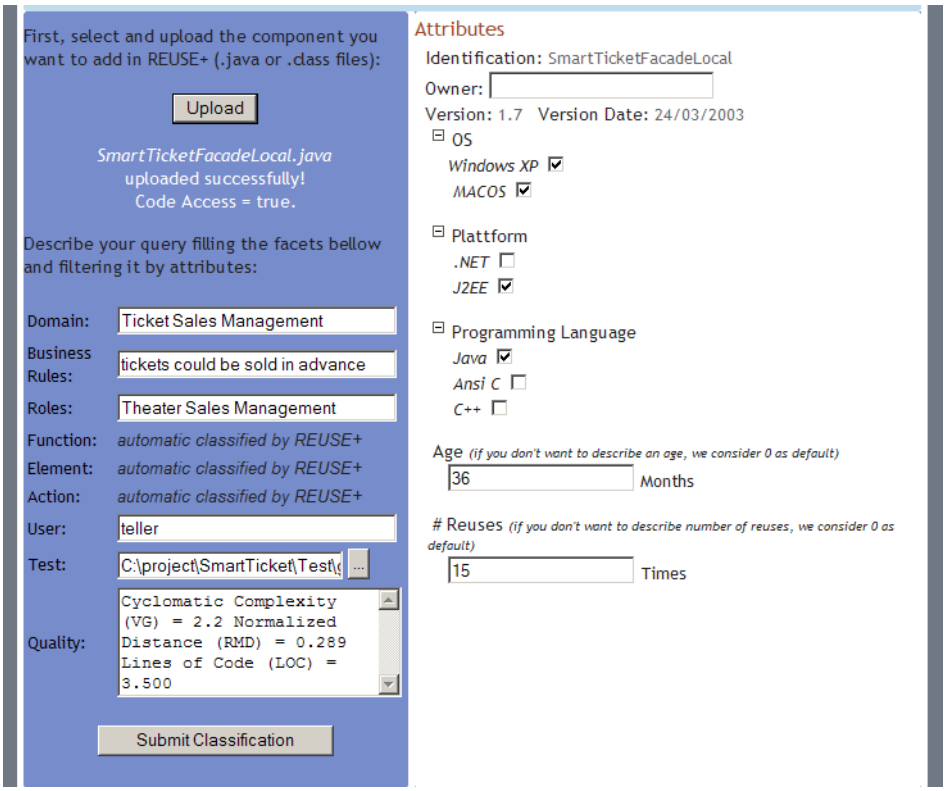

Figura 5.7: Ambientes do classificador de componentes no REUSE+

1. *Seção de Consulta:* o usuário final utiliza esse ambiente para criar as consultas do repositório do REUSE+, guiado pelas facetas disponíveis;

2. *Seção de Filtro:* nessa seção é possível restringir as consultas por meio de filtros realizados pelos possíveis atributos de um componente;

3. *Seção de Resultados:* nessa seção o sistema exibe os resultados obtidos após a consulta ao Repositório do REUSE+. Os resultados são ordenados pelo grau de similaridade com a consulta realizada.

Supondo que o usuário se interesse pelos resultados apresentados e decida avaliar os detalhes de certo componente, o sistema possibilita que ele navegue para outra página, na qual toda a classificação disponível é apresentada. A Figura 5.9 demonstra a interface e os dados referentes ao detalhamento

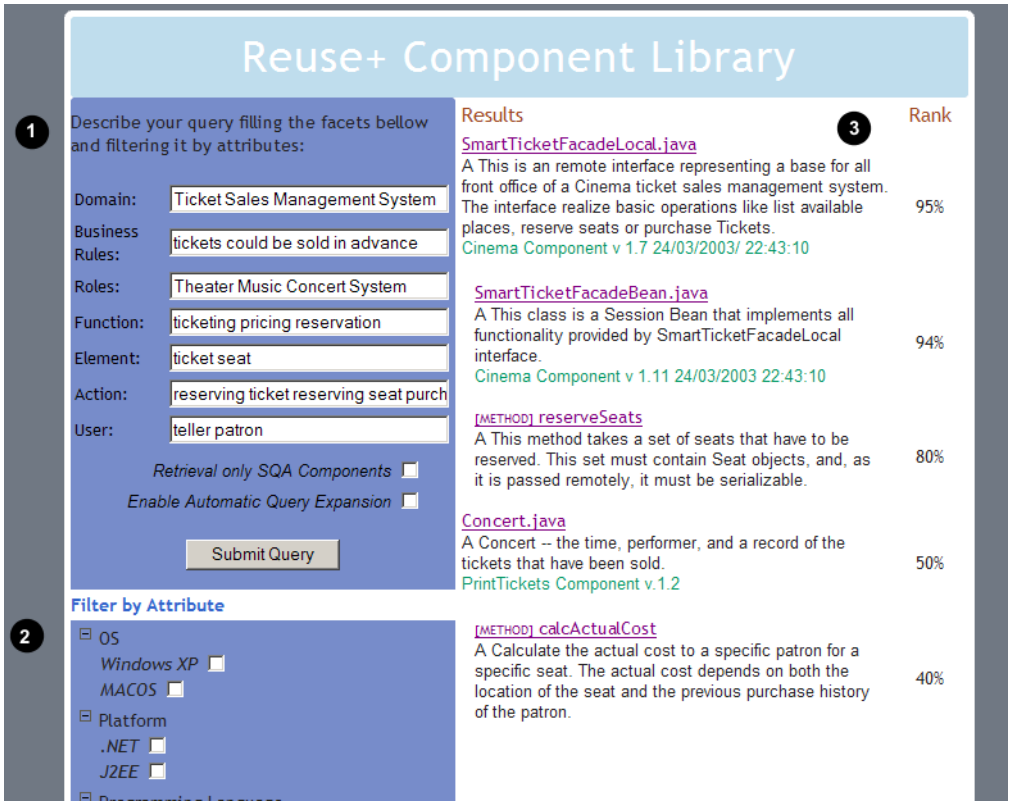

Figura 5.8: Ambientes do usuário final no REUSE+

do componente `SmartTicketFacadeLocal`, o mais similar à consulta que foi encontrado.

5.7 Exemplo de uso

5.7.1 A coleção de testes

Criar uma coleção de testes é uma tarefa essencial para a avaliação de um sistema de RI que requer tempo e esforço adicionais. A área científica que estuda RI organizou, durante seus eventos TREC[3] e Workshop TIPSTER [4], a coleção de testes TIPSTER, que reúne milhares de documentos, consultas e julgamentos de relevância prontos. Dessa maneira, é possível realizar testes automatizados e de

[3] http://trec.nist.gov
[4] http://www-nlpir.nist.gov/related_projects/tipster

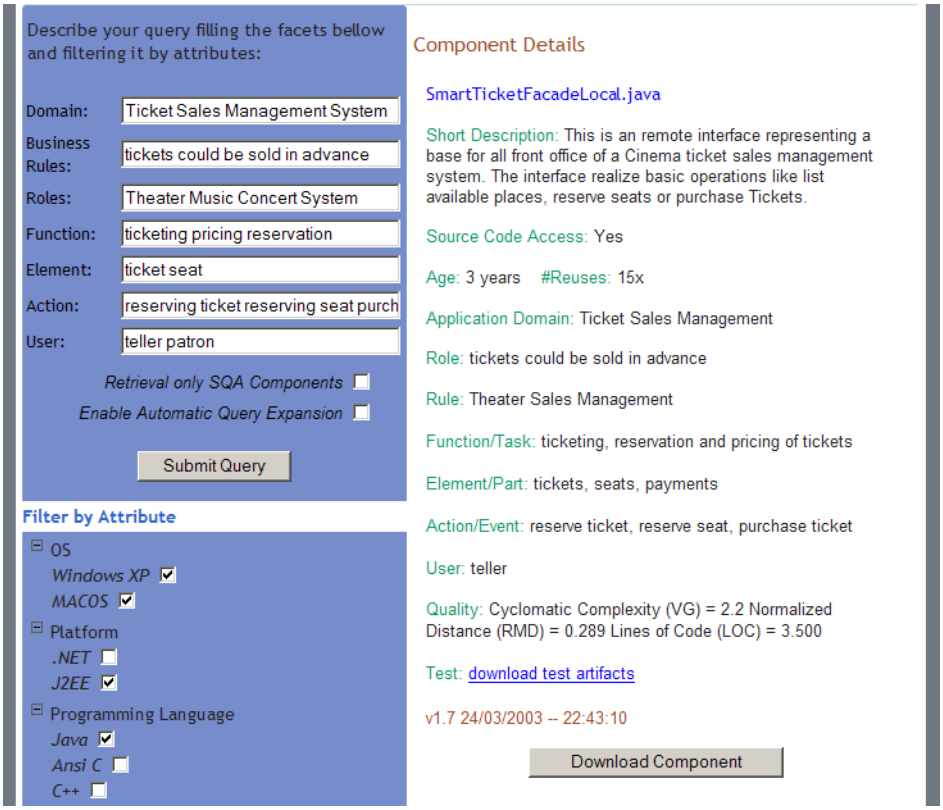

Figura 5.9: Informações detalhadas sobre o componente

maior escala, além de possibilitar a comparação de resultados com outros pesquisadores.

Porém, na subárea de classificação e recuperação de software por meio de técnicas de RI, não existe nenhuma biblioteca similar ao TIPSTER, tornando necessária a construção de uma coleção com essa finalidade.

O Conjunto de Componentes

Foram classificados e armazenados um total de 50 componentes por meio do REUSE+. Eles fazem parte de diferentes domínios, predominantemente o Financeiro e o de Automação de Vendas.

As Consultas de Teste

Para efeito de teste do REUSE+, foram descritos 20 pares de consulta `<consulta,componente relevante>`. As consultas estão divididas em dois tipos:

- **SE:** consultas compostas por palavras-chave e filtros de atributos, aplicadas *sem* expansão automática;
- **CE:** consultas compostas por palavras-chave e filtros de atributos, aplicadas *com* expansão automática;

Ambos os grupos não consideram obrigatório o filtro de Garantia de Qualidade de Software (facetas *Test* e *Quality*), representado pelo check da interface de usuário *"Retrieve only SQA Components"*.

As consultas foram definidas a partir do conhecimento prévio da coleção de componentes, de maneira a possibilitar o julgamento de relevância necessário para compor o par `<consulta,componente relevante>`. Apesar de impraticável em grandes coleções, para cada consulta de teste houve uma classificação manual dos componentes que são relevantes ou irrelevantes em relação àquela consulta.

5.7.2 Tipos de Avaliação

Na fase de avaliação são feitos testes e medições capazes de certificar que dado método ou ferramenta atingiu seu objetivo inicial. Para o protótipo REUSE+ foram inicialmente avaliadas as seguintes características:

1. *Precisão:* número de componentes realmente utilizáveis frente ao número de componentes obtidos no processo de seleção (ver Seção 3.5).
2. *Cobertura:* número de componentes relevantes obtidos no processo de seleção versus o número total de candidatos que obedecem aos requisitos (ver Seção 3.5).
3. *Usabilidade:* A usabilidade mede a eficácia, eficiência e satisfação com que um usuário pode realizar um conjunto específico de tarefas em um ambiente particular [44].

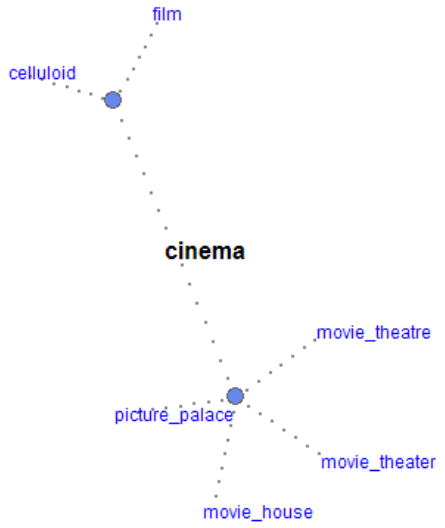

Figura 5.10: Termo *Cinema* Expandido.

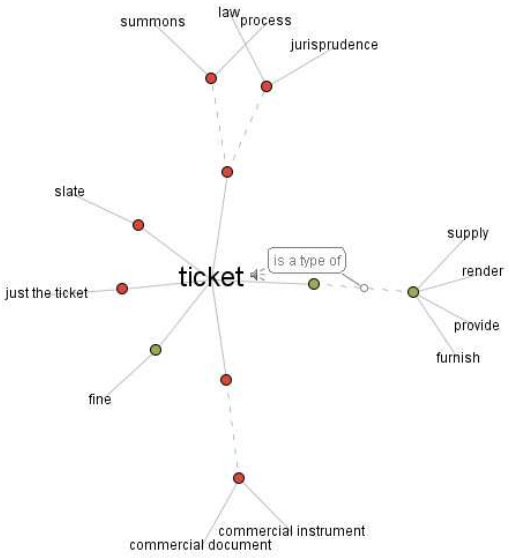

Figura 5.11: Termo *Ticket* Expandido.

Precisão e Cobertura

Para realizar os testes de cobertura e precisão foi usada a mesma técnica empírica utilizada em experimentos similares que baseia-se na prévia identificação do par `<consulta,componente relevante>` e posterior submissão da consulta e coleta de resultados. Com esse processo é possível calcular a Precisão (P) e a Cobertura (C) do sistema. Obviamente um maior número de pares `<consulta,componente relevante>` possibilitam a geração de mais resultados e torna a estimativa mais confiável.

Para esses testes também foi aplicado o mecanismo de expansão de consultas - realizado automaticamente, bastando que o usuário explicite que o deseja. Conforme citado na Seção 5.4, o WordNet [27] foi utilizado como recurso lingüístico para determinar os sinônimos e hiperônimos dos termos da consulta. Exemplos de utilização do Wordnet são mostrados na Figura 5.10 e 5.11. No primeiro caso, apenas sinônimos foram encontrados e, no segundo caso, existem também hiperônimos (ligados ao termo pela linha tracejada).

Para determinar a melhor configuração da expansão automática, foram considerados os resultados obtidos em [71], que mostraram que a expansão com sinônimos e/ou hiperônimos de primeiro nível melhora a cobertura. Eles também mostraram que a expansão com sinônimos associados ao

significado mais comum das palavras-chave das consultas melhora a a cobertura e a precisão para os primeiros 20, 30, 40 e 50 resultados ordenados da consulta. Sendo assim, a expansão do REUSE+ utiliza sinônimos e hiperônimos de primeiro nível (onde está o mais freqüente sentido de um termo). Baseado também nos resultados de [71], o número de sinônimos e hiperônimos usados limita-se cada um a 2 (dois). O motivo para isso está no equilíbrio entre o desempenho do sistema e o ganho obtido durante a expansão.

Testes de Usabilidade

A ferramenta foi submetida ao uso por profissionais e estudantes da área de computação, com ou sem experiência em desenvolvimento baseado em componentes (CBD) ou Java. O objetivo foi identificar problemas básicos de interação humano-computador do protótipo e a satisfação do usuário.

Para avaliar a experiência do usuário com o REUSE+ foram aplicados questionários criados pelo Projeto Ergolist [88], que tem por objetivo determinar a ergonomia de interfaces humano-computador (ver Apêndice A). Ergonomia é o conjunto de conhecimentos necessários para conceber ferramentas, máquinas e dispositivos que possam ser utilizados com o máximo de conforto, segurança e eficiência [5].

O Ergolist disponibiliza na web uma série *checklists*[5] para avaliação de conformidade ergonômica de software. Eles são divididos de acordo com os critérios definidos em [5], totalizando oito critérios principais: condução, carga de trabalho, controle explícito, adaptabilidade, gestão de erros, consistência, significância dos códigos e compatibilidade. Dentre os subcritérios foram escolhidos:

- *Legibilidade:* diz respeito às características lexicais das informações apresentadas na tela que possam dificultar ou facilitar a leitura dessa informação (brilho do caractere, contraste letra/fundo, tamanho da fonte, espaçamento entre palavras, espaçamento entre linhas, espaçamento de parágrafos, comprimento da linha, etc.). Por definição, o critério Legibilidade não abrange mensagens de erro ou de *feedback*.
- *Densidade Informacional:* diz respeito à carga de trabalho do usuário de um ponto de vista perceptivo e cognitivo, com relação ao conjunto total de itens de informação apresentados aos usuários, e não a cada elemento ou ítem individual.

[5] *Checklists* são usados para compensar as falhas da memória humana em assegurar consistência e completude na execução de uma tarefa. Geralmente são listas com perguntas ou itens essenciais que devem ser checados durante a execução.

- *Feedback:* diz respeito às respostas do sistema às ações do usuário. Tais entradas podem ir do simples pressionar de uma tecla até uma lista de comandos. Em todos os casos, respostas do computador devem ser fornecidas, de forma rápida, com passo (*timing*) apropriado e consistente para cada tipo de transação. De todo modo, uma resposta rápida deve ser fornecida com informações sobre a transação solicitada e seu resultado.

- *Experiência do Usuário:* diz respeito aos meios implementados que permitem que o sistema respeite o nível de experiência do usuário.

- *Consistência:* refere-se à forma na qual as escolhas na concepção da interface (códigos, denominações, formatos, procedimentos, etc.) são conservadas idênticas, em contextos idênticos, e diferentes, em contextos diferentes.

Os *checklists* disponíveis estão acompanhados de definições e explicações mais detalhadas e um laudo técnico é gerado a partir das respostas dos avaliadores.

5.7.3 Resultados

Nesta seção são apresentados os resultados dos experimentos realizados com o protótipo REUSE+, tendo como objetivo avaliá-lo quando à precisão, cobertura e usabilidade do sistema.

Testes de Precisão e Cobertura

A coleção de testes criada foi submetida ao REUSE+, que calculou e armazenou as medidas de precisão e cobertura para diferentes graus de similaridade. Para efeitos de teste, não foi aplicado nenhum fator limitante de similaridade entre as consultas e os componentes do REUSE+.

A Figura 5.12 apresenta os resultados obtidos após o experimento com as consultas de teste **SE** e o conjunto de componentes descritos anteriormente.

A Figura 5.13 apresenta os resultados obtidos após o experimento com as consultas de teste **CE** e o conjunto de componentes *versus* o resultado obtido com as consultas de teste **SE**.

Os resultados obtidos apontam que a expansão de fato aumentou a cobertura do sistema, o que fatalmente compromete a precisão - fato este bem conhecido na literatura de RI [71]. Seria importante comparar os resultados obtidos nos experimentos do REUSE+ com os do CKBR-P [91] que também implementa um modelo de classificação para componentes de negócio, porém totalmente manual.

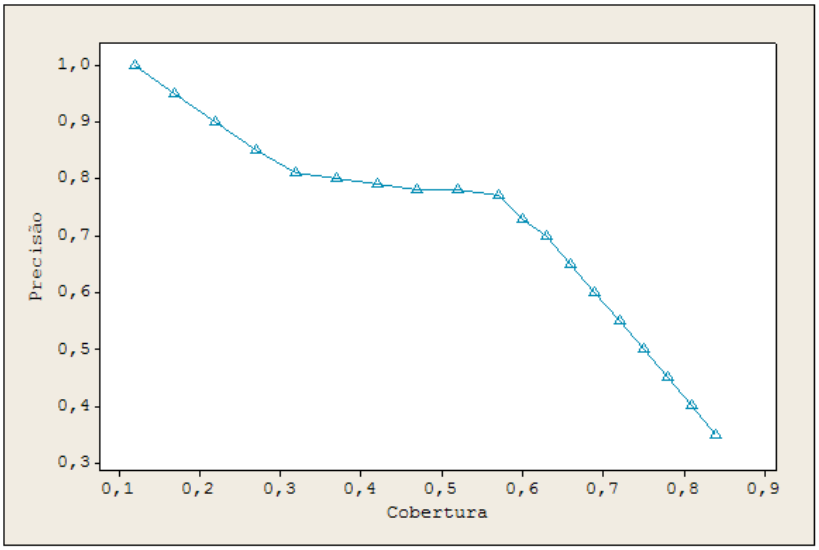

Figura 5.12: Curva de Precisão x Cobertura - Consultas **SE**

Porém, existem os seguintes impeditivos:

- Os resultados de [91] estão focados em avaliar a relação entre as métricas de recuperação (cobertura e precisão), os diferentes níveis de tarefas do usuário e as versões do protótipo (tratamento e controle). Já os experimentos conduzidos neste trabalho tem como objetivo demonstrar as melhorias trazidas por um mecanismo semi-automático de classificação, além do uso de um modelo mais rico em informações sobre componentes de software.
- Mesmo que o foco fosse o mesmo, sistemas de RI só podem ser comparados quando todas as consultas e critérios de relevância são iguais. Neste caso, seria necessário obter a coleção de testes de [91], que não está disponível.
- Outra possibilidade seria usar o Java SDK, coleção representada pela API da linguagem Java, utilizada em vários experimentos de classificação. Porém, classificar essa coleção significaria abrir mão de todas as facetas de maior nível de abstração - um dos maiores focos deste trabalho.

Mesmo sem essa essa comparação, é possível determinar resultados significativos do REUSE+. O tempo médio de classificação foi reduzido em 33,3%, correspondente à proporção das facetas

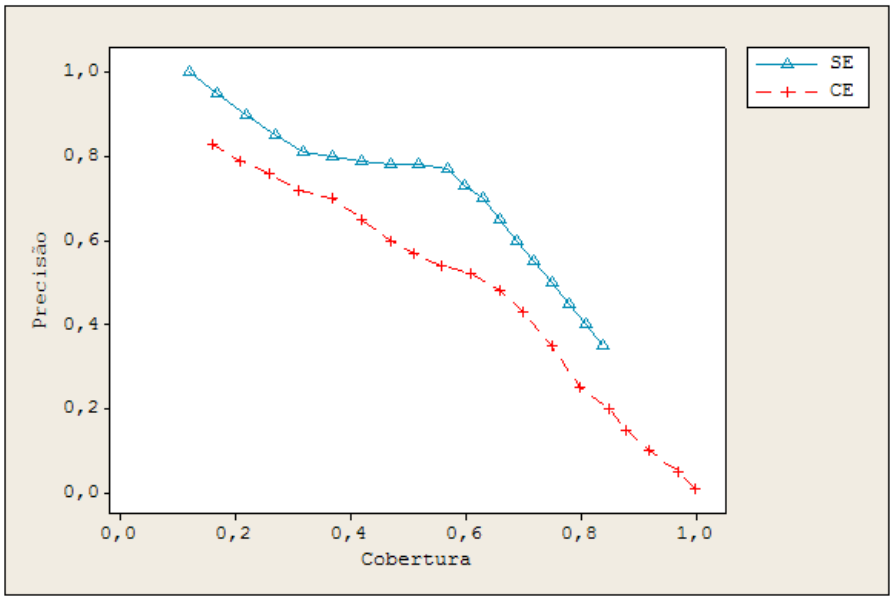

Figura 5.13: Curva de Precisão x Cobertura - Consultas **CE** *versus* **SE**

preenchidas automaticamente. Além disso, os resultados demonstraram que tal automatização não refletiu negativamente na qualidade da classificação, pois os resultados de precisão e cobertura foram satisfatórios se comparados qualitativamente aos demonstrados anteriormente [32].

Testes de Usabilidade

O experimento foi conduzido da seguinte maneira: após a utilização do REUSE+ por dois (2) dias, os usuários foram convidados a responder os questionários selecionados do Ergolist - determinados na seção anterior. Ao todo, quinze (15) usuários participaram, dentre profissionais de Engenharia de Software, gerentes de projeto, desenvolvedores e estudantes de exatas.

O Ergolist oferece um bom suporte ao usuário, contendo explicações detalhadas sobre as perguntas e termos nelas utilizados. Mesmo assim, o experimento de avaliação foi explicado para os participantes, que estiveram acompanhados durante todo o tempo dado. A única orientação durante o preenchimento dos questionários foi quanto ao tempo: eles tiveram uma hora para responder todas as questões. A maior parte dos acontecimentos observados durante o preenchimento do questionário

esteve ligada à correta compreensão de termos específicos da área de ergonomia de interface humano-computador, como o termo "vídeo reverso", que é a inversão dos pixels claros e escuros que formam os caracteres selecionados na tela do computador de forma a chamar a atenção do usuário.

A média dos laudos gerados pelo Ergolist está sumarizado na Figura 5.14.

	Total	Respondidas	Não Respondidas	Conformes	Não Conformes	Não aplicáveis	Adiadas
Legibilidade	27	100%	0%	52%	7%	41%	0%
Experiência do Usuário	6	100%	0%	50%	0%	50%	0%
Consistência	11	100%	0%	82%	0%	18%	0%
Densidade Informacional	9	100%	0%	78%	0%	22%	0%
Feedback	12	100%	0%	50%	25%	25%	0%

Figura 5.14: Resultados do Estudo de Usabilidade

Os resultados obtidos apontam que, dentre as questões que se aplicam ao sistema, a maior parte está conforme, ou seja, de acordo com os padrões de usabilidade. O número elevado de questões não aplicáveis deve-se à abrangência do Ergolist dentro da atividade que se propõe - avaliar ergonomia de interfaces. Embora os resultados sejam positivos, o mecanismo de avaliação necessita de refinamentos. O ambiente de teste não foi controlado por câmeras e não foi feita uma sessão de discussão sobre o método de avaliação. Um maior número de usuários também seria essencial para gerar dados conclusivos sobre o REUSE+ como solução com usabilidade para Reuso de Software.

5.8 Considerações finais

Neste capítulo foi apresentado o protótipo REUSE+, criado para validar o novo modelo de classificação de componentes proposto, juntamente com seu método de automatização. O uso da ferramenta e seus resultados foram demonstrados.

Capítulo 6

Conclusões

Neste capítulo serão apresentadas as conclusões alcançadas no desenvolvimento do trabalho, assim como suas contribuições. As limitações também são discutidas, apontando finalmente para algumas possibilidades de trabalhos futuros.

6.1 Considerações Gerais

Reutilização é uma solução para o aumento da produtividade e diminuição de custos na Engenharia em geral. Especificamente na área de software, o reuso apareceu como uma forte possibilidade de melhoria no cenário de desenvolvimento de projetos, que ainda apresentam fragilidades quanto ao cumprimento de prazos e à qualidade do produto gerado.

Existem porém, por vários fatores descritos ao longo deste trabalho, barreiras à adoção do reuso como prática habitual no desenvolvimento. O foco deste trabalho está ligado ao estudo das ferramentas de apoio ao reuso, ao entendimento dos porquês da não adoção e, por fim, ligado à proposição de melhorias nessas ferramentas. Para isso, foi necessário rever o histórico de pesquisas na área, compreender as tendências atuais e o contexto das demandas do mercado.

Esse estudo revelou que uma das possibilidades de melhoria estava ligada à classificação de componentes, pois o software hoje apresenta maior variedade de nível de abstração e as classificações em geral não acompanharam essa evolução. Dentre elas, uma chamou atenção pelo potencial de classificação de componentes de negócio [91], tendência de mercado fortemente relacionada à tecnologia de *Web Services*. Esse modelo, descrito em detalhes no Capítulo 3, inspirou parte deste trabalho, que procurou aproveitar as vantagens percebidas no modelo de [91], mas também resolver suas limitações - referentes principalmente ao alto custo da classificação e a não preocupação com

aspectos de qualidade.

6.2 Contribuições e Limitações do Trabalho

Este trabalho apresentou um novo modelo de classificação de componentes de software e um método de preenchimento semi-automático baseado em informações contidas no código-fonte e sua documentação. Além disso, um protótipo - REUSE+ - foi construído para validar a proposta. O REUSE+, além de implementar o modelo e o método semi-automático, levou em conta aspectos de expansão de consultas, acessibilidade e usabilidade necessários a um bom repositório de componentes. A arquitetura do REUSE+ foi apresentada em detalhes no Capítulo 5.

A implementação do método semi-automático de classificação foi construído para avaliar código Java. Sua estratégia de extração de informação levou em conta a estrutura da documentação interna do código-fonte, norteada pelo padrão javadoc [84]. Nenhum trabalho até então explorou a semântica obtida por meio desse padrão de documentação, que aponta com maior clareza as funções de cada elemento de um software OO, como classes, métodos, atributos, parâmetros ou exceções. A função é um dos fatores mais importantes na maior parte dos modelos de classificação.

O trabalho procurou detalhar a arquitetura do protótipo REUSE+, cuja proposta se baseia na necessidade de maior acessibilidade do sistema. O REUSE+ é um sistema multiplataforma, distribuído e escalável e sua arquitetura pode ajudar na construção de sistemas maiores com o mesmo propósito.

A preocupação com a qualidade do software foi refletida no modelo que habilita o classificador a inserir métricas de qualidade, tais como as definidas pela ISO 9126 [45]. Além disso, é possível também anexar artefatos de teste ao modelo, sejam eles casos de teste, testes unitários ou de performance.

Por fim, conseguiu-se atingir o último objetivo deste trabalho: validar o modelo e o método por meio do REUSE+. Para tanto, foi definido um conjunto de testes e durante sua execução, algumas métricas foram avaliadas. Elas demonstraram que o sistema, além de melhorar o custo de classificação em cerca de 1/3 se comparado ao de [91], apresentou resultados de recuperação de informação satisfatórios. A usabilidade do sistema também foi avaliada por meio de questionários submetidos aos usuários do REUSE+, também com resultados satisfatórios.

Existem porém, como em qualquer trabalho de pesquisa, limitações em alguns pontos. O primeiro deles está na capacidade limitada do REUSE+ em classificar automaticamente apenas classes iso-

ladas, sem um mecanismo apropriado para lidar com pacotes de classes - estrutura comum para um componente. Essa simplificação ajudou no desenvolvimento do protótipo e na avaliação dos resultados, mas não corresponde totalmente ao modelo proposto.

Outra limitação está na avaliação realizada, que precisaria ser aplicada com um maior número de usuários. Além disso, não foi possível comparar quantitativamente os resultados com os de trabalhos relacionados, pois os dados utilizados por eles não estavam disponíveis.

6.3 Trabalhos futuros

Nesta seção são apresentados alguns pontos identificados como oportunidade de pesquisas e aprofundamento. Alguns deles são conseqüência direta das limitações descritas anteriormente.

Mecanismo de empacotamento de classes
Para resolver limitações do REUSE+ em trabalhar com componentes de fato, um mecanismo de empacotamento de classes seria interessante. Em [92] foi realizado um estudo não apenas sobre empacotamento, mas também sobre a criação automática de novos componentes por meio da análise do código-fonte de um conjunto de classes e da criação automática de interfaces que o definem. Neste caso, o mecanismo de classificação precisaria ser alterado para trabalhar com múltiplas classes e garantir a classificação de todas aquelas que fazem parte de um mesmo componente.

Mecanismo de Inferência
Apesar de descrito brevemente no modelo, o REUSE+ não implementa nenhum mecanismo de inferência sobre o conhecimento obtido durante a classificação. Esta seria uma oportunidade de extensão da ferramenta.

Integração com outros fases do ciclo de vida do desenvolvimento:
Seria interessante possibilitar o desenvolvimento de componentes *para* e *com* reuso por meio da integração com ferramentas CASE e ferramentas de gestão de configuração. Dessa forma, o desenvolvedor teria um ambiente unificado para desenvolver novos componentes, classificá-los, buscá-los e modificá-los de maneira totalmente conectada ao repositório, tornando-o mais acessível, completo e consistente.

Geração automática de métricas de qualidade:
Muitas oportunidades de pesquisa estão associadas ao desenvolvimento baseado em componentes e, especificamente no contexto de classificação, seria interessante poder gerar automaticamente métricas de qualidade a partir do código-fonte do componente. Outra possibilidade é refinar o método de

preenchimento automático das facetas, explorando não só o código-fonte e sua documentação, mas documentações externas.

Extensão do mecanismo de classificação para uma Arquitetura Orientada a Serviços (SOA)

Existe uma forte tendência atual no mercado em enxergar os negócios de maneira isolada da tecnologia que apóiam o negócio. Para lidar com isso, a área de tecnologia precisou migrar para um novo modelo baseado em gestão de serviços. Foi criado o conceito de SOA (*Service Oriented Architecture*), que modela essa relação produtor *versus* consumidor que o mercado demanda. Uma das tecnologias utilizadas para implementar o SOA é a de *Web Services* - Serviços Web. Os *Web Services* são componentes que permitem às aplicações enviar e receber dados em formato XML. Em uma visão mais negocial, os *Web Services* representam o serviço que deseja-se contratar para dar suporte ao negócio. Surge então uma outra possibilidade de pesquisa: estender o modelo e o método semi-automático de classificação para lidar com *Web Services*.

Classificação de Modelos

Com o surgimento da MDA (*Model Driven Architecture*), uma nova arquitetura de desenvolvimento de sistema foi introduzida. A MDA fortalece os conceitos de reutilização generativa (visto no Capítulo 2), onde o objeto de reuso é um modelo. Nessa arquitetura, o código-fonte é totalmente gerado a partir de modelos, elevando o desenvolvedor para um nível mais alto de abstração - o de projeto de software. Essa abordagem também sofre do problema de se encontrar os modelos apropriados para atender certos requisitos. Uma possibilidade de estudo seria estender a classificação proposta para lidar com modelos (diagramas) de sistemas.

Apêndice A

Questionário de usabilidade

Este anexo descreve o questionário de usabilidade aplicado na avaliação do protótipo REUSE+, criado a partir do checklist disponibilizado por [88].

A.1 Questionário de legibilidade

Avaliação de Legibilidade do REUSE+
Data:
Cargo/Ocupação atual:
Instruções de preenchimento:
(1) Você deve responder todas as perguntas em no máximo 1 hora;
(2) Você deve selecionar apenas uma resposta para cada pergunta.

1. As áreas livres são usadas para separar grupos lógicos em vez de tê-los todos de um só lado da tela, caixa ou janela?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

2. Os grupos de objetos de controle e de apresentação que compõem as caixas de diálogo e outros objetos compostos encontram-se alinhados vertical e horizontalmente?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

3. Os rótulos de campos organizados verticalmente e muito diferentes em tamanho estão justifi-

cados à direita?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

4. A largura mínima dos mostradores de texto é de 50 caracteres?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

5. A altura mínima dos mostradores de texto é de 4 linhas?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

6. Os parágrafos de texto são separados por, pelo menos, uma linha em branco?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

7. O uso exclusivo de maiúsculas nos textos é evitado?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

8. O uso do negrito é minimizado?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

9. O uso do sublinhado é minimizado?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

10. Nas tabelas, linhas em branco são empregadas para separar grupos?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

11. As listas de dados alfabéticos são justificadas à esquerda?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

12. As listas contendo números decimais apresentam alinhamento pela vírgula?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

13. As linhas empregadas para o enquadramento e segmentação de menus (separadores, delimitadores etc.) são simples?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

14. As bordas dos painéis dos menus estão suficientemente separadas dos textos das opções de modo a não prejudicar a sua legibi lidade?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

15. O uso de abreviaturas é minimizado nos menus?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

16. Os nomes das opções estão somente com a inicial em maiúsculo?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

17. Os números que indicam as opções de menu estão alinhados pela direita?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

18. Se a enumeração alfabética é utilizada, então as letras para seleção estão alinhadas pela esquerda?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

19. As opções de uma barra de menu horizontal estão separadas por, no mínimo, 2 caracteres brancos?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

20. Os rótulos de campos começam com uma letra maiúscula, e as letras restantes são minúsculas?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

21. Os itens de dados longos são particionados em grupos mais curtos, tanto nas entradas como nas apresentações?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

22. Os códigos alfanuméricos do sistema agrupam separadamente letras e números?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

23. Os ícones são legíveis?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

24. O sistema utiliza rótulos (textuais) quando pode existir ambiguidade de ícones?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

25. A informação codificada com o vídeo reverso está sempre legível?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

26. O uso de vídeo reverso está restrito à indicação de feedback de seleção?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

27. Os dados a serem lidos são apresentados de forma contínua, não piscantes ?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

A.2 Questionário de Experiência do Usuário

Data:
Cargo/Ocupação atual:
Instruções de preenchimento:
(1) Você deve responder todas as perguntas em no máximo 1 hora;
(2) Você deve selecionar apenas uma resposta para cada pergunta.

1. Caso se trate de um sistema de grande público, ele oferece formas variadas de apresentar as mesmas informações aos diferentes tipos de usuário?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

2. Os estilos de diálogo são compatíveis com as habilidades do usuário, permitindo ações passo-a-passo para iniciantes e a entrada de comandos mais complexos por usuários experimentados?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

3. O usuário pode se deslocar de uma parte da estrutura de menu para outra rapidamente?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

4. O sistema oferece equivalentes de teclado para a seleção e execução das opções de menu, além do dispositivo de apontamento (mouse,...)?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

5. O sistema é capaz de reconhecer um conjunto de sinônimos para os termos básicos definidos na linguagem de comando, isto para se adap tar aos usuários novatos ou ocasionais?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

6. O usuário experiente pode efetuar a digitação de vários comandos antes de uma confirmação?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

A.3 Questionário de Consistência do REUSE+

Data:
Cargo/Ocupação atual:
Instruções de preenchimento:
(1) Você deve responder todas as perguntas em no máximo 1 hora;
(2) Você deve selecionar apenas uma resposta para cada pergunta.

1. A organização em termos da localização das várias características das janelas é mantida consistente de uma tela para outra?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

2. A posição inicial do cursor é mantida consistente ao longo de todas as apresentações de formulários?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

3. Uma mesma tecla de função aciona a mesma opção de uma tela para outra?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

4. Os ícones são distintos uns dos outros e possuem sempre o mesmo significado de uma tela para outra?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

5. A localização dos dados é mantida consistente de uma tela para outra?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

6. Os formatos de apresentação dos dados são mantidos consistentes de uma tela para outra?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

7. Os rótulos estão na mesma posição em relação aos campos associados?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

8. O símbolo para convite à entrada de dados é padronizado (por exemplo ": ")?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

9. As áreas de entrada de comandos estão na mesma posição de uma tela para outra?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

10. Os significados dos códigos de cores são seguidos de maneira consistente?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

A.4 Questionário de Densidade informacional do REUSE+

Data:
Cargo/Ocupação atual:
Instruções de preenchimento:
(1) Você deve responder todas as perguntas em no máximo 1 hora;
(2) Você deve selecionar apenas uma resposta para cada pergunta.

1. A densidade informacional das janelas é reduzida?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

2. As telas apresentam somente os dados e informações necessários e indispensáveis para o usuário em sua tarefa?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

3. Na entrada de dados codificados, os códigos apresentam somente os dados necessários estão presentes na tela de uma maneira distinguível?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

4. O sistema minimiza a necessidade do usuário lembrar dados exatos de uma tela para outra?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

5. Na leitura de uma janela, o usuário tem seus movimentos oculares minimizados através da distribuição dos objetos principais segundo as linhas de um "Z"?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

6. O sistema evita apresentar um grande número de janelas que podem desconcentrar ou sobre-

carregar a memória do usuário?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

7. Na manipulação dos dados apresentados pelo sistema, o usuário está liberado da tradução de unidades?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

8. As listas de seleção e combinação apresentam uma altura correspondente a um máximo de nove linhas?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

9. Os painéis de menu apresentam como ativas somente as opções necessárias?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

A.5 Questionário de Feedback do REUSE+

Data:
Cargo/Ocupação atual:
Instruções de preenchimento:
(1) Você deve responder todas as perguntas em no máximo 1 hora;
(2) Você deve selecionar apenas uma resposta para cada pergunta.

1. O sistema fornece feedback para todas as ações do usuário?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

2. Quando, durante a entrada de dados, o sistema torna-se indisponível ao usuário, devido a algum processamento longo, o usuário é avisado desse estado do sistema e do tempo dessa indisponibilidade?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

3. O sistema fornece informações sobre o estado das impressões?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

4. Os itens selecionados de uma lista são realçados visualmente de imediato?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

5. A imagem do cursor fornece feedback dinâmico e contextual sobre a manipulação direta?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

6. O sistema fornece ao usuário informações sobre o tempo de processamentos demorados?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

7. O sistema apresenta uma mensagem informando sobre o sucesso ou fracasso de um processamento demorado?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

8. O sistema fornece feedabck imediato e contínuo das manipulações diretas?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

9. O sistema define o foco das ações para os objetos recém criados ou recém abertos?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

10. O sistema fornece feedback sobre as mudanças de atributos dos objetos?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

11. Qualquer mudança na situação atual de objetos de controle é apresentada visualmente de modo claro ao usuário?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

12. O sistema fornece um histórico dos comandos entrados pelo usuário durante uma sessão de trabalho?
( )Conforme ( )Não-conforme ( ) Não-aplicável ( )Adiar resposta

Referências Bibliográficas

[1] S. P. Arnold and S. L. Stepoway, *The reuse system: cataloging and retrieval of reusable software*, (1988), 138–141.

[2] David Axmark, Michael Widenius, Jeremy Cole, and Paul DuBois, *Mysql reference manual*, 2001, [Online; accessed 2-Agosto-2006].

[3] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto, *Modern information retrieval*, ACM Press / Addison-Wesley, 1999.

[4] L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, and K. Wallnau, *Market assessment of component based software engineering*, Internal Research and Development (2001).

[5] J. M. C. Bastien and D. L. Scapin, *Preliminary findings on the effectiveness of ergonomic criteria for the evaluation of human-computer interfaces*, Adjunct Proceedings of the Conference on Human Factors in Computing Systems (INTERACT'93) (Amsterdam, The Netherlands), 1993, pp. 187–188.

[6] T. J. Biggerstaff and A. J. Perlis, *Software reusability: vol. 1, concepts and models*, ACM Press, 1989.

[7] T. J. Biggerstaff and C. Richter, *Reusability framework, assessment, and directions*, (1989), 1–17.

[8] RTV Braga, FSR Germano, PC Masiero, and JC Maldonado, *Introdução aos padrões de software*, First Latin American Conference on Pattern Languages of Programming, SugarLoaf-PLoP'2001, October 2001.

[9] T. Bray, J. Paoli, and C. Sperberg-McQueen, *Extensible Markup Language (XML)*, The World Wide Web Journal **2** (1997), no. 4, 29–66.

[10] Alan W. Brown and Kurt C. Wallnau, *The current state of cbse*, IEEE Softw. **15** (1998), no. 5, 37–46.

[11] Islam Choudhury, *A perspective on software reuse*, 1998, [Online; accessed 2-Agosto-2006].

[12] Paul Clements, Linda Northrop, and Linda M. Northrop, *Software product lines : Practices and patterns*, Addison-Wesley Professional, August 2001.

[13] World Wide Web Consortium, *Simple object access protocol*, 2005, [Online; accessed 2-Agosto-2006].

[14] ———, *Web service description language*, 2005, [Online; accessed 2-Agosto-2006].

[15] D. R. Cutting and J. O. Pedersen, *Space optimizations for total ranking*, RAIO Conference, 1997, pp. 401–412.

[16] Jacob L. Cybulski, *Management of reuse libraries*, Seminar Paper (1999).

[17] Glaydson Elias da Silveira, *Spontaneous software: a web-based, object computing paradigm*, ICSE '00: Proceedings of the 22nd international conference on Software engineering (New York, NY, USA), ACM Press, 2000, pp. 719–721.

[18] Massimo D'Alessandro, Pier Luigi Iachini, and Alessandra Martelli, *"the generic reusable component: an approach to reuse hierarchical oo designs"*, IEEE Computer Society Press: Los Alamitos, pp. 39–46.

[19] Eduardo Santana de Almeida, Calebe De Paula Bianchini, Antônio Francisco do Prado, and Luís Carlos Trevelin, *Distributed component-based software development strategy integrated by mvcase tool.*, JIISIC, 2002, pp. 97–106.

[20] P. T. Devanbu, R. J. Brachman, P. G. Selfridge, and B. W. Ballard, *Lassiea knowledge-based software information system*, ICSE '90: Proceedings of the 12th international conference on Software engineering, IEEE Computer Society Press, 1990, pp. 249–261.

[21] Tharam S. Dillon and Poh L. Tan, *Object-oriented conceptual modeling*, Prentice Hall PTR, 1993.

[22] DoD, *Software technology for adaptable reliable systems*, Tech. report, Unisys, STARS Technology Center, Arlington VA 22203, July 1993.

[23] DoD-SRI, *Software reuse initiative: Technologyroadmap,v2.2*, Tech. report, Department of Defense, 1995, V2.2.

[24] Desmond F. D'Souza and Alan Cameron Wills, *Objects, components, and frameworks with uml: the catalysis approach*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[25] Letha H. Etzkorn and Carl G. Davis, *Automatically identifying reusable oo legacy code*, Computer **30** (1997), no. 10, 66–71.

[26] Letha Hughes Etzkorn, *A metrics-based approach to the automated identification of object-oriented reusable software components*, Ph.D. thesis, University of Alabama in Huntsville, Huntsville, AL, USA, 1998.

[27] C. Fellbaum, *Wordnet an electronic lexical database*, 1998.

[28] Apache Software Foundation, *Commons digester*, 2006, [Online; accessed 2-Agosto-2006].

[29] ———, *Lucene java release 2.0*, 2006, [Online; accessed 2-Agosto-2006].

[30] Martin Fowler, *Analysis patterns: reusable objects models*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[31] W. B. Frakes and B. A. Nejmeh, *Ieee tutorial: Software reuse: Emerging technology*, ch. An information system for software reuse, IEEE Computer Society, 1988.

[32] W. B. Frakes and T. P. Pole, *An empirical study of representation methods for reusable software components*, IEEE Trans. Softw. Eng. **20** (1994), no. 8, 617–630.

[33] William Frakes and Carol Terry, *Software reuse and reusability metrics and models*, Tech. Report TR-95-07, 4, 1995.

[34] M.G. Fugini and S. Faustle, *Retrieval of reusable components in a development information system*, IEEE 2nd Workshop on Software Reusability, IEEE Press, 1993, pp. 110–115.

[35] WEISS G. M, *Adaptação de componentes de software para o desenvolvimento de sistemas confiáveis*, Master's thesis, Instituto de Computação - UNICAMP, 2001.

[36] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[37] M. R. Girardi and Bertrand Ibrahim, *A software reuse system based on natural language specifications*, ICCI '93: Proceedings of the Fifth International Conference on Computing and Information, IEEE Computer Society, 1993, pp. 507–511.

[38] Robert L. Glass and Iris Vessey, *Contemporary application-domain taxonomies*, IEEE Softw. **12** (1995), no. 4, 63–76.

[39] Adele Goldberg, *The influence of an object-oriented language on the programming environment*, CSC-83: Proceedings of the 1983 computer science conference, ACM Press, 1983, pp. 35–54.

[40] M Gonzalez and V. Lima, *Recuperação de informação e expansão automática de consulta com thesaurus: uma avaliação*, XXVII Conferencia Latinoamericana de Informatica (CLEI'2001) (2001).

[41] I. Graham, *Object oriented methods*, Addison-Wesley Pub. Co., 1994.

[42] S. Henninger, *Using iterative refinement to find reusable software*, IEEE Softw. **11** (1994), no. 5, 48–59.

[43] K. Inoue, *Component rank: Relative significance rank for software component search*, 2003.

[44] ISO, *En iso 9241-11. ergonomic requirements for office work with visual display terminals (vdt's). part 11*, International Standards Organization, 1997.

[45] ISO/IEC-9126-1, *Part 1: Quality model*, Software engineering - Product quality, Geneva, International Organization for Standardization and International Electrotechnical Commission, 2001.

[46] Peter Jackson and Isabelle Moulinier, *Natural language processing for online applications: Text retrieval, extraction and categorization*, Natural Language Processing, vol. 5, John Benjamins Publishing Company, Amsterdam/Philadelphia, 2002.

[47] Valdo Noronha Péres Júnior, *Estratégias para a utilização da tecnologia j2ee com a arquitetura de cinco camadas*, Master's thesis, UFMG, 2003.

[48] Hsian-Chou Liao and Feng-Jian Wang, *Software reuse based on a large object-oriented library*, SIGSOFT Softw. Eng. Notes **18** (1993), no. 1, 74–80.

[49] J. Lonchamp, *A structured conceptual and terminological framework for software process engineering*, Proceedings of the 2^{nd} International Conference on the Software Process - Continuous Software Process Improvement (Berlin, Germany), 1993.

[50] Andrea De Lucia, *Reuse reengineering*, 1993, [Online; accessed 2-Agosto-2006].

[51] Daniel Lucrédio, Antônio Francisco do Prado, and Eduardo Santana de Almeida, *A survey on software components search and retrieval.*, EUROMICRO, 2004, pp. 152–159.

[52] R. Malan, , and D. Bredemeyer, *Defining non-functional requirements*, 2001.

[53] Tony Mason and Doug Brown, *Lex & yacc*, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1990.

[54] M. D. McIlroy, *Mass produced software components*, Software Engineering, 1969.

[55] B. Meyer, *Reusability: the case for object-oriented design*, (1989), 1–33.

[56] H. Mili, F. Mili, and A. Mili, *Reusing software: Issues and research directions*, Software Engineering **21** (1995), no. 6, 528–562.

[57] Hafedh Mili, Estelle Ah-Ki, Robert Godin, and Hamid Mcheick, *Another nail to the coffin of faceted controlled-vocabulary component classification and retrieval*, SIGSOFT Softw. Eng. Notes **22** (1997), no. 3, 89–98.

[58] R. Mittermeir, H. Pozewaunig, A. Mili, and R. Mili, *Uncertainty aspects in component retrieval*, 1998.

[59] C.N. Moores, *Datacoding applied to mechanical organization of knowledge*, Am. Doc. 2 (1951), 20–32.

[60] Peter Naur and Brian Randell (eds.), *Software engineering: Report of a conference sponsored by the nato science committee, garmisch, germany, 7-11 oct. 1968, brussels, scientific affairs division, nato*, 1969.

[61] James M. Neighbors, *Draco: A method for engineering reusable software systems*, Software Reusability – Concepts and Models (Ted J. Biggerstaff and Alan J. Perlis, eds.), vol. I, ACM Press, 1989, pp. 295–319.

[62] OASIS, *Universal description, discovery integration*, 2005, [Online; accessed 2-Agosto-2006].

[63] Wilson de Pádua Paula Filho, *Engenharia de software - fundamentos, métodos e padrões*, LTC, 2003.

[64] R. Prieto-Diaz, *Classification of reusable modules*, (1989), 99–123.

[65] ———, *Domain analysis: an introduction*, SIGSOFT Softw. Eng. Notes **15** (1990), no. 2, 47–54.

[66] ———, *Implementing faceted classification for software reuse*, Commun. ACM **34** (1991), no. 5, 88–97.

[67] ———, *Status report: Software reusability*, IEEE Softw. **10** (1993), no. 3, 61–66.

[68] R. Prieto-Diaz and P. Freeman, *Classifying software for reusability*, IEEE Software **4** (1987), no. 1, 6–16.

[69] M. R. Quillian, *Semantic memory*, Semantic Information Processing (1968), 227–270.

[70] R.T. Mittermeir R. Mili, A. Mili, *A survey of software storage and retrieval*, Annals of Software Engineering **05** (1998), no. 02, 349–414.

[71] Franklin Ramalho and Jacques Robin, *Avaliação empírica da expansão de consultas baseada em um thesaurus: aplicação em um engenho de busca na web.*, RITA **10** (2004), no. 2, 9–28.

[72] C. V. Ramamoorthy, Vijay Garg, and Atul Prakash, *Support for reusability in genesis*, IEEE Trans. Softw. Eng. **14** (1988), no. 8, 1145–1154.

[73] S.R. Ranghanathan, *Prolegomena to library classification*, (1957).

[74] T. Ravichandran and Marcus A. Rothenberger, *Software reuse strategies and component markets*, Commun. ACM **46** (2003), no. 8, 109–114.

[75] G. Salton, *Automatic text processing: the transformation, analysis, and retrieval of information by computer*, Addison-Wesley Longman Publishing Co., Inc., 1989.

[76] G. Salton and M. Smith, *On the application of syntactic methodologies in automatic text analysis*, SIGIR '89: Proceedings of the 12th annual international ACM SIGIR conference on Research and development in information retrieval, ACM Press, 1989, pp. 137–150.

[77] J. Sametinger, *Software engineering with reusable components*, Springer-Verlag New York, Inc., 1997.

[78] P. R. Santos, *Reutilização de componentes de software com base em identificadores hierárquicos*, Ph.D. thesis, Instituto Superior Tecnico, Maio 2000.

[79] Robert C. Seacord, *Software engineering component repositories*, Tech. report, Software Engineering Institute (SEI), 1999.

[80] Robert C. Seacord, Scott A. Hissam, and Kurt C. Wallnau, *Agora: A search engine for software components*, Tech. report, Software Engineering Institute (SEI), 1998.

[81] Snowball, *Snowball: Quick introduction*, 2005, [Online; accessed 2-Agosto-2006].

[82] L.S. Sorumgard, G. Sindre, and F. Stokke, *Experiences from application of a faceted classification scheme*, Second International Workshop on Software Reusability (Los Alamitos, California) (P.-D. Ruben and Editors. B.F. William, eds.), IEEE Computer Society Press, 1993, pp. 116–124.

[83] Vijayan Sugumaran and Veda C. Storey, *A semantic-based approach to component retrieval*, SIGMIS Database **34** (2003), no. 3, 8–24.

[84] Inc Sun Microsystems, *How to write doc comments for the javadoc tool*, 2005, [Online; accessed 2-Agosto-2006].

[85] ———, *Java 2 platform, enterprise edition (j2ee)*, 2006.

[86] Clemens Szyperski, *Component software: Beyond object-oriented programming*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[87] James E. Tomayko, *Forging a discipline: An outline history of software engineering education*, Ann. Softw. Eng. **6** (1999), no. 1-4, 3–18.

[88] LabIUtil UFSC, *Ergolist checklist*, 2006.

[89] Visual Numerics, Inc, *The imsl product family*, 1995.

[90] Padmal Vitharana, Fatemah "Mariam" Zahedi, and Hemant Jain, *Design, retrieval, and assembly in component-based software development*, Commun. ACM **46** (2003), no. 11, 97–102.

[91] Padmal Vitharana, Fatemeh Mariam Zahedi, and Hemant Jain, *Knowledge-based repository scheme for storing and retrieving business components: A theoretical design and an empirical analysis*, IEEE Trans. Softw. Eng. **29** (2003), no. 7, 649–664.

[92] Hironori Washizaki and Yoshiaki Fukazawa, *Component-extraction-based search system for object-oriented programs.*, ICSR, 2004, pp. 254–263.

[93] Wikipédia, *Objeto — wikipédia, a enciclopédia livre*, 2006, [Online; accessed 2-Agosto-2006].

[94] Wikipedia, *Naics — wikipedia, the free encyclopedia*, 2006, [Online; accessed 02-August-2006].

[95] Yunjiao Xue, Leqiu Qian, Ruzhi Xu, Bin Tang, and Xin Peng, *An adaptive agent-based network for distributed component repositories*, First International Multi-Symposiums on Computer and Computational Sciences **1** (2006), 458–465.

[96] Haining Yao and Letha Etzkorn, *Towards a semantic-based approach for software reusable component classification and retrieval*, ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference, ACM Press, 2004, pp. 110–115.